How LinkedIn uses Event Driven Architectures to Scale

An introduction to EDAs and the Actor Model. Plus, how to ship projects at big tech companies, how Coinbase uses ML to predict traffic patterns and more.

November 15, 2024

Hey Everyone!

Today we'll be talking about

- **How LinkedIn uses Event Driven Architectures to Scale their Infrastructure**
  - An Introduction to the