Kafka Synth Client

Table of Contents

1. Introduction

Kafka-synth-client is a simple Kafka client (producer+consumer) that measures end-to-end latency of using Kafka. The latencies are measured by sending messages to the given Kafka topic via all the brokers, consuming them and measuring the time it took for the message to be produced and consumed. The recorded latencies are reported in prometheus format on a metrics endpoint (`/q/metrics`). The metrics contain values for the median, percentiles like 95th percentile, and 99th percentile of the latencies.

If the client is given the required ACLs, it will report latencies per broker. It is also able to automatically increase the number of partitions in a topic to match the number of brokers and automatically reassigns partitions to brokers to ensure that each broker will be produced to and consumed from.

2. Getting started

2.1. Using docker-compose

Use the docker compose file found at the root of this repository.

Start everything using

```
docker-compose up -d
```

This will start a kafka broker, the kafka-synth-client and prometheus so you can visualize the metrics. Prometheus will be available at `http://localhost:9090`. You can try the following queries:

```
synth_client_e2e_latency_ms
```

If you want to test against your own kafka cluster, be sure to change the kafka parameters in the docker-compose file. Everything prefixed with `KAFKA_` will be used in the kafka client.

Refer to the chapter Parameters for more information on the available parameters.

2.2. Required Topic

The synth-client requires a topic to produce and consume messages from configured by `synth-client.topic` property. The topic needs to have the number of partitions equal to the number of brokers in the Kafka cluster. (If synth-client notices that this is not the case, it will attempt to increase the number of partitions to match the number of brokers, if it has the required permissions.)

As data is random we recommend short retention times for the topic.

If you want the client to automatically create the topic with the correct number of partitions, set the `synth-client.auto-create-topic` property to `true`. For this to work the client needs `Create` permissions on the configured topic.

2.3. Required ACLs

In order to work properly, the client's Kafka user needs to have ACL permissions to do the following operations:

- Describe, Read, Write, Alter the configured topic
- Read the configured consumer group
- Describe the cluster

For information on how to configure the consumer group and the topic, see the Parameters section. An example for setting this up for a Strimzi cluster is available in the `examples/` folder.

2.4. Deployment locations

The synth client is designed to be deployed in all the locations where your Kafka users run their applications. The better you cover all the used network zones and paths, the better you can measure the end-to-end latency of your Kafka messages.

