Sepehr Mahmoodian

Towards HPC integration: Kafka-based ingestion of edge/IoT data streams

Experiences from a benchmarking testbench on OpenStack

Table of contents

Motivation: Reliable Data Ingestion for HPC

- Modern systems continuously generate data (sensors, scientific instruments, distributed services)
- HPC workflows increasingly rely on timely and reliable data input
- Before computation can start, incoming data must:
  - be ingested at sustained high throughput
  - be buffered and replicated safely
  - remain available for downstream processing

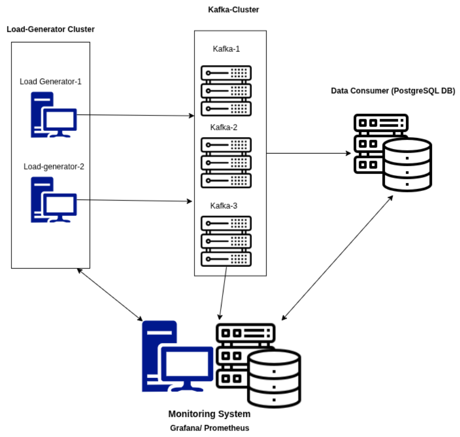
Core Challenge

How can we push the system to high ingestion rates without losing reliability, stability, or visibility into what is happening?

Scope of This Work

- **Implemented a full edge-to-database ingestion pipeline:**
  - Simulated edge devices generating telemetry data
  - Kafka as ingestion and buffering layer
  - Consumer writing data to PostgreSQL for storage
- **Primary focus of the evaluation:**
  - Load testing the ingestion layer under controlled producer pressure
  - Measuring sustained throughput
  - Identifying system bottlenecks with full observability

System Architecture Overview



Simulated edge generators produce telemetry data, which is ingested by a 3-node Kafka cluster and persisted in PostgreSQL. Monitoring is performed via Prometheus and Grafana.

Deployment Configuration

- **Load Generators (2 VMs)**
  - Ubuntu 22.04
  - Rate-controlled producers
  - Stateless design
- **Kafka Cluster (3 nodes)**
  - Kafka 4.1.1, KRaft combined mode
  - Replication factor = 3
  - Dedicated 300 GB persistent volume per broker
- **Consumer + PostgreSQL**
  - Data validation and persistent storage
- **Monitoring VM**
  - Prometheus 3.5.0 LTS
  - Grafana 12.3.1
  - Node / JMX / Kafka exporters

Infrastructure Environment

- Deployed on OpenStack private network
- Persistent block storage for Kafka logs
- SSH bastion-based access model
- Internal-only Kafka ports (no public exposure)
- Full hostname resolution via /etc/hosts

Design Principle

Reproducible, isolated, and production-like deployment for benchmarking.

Kafka as the Ingestion Layer

What is Kafka?

- A distributed event streaming platform
- Stores data as an ordered, replicated log
- Designed for high-throughput message ingestion

Apache Kafka® logo

Role in this project:

- Receives telemetry data from generators
- Buffers and replicates messages (RF=3)
- Decouples producers from the database layer

Kafka: Producers, Topics and Partitions

Basic Data Flow

- Producers send messages to Kafka topics
- Consumers read topics independently

Topics and Partitions

- A topic is split into multiple **partitions**
- Each partition keeps messages in strict order
- New messages are always written at the end
- Ordering is guaranteed **within a partition**

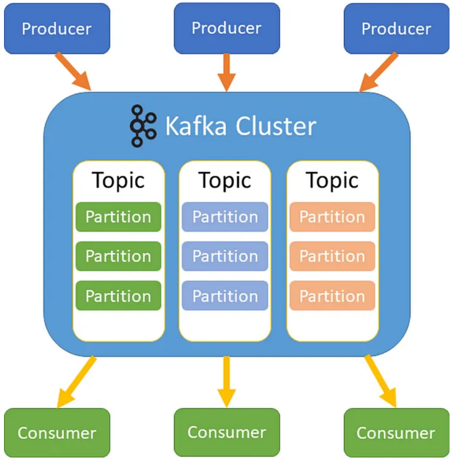


Figure: Kafka producers, topics and partitions

Source: DevGenius Blog

Kafka as a Central Data Backbone

- Kafka sits between producers and downstream systems
- Decouples data producers from consumers
- Enables independent scaling of each component

Key Idea: Kafka acts as a central data backbone that can connect to many systems.

