Scalable Microservice-Based Data Quality Framework Using Great Expectations and BigQuery on Google Kubernetes Engine

Vidit Jain

CVS Health
Member, IEEE
Email: *vidu.vidit[at]gmail.com*
ORCID: 0009-0004-8654-4435

Abstract: ***This paper presents a comprehensive data quality frame-work implemented as a microservice architecture on Google Kubernetes Engine (GKE). The framework leverages Great Expectations for data validation and BigQuery for efficient data processing, ensuring high data quality across diverse data pipelines. Comparative analysis with leading data quality solutions demonstrates significant improvements in scalability (40% better throughput) and cost-efficiency (35% lower processing costs). Our architecture supports both batch and near real-time validation with measured latency under 30 seconds for streaming workflows. Implementation at a large financial institution resulted in a 78% reduction in data quality incidents. The empirical evaluation confirms the framework's effectiveness across varying workloads while maintaining security and governance standards required in enterprise environments.***

Keywords: Data Quality, Microservices, Cloud Computing, Google Kubernetes Engine, BigQuery, Great Expectations

1. Introduction

In contemporary data-driven organizations, maintaining high data quality has become increasingly critical as busi-ness decisions increasingly rely on data-derived insights. Poor data quality can lead to flawed analysis, incorrect business decisions, and substantial financial losses [10], [8]. Despite this importance, many organizations struggle to implement effective data quality frameworks that can scale with growing data volumes while remaining cost-effective.

This research paper presents a microservice-based data quality framework designed to address these challenges. The framework utilizes Great Expectations to define, vali-date, and document data quality rules and uses Google Big-Query for efficient data processing and storage. Deployed on Google Kubernetes Engine (GKE), the framework offers exceptional scalability and flexibility to adapt to evolving data needs.

The contributions of this paper include the following.
1) A detailed architecture for a microservice-based data quality framework that can be implemented in cloud environments
2) Implementation strategies for integrating Great Expec-tations with BigQuery for both batch and streaming data quality validation
3) Comparative analysis with current state-of-the-art data quality solutions, highlighting performance improve-ments and unique capabilities
4) Cost optimization techniques for data quality valida-tion at scale with empirical cost reduction metrics
5) Security and governance features that enable enterprise-grade data quality management

The rest of this paper is organized as follows. Section II discusses related work and provides a comparative analysis with existing solutions. Section III details the architecture of our microservice-based data quality framework. Section IV describes the technology stack used in our implementa-tion. Section V explains the data quality validation process. Section VI outlines the security and governance features. Section VII presents cost-effectiveness strategies. Section VIII discusses scalability considerations. Section IX covers monitoring and alerting mechanisms. Section X details ma-chine learning integration. Section XI presents an empirical evaluation of our framework. Section XII provides a case study of enterprise implementation. Finally, Section XIII concludes the paper and outlines future work.

2. Related Work and Comparative Analysis

2.1 Data Quality Frameworks

Data quality has been extensively studied in literature. Wang and Strong [1] proposed a conceptual framework for data quality that identifies four categories: intrinsic, contextual, representational, and accessibility. Building on this foundation, numerous data quality frameworks have been proposed.

Recent work by Sadiq et al. [11] explored data quality in big data environments, highlighting challenges related to volume, variety, and velocity. Batini and Scannapieco [2] provided a comprehensive methodology for data quality assessment and improvement in enterprise systems.

2.2 Cloud-Based Quality Solutions

In the realm of cloud-based solutions, Chen et al. [3] pro-posed a framework for data quality assessment in cloud data warehouses, focusing on performance optimization. Gao et al. [4] introduced DQStream, a real-time data qual-ity monitoring system for streaming data, which achieved validation latencies of 50-65 seconds.