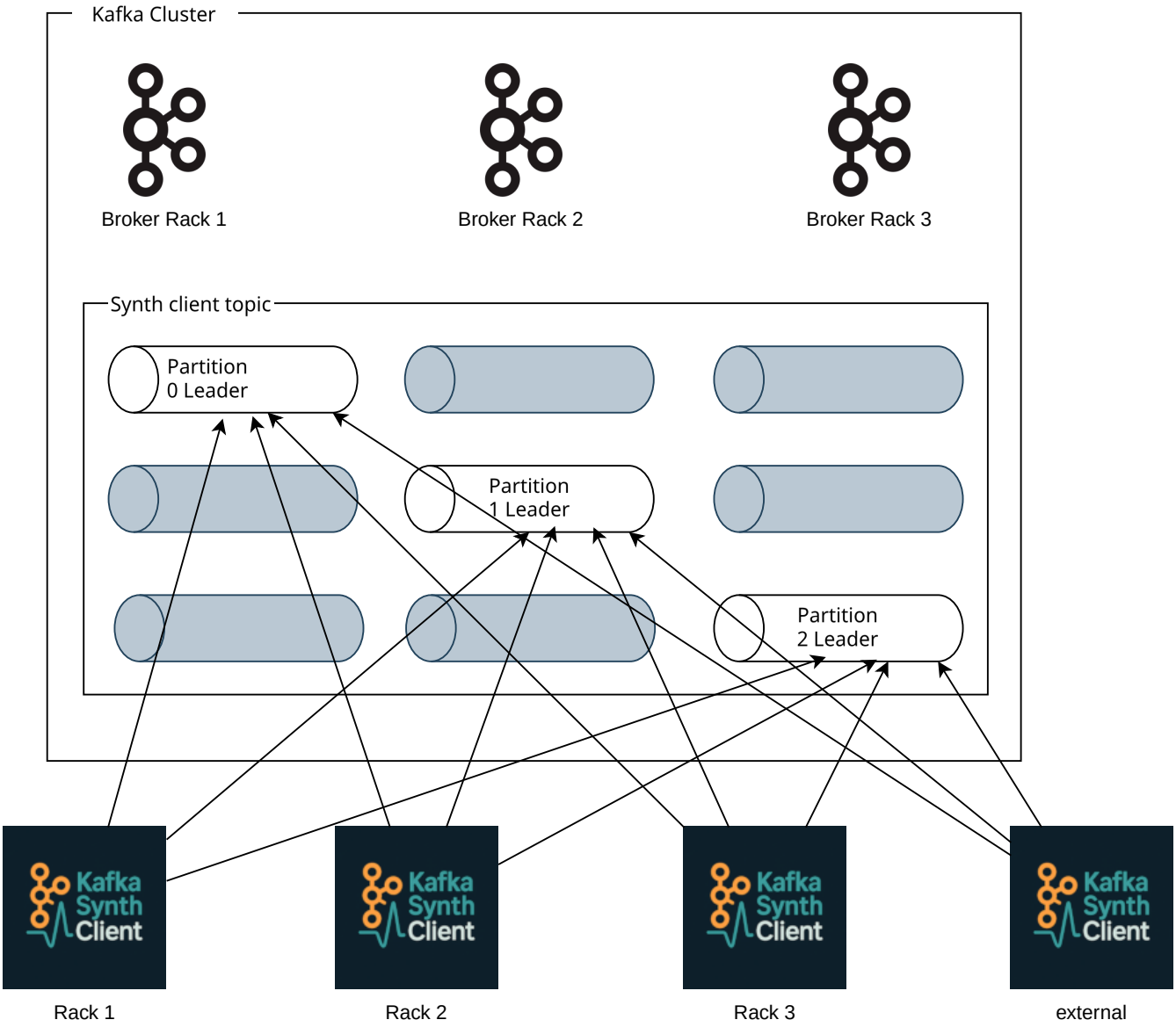

Figure 1. Deployment in different locations

2.5. Available Metrics

The synth client exposes the following metrics in prometheus format (on the `/q/metrics` endpoint):

2.5.1. End to end produce/consume latency in milliseconds

```
# HELP synth_client_e2e_latency_ms End-to-end latency of the synthetic client
# TYPE synth_client_e2e_latency_ms summary
synth_client_e2e_latency_ms{broker="2",fromRack="rack1",partition="0",toRack="rack0",t
opic="kafka-synth-client-test-ackone",viaBrokerRack="rack2",quantile="0.5",} 2.0625
synth_client_e2e_latency_ms{broker="2",fromRack="rack1",partition="0",toRack="rack0",t
```

```
opic="kafka-synth-client-test-ackone",viaBrokerRack="rack2",quantile="0.8",} 2.0625
synth_client_e2e_latency_ms{broker="2",fromRack="rack1",partition="0",toRack="rack0",t
opic="kafka-synth-client-test-ackone",viaBrokerRack="rack2",quantile="0.9",} 3.0625
synth_client_e2e_latency_ms{broker="2",fromRack="rack1",partition="0",toRack="rack0",t
opic="kafka-synth-client-test-ackone",viaBrokerRack="rack2",quantile="0.95",} 3.0625
synth_client_e2e_latency_ms{broker="2",fromRack="rack1",partition="0",toRack="rack0",t
opic="kafka-synth-client-test-ackone",viaBrokerRack="rack2",quantile="0.99",} 3.0625
synth_client_e2e_latency_ms_count{broker="2",fromRack="rack1",partition="0",toRack="ra
ck0",topic="kafka-synth-client-test-ackone",viaBrokerRack="rack2",} 717.0
synth_client_e2e_latency_ms_sum{broker="2",fromRack="rack1",partition="0",toRack="rack
0",topic="kafka-synth-client-test-ackone",viaBrokerRack="rack2",} 4409.0
```

This latency describes the time it took for a produced message to be consumed. The latency is measured in milliseconds. The synth client reports the median, 80th, 90th, 95th, and 99th percentile of the latencies. These metrics are handy if you have a service level objective (SLO) for the end-to-end latency of your Kafka messages (e.g. 95% of messages should be consumable within 30ms of being produced).

