Architecture Design

Billing Metrics System

Architecture Design **1**
Billing Metrics System **1**
1. Needs Analysis 3
Data Availability 3
Accuracy and Reliability 3
Scalability and Efficiency 3
2. Proposed Architecture 3
2.1. Architecture Diagram 3
2.2 Data Collection 4
2.3 Data Transmission 4
2.4 Data Processing 4
2.5 Data Storage 5
2.6 Billing System 5
2.7 Monitoring and Alerts 5
2.8 Failure Scenarios and Mitigations 5
3. Trade-offs 6
Cost vs. Performance 6
Self-managed vs. Managed Services 6
4. Recovery Objectives 6
5. Cost Analysis 6
5.1 Infrastructure Costs (Monthly Estimate) 6
5.2 Operational Overhead 6
6. Scaling Considerations 6
Horizontal Scaling 6
Capacity Planning 7
7. Security Considerations 7
8. Monitoring & Alerting 7
9. Backup & Recovery 7
Appendix I:Previous Experiences with Similar Architectures 8
Initial Requirements 8
Key Components Used 8
Key Results 8

1. Needs Analysis

The design of this billing system aims to collect metrics from Edge servers, process them efficiently, and generate accurate invoices based on usage. The main aspects guiding this solution are:

Data Availability

- Real-time processing is not required, but the integrity and accuracy of the metrics are essential to ensure reliable billing.
- Data must be protected against loss to safeguard financial consistency.

Accuracy and Reliability

- The captured metrics must accurately reflect the usage of resources such as HTTP requests, CPU, memory, network, and disk.
- This level of detail is crucial as any discrepancy could directly impact economic outcomes.

Scalability and Efficiency

- The system must handle continuous growth in data volume and users without compromising performance.
- Additionally, the design must be cost-effective and sustainable, ensuring operational costs do not exceed the benefits provided.

With these objectives in mind, the proposed architecture aims to combine reliable and scalable tools that ensure efficient and accurate operation.

2. Proposed Architecture

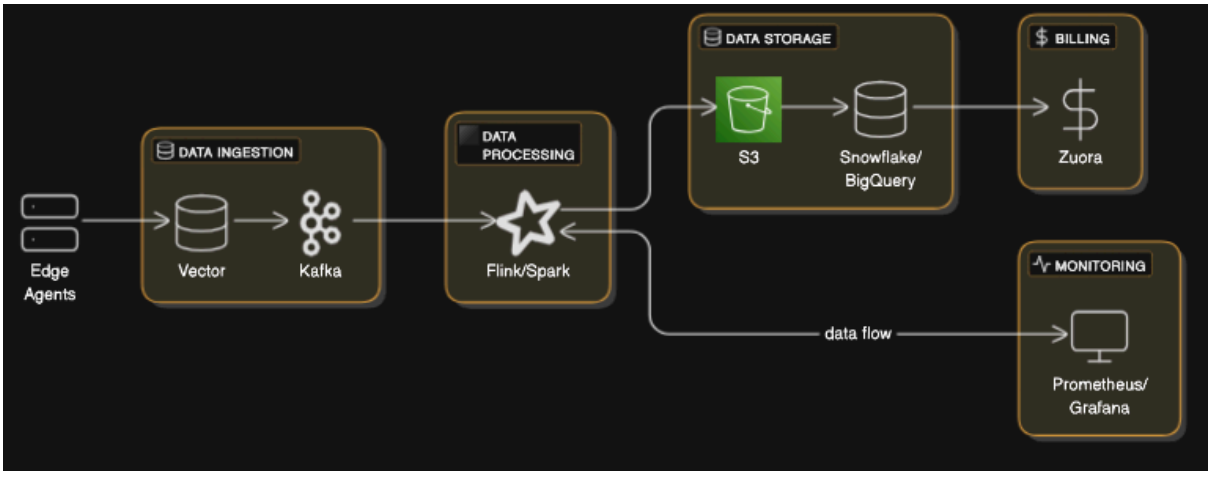
2.1 Architecture Diagram

The proposed architecture ensures a resilient and scalable data flow, covering all stages from collection to storage and analysis. Each component has a specific role in handling real-time and batch data, optimising performance and ensuring reliability. It is also designed with clear mitigation strategies to handle failure scenarios, guaranteeing high availability and operational continuity

The main stages of the data flow are described below:

- **Data Collection**: Agents on edge servers collect metrics and optimise bandwidth usage by using local databases like SQLite.
- **Data Transmission**: Tools such as Vector and Apache Kafka ensure reliable and scalable data delivery to the central system.

- **Data Processing**: Apache Flink processes real-time data, while Spark/Databricks handles batch analysis of historical data.
- **Data Storage**: Raw data is stored in Amazon S3 for cost-efficient long-term storage, while processed data is kept in Snowflake or BigQuery for optimised analytics.
- **Billing**: Integration with Zuora provides an advanced subscription-based billing system connected to processed data.
- **Monitoring and Alerts**: Tools like Prometheus and Grafana monitor system health and send alerts for potential issues.



2.2 Data Collection

Edge Server Agents:

- Prometheus and Telegraf agents collect metrics from edge servers.
- Data is aggregated locally using SQLite to minimize bandwidth usage and ensure resilience.