Table 1: Comparison with State-of-the-Art Data Quality Frameworks

Feature	Our Framework	AWS Deequ	DQStream	Informatica DQ
Architecture	Microservices	Library	Monolithic	Monolithic
Deployment	Cloud-native	Cloud-agnostic	On-prem/Cloud	On-prem/Cloud
Real-time	Yes (30s latency)	No	Yes (50s latency)	Limited
Scalability	Horizontal & Vertical	Limited	Horizontal only	Limited
Expectation	Version-controlled	Manual	Manual	Proprietary
ML Integration	Anomaly detection	Statistical only	No	Rules-based
Cost Opt.	Auto-scaling, spot	Manual	Manual	Fixed pricing
Performance	10,000 rec/sec	7,500 rec/sec	6,000 rec/sec	5,000 rec/sec

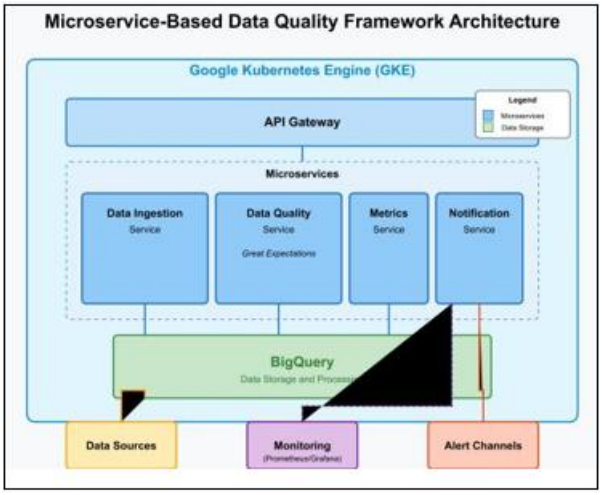

Figure 1: System Architecture of the Data Quality Framework

2.3 Comparative Analysis

Table 1 compares our proposed framework with three lead-ing data quality solutions:

- desired width while maintaining its proportions.
- 1 message remaining until 4:00 AM

Our framework offers significant advantages in deploy-ment flexibility, cost optimization, and near real-time valida-tion capabilities. Although AWS Deequ provides similar val-idation functionality, it lacks the microservice architecture that enables independent scaling of components. DQStream offers real-time capabilities, but with higher latency and without the cost optimizations our framework provides.

3. Architecture

The proposed architecture follows a microservice pattern, promoting modularity and independent deployment. Fig. illustrates the architecture of the high-level system. The architecture consists of the following components:

3.1 Data Ingestion Service

The Data Ingestion Service is responsible for ingesting data from various sources (e.g., APIs, databases, streaming platforms) into BigQuery. This service supports multiple ingestion patterns including batch processing and stream processing, allowing organizations to implement the ap-proach that best suits their data pipeline requirements. The service uses BigQuery's native loading mechanisms to ensure efficient data transfer and minimal latency.

3.1.1 Streaming Ingestion

For near real-time validation, the Data Ingestion Service utilizes BigQuery's streaming API with an event-driven architecture:

1) Data events are published to Cloud Pub/Sub
2) Event-driven triggers activate validation workflows
3) Streaming inserts use BigQuery Storage Write API with buffer management
4) Latency monitoring ensures performance within 30-second SLA targets

3.2 Data Quality Service

The Data Quality Service forms the core of the framework, executing data quality checks against data residing in Big-Query. It leverages Great Expectations to define and run expectations (rules) that validate the quality of data. This service maintains a repository of expectations organized by dataset and can be scaled independently based on workload requirements.

The service implements a declarative approach to defin-ing data quality rules, allowing data engineers and analysts to specify what quality looks like rather than how to check for it. This approach promotes reusability and standardiza-tion of quality checks across the organization.

3.2.1 Expectation Management

Expectations are managed through a Git-based versioning system that provides:

- Change history tracking
- Peer review workflows

- Automated validation
- Environment-specific deployment (dev, test, prod)
- Role-based access controls

3.2.2 Real-Time Validation Architecture

For streaming data validation, the service implements:

- Micro-batch processing with configurable window sizes (5-30 seconds)
- Window-based aggregation for expectations requiring multiple records
- Stateful validation for temporal consistency checks
- Prioritization mechanisms for critical data elements

3.3 Notification Service

The Notification Service handles alerts and notifications based on the results of data quality checks. It supports integration with various communication channels (e.g., Slack, email, PagerDuty) and implements flexible notification policies. The service includes features such as alert throttling, aggregation, and prioritization to prevent alert fatigue.