2.5.2. Acknowledgement latency in milliseconds

```
# HELP synth_client_ack_latency_ms Ack latency of the synthetic client
# TYPE synth_client_ack_latency_ms summary
synth_client_ack_latency_ms{broker="0",partition="1",rack="rack0",topic="kafka-synth-
client-test-ackone",viaBrokerRack="rack0",quantile="0.5",} 1.0
synth_client_ack_latency_ms{broker="0",partition="1",rack="rack0",topic="kafka-synth-
client-test-ackone",viaBrokerRack="rack0",quantile="0.8",} 1.0
synth_client_ack_latency_ms{broker="0",partition="1",rack="rack0",topic="kafka-synth-
client-test-ackone",viaBrokerRack="rack0",quantile="0.9",} 1.0
synth_client_ack_latency_ms{broker="0",partition="1",rack="rack0",topic="kafka-synth-
client-test-ackone",viaBrokerRack="rack0",quantile="0.95",} 2.0625
synth_client_ack_latency_ms{broker="0",partition="1",rack="rack0",topic="kafka-synth-
client-test-ackone",viaBrokerRack="rack0",quantile="0.99",} 2.0625
synth_client_ack_latency_ms_count{broker="0",partition="1",rack="rack0",topic="kafka-
synth-client-test-ackone",viaBrokerRack="rack0",} 1092.0
synth_client_ack_latency_ms_sum{broker="0",partition="1",rack="rack0",topic="kafka-
synth-client-test-ackone",viaBrokerRack="rack0",} 1292.0
```

This latency describes the time it took for a produced message to be acknowledged by the broker. The latency is measured in milliseconds. The synth client reports the median, 80th, 90th, 95th, and 99th percentile of the latencies. These metrics are handy if you would like to know how long it takes for your message to be acknowledged by the broker. This is especially interesting if you configure the producer with `acks=all` (you can do this in the synth client by setting the `KAFKA_ACKS` environment variable to `all`), as this will only acknowledge the message once it has been received by all replicas. In this case, we are effectively monitoring the time it takes to replicate the message across all replicas. The `rack` label indicates the "rack" or environment in which the client is running. This label is useful for distinguishing between latencies in different environments.

2.5.3. Time since last message in seconds

```
# HELP synth_client_time_since_last_consumption_seconds
# TYPE synth_client_time_since_last_consumption_seconds gauge
synth_client_time_since_last_consumption_seconds{rack="rack1",} 0.175
```

This metric describes the amount of seconds since the last message was consumed by the client. Since the client is producing messages at a constant rate (at least one message per second), a value that is much higher than 1 second indicates that there are issues either with the production or consumption of messages. This metric is a good candidate for alerting, as it can indicate that the Kafka cluster is not functioning as expected or unreachable.

2.5.4. Producer error rate per second

```
# HELP synth_client_producer_error_rate
# TYPE synth_client_producer_error_rate gauge
synth_client_producer_error_rate{rack="rack1",} 0.0
```

The average per-second number of record sends that resulted in errors. An increase in this metric can indicate issues with reaching the Kafka cluster or issues with the Kafka cluster itself. This is another good candidate for alerting.

2.6. Web UI

The Kafka Synth Client includes a built-in web interface that provides real-time visualization of latency metrics and message paths across your Kafka cluster.