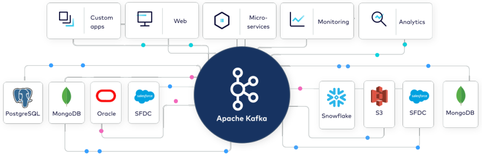


Figure: Kafka ecosystem and connectors
Source: confluent.io

How Systems Connect to Kafka

- Applications use Kafka client APIs (Producer / Consumer)
- Kafka Connect integrates external systems
- Stream processing frameworks consume and transform data

In this project: Generators → Kafka → Consumer → PostgreSQL

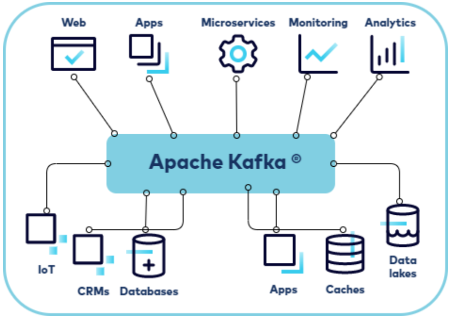


Figure: Kafka integration model
Source: Confluent Documentation

Kafka Deployment Choices

- 3-node Kafka cluster in **KRaft mode**
- No ZooKeeper dependency
- Separate persistent storage for data files
- Private internal networking on OpenStack

Why this design?

A simple and isolated setup makes benchmark results easier to interpret.

Synthetic Workload Model

Telemetry Event Structure

- Simulated IoT-like sensor events
- Small, frequent JSON messages
- Per-device ordering via seq
- Globally unique event_id

Purpose

- Emulate realistic edge workloads
- Enable correctness validation

```
{
  "event_id": "<uuid>",
  "device_id": "sensor-0123",
  "timestamp": "2026-01-22T15:42:11Z",
  "seq": 1842,
  "metrics": {
    "temperature": 21.4,
    "humidity": 48.2,
    "pressure": 1012.4
  },
  "status": "OK"
}
```

Generator Design

- Stateless generator nodes
- Horizontally scalable by adding VMs
- Rate-controlled message production (msg/s)
- Per-device ordering and correctness validation

Key Idea

Generators emulate realistic edge behavior without becoming bottlenecks.

Observability Stack

- **Prometheus:** time-series collection of system and Kafka metrics
- **Grafana:** real-time visualization and dashboards
- **Node Exporter:** CPU, memory, disk, and network metrics
- **JMX Exporter:** Kafka broker and JVM internals
- **Kafka Exporter:** topic throughput and consumer lag

Purpose

Correlate ingestion throughput with CPU, disk, and network utilization to identify the actual system bottleneck.

Evaluation Methodology

- Establish baseline with Kafka performance tools
- Run sustained and ramped producer workloads
- Measure:
  - throughput (msg/s, MB/s)
  - resource utilization (CPU, disk, network)
  - consumer lag and stability

Goal

Correlate ingestion throughput with system bottlenecks.

Evaluation Methodology

- Two independent approaches were used to evaluate Kafka throughput
- **Approach 1: Kafka internal performance test**
  - kafka-producer-perf-test.sh
  - Controlled synthetic workload
  - Used to identify cluster limits
- **Approach 2: Custom Kafka Test Bench**
  - Multi-generator architecture
  - Realistic ingestion logic
  - Integrated monitoring and validation

Kafka Internal Benchmark: Setup and Monitoring

Benchmark Configuration

- Tool: `kafka-producer-perf-test.sh`
- Topic: 12 partitions, Replication Factor (RF)=3
- Acks=all, idempotence enabled
- Controlled message size and send rate

System Monitoring

- CPU and memory usage (Node Exporter / `htop`)
- Disk I/O and throughput (`iostat`)
- Network utilization
- Kafka broker metrics (JMX Exporter)

Goal

Correlate producer throughput with resource saturation to identify the true bottleneck.

