Published Published onNovember 21, 2017 Unknown author

1. No Relational Database

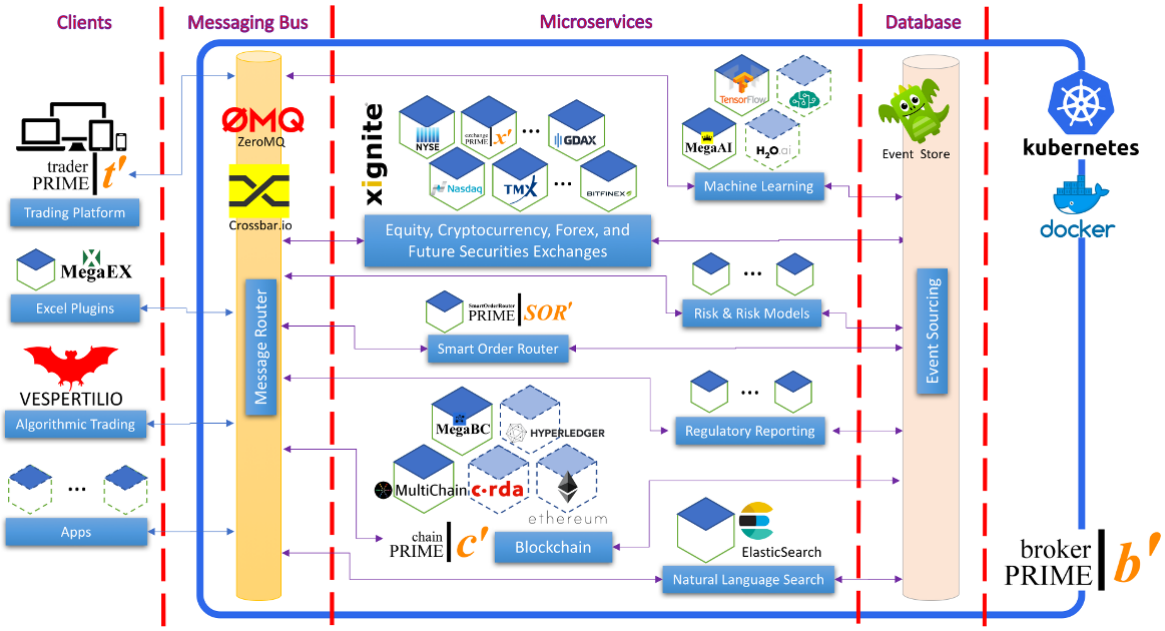
brokerPRIME is a resilient, self-healing, elastic, and horizontally scalable collection of loosely coupled message-driven microservices, deployed as containers in a clustered environment, backed by a functional database.

All of the above comes down to a simple and rather controversial design decision that made such an architecture possible, as well as easy to implement and develop. Here is the **architecture's punch line***:*

> **"Events & Aggregates" are more important than "Data & Relations".**

This statement goes against common and established architectural and design practices. The traditional approach values data and relations and the result is generally a centralized system, which was not our goal.

Another reason for our approach is that the impedance mismatch between the business model and the relational data and databases has a large cost for organizations. One of the most obvious and painful results of such mismatch - and I am pretty sure we have all been there - is that, no matter how well organized the data and its relations are, a simple report still requires a very long SQL statement with numerous join statements, and it gets more complex as time goes by. However, "Events" are themselves a domain concept and therefore such mismatch between events and the business model doesn't occur. If interested, I suggest reading more about Domain-Driven Design and related concepts.



How does "Events & Aggregates" manifest itself in the design?

1. No Relational Database
2. Command Query Responsibility Segregation
3. Message Driven
4. Microservices
5. Orchestration
6. Architecture and Design Sessions

First and foremost, we ditched relational database (RDBMS) altogether, opting instead for a functional database and in particular **Event Sourcing**. The database is essentially a sequence of state-changing events. Event transactions are immutable. There is no delete operation. The store acts as the system of record and can be used to materialize the domain objects. This can simplify tasks in complex domains by avoiding the need to synchronize the data model and the business domain, while improving consistency, performance, scalability, and responsiveness.

- **High Performance and Scalability**: As an append-only model, storing events is a far easier model to scale and optimize compared with a stereotypical relational model. Methods such as Horizontal Partitioning or Sharding can be easily deployed.
- **Superior Business Intelligence**: The focus of event-based database is on how something came into being as opposed to what it came out to be. Storing only current state limits organizations to asking "what" questions about the data, while keeping the history provides both audit-trail and answers to “why”, "when", and “how” questions.
- **Blockchain**: Suffice it to say, blockchian sits well with the model, as

Blockchain-based platforms are used to provide both 1. Multi-asset ledger capabilities; and 2. Storage infrastructure.