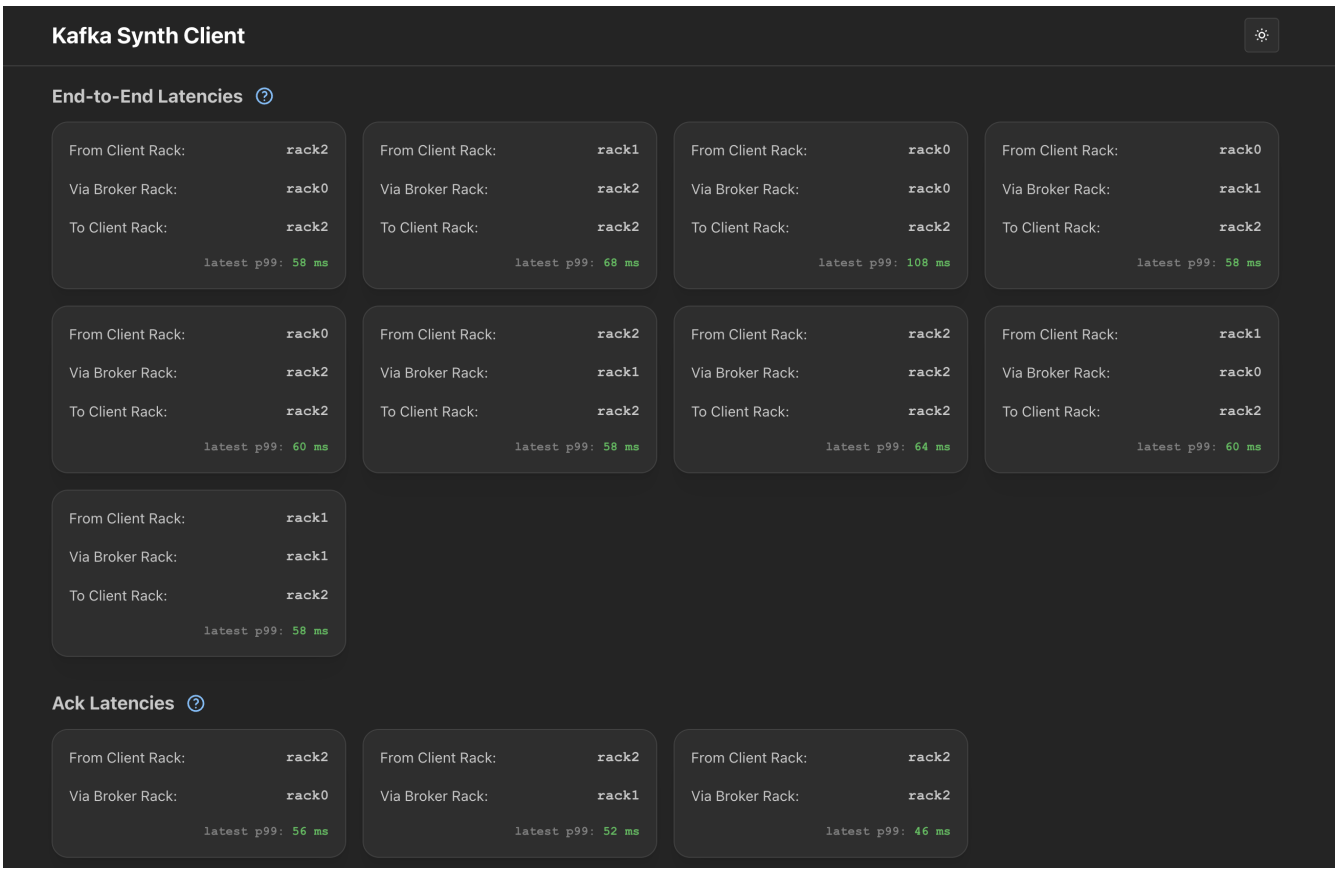

Figure 2. Web UI Message Paths Dashboard

2.6.1. Accessing the Web UI

The web UI is available per default on port `8081` of the synth client deployment (the same port that serves Prometheus metrics). When running with docker-compose, you can access it at:

```
http://localhost:8081
```

When deployed to Kubernetes, the web UI can be exposed through an ingress or service. See the `examples/strimzi.yaml` file for a complete example including ingress configuration.

2.6.2. Features

The web UI provides three main dashboards:

Message Paths Dashboard

The home dashboard displays all active message paths in your Kafka cluster, showing:

- **End-to-End (E2E) Latency Paths**: Visual representation of message paths from producer rack through broker rack to consumer rack, with current latency measurements

- **Acknowledgement (Ack) Latency Paths**: Direct paths from producer to broker showing acknowledgement latencies

Each path card shows:

- Source and (in the case of E2E-latencies) destination racks
- Broker rack (via rack)
- Current latency in milliseconds

E2E Latency Dashboard

Clicking on an E2E latency path takes you to a detailed dashboard showing:

- Historical latency trends over time
- Configurable time ranges (last 5 minutes to 24 hours)
- Interactive charts with hoverable data points
- Multiple percentiles