Results – Kafka Internal Benchmark

- Topic configuration: 12 partitions, RF=3
- Two parallel generators
- Aggregate throughput: ∼**48k msg/s**
- Effective bandwidth: ∼23 MB/s
- Average producer latency: 2–3 seconds under load

Observations

In one test run, disk burst saturation occurred on a single broker, temporarily increasing replication lag.
However, the sustained throughput plateau suggested a possible network-level constraint rather than a Kafka-internal limit.

VM-to-VM Network Benchmarking

Goal: Verify whether Kafka throughput was constrained by the VM network link.

Method

- Direct VM-to-VM benchmarking using `iperf3`
- TCP and UDP tests between OpenStack instances

Results

- Sustained bandwidth: $\sim$100–120 Mbit/s
- Equivalent to $\sim$12–15 MB/s
- UDP packet loss: $\sim$65% at higher rates

Conclusion

The VM-to-VM link imposes a bandwidth ceiling, indicating the workload is network-bound rather than Kafka-bound.

Web-Based Kafka Benchmarking Platform

Controller Architecture

- Centralized web-based control interface (Streamlit)
- Lightweight HTTP agents running on generator VMs
- Separation of control plane and data plane

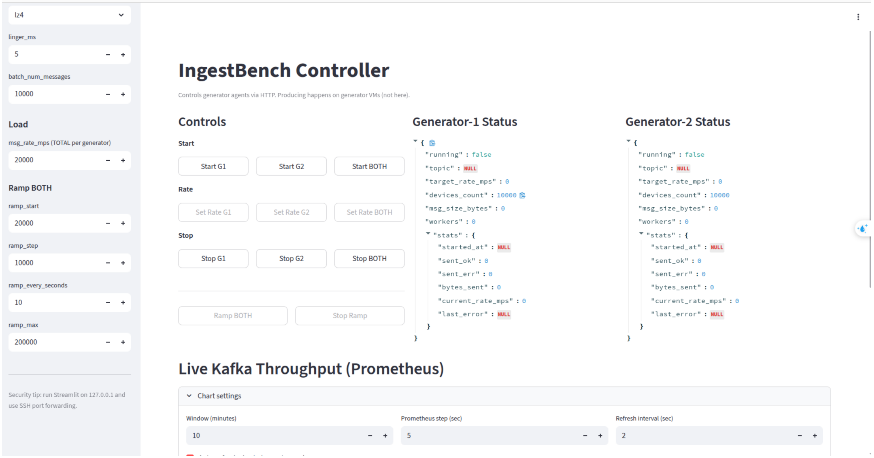
Capabilities

- Start/stop generators remotely
- Dynamic rate control and ramp testing
- Real-time Prometheus-based monitoring
- Integrated validation via downstream consumer

Purpose

Enable reproducible, observable, and controlled ingestion experiments.

IngestBench Controller – Web Interface



lz4

linger_ms
5

batch_num_messages
10000

Load

msg_rate_mps (TOTAL per generator)
20000

Ramp BOTH

ramp_start
20000

ramp_step
10000

ramp_every_seconds
10

ramp_max
200000

Security tip: run Streamlit on 127.0.0.1 and use SSH port forwarding.

IngestBench Controller

Controls generator agents via HTTP. Producing happens on generator VMs (not here).

Controls

Start: Start G1 | Start G2 | Start BOTH

Rate: Set Rate G1 | Set Rate G2 | Set Rate BOTH

Stop: Stop G1 | Stop G2 | Stop BOTH

Ramp BOTH | Stop Ramp

Generator-1 Status

```
{
  "running" : false
  "topic" : NULL
  "target_rate_mps" : 0
  "devices_count" : 10000
  "msg_size_bytes" : 0
  "workers" : 0
  "stats" : {
    "started_at" : NULL
    "sent_ok" : 0
    "sent_err" : 0
    "bytes_sent" : 0
    "current_rate_mps" : 0
    "last_error" : NULL
  }
}
```

