CASE STUDY

Bitwyre

Bitwyre Trades Redis for SingleStore — and Powers an Ultra-fast, Scalable, Resilient, Secure Cryptocurrency Exchange

1ms
average latency

1M
orders per second

"Most cryptocurrency exchanges create their own challenges and risks. They focus on rapid bootstrapping and delivery, and fail to focus on the performance and scalability of the data platform upon which every trade — and their business — depends."

Dendi Suhubdy,
CEO and Co-Founder

Business Goals

Bitwyre initially used Redis. Redis was single-threaded. It was optimized for fast writes but not simultaneous reads and writes, and trading applications need both for order updates. As a result, Suhubdy and Bitwyre found themselves racing headlong into major issues.

A race condition occurs when two or more parts of an app that rely on each other get locked in an infinite loop, bringing the app to a standstill. "Dirty reads" occur when a transaction is allowed to read data from a row that has been modified by another running transaction and not yet committed. When multiple users or sessions are trying to update or delete the same data in a table, it creates concurrency update problems, so the database locks the data for the first user or session and allows them to update/delete the data — while temporarily locking out other users/sessions. This slows down the database.

With Redis, Bitwyre was hit with race conditions, dirty reads, and database locks. This negatively impacted Bitwyre's performance, accuracy, durability, and resilience. One real-world example of the impacts this had on the business came in the form of stranded sell and buy orders. The Risk Engine in Bitwyre's architecture (more on this below) could not consistently define whether an order needed to be sent to the Matching Engine, and orders failed. On Redis, not only did orders fail, but the entire exchange experienced failures and outages. Redis lacked durability and resilience in the face of those outages.

All in all, Bitwyre knew it was impossible to run its growing cryptocurrency exchange on Redis going forward.

Bitwyre's business goals for its exchange included:

- Minimizing potential outages through a reliable system
- Achieving millisecond performance to support MTM and the other demands of a modern financial exchange
- Achieving massive scalability to support rapid growth in cryptocurrency and its customer base
- Avoiding lawsuits and loss of trading volume due to outages and poor performance

SingleStore

Technology Requirements

Bitwyre had the vision to set itself apart from other crypto exchanges in two crucial ways. It first made the decision to own its data center and use baremetal servers rather than the cloud. It also wanted to avoid the monolithic software architecture that other popular exchanges use, and instead implement a microservices architecture for improved scalability, resiliency, and availability.

Figure 1 illustrates the design decisions Bitwyre has made to ensure scalability, top performance, and reliability compared with the design tradeoffs being made at other cryptocurrency exchanges.

Design Trade-Offs		
Design/Infrastructure Difference	**Other Exchanges**	**Bitwyre**
Monolithic vs. Microservices	Monolithic - Single Database	Microservices - Multiple Databases
Single Threaded vs. Multi Threaded	Multi Threaded	Single Threaded containers with multiple partitioned message queues
Matching Engine	Database Operations (eg. Big Query, KX Systems kdb+	All in-memory data structures with event sourcing
Cloud vs. Baremetal	Cloud	Baremetal

Figure 1: Cryptocurrency Exchange Design Tradeoffs

Building the exchange on a microservices foundation addresses a large or complex problem by using a set of smaller, simpler services that work together to achieve the business goal. Each microservice is a self-contained process delivering a specific capability that rolls up into the business application. Each sprint team operates its own microservice, managing its own product backlog, developing its portion of the app, and implementing updates and fixes as needed, all independently from the other sprint teams. No team has to wait for another team to complete its work — and no one microservice can take down the entire business application.

Why SingleStore?

Bitwyre wanted an in-memory database that worked with RedPanda, the Kafka-compatible event streaming platform, as well as a Kafka-compatible API and REST API gateways. The database had to support baremetal deployment in Bitwyre's own data center running a Kubernetes operator; support a distributed, microservices architecture; and deliver ultra-fast performance to keep up with the millisecond market demands of all financial markets, but particularly cryptocurrencies.