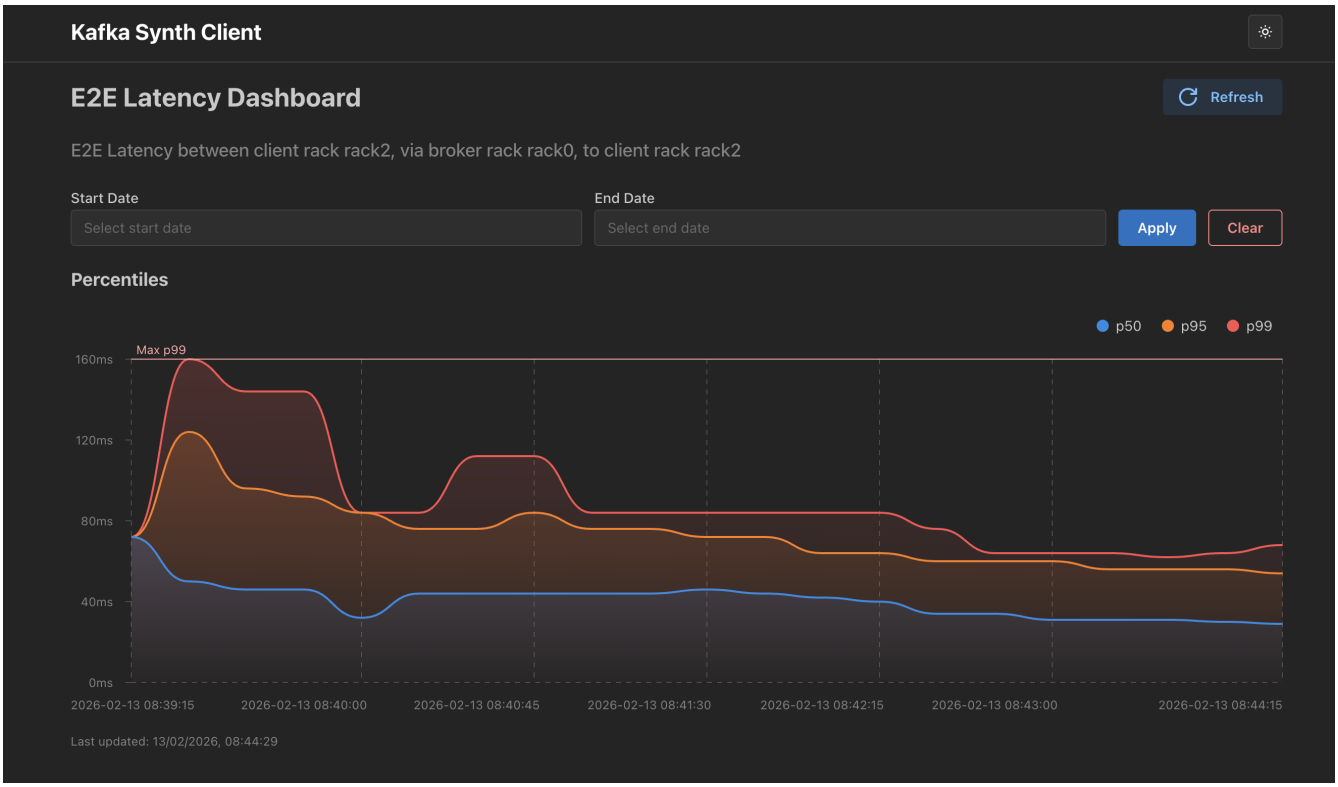

Figure 3. E2E Latency Chart with Time Series Data

Ack Latency Dashboard

Clicking on an Ack latency path shows detailed acknowledgement latency metrics:

- Producer to broker acknowledgement times
- Historical trends
- Percentile breakdowns
- Time range selection

2.6.3. Multi-DC Setup

When using the `multi-dc-docker-compose.yaml` setup, each synth client instance has its own web UI exposed on different ports:

Table 1. Synth Client Ports for ack=all configuration

Instance	Port (UI & Metrics)
rack0	18088
rack1	28088
rack2	38088

Table 2. Synth Client Ports for ack=1 configuration

Instance	Port (UI & Metrics)
rack0	18089
rack1	28089
rack2	38089

This allows you to view latency data from the perspective of each individual rack or datacenter location.