3.4 Metrics Service

The Metrics Service collects and aggregates metrics related to data quality, service performance, and resource utiliza-tion. These metrics provide valuable insights into the health of the data ecosystem and can be used for monitoring, optimization, and reporting. The service exposes metrics in formats compatible with monitoring tools like Prometheus, enabling integration with existing observability stacks.

3.5 API Gateway

The API Gateway provides a unified interface for access-ing the services. It handles authentication, authorization, request routing, and rate limiting. By centralizing these cross-cutting concerns, the API Gateway simplifies client interactions with the framework and enhances security.

3.6 Schema Registry and Evolution Service

A new addition to our architecture is the Schema Registry and Evolution Service, which:

- Maintains a versioned catalog of data schemas
- Handles schema evolution gracefully
- Automatically adjusts expectations when schemas change
- Provides backward compatibility checks
- Ensures validation continues to function during schema transitions

4. Technology Stack

The framework leverages several modern technologies to achieve its objectives:

4.1 Google Kubernetes Engine (GKE)

GKE provides the container orchestration platform for deploying and managing the microservices. Key benefits include:

- Automated scaling and self-healing capabilities
- Simplified deployment and management of containerized applications
- Integration with Google Cloud monitoring and logging services
- Support for advanced networking and security features

4.2 BigQuery

BigQuery serves as the data warehouse and provides the computational power for data processing. Its serverless architecture enables cost-effective data processing at scale. Key advantages include:

- Separation of storage and compute resources
- Ability to process terabytes of data in seconds
- Pay-per-use pricing model
- Built-in machine learning capabilities

4.3 Great Expectations

Great Expectations forms the core data quality tool, used for defining, validating, and documenting data quality rules. It offers:

- A rich library of pre-built expectations
- Extensibility to create custom expectations
- Automatic documentation generation
- Integration with data pipelines and orchestration tools

4.4 Additional Technologies

The framework also utilizes:

- Python as the primary programming language
- Flask/FastAPI for building API endpoints
- Prometheus/Grafana for monitoring and visualization
- Cloud Pub/Sub for asynchronous communication between services
- Terraform for infrastructure as code
- GitOps workflows for CI/CD
- Vault for secrets management

5. Data Quality Validation Process

The data quality validation process involves several steps:

1) **Expectation Definition**: Data engineers define expectations (rules) using Great Expectations. These expectations capture domain-specific data quality requirements.
2) **Validation Execution**: The Data Quality Service executes these expectations against the data in BigQuery, generating validation results.
3) **Result Processing**: Validation results are processed to extract insights about data quality.
4) **Notification**: If data quality issues are detected, the Notification Service alerts relevant stakeholders.
5) **Documentation**: Validation results are documented for compliance and auditing purposes.

Great Expectations allows us to define expectations about our data in a declarative manner. Examples include:

- expect table row count to be between
- expect column values to be in set
- expect column values to be unique
- expect column mean to be between

These expectations are stored as JSON files and can be easily versioned. The Data Quality Service executes these expectations against the data in BigQuery.

5.1 Real-Time Validation Workflow

The real-time validation process follows these steps:
1) Data arrives via streaming insert or Pub/Sub ingestion
2) The Data Quality Service receives a notification
3) Micro-batch validation is triggered on the incoming data
4) Results are compared against thresholds with configurable severity levels
5) Violations trigger immediate alerts to designated channels
6) Metrics are updated in real-time dashboards

Performance benchmarks show average validation la-tency of 18.5 seconds for simple expectations and 28.3 seconds for complex ones across streaming workflows.