SingleStore met all of these requirements and more. It supports microservices design and Kubernetes on baremetal machines, and is designed for scaling simultaneous reads, writes, and order updates.

Another key factor in Bitwyre's strategy was cost. "We are not VC funded; we're bootstrapping the company to profitability, but unlike others in our position, we're doing it with a vision and a scalable data strategy to actually help us get there," said Suhubdy. "SingleStore offered the millisecond performance, scalability, high availability, and fault tolerance upon which we can confidently run a global cryptocurrency exchange."

Solution

Bitwyre is running SingleStore DB on baremetal hardware in Bitwyre's own data center, on top of four virtual machines (VMs). The application is built with 100 microservices running Risk Engines, Matching Engines, Feed Engines, and a Gateway. Every engine except the matching engine is running in Kubernetes for added resiliency.

Figure 2 illustrates Bitwyre's architecture.

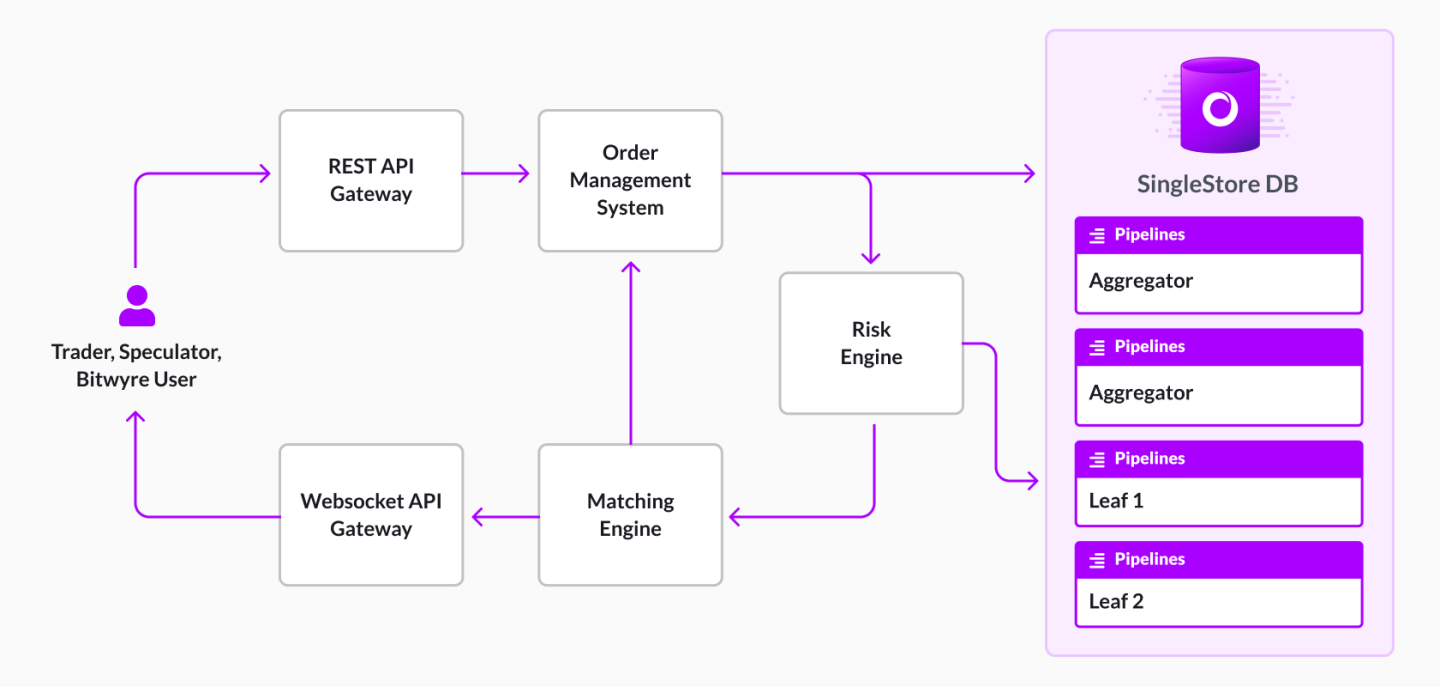

Figure 2: Bitwyre's Architecture, Powered by SingleStore

Suhubdy explained a key feature of Bitwyre's architecture. The Order Management System (OMS) and Risk Engine each have their own state, and with SingleStore, "You can support microservices principles, with each microservice reporting its own state, while connecting to one SingleStore cluster."

The concept of a pod in Kubernetes makes it easy to tag multiple containers for treatment as a single unit of deployment. They are co-located on the same host and share the same network, memory and storage resources. Essentially, a pod is the new VM in the context of microservices and Kubernetes.

Financial exchanges are graded on round-trip time: total elapsed time from the moment an order enters the system to the moment it flashes back out to the customer. Bitwyre's Matching Engine built on SingleStore delivers matching throughput of six million orders/second and event consumers and producers are capable of one million orders/second, respectively; so resulting system throughput, or round-trip time, is 1M orders/second.

"We're self-hosting SingleStoreDB on four VMs on baremetal. Our SingleStore Premium Enterprise license enabled us to scale our trading engine to one million orders/second with one millisecond latency, or round-trip time. Other crypto exchanges have an 18 millisecond round-trip time," said Suhubdy. "SingleStore is able to deliver this level of latency even while supporting multiple pods with parallel reads and writes," said Suhubdy.