2.6.4. Configuration

Historical data for the last 7 days is stored in-memory inside a DuckDB. If you wish to retain this data across restarts of the synth client, override the `SYNTH_CLIENT_HISTORY_DATABASE_PATH` env variable to point to an actual path, e.g. `jdbc:duckdb:/my/volume/duck.db`. The default retention period of 7 days can be modified using the `SYNTH_CLIENT_HISTORY_RETENTION_PERIOD` environment variable (value must be a valid ISO-8601 duration like `PT2D`)

2.6.5. Technology Stack

The web UI is built with:

- React
- Mantine UI components
- React Router for navigation
- Recharts for data visualization
- DayJS for time handling

The UI is served directly by the Quarkus application using Quinoa, which means no separate deployment is needed.

3. Parameters

The preferred way to configure the client is to use environment variables. Below is an overview of some key configuration parameters.

Parameter	default	Description
KAFKA_BOOTSTRAP_SERVERS	<*mandatory*>	Kafka bootstrap servers
SYNTH_CLIENT_TOPIC	<*mandatory*>	The Kafka topic to produce to and consume from.
SYNTH_CLIENT_CONSUMER_TOPIC_REGEX	a regex pattern that matches only the value of SYNTH_CLIENT_TOPIC	A regex pattern describing which topics to consume from. This is only needed if the topics you want to consume from differ from the topic you produce to.
SYNTH_CLIENT_RACK	"default"	Some identifier of the environment in which the client is running. For example "eu-west-1a". This is useful for measuring latencies between clients that are running in different environments. Can be left unset if not needed. If you have multiple racks, then be sure to assign a unique consumer group ID to each rack.
SYNTH_CLIENT_MESSAGES_MESSAGE_SIZE_BYTES	8	The size of each Kafka message in bytes.
SYNTH_CLIENT_MESSAGES_MESSAGES_PER_SECOND	10	The number of messages to produce per second.
SYNTH_CLIENT_MESSAGES_IGNORE_FIRST_N_MESSAGES	10	The number of messages (per partition) to ignore before starting to measure latencies. This is useful for avoiding adding noise to the metrics when the consumer group is being rebalanced. The default value should be sufficient.
SYNTH_CLIENT_SAMPLING_TIME_WINDOW	2m	Duration length of the time window for which to calculate the latencies. The default value should be sufficient.

Parameter	default	Description
SYNTH_CLIENT_MIN_SAMPLES_FIRST_WINDOW	100	Number of samples before percentiles are calculated. The default value should be sufficient.
SYNTH_CLIENT_AUTO_CREATE_TOPIC	true	If the topic is managed by the synth client and also created by the synth client. This requires CREATE permissions.
SYNTH_CLIENT_TIME_SERVERS	time.google.com	NTP server to use for time synchronization.
SYNTH_CLIENT_TOPIC_REPLICATION_FACTOR	1	Replication factor to use when creating the topic.
SYNTH_CLIENT_PUBLISH_HISTOGRAM_BUCKETS	false	Will publish the histogram buckets to the metrics endpoint if enabled. ⋯, synth_client_e2e_latency_ms_bucket{⋯,le="41.0",} 902.0, synth_client_e2e_latency_ms_bucket{⋯,le="46.0",} 902.0, ⋯
SYNTH_CLIENT_EXPECTED_MIN_LATENCY	1.0	Minimum expected latency in milliseconds. Used to create histogram buckets when enabled.
SYNTH_CLIENT_EXPECTED_MAX_LATENCY	1.0	Maximum expected latency in milliseconds. Used to create histogram buckets when enabled.
SYNTH_CLIENT_HISTORY_DATABASE_PATH	jdbc:duckdb:	DuckDB connection url for storing historical latency data. Append a path (e.g. jdbc:duckdb:/my/volume/duck.db) to persist the data across restarts. By default, the database is in-memory and data will be lost when the client restarts.
SYNTH_CLIENT_HISTORY_RETENTION_PERIOD	P7D	How long to retain historical latency data. The value must be a valid ISO-8601 duration (e.g. P2D for 2 days).