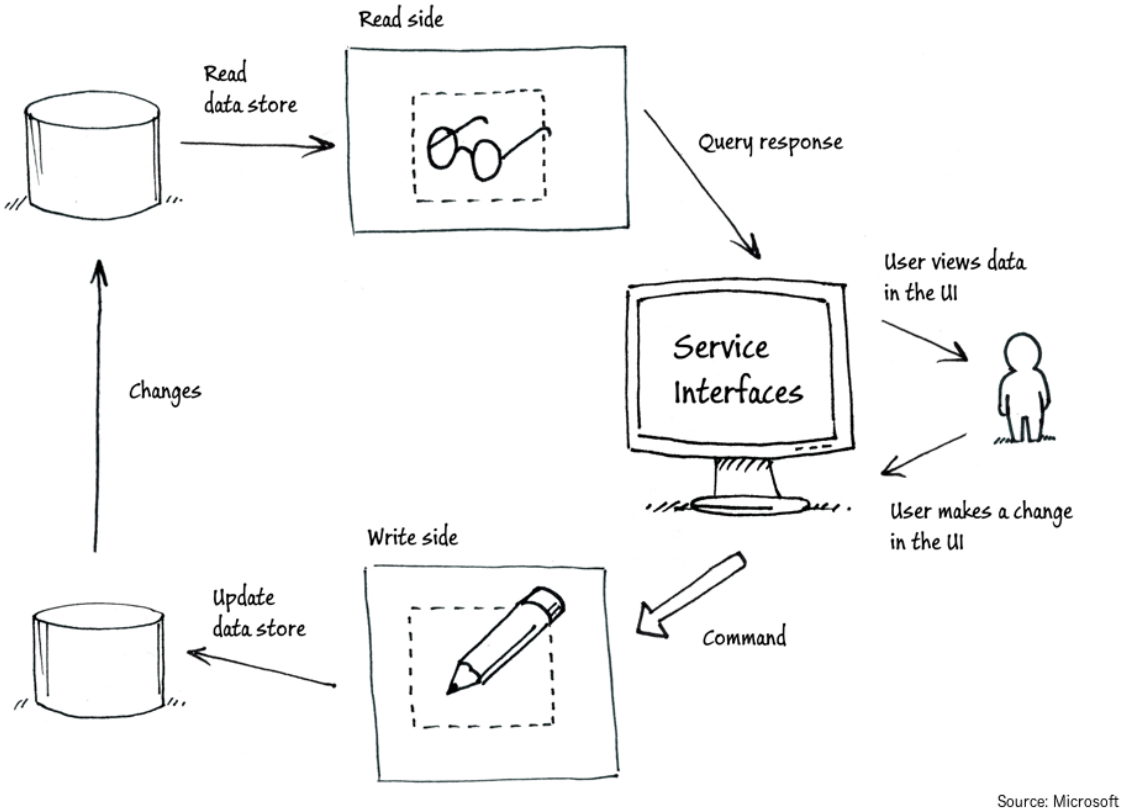
- **Natural Language Search:** As there are no fields, keys, keywords, or indices, Natural Language Search with ElasticSearch provides a superior reporting system. End-users can simply query the database, without the need for a database expert to create a complex SQL statements.

2. Command Query Responsibility Segregation Model

CQRS demands every method should either be a ***command*** that performs an action, or a ***query*** that returns data to the caller, but not both. In Bertrand Meyer's words: "Asking a question should not change the answer."

As CQRS splits the system into separate services, it allows to easily take advantage of the Event Sourcing concept. Microsoft has an excellent article on the subject.

> CQRS pattern enhances scalability, simplifies complexity of the domain, increases flexibility of the solution, and provides greater adaptability to changing business requirements.



Source: Microsoft

3. Message Driven

"Message driven" should not be mistaken with "Event driven". A message is an item of data sent to a specific destination. An event is a signal emitted by a component upon reaching a given state. A message payload can contain an encoded event.

Using only non-blocking and asynchronous messages among components guaranties loose coupling, isolation, and decoupling of the runtime instances from their references. It also allows recipients to only consume resources while active, hence less overhead.

> Monitoring of message queues leads to load management, elasticity, and flow control.

4. Microservices

Microservices provides agility to a complex development, more so than the monolithic large and hard-to-move codebases that traditionally power enterprise applications. Nevertheless, beware of the challenges ahead: Microservices without **Orchestration** ends in disaster.

