Health-check Heartbeat App

A high-level specification for an end-to-end health-check for Confluent Cloud

Confluent Cloud ("CC") is supported 24x7 by Confluent's engineers and there are many automated health checks and alerts in place to ensure any problems are investigated immediately and resolved as soon as possible, often before our customers are aware something has gone wrong.
There is more involved in an application than just the cluster though - networks and DNS tend to have faults more frequently than the source application or the destination cluster. Therefore, the best metric to alert on for customer-side on-call teams is an end-to-end health check that ensures every part of the chain from the client hosting location to the CC cluster and back is functioning correctly.

In essence this is simply a case of producing a message every second or so to the CC cluster and ensuring the messages can be consumed as well. This is a tiny amount of throughput so there will be no impact on the cluster and minimal cost, but it will give on-call engineers on the customer side initial alerts to an issue, as well as information to aid diagnosis of where the problem lies.

Table of Contents

Requirements

- A topic on the cluster ("`cc-heartbeat`" used here)
- Credentials to produce into this topic and consume from it

Ensure the topic has the same number of partitions as there are brokers in the cluster. It's not possible to guarantee these will be spread across the brokers evenly as you can't define rack placement in Confluent Cloud, but it's a sane default either way. Retention can be short (e.g. 24 hours) and the Replication Factor needs to be 3 in Confluent Cloud.

Functionality

Startup:

- Read one or more bootstrap IPs + credentials from config file
- Resolve bootstrap & use AdminAPI describe cluster to get nodes / IPs / rack (AZ)
- Ensure `cc-heartbeat` topic exists and has same number of partitions as there are brokers
  The app can create the topic by default but a customer may choose to run with reduced permissions in which case they will have to create it themselves and adjust the partitions if the cluster is resized.

Every second:

- Produce a message into each partition every second with a value containing the timestamp the message was generated. Temporarily store the latency reported for that message.
    - Configure 0 retries with a short timeout.

Every 10 seconds:

- Test network reachability for each broker on port 9092. Temporarily store the round trip times.
- Re-fetch the cluster metadata; flag any changes from the prior metadata and trigger the producer and consumer to update / reconnect.

Start a Consumer, and for each message received calculate the end-to-end latency from (current time - message payload time). Temporarily store the latency per message.

Start an HTTP listener to respond to polls from Prometheus in OpenTelemetry format.
The output should be counts of metrics since the last Prometheus scrape: (so clear the store after each scrape)

- Label all metrics with the clusterId (`lkc-abcde`) to ensure all the metrics can be related to the correct cluster
- Broker count
- Number of messages produced
- Number of messages consumed
- Average end-to-end latency for messages consumed

- For each broker:
    - Label: broker name
    - Label: broker rack (from `describe_cluster`)
    - Number of connectivity checks performed
    - Number of connectivity checks failed / timed out
    - Average latency for the successful checks

The producer and consumer should be long-running to avoid the overhead of recreating connections or consumer group rebalancing noise, but when the cluster experiences changes (such as a failed broker) there will be rebalancing and the heartbeat app needs to cope with this. It may be best to periodically re-fetch the metadata

Consumer Group Lag

Given this app has an AdminAPI session, it would be great to offer consumer group lag monitoring as well. It can be included in the Prometheus metrics exporter and offer much more responsive and fine-grained lag monitoring than is available with CC Metrics API.
Performance for this could get slow on a large cluster so it should be implemented in such a way that it doesn't affect the heartbeat health check functionality at all.