Generator-2 Status

```
{
  "running" : false
  "topic" : NULL
  "target_rate_mps" : 0
  "devices_count" : 10000
  "msg_size_bytes" : 0
  "workers" : 0
  "stats" : {
    "started_at" : NULL
    "sent_ok" : 0
    "sent_err" : 0
    "bytes_sent" : 0
    "current_rate_mps" : 0
    "last_error" : NULL
  }
}
```

Live Kafka Throughput (Prometheus)

Chart settings

Window (minutes): 10
Prometheus step (sec): 5
Refresh interval (sec): 2

Screenshot of the Streamlit-based IngestBench Controller

Results – Custom Kafka Test Bench

- Multi-generator ingestion controlled via web interface
- Realistic telemetry workload (JSON events)
- End-to-end monitoring (CPU, disk, network, broker metrics)

Observed Performance

- Sustained throughput: ∼12–15 MB/s
- Corresponding to VM-to-VM bandwidth cap (100–120 Mbit/s)
- Stable throughput plateau under continuous load
- Increased latency once network saturation was reached

Insight

Under realistic ingestion conditions, the workload became network-bound rather than Kafka-bound, confirming infrastructure constraints identified via `iperf3`.

Lessons Learned

- Apparent throughput plateaus may originate outside Kafka
- Infrastructure constraints (e.g., VM network bandwidth) can dominate system performance
- Replication (RF=3) amplifies network and I/O pressure
- Transient disk burst events can mislead root-cause analysis
- End-to-end observability is essential for correct diagnosis

Outlook

- Deploy on infrastructure without VM bandwidth caps
- Evaluate performance on dedicated HPC network fabrics
- Integrate ingestion pipeline with real HPC workflows
- Study behavior under scientific workload patterns

Next Step

Move from controlled benchmarking toward edge-to-HPC workflow integration under realistic conditions.

Summary & Key Takeaways

- Built a **Kafka-based ingestion testbench** on OpenStack with reproducible deployment and clear separation of concerns
- Designed a **realistic, rate-controlled generator model** to emulate edge/IoT-style data streams at scale
- Deployed a **production-grade monitoring stack** (Prometheus, Grafana, Kafka/JMX/Node exporters) for end-to-end observability
- Established a **structured evaluation methodology** combining baseline Kafka tools with custom workloads
- Identified **practical bottlenecks and limits** (producer CPU, networking, validation overhead)
- The ingestion layer is **ready to be coupled** with real HPC workflows; integration and scaling are the next steps

References

Apache Software Foundation. *Apache Kafka Documentation*. Accessed: 2026-02-18. 2026. URL: `https://kafka.apache.org/documentation/`.

ESnet. *iperf3: A TCP, UDP, and SCTP Network Bandwidth Measurement Tool*. Accessed: 2026-02-18. 2026. URL: `https://iperf.fr/`.

Grafana Labs. *Grafana Documentation*. Accessed: 2026-02-18. 2026. URL: `https://grafana.com/docs/`.

htop-dev. *htop - Interactive Process Viewer*. Accessed: 2026-02-18. 2026. URL: `https://htop.dev/`.

OpenStack Foundation. *OpenStack Documentation*. Accessed: 2026-02-18. 2026. URL: `https://docs.openstack.org/`.

Prometheus Authors. *Prometheus Monitoring System*. Accessed: 2026-02-18. 2026. URL: `https://prometheus.io/docs/`.

Ramírez, Sebastián. *FastAPI Documentation*. Accessed: 2026-02-18. 2026. URL: `https://fastapi.tiangolo.com/`.

Streamlit Inc. *Streamlit: The Fastest Way to Build Data Apps in Python*. Accessed: 2026-02-18. 2026. URL: `https://docs.streamlit.io/`.