While microservices is a variant of the service-oriented architecture (SOA) architectural style, there are however a few differences, e.g. microservices is loosely coupled. Another significant difference is the granularity of functionality and scope for a given service: Microservices tends to have fewer operations than its SOA cousin, therefore - and in theory - any single SOA service can be divided into multiple microservices.

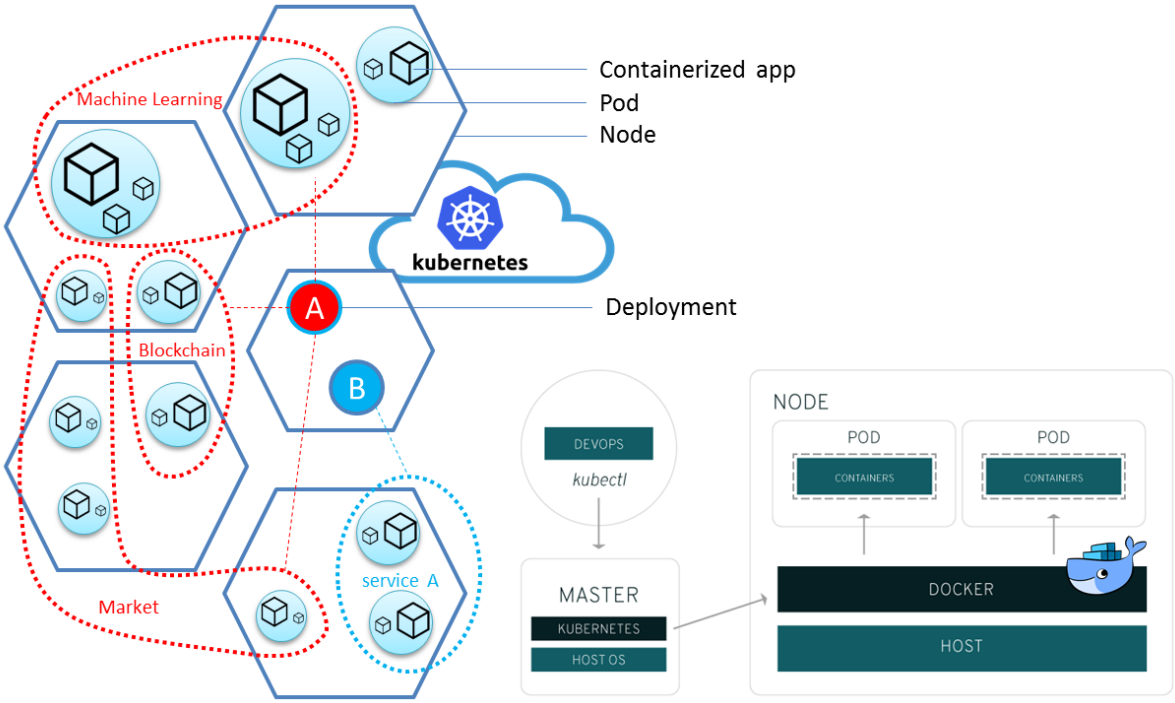
> Microservices provides autonomy, enables greater agility, and requires orchestration.

5. Orchestration

brokerPRIME's microservices spans multiple containers. Those containers are deployed across multiple server hosts. Kubernetes provides the orchestration and management capabilities required to deploy containers, at scale, for these workloads. Orchestration allows building services that

span multiple containers, schedule these containers across a cluster, scale them, and manage the health of the containers over time.

> Self-healing is achieved by auto-placement, auto-restart, auto-replication, and auto-scaling.



6. Architecture and Design Sessions

Traditionally, for such complex domains, it would take weeks to create the architecture blueprints. This can be reduced to days. "**Event storming**" is one way to approach the design and architecture, as explained by Alberto Brandolini with an exploratory way of finding the problems in a domain. Also Greg Young suggests focusing only on one process, the purpose being to discover service boundaries.

Related Projects

- **exchangePRIME**: An Advanced Hybrid Securities Exchange
- **Smart Order Routing**: Machine Learning Smart Order Router
- **traderPRIME**: In a future post, I will discuss the architecture behind it, also a client of brokerPRIME. A cross-asset blockchain-enabled trading platform and execution management system, traderPRIME provides quantitative operations, artificial intelligence, Excel integration, and a unique GUI running natively on Mac, Windows

- **Vespertilio**: is a machine learning algorithmic trading platform by logicallyMAGIC. To be announced and discussed in a later post.
- Many open source frameworks and libraries, among others are: Kubernetes, Docker, Event Store, Crossbar.io, ZeroMQ, Multichain, TensorFlow, H2O.ai, ElasticSearch.

As always, please feel free to contact me directly at HamidJ@logicallyMAGIC.com, I will be delighted to hear your comments and answer questions!
