MATERA

Modern Technology of Digital Twin

Digital Twin is API and event-driven software written in Java with four microservices:

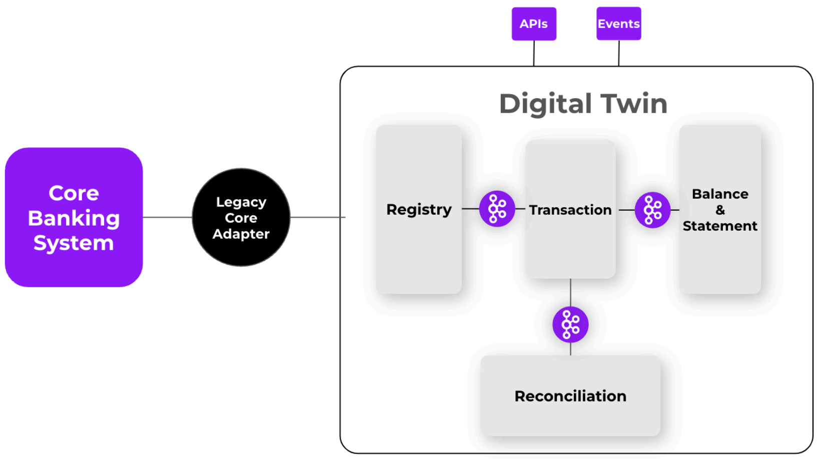
- ***Registry:*** stores account information
- ***Transaction Service:*** approves/denies transactions, stores transaction records and keeps balances up-to-date. It also orchestrates operations across one or more accounts
- ***Balance & Statements:*** used for querying balance and transaction history
- ***Reconciliation module:*** ensures that data is consistent between Digital Twin modules and/or is consistent with the data in the Core.

Each microservice has its own database so activity in one module doesn't impact performance of another. For example, the Balance & Statements module can handle a high volume of queries without slowing down the ability of the Transaction Service to authorize transactions.

Digital Twin is high performance, but doesn't consume unnecessary resources; rather it's installed in the cloud so it scales up or down as needed. The same instance of Digital Twin can be replicated in multiple clouds and/or across different cloud platforms.

It's also architected to hold minimal logic and perform few, but critical functions to achieve extremely high throughput and response times while maintaining resiliency.

Redundancy is built in thanks to this "shared nothing" infrastructure. The same instance of Digital Twin can be replicated in multiple clouds and/or across different cloud platforms.



Digital Twin's microservices communicate by sharing data through Kafka topics[1]. When changes occur in one microservice or when an operation is initiated, such as account creation, events are created and published to a Kafka topic where they are consumed by other microservices to react accordingly.

For example, if there is a transaction event, it is added to a dedicated topic. The Balance and Statement module consumes this event and stores the data in its own database and then uses this information to generate statements. Another module can consume this event as needed.

APIs and Kafka Topics

Digital Twin interfaces with other systems via APIs and Kafka topics.

REST APIs

Two types of REST APIs are supported by Digital Twin:

- **Online:** traditional request-response model
- **Real-time:** allows for continuous data flow without the need for continued requests so data is available in real-time

Kafka Topics

For handling large volumes of data requiring fast and flexible sharing, Digital Twin leverages Kafka, technology able to distribute information to multiple systems simultaneously.

Kafka topics can be expressed as both events and commands.

- **Events**: notifications that an action has been performed.
- **Commands**: initiated by a user or system and requests a system to perform an action. The requesting system waits for a return to know if the action was executed.

Events

Kafka events can be consumed in stream, batch or near real-time. Digital Twin creates events.

Stream processing

Stream processing is often used for applications that need immediate insights, such as fraud detection or live monitoring. Data is processed in real-time or near-real-time as it flows through the system.

Batch processing

[1] https://www.redhat.com/en/blog/apache-kafka-10-essential-terms-and-concepts-explained

Batch processing is often used for tasks that don't require immediate results, such as generating reports or analyzing historical data. Data is processed in large chunks, or batches, after it has been collected over a period of time.

Near real-time processing

Near-real-time processing is used when up-to-date data is a priority, but there's no need to intervene immediately. Data is processed with a few seconds to a few minutes of delay.

Events that are processed on Digital Twin may include completed deposits or payments that occurred in one of Digital Twin’s modules that need to be communicated to other modules or external systems.

Even payments initiated via commands generate events. The consuming system does not need to know whether the payment was made via API or command, or whether it was processed synchronously or asynchronously. It only needs to register that a payment occurred in the specified account. Since events are purely notifications, no response is expected.

For instance, a Core integrated with DTW may subscribe to the Events Topic to receive updates on all transactions processed in DTW, so it can keep its transactional database up to date.