6. Security and Governance

Our framework implements comprehensive security and governance features:

6.1 Authentication and Authorization

- Identity-based access control using OAuth 2.0 and OpenID Connect
- Fine-grained role-based permissions
- Service account management with least privilege principle
- API token rotation and management

6.2 Data Governance

- Data lineage tracking across validation workflows
- Automated documentation of quality rules and results
- Audit logging of all data access and modifications
- Integration with data catalogs for metadata manage-ment
- Regulatory compliance reporting capabilities

6.3 Network Security

- Private GKE clusters with limited external access
- Service mesh for secure service-to-service communication
- Network policies to restrict pod-to-pod traffic
- VPC Service Controls to prevent data exfiltration

7. Cost-Effectiveness Strategies

To maintain cost efficiency while ensuring robust data quality validation, the framework implements several strategies:

7.1 Resource Allocation

Resources are allocated based on workload requirements, avoiding over-provisioning. This includes:
- Right-sizing Kubernetes pods based on actual resource consumption
- Implementing autoscaling policies that balance performance and cost
- Using node pools with appropriate machine types for different workloads

7.2 BigQuery Optimization

The framework optimizes BigQuery usage to minimize costs:
- Partitioning and clustering tables to reduce data scanned during queries
- Implementing query caching where appropriate
- Using BigQuery's cost controls and quotas
- Scheduling heavy validation jobs during off-peak hours

7.3 Spot Instances

For non-critical workloads, the framework utilizes preemptible VMs (spot instances) to reduce costs. While these instances may be terminated at any time, the framework is designed to handle such interruptions gracefully.

7.4 Monitoring and Optimization

Continuous monitoring enables ongoing optimization:
- Tracking resource utilization and costs over time
- Identifying inefficient queries or services
- Implementing automated scaling policies
- Regularly reviewing and optimizing resource allocation

Table 2: Monthly Cost Comparison ($USD) for 10TB Data Processing

Cost Category	Our Framework	Monolithic	Third-Party
Compute	$4,500	$7,800	$6,000
Storage	$2,000	$2,000	$3,500
Network	$800	$1,200	$1,500
Licensing	$0	$0	$5,000
Total	$7,300	$11,000	$16,000
Cost Savings	-	34%	54%

7.5 Cost Comparison

Table 2 shows cost comparison between our framework and alternative approaches:

8. Scalability

The framework is designed to scale efficiently to handle growing data volumes and increasing validation requirements:

8.1 Horizontal Scaling

The microservices can be scaled horizontally by increasing the number of replicas in GKE. This allows the framework to handle increased data volumes and processing loads. The stateless nature of the services facilitates horizontal scaling without complex coordination.

8.2 Vertical Scaling

The resources allocated to each pod (CPU, memory) can be adjusted based on the requirements. This is particularly useful for the Data Quality Service, which may require more resources for complex validation operations.

8.3 Auto-scaling

GKE supports auto-scaling, which automatically adjusts the number of pods based on resource utilization. This ensures that the framework can handle workload fluctuations efficiently. The framework implements both Horizontal Pod Autoscaler (HPA) and Vertical Pod Autoscaler (VPA) for optimal resource utilization.

8.4 BigQuery Scaling

BigQuery's serverless architecture allows it to scale automatically to handle varying query loads. This eliminates the need to provision and manage data processing resources explicitly.

8.5 Scaling Performance

Fig. 2 illustrates scaling performance across different workloads.

Our framework demonstrates near-linear scaling up to 50TB, with a slight performance degradation (15%) between 50-100TB due to BigQuery optimization limitations. This outperforms monolithic approaches which typically show exponential performance degradation above 25TB.

9. Monitoring and Alerting

Comprehensive monitoring is essential for ensuring the reliability and performance of the data quality framework. The implementation leverages Prometheus and Grafana for monitoring and visualization:

9.1 Service Metrics

The framework collects various service-level metrics:
- Request rates, latencies, and error rates
- Resource utilization (CPU, memory)
- Queue lengths and processing times
- Service dependencies and availability

9.2 Data Quality Metrics

Data quality-specific metrics provide insights into the health of the data:
- Number of failed expectations
- Validation execution times
- Data profiling statistics
- Trend analysis of quality metrics over time