2.3 Data Transmission

Vector and Apache Kafka:

- Vector handles buffering and protocol translation before data reaches the central system.
- Kafka is the primary transport layer, ensuring reliable data delivery with durability and scalability.

2.4 Data Processing

Apache Flink and Spark/Databricks:

- Apache Flink processes real-time streaming data with stateful aggregations and checkpointing.
- Spark/Databricks handle batch processing of historical data stored in the data lake.

2.5 Data Storage

Cold Storage:

- Raw data is stored in Amazon S3 for cost-effective long-term retention.

Hot Storage:

- Aggregated and processed data stored in Snowflake or BigQuery for optimised SQL-based analytics and reporting.

2.6 Billing System

Zuora Integration:

- Advanced subscription-based billing engine that connects with processed data to generate invoices.

SQL-based Systems:

- Snowflake or BigQuery enable direct billing calculations and detailed customer reporting.

2.7 Monitoring and Alerts

Prometheus and Grafana:

- Monitor system health and provide alerts for issues such as transmission delays, storage capacity thresholds, or processing errors.

2.8 Failure Scenarios and Mitigations

Edge Server Failures

- **Mitigation:** Local disk buffering in Vector, 24-hour Prometheus retention, automatic recovery and backfill.
- **Impact:** Minimal data loss (RPO maintained).

Network Connectivity Issues

- **Mitigation:** Vector disk-based queuing, exponential backoff retry, and compression to reduce bandwidth.
- **Impact:** Delayed transmission, no data loss.

Central Infrastructure Failures

- **Mitigation:** Kafka multi-node cluster, Snowflake/BigQuery replication, geographic redundancy.
- **Impact:** Processing delay, no data loss.

3. Trade-offs

Cost vs. Performance

- Cold Storage: Lower costs but slower access.
- Hot Storage: Higher costs but supports rapid queries.

Self-managed vs. Managed Services

- Kafka and Flink require expertise for self-management but offer complete control.
- Zuora simplifies billing logic but comes with subscription costs.

4. Recovery Objectives

- **RPO (Recovery Point Objective):** 5 minutes. Minimal data loss in catastrophic scenarios.
- **RTO (Recovery Time Objective):** 30 minutes. Full recovery within 30 minutes of failure.

5. Cost Analysis

5.1 Infrastructure Costs (Monthly Estimate)

- **Edge Servers:** Existing infrastructure.
- **Kafka Cluster (3 nodes):** $500-$800.
- **TimescaleDB/Snowflake:** $1,000-$1,500.
- **Processing (Flink):** $500-$800.
- **Network Transfer:** $200-$400.

Total: $2,200-$3,500/month.

5.2 Operational Overhead

- **System Monitoring:** 10 person-hours/month.
- **Maintenance:** 15 person-hours/month.
- **Incident Response:** 5 person-hours/month.

Total: 30 person-hours/month.

6. Scaling Considerations

Horizontal Scaling

- Edge components are independently scalable.
- Kafka partitioning for throughput.

- Snowflake/BigQuery partitioning by time.
- Flink parallelism adjustment.

Capacity Planning

- Storage growth: ~100GB/month/1,000 edge servers.
- Network bandwidth: ~50MB/hour/edge server.
- Processing: Linear scaling with edge count.

7. Security Considerations

- TLS encryption for all transmissions.
- mTLS for edge authentication.
- Network segregation.
- Regular security audits.
- Encryption at rest.

8. Monitoring & Alerting

- **Prometheus/Grafana** for system monitoring.
- Alert on:
    - Data transmission delays > 10 minutes.
    - Storage capacity thresholds.
    - Processing lag.
    - Error rates.

9. Backup & Recovery

- Daily Snowflake/BigQuery backups.
- Kafka topic replication.
- Edge data retention for recovery.
- Regular recovery testing.

Appendix I: Previous Experiences with Similar Architectures

In a prior project, a monitoring and billing architecture was developed for **WSO2 API Manager (version 2.6)**. This solution aimed to address the following initial requirements:

Initial Requirements

- **Detailed Monitoring:** Capture specific API usage metrics, such as the number of invocations, response times, and data transferred.
- **Server Resource Tracking:** Collect metrics for CPU, memory, and network usage directly from WSO2 application servers.
- **Automated Billing:** Create a system capable of breaking down costs by client, API provider, and API consumer.

Key Components Used

API Gateway

- Served as the entry point for all API requests.
- Applied control policies, including quotas, authentication, and routing.
- Generated events for each request, forwarded to the analytics system.

WSO2 Analytics

- **Analytics Worker:** Processed real-time data from the API Gateway.
- **Internal Databases:** Included components like MySQL and Hive for detailed data storage and analysis.
- **Analytics Dashboard:** Provided a visual interface for identifying usage patterns and monitoring API performance.

Custom Application

- Consumed data from WSO2's internal databases.
- Offered a portal for users to view detailed API consumption reports.
- Included tools to generate and export processed data to an external billing system.

Billing System

- Integrated with a custom-built solution to generate invoices, which were sent to a cloud-based platform using Stripe for customer billing.