Commands

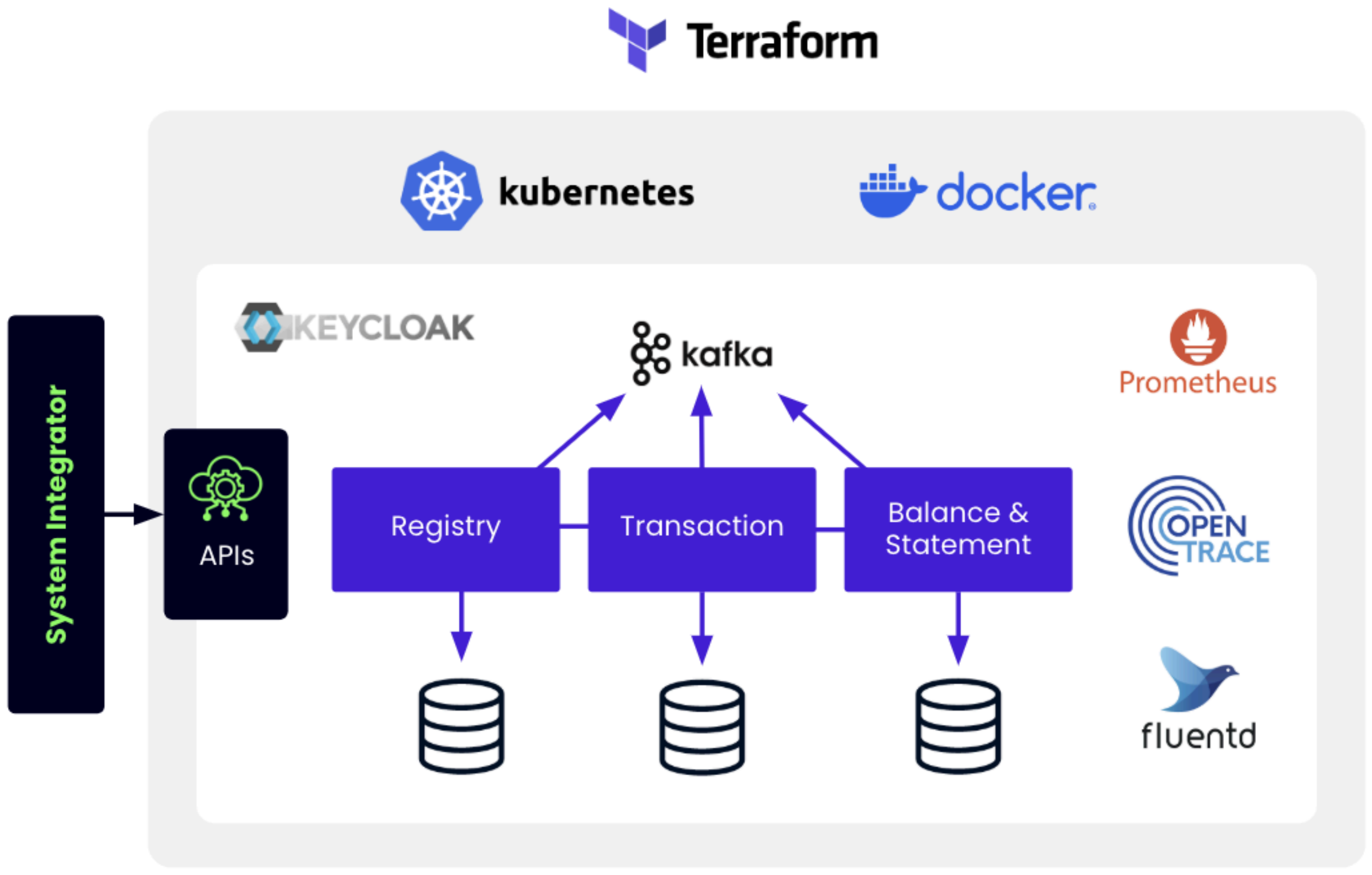
Commands are specific requests one system communicates to another through a request topic and a response topic. For Digital Twin, this could be a request from the Core to make a debit or credit entry, place a balance block or account monitoring. Each command is associated with a return topic, which indicates whether that requested operation was successfully executed or not.

Another example is scheduled payments. For those, the Core sends an asynchronous request to DTW on the scheduled date to debit the relevant accounts. DTW responds with a response command, indicating the payments that were successfully processed. If a transaction is scheduled for the 19th of the month, the debit attempt will occur at some point during that day, but there’s no exact time required. The Core attempts the debit and waits for DTW to confirm whether the account was debited successfully.

Architecture and Technology

The technologies underlying Digital Twin are a collection of modern, scalable, and interoperable tools tailored for high-performance distributed systems. They support real-time data processing, authentication, system observability, and efficient resource orchestration.

Together, these tools empower Digital Twin to handle high transaction volumes, maintain availability during peak demand, and integrate seamlessly with existing infrastructure, making them ideal for instant, secure, and reliable performance required for financial institutions.



- **Microservice and container-based**

  Digital Twin employs a microservice and container-based architecture, allowing for fast and flexible application creation and deployment. Additionally, the application is packaged into Helm Charts and runs on Docker through Kubernetes, making it easier to deploy and operate across different environments. Meanwhile, Kubernetes clusters are created and managed using Terraform technology.

