Generating log data for testing

- Using pushParameterized
  - Argument streams
  - Arguments minSize and maxSize
- Log format
- Labels
- Payload encoding

You can use k6 to generate log data for load testing.

Using `pushParameterized`

Push logs to Loki with `pushParameterized`. This method generates batches of streams in a random fashion. This method requires three arguments:

name	description
`streams`	number of streams per batch
`minSize`	minimum batch size in bytes
`maxSize`	maximum batch size in bytes

JavaScript example code fragment:

```
import loki from 'k6/x/loki';

const KB = 1024;
const MB = KB * KB;

const conf = loki.Config("http://localhost:3100");
```

```
const client = loki.Client(conf);

export default () => {
   client.pushParameterized(2, 500 * KB, 1 * MB);
};
```

Argument `streams`

The first argument of the method is the desired amount of streams per batch. Instead of using a fixed amount of streams in each call, you can randomize the value to simulate a more realistic scenario.

JavaScript example code fragment:

```
function randomInt(min, max) {
  return Math.floor(Math.random() * (max - min + 1) + min);
};

export default () => {
   let streams = randomInt(2, 8);
   client.pushParameterized(streams, 500 * KB, 1 * MB);
}
```

Arguments `minSize` and `maxSize`

The second and third argument of the method take the lower and upper bound of the batch size. The resulting batch size is a random value between the two arguments. This mimics the behavior of a log client, such as Grafana Alloy, where logs are buffered and pushed once a certain batch size is reached or after a certain size when no logs have been received.

The batch size is not equal to the payload size, as the batch size only counts bytes of the raw logs. The payload may be compressed when Protobuf encoding is used.

Log format

`xk6-loki` can emit log lines in seven distinct formats. The label `format` of a stream defines the format of its log lines.

- Apache common (`apache_common`)

- Apache combined (`apache_combined`)
- Apache error (`apache_error`)
- BSD syslog (`rfc3164`)
- Syslog (`rfc5424`)
- JSON (`json`)
- logfmt (`logfmt`)

Under the hood, the extension uses a fork the library flog for generating log lines.

Labels

`xk6-loki` uses the following label names for generating streams:

name	type	cardinality
instance	fixed	1 per k6 worker
format	fixed	7
os	fixed	3
namespace	variable	>100
app	variable	>100
pod	variable	>100
language	variable	>100
word	variable	>100

By default, variable labels are not used. However, you can specify the cardinality (quantity of distinct label values) using the `cardinality` argument in the `Config` constructor.

JavaScript example code fragment:

```
import loki from 'k6/x/loki';

const cardinality = {
   "app": 1,
   "namespace": 2,
   "language": 2,
   "pod": 5,
};
const conf = loki.Config("http://localhost:3100", 5000, 1.0, cardinality);
const client = loki.Client(conf);
```

The total quantity of distinct streams is defined by the cartesian product of all label values. Keep in mind that high cardinality negatively impacts the performance of the Loki instance.

Payload encoding

Loki accepts two kinds of push payload encodings: JSON and Protobuf. While JSON is easier for humans to read, Protobuf is optimized for performance and should be preferred when possible.

To define the ratio of Protobuf to JSON requests, the client configuration accepts values of 0.0 to 1.0. 0.0 means 100% JSON encoding, and 1.0 means 100% Protobuf encoding.

The default value is 0.9.

JavaScript example code fragment:

```
import loki from 'k6/x/loki';


const ratio = 0.8; // 80% Protobuf, 20% JSON
const conf = loki.Config("http://localhost:3100", 5000, ratio);
const client = loki.Client(conf);
```

□□□ #2
□ □□□ □□□ 2026-10-31 13:49:22 UTC
□ □□□ □□□ 2026-10-31 13:49:23 UTC