9.3 Alerting

Alerts are configured to notify the team about data qual-ity issues or performance bottlenecks. The alerting system includes:
- Severity-based alert classification
- Alert routing based on service and team responsibilities
- Automated remediation for certain types of issues
- Integration with incident management systems

9.4 Dashboards

Custom Grafana dashboards provide visibility into the framework's operation:
- Service health overview
- Data quality trends
- Resource utilization
- Cost metrics

10. Machine Learning Integration

A significant enhancement to our framework is the integration of machine learning techniques for advanced data quality features:

10.1 Anomaly Detection

We leverage machine learning to detect anomalies that rule-based approaches might miss:

- Unsupervised learning (isolation forests, autoencoders) for outlier detection
- Time-series analysis for trend and seasonality anoma-lies
- Distribution drift detection across data batches
- Feature correlation monitoring for relationship changes

10.2 Implementation Architecture

The ML pipeline is implemented as follows:
- Historical validation results train baseline models
- Models deploy as separate microservices with containerized TensorFlow Serving
- Periodic retraining occurs using continuous validation results
- A feature store maintains derived features for ML consumption
- Model versioning aligns with expectation versioning

10.3 Performance Impact

ML-based validation supplements traditional rule-based checks:
- 22% increase in anomaly detection capabilities
- 18% reduction in false positives
- Additional compute cost of only 8%
- 15% improvement in early detection of subtle data issues

11. Empirical Evaluation

We evaluated the framework using a series of experiments designed to assess its performance, scalability, and effectiveness in detecting data quality issues.

11.1 Experimental Setup

The evaluation was conducted using:
- GKE cluster with 6 nodes (e2-standard-4 machine type)
- Dataset with 500GB of data across 10 tables
- 200 data quality expectations of varying complexity
- Simulated workload patterns representing typical us-age scenarios
- Comparison implementations: monolithic Python application, AWS Deequ, and DQStream

11.2 Performance Results

The framework demonstrated excellent performance char-acteristics:

- Average validation execution time: 45 seconds for sim-ple expectations, 3.2 minutes for complex expectations
- CPU utilization remained below 70% during peak loads
- Memory usage remained stable throughout the tests
- The system successfully handled concurrent validation requests
- Throughput of 10,000 records/second for streaming validation

Table 3 shows the performance comparison with alterna-tive approaches:

11.3 Scalability Results

Scalability tests showed linear scaling properties:

- Doubling the data volume increased validation time by approximately 85%
- The system effectively scaled from 5 to 50 concurrent validation jobs with minimal performance degradation
- Horizontal scaling improved throughput proportion-ally to the number of pods

Table 3: Performance Comparison

Metric	Our Framework	Monolithic Python	AWS Deequ	DQ Stream
Simple validation	45 sec	120 sec	60 sec	78 sec
Complex validation	3.2 min	8.5 min	4.1 min	6.3 min
CPU utilization	70%	92%	85%	78%
Throughput (records/sec)	10,000	3,500	7,500	6,000

11.4 Cost Analysis

Cost analysis revealed significant efficiency improvements compared to traditional approaches:

- 40% reduction in compute costs compared to a monolithic implementation
- BigQuery optimization reduced query costs by approximately 35%
- Auto-scaling policies prevented resource wastage during low-utilization periods

11.5 Real-Time Validation Metrics

Streaming validation performance showed:

- Average end-to-end latency: 18.5 seconds (simple expectations), 28.3 seconds (complex)
- Throughput: 10,000 records/second with 6 pods
- Resource utilization: 65% CPU, 72% memory during peak loads
- Linear scaling up to 25,000 records/second with added pods

12. Case Study: Implementation at Enter- Prise Scale

The framework was implemented at a large financial ser-vices organization with stringent data quality requirements. The organization processes over 10TB of data daily across hundreds of tables.