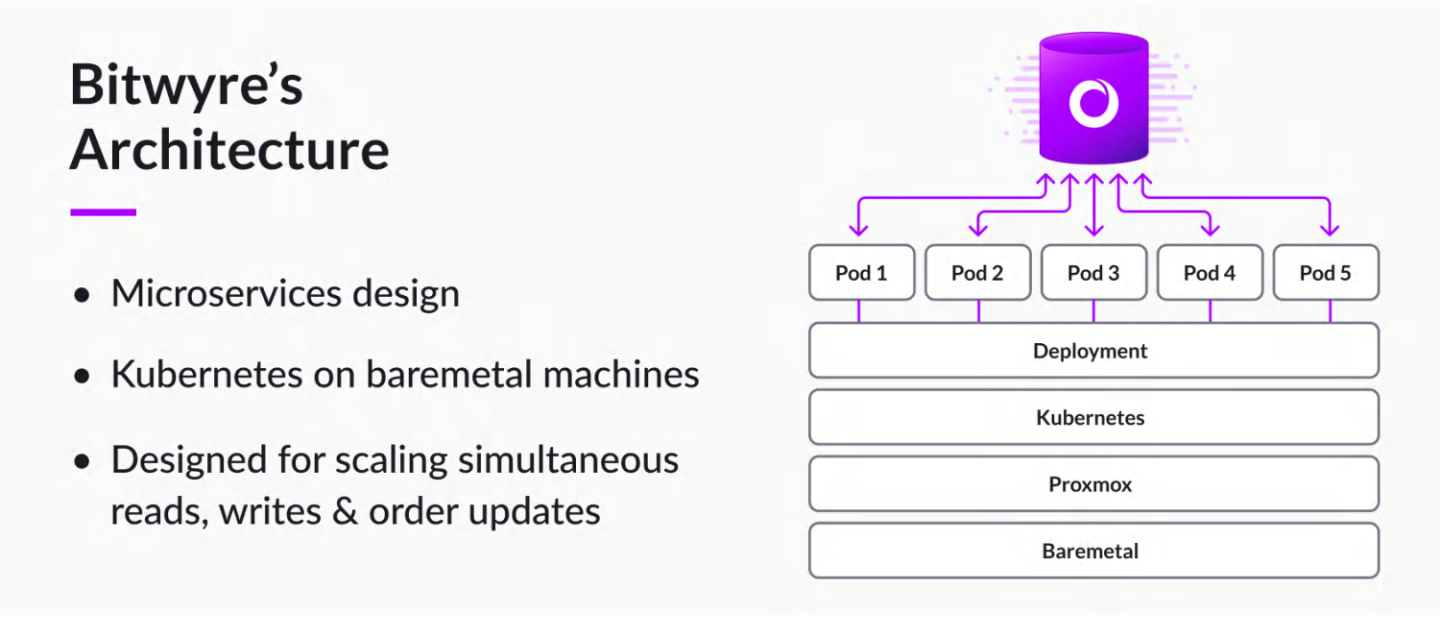

Figure 3: Bitwyre's Order Management System - 1ms latency across multiple pods; parallel reads/writes

Speed, scalability, and high availability are critical at Bitwyre. So, too, is security & compliance.

'Wash trading' is the illegal process of buying shares of a company through one broker while selling shares through a different broker so the trades essentially cancel each other out. HFT firms and cryptocurrency exchanges can also use wash trading to manipulate prices, e.g., by creating fake volumes for a stock at one investment firm to pump its price — then shorting the stock at another firm to reap a windfall.

Bitwyre manages time series data inside ColumnStore for further analysis to detect and prevent fraud including 'wash trades' and other insider trading, and to support various forms of real-time market analytics.

"One thing we are particularly proud of is that we do not use soft matching of data," said Suhuby. "Our matching algorithm ensures speed and precision in our system to help prevent insider trading and fraud."

Suhubdy added, "We migrated approximately 100 microservices from Redis to SingleStore in less than two months. We didn't have any major challenges moving to SingleStore because the developer experience was that good."

Technology Requirements

The visionaries at Bitwyre have implemented SingleStore to power their ultra-fast, massively scalable, resilient, and secure cryptocurrency exchange that differentiates them from competitive exchanges and future-proofs their business.

A millisecond-speed, and durable, exchange. And no "dirty reads"

"After we made the move to SingleStore, we got no more dirty reads. It really scales for simultaneous reads and writes. The data uses tiered storage, so order states are very recent. Since we have nanosecond timestamps, we see everything that happened within the last second. It's very fast," said Suhubdy.