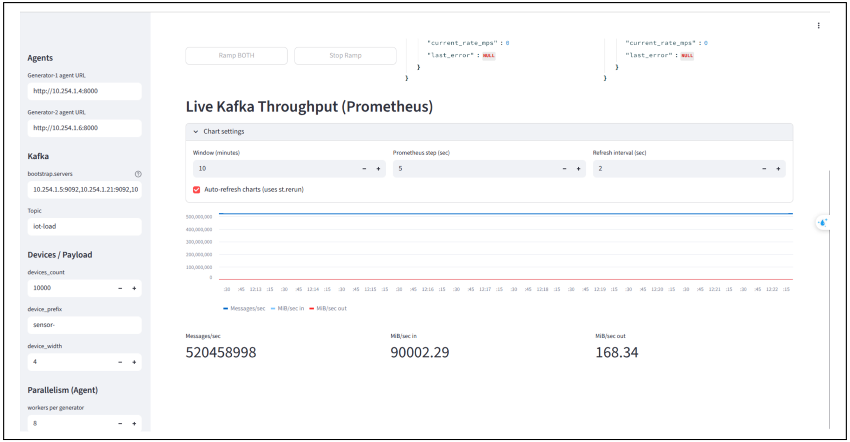
sysstat project. *sysstat Utilities (iostat)*. Accessed: 2026-02-18. 2026. URL: `https://github.com/sysstat/sysstat`.

Uvicorn Contributors. *Uvicorn ASGI Server*. Accessed: 2026-02-18. 2026. URL: `https://www.uvicorn.org/`.

Backup Slides

Additional Material and Technical Details

Live Kafka Throughput (Prometheus)



```
    "current_rate_mps" : 0
    "last_error" : NULL
  }
}
```

```
    "current_rate_mps" : 0
    "last_error" : NULL
  }
}
```

Agents

Generator-1 agent URL: http://10.254.1.4:8000

Generator-2 agent URL: http://10.254.1.6:8000

Kafka

bootstrap.servers: 10.254.1.5:9092,10.254.1.21:9092,10

Topic: iot-load

Devices / Payload

devices_count: 10000

device_prefix: sensor-

device_width: 4

Parallelism (Agent)

workers per generator: 8

Live Kafka Throughput (Prometheus)

Window (minutes): 10

Prometheus step (sec): 5

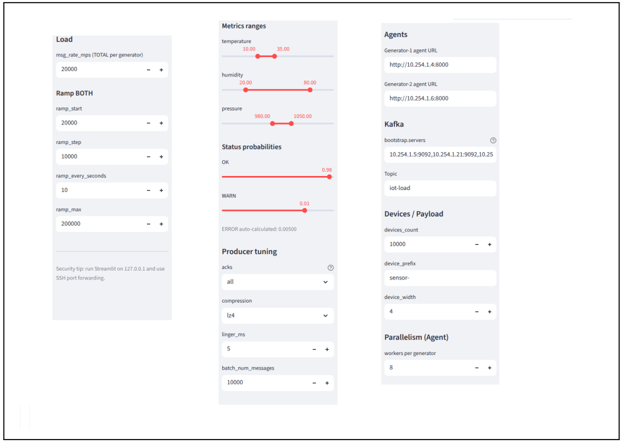
Refresh interval (sec): 2

✓ Auto-refresh charts (uses st.rerun)

Messages/sec	MiB/sec in	MiB/sec out
520458998	90002.29	168.34

Figure: Real-time Kafka throughput measured via Prometheus. Throughput plateau reflects the VM network bandwidth limit.

IngestBench Controller Interface



Load

msg_rate_mps (TOTAL per generator): 20000

Ramp BOTH

ramp_start: 20000

ramp_step: 10000

ramp_every_seconds: 10

ramp_max: 200000

Security tip: run Streamlit on 127.0.0.1 and use SSH port forwarding.

Metrics ranges

temperature: 10.00 – 35.00

humidity: 20.00 – 80.00

pressure: 980.00 – 1050.00

Status probabilities

OK: 0.98

WARN: 0.01

ERROR auto-calculated: 0.00500

Producer tuning

acks: all

compression: lz4

linger_ms: 5

batch_num_messages: 10000

Agents

Generator-1 agent URL: http://10.254.1.4:8000

Generator-2 agent URL: http://10.254.1.6:8000

Kafka

bootstrap.servers: 10.254.1.5:9092,10.254.1.21:9092,10.25

Topic: iot-load

Devices / Payload

devices_count: 10000

device_prefix: sensor-

device_width: 4

Parallelism (Agent)

workers per generator: 8

Figure: Web-based controller for load configuration and ramp testing. Supports controlled and reproducible ingestion experiments.