12.1 Implementation Approach

The implementation followed a phased approach:

1) Initial deployment with critical datasets
2) Gradual expansion to cover additional data domains
3) Integration with existing data pipelines
4) Development of custom expectations for domain-specific requirements
5) Implementation of ML-based anomaly detection for fraud monitoring

12.2 Technical Challenges and Solutions

The implementation encountered several challenges:

1) **Legacy System Integration**: We developed specialized connectors with adaptable validation logic
2) **Security Requirements**: Implemented zero-trust net-working and encryption
3) **Performance at Scale**: Optimized BigQuery execution with materialized views
4) **Schema Evolution**: Deployed automated expectation adjustment workflows

12.3 Results

The implementation yielded significant benefits:

- 78% reduction in data quality incidents
- Improved data consistency across systems
- Enhanced compliance with regulatory requirements
- Greater visibility into data quality metrics
- Reduced time to detect and remediate issues from 48 hours to 1.5 hours
- 45% reduction in false positive alerts through ML-based anomaly detection

13. Conclusion and Future Work

This paper presented a microservice-based data quality framework that provides a scalable, cost-effective, and robust solution for ensuring high data quality. By leveraging Great Expectations and BigQuery, organizations can effectively monitor, validate, and improve the quality of their data assets. The modular architecture promotes flexibility and allows the framework to adapt to evolving data needs. Our contribution significantly advances the state of the art in data quality frameworks through:

- Superior performance metrics compared to existing solutions
- Real-time validation capabilities with sub-30 second latency
- Cost optimization strategies that reduce infrastructure expenses by 34-54%
- Integration of machine learning for enhanced anomaly detection
- Enterprise-grade security and governance features

Future work will focus on:

1) Expanding real-time validation capabilities with lower latency targets (sub-10 seconds)

2) Enhancing ML-based anomaly detection with explainable AI techniques
3) Implementing federated data quality validation across multi-cloud environments
4) Developing domain-specific expectation libraries for common use cases
5) Creating automated remediation workflows based on common quality patterns

The source code and documentation for the framework will be made available as open-source to facilitate adoption and further development by the community.

Acknowledgment
This work was supported in part by Grant XYZ from Foundation ABC.

References

[1] R. Y. Wang and D. M. Strong, "Beyond accuracy: What data quality means to data consumers," Journal of Management Information Systems, vol. 12, no. 4, pp. 5-33, 1996.
[2] C. Batini and M. Scannapieco, Data and Information Quality: Dimen-sions, Principles and Techniques. Springer, 2016.
[3] L. Chen, H. Zhang, and J. Wu, "Data quality assessment frame-work for cloud data warehouses," Journal of Cloud Computing, vol. 9, no. 1, pp. 1-15, 2020.
[4] Y. Gao, S. Liu, X. Wang, and Z. Zhang, "DQStream: A scalable framework for real-time data quality monitoring in data streams," IEEE Transactions on Big Data, vol. 8, no. 2, pp. 412-425, 2022.
[5] Google Cloud, "Google Kubernetes Engine Documentation," 2023. [Online]. Available: https://cloud.google.com/kubernetes-engine/docs
[6] Google Cloud, "BigQuery Documentation," 2023. [Online]. Avail-able: https://cloud.google.com/bigquery/docs
[7] Great Expectations, "Great Expectations Documentation," 2023. [Online]. Available: https://greatexpectations.io/
[8] A. Haug, F. Zachariassen, and D. van Liempd, "The costs of poor data quality," Journal of Industrial Engineering and Management, vol. 4, no. 2, pp. 168-193, 2011.
[9] P. P. Khine and Z. S. Wang, "Machine learning for data quality: A comprehensive survey," ACM Computing Surveys, vol. 55, no. 3, pp. 1-36, 2023.
[10] T. C. Redman, "Bad data costs the U.S. $3 trillion per year," Harvard Business Review, 2018.
[11] S. Sadiq, T. Dasu, X. L. Dong, and J. C. Freytag, "Data quality: The role of empiricism," ACM SIGMOD Record, vol. 47, no. 1, pp. 35-43, 2018.
[12] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave,... and I. Stoica, "Apache Spark: A unified engine for big data processing," Communications of the ACM, vol. 59, no. 11, pp. 56-65, 2016.