CockroachDB

Payments

> Payments is the process of initiating, authorizing, capturing, and exchanging a monetary value.

Virtually every business in one way or another has to deal with payment processing. It's easy to understand how a missed or incorrect payment impacts a company's bottom line and credibility. It's also easy to understand how frustrating it is when a payment system isn't working or charges you incorrectly.

Typically there are two types of payment use cases: payment processing and payment gateways. A payment processing system resolves money movement between financial institutions, while a payment gateway accepts payments. There are also subcategories within the payments ecosystem which include use cases such as card authorization, fraud, rewards, and gift cards.

Challenges

The speed and correctness of a payment matters during the time in which a payment transaction takes place. However, because of the possibility of things like returns, refunds, disputes, and audits, payment data typically needs to remain available for years.

Customers (and auditors) are not forgiving when it comes to money which means data correctness and availability is critical. Even a very small amount of downtime or a missed calculation could equate to financial fines, lost revenue, brand deterioration, and customer churn.

Without question, security and audit capabilities are table stakes for these systems. Successful businesses utilize key audit information generated from payment transactions to ensure quality, protect against fraud, conduct root cause analysis, and instrument the overall system. Ensuring that logging, security and auditing doesn't overwhelm the system is a constant trade off with system performance.

Lastly, relying on your application (as opposed to your infrastructure) to handle consistency, resiliency, and scale creates challenges for your operations and development teams. Building consistency checks in your application takes effort and creates excess code. Building highly resilient applications without an active-active setup takes a lot of toil and complex operational runbooks. Manually sharding in order to scale requires teams to re-architect the application so it will work in the future.

Database requirements

Security and auditing of all interactions, including data mutations and data access, need to be available and integrated into the organizational authentication, authorization, auditing, and encryption systems. Security and audit is the one feature that gives business owners a peace of mind about the database.

The actual process of storing payment data in a database isn't complex or particularly different from storing other types of data. At smaller scales, the consistency and correctness requirements for payments can be met with the ACID transactional guarantees offered by most SQL databases. For larger systems, it's wise to build a payment infrastructure with correctness in mind which means using a database that allows you to achieve idempotency.

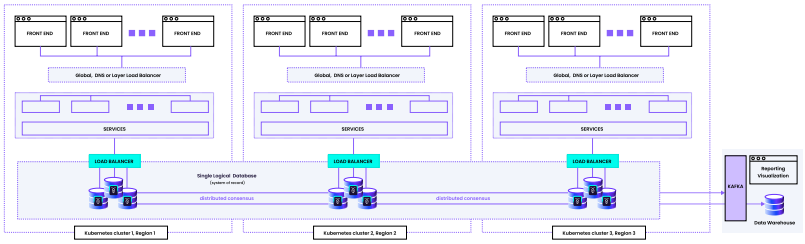
As a company scales up, the regular transactional load placed on the database, as well as the ever-increasing total volume of data, can quickly become a performance bottleneck. Maintaining performance as your business grows requires a solution that allows you to scale up your payments database without sacrificing correctness or availability. You'll also want a database that doesn't require manual sharding.

Why CockroachDB for payments?

Payments systems rely on CockroachDB to handle the complexity in the database layer so that they can deliver customers real-time transactional correctness all the time, every time.

Reference architecture

Visualize how CockroachDB's dramatically simple multi-region architecture enables payment systems to be highly available and strongly consistent with greatly reduced operational upkeep.



Shipt

Customer Story: Shipt relies on CockroachDB to deliver correct payments

Shipt, a grocery ecommerce company owned by Target, maintains a suite of payment services that are crucial to their business model. Here's a basic model of how a transaction flows through their payment service and how they set up idempotency.

For Shipt, it was crucial to have concurrency control in a distributed system so that they could block concurrent requests for the same payment. They built their system on CockroachDB which allows them to achieve guaranteed atomicity, consistency, isolation, and durability (ACID) all the way down to the row level.

> *When working with CockroachDB you have to develop a distributed mindset. We knew we wanted to build a multi-region application so we started to ask ourselves, what part of the application layer do we put in the complexity to make this work? But with CockroachDB, the complexity is not in the application layer, a majority of it is handled by the database. This makes it substantially easier to build multi-region services.*

David Templin
Senior Software Engineer

To learn more about Shipt's use case and read other customer stories, visit cockroachlabs.com/customers