Parameter	default	Description
SYNTH_CLIENT_ADVERTISED_LISTENER	*<optional>*	A URL under which this synth client instance is accessible/resolvable via browser (must include the protocol, e.g. http://localhost:8081). The synth-client will advertise this URL as part of the Kafka messages it sends out. This will be used by the UI to pull metrics from all synth-client deployments.
SYNTH_CLIENT_CONTEXT_PATH	/	Context path of the application. All endpoints (e.g. /q/metrics) will be relative to this path.
QUARKUS_HTTP_PORT	8081	The port on which the metrics endpoint will be exposed.
QUARKUS_LOG_LEVEL	INFO	Set to DEBUG to get more insights.
QUARKUS_LOG_CONSOLE_JSON_ENABLED	false	Set to true to enable json logging.
QUARKUS_LOG_CONSOLE_JSON_LOG_FORMAT	default	Set to ecs for elastic common schema.
JAVA_MAX_MEM_RATIO	50	This is used to calculate the default maximal heap memory based on a container's restriction. If used in a container without any memory constraints for the container then this option has no effect. The default is 50 which means 50% of the available memory is used as an upper boundary.

Furthermore, any environment variable prefixed with KAFKA_ will be interpreted as a Kafka Producer/Consumer configuration property. For example, setting KAFKA_GROUP_ID will set the value of the group.id consumer property.

KAFKA_CONFIG_PROVIDERS are supported like file or env. For example, setting KAFKA_CONFIG_PROVIDERS=file will load the configuration from a file. KAFKA_CONFIG_PROVIDERS_FILE_CLASS will be used to define the provider class. For example, setting KAFKA_CONFIG_PROVIDERS_FILE_CLASS=org.apache.kafka.common.config.provider.FileConfigProvider will load the configuration from a file.

Be aware that e.g., ${file:/path/to/file} needs to be escaped in the environment variable. For

example, `KAFKA_CONFIG_PROVIDERS_FILE_PATH=\${file:/path/to/file}` otherwise quarkus will try to resolve it as a variable and you will see `Error injecting java.util.Map<java.lang.String, java.lang.String> io.spoud.kafka.KafkaFactory.config`.

Use the following rule to convert a config property name to an environment variable name: - Replace all dots (`.`) with underscores (`) - Convert to uppercase - Prefix with KAFKA`

For example, the Kafka property `bootstrap.servers` would be set as the environment variable `KAFKA_BOOTSTRAP_SERVERS`.

For a list of Kafka configuration options, see the following links:

- Kafka Producer Configuration
- Kafka Consumer Configuration

Note that key/value serializers and deserializers are already configured and must not be overridden.

When deploying a synth-client instance, be sure that it is not completely utilizing the CPU. If the CPU is fully utilized, then the client will not be able to consume messages in a timely manner, which will result in skewed latencies.

4. Multi DC simulation

We have created a docker-compose file that simulates a multi-DC Kafka cluster with 3 DCs.

Ensure you have built the image with maven.

Startup:

```
docker-compose -f multi-dc-docker-compose.yaml up -d
```

Dashboard can be found at `http://localhost:3000` with username `admin` and password `admin`. Prometheus can be found at `http://localhost:9090`. There are 2 sets of synth clients deployed one reading and writing to a not replicated topic `⋯ackone` and one to a fully replicated topic `⋯ackall`.

Introduce latency on the network in DC2 (requires pumba to be installed https://github.com/alexei-led/pumba):

```
pumba netem --tc-image ghcr.io/alexei-led/pumba-alpine-nettools:latest \
    --duration 5m \
    delay --time 1000 \
    --jitter 100 \
    redpanda-2

# you may want to enter another broker and test the delay with pings
docker exec -u 0 -ti redpanda-1 bash
apt update; apt install -y iputils-ping
ping redpanda-2
```

Tear down:

```
docker-compose -f multi-dc-docker-compose.yaml down -v
```

5. Dashboards

This repository contains example dashboards you can use to visualize the metrics exposed by the synth client.

On the first panel you can see the e2e latency between producer (source rack) and consumer (destination rack).

As the image shows you can see from the legend or tool tip which path between source and destination rack the message took and what the latency was.

In this image we see the latencies that were recorded by messages that were passing through any broker in the selected `via Broker Rack` (in this case `rack2`).

This image shows the ack latency of the messages produced by the synth client. The ack latency is the time it takes for the broker to acknowledge the message. This is especially interesting if you have configured the producer with `acks=all` as this will only acknowledge the message once it has been received by all replicas.

If you want to compare the acks settings it's best to deploy 2 sets of synth clients with different acks settings and a dedicated topic with no replication for `acks=1` setting.