- **Event-based model**

  Digital Twin consists of microservices that communicate by sharing data through Kafka topics. When changes occur in one microservice or when an operation is initiated, such as an account creation, events are created and published to a Kafka topic, where they are consumed by other microservices to react accordingly. For example, if there is a transaction event, it is added to a dedicated topic. The Balance and Statement module consumes and stores it in its own database and then uses this event to generate statements. Another module can consume this event as needed.

- **Infrastructure as Code**

  Environments are provisioned using the infrastructure as code (IaC) approach, replacing manual processes with code. This approach allows for the rapid creation and deletion of environments at scale, ensuring that they are consistent and accurately reflect the desired state as defined in the code.

- **APIs First**

  Digital Twin operations are fully available through standardized APIs. These APIs are easy to use, maintain and integrate with other systems.

- **Streamlined installation**

  Digital Twin utilizes Liquibase for controlling database versions. This method replaces the scattered scripts and robust deltas commonly found in applications such as Oracle. Instead, Liquibase includes all migration scripts within the application. This allows the application to verify whether the database version is up-to-date during deployment. If it is not current, Liquibase runs the appropriate scripts to update the database to the latest version.

- **Observability**

  Digital Twin offers extensive service metrics that can be gathered by Prometheus or an Application Performance Monitoring (APM) system due to its microservice and event-driven architecture.

  OpenTrace provides a standardized approach to distributed tracing, offering a high-level view of how requests are processed. Mapped Diagnostic Context (MDC) complements these efforts by enhancing log messages with contextual details, making it easier to trace the origin of events of an event and troubleshoot issues. Tools like Fluentd can be used to collect and transport logs and trace, centralizing this data.

The combination of metrics, OpenTrace, and logs enhanced by MDC creates a powerful observability stack, enabling development and operations teams to effectively monitor, troubleshoot, and analyze Matera Digital Twin's behavior.

Security & Risk Management

- **Authentication and event tracking**

  Matera Digital Twin uses the OpenID Connect (OIDC) technology to connect authentication servers to the application. It also uses unique numbers called Monotonic Identifiers (UUIDs) to keep track of events in the database and sort them based on when they happened. Matera Digital Twin is designed based on Domain-Driven Design (DDD), where accounts are treated as the main unit. This ensures that events related to a specific account are processed sequentially, so an event that changes an account never arrives before an event that creates it.

- **Data encryption and security**

  While Digital Twin may not provide encryption at the application level, it supports the implementation of additional security measures, such as mutual Transport Layer Security (mTLS) and Kafka's native encryption, to ensure secure communication between different systems and to protect data integrity and confidentiality.

Additional applications to support Digital Twin

- **Prometheus** is an open-source monitoring system used to collect and store time-series data for analyzing system performance. Digital Twin exposes its metrics using the Spring Boot Actuator endpoint. Prometheus collects these metrics from the configured applications and acts as a central storage for the metrics data as well as the data source for Grafana.

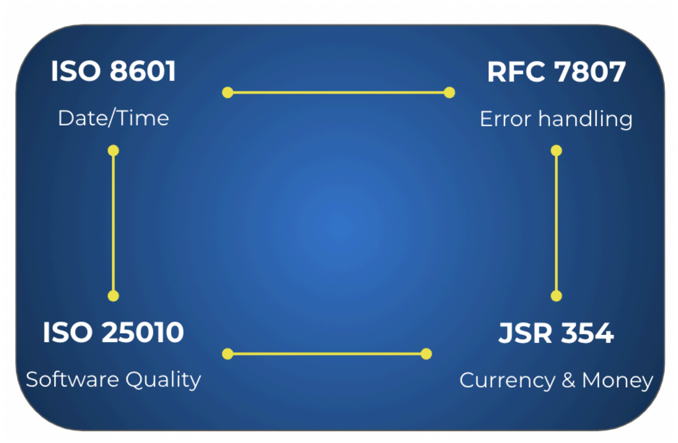
- **Grafana** is an observability stack for observing and analyzing data related to metrics, logs, and traces. It builds dashboards with custom visualizations with data collected and stored by Prometheus. This allows monitoring and analyzing metrics effectively.

- **Keycloak** is employed to manage user authentication, authorization, and security aspects within Digital Twin to ensure only authorized users and systems can interact with its APIs and resources.

- **Fluentd** is a tool used in Digital Twin to gather data and store logs in an external repository. This approach allows a complete record of every request, transaction, or other activity that takes place.

Standards

Digital Twin is built according to common standards to maximize interoperability with other systems.



ISO 8601 - the global standard to represent dates and times that follow the YYYY–MM-DD format for dates, and the 24-hour clock notation for times.

RFC 7807 - standard way to convey error messages in API responses.

ISO 25010 - guidelines for evaluating functionality, reliability, usability, efficiency, maintainability, and portability.

- JSR 354 - set of APIs for handling monetary values and currencies in Java applications.