Better experience for traders

"Throughput allows you to monetize volume, but speed is what gets people to keep trading. If they don't have fast execution, they'll probably look for a different exchange," said Suhubdy.

Bitwyre's Matching Engine design leverages SingleStore's all in-memory data structures with event sourcing to keep existing traders trading and new ones coming in the door.

Increased profitability for Bitwyre

Bitwyre's ability to handle higher daily trading volumes drives more revenue and is supporting its bootstrapping efforts to drive trading volume in excess of $1.5 million per day. $1,500,000/day

"We can handle more daily trading volumes, and our net profit is determined by 4.5 basis points of that volume. The higher the throughput is, the higher our profitability," concluded Suhubdy.

Reduced risk of disruptions and outages

Bitwyre put SingleStore's High Availability nodes to the test — not in test, but in production — to confirm that the system would be resilient and accurate in the event of failure. "We deleted one leaf and multiple pods to see if it impacted the trading system in live production," said Suhubdy. "The system performed like there was no problem with the Order Management System."

Fraud detection to ensure trader confidence and regulatory compliance

Bitwyre is using SingleStore to manage time series data in order to detect and prevent fraud. Beyond that, this same time series data enables Suhubdy and team to support a range of other analytics about the market.

LEARN MORE

To hear Bitwyre discuss its innovative implementation of SingleStore, watch this on demand webinar: Launching a Cryptocurrency Exchange >

Check out this Bitwyre Guest Blog Post and video: How We Reverse-Engineered the Chicago Mercantile Exchange >

Bitwyre designed a C++ program, in Suhubdy's words, "for blasting data at scale to websocket clients." They invite you to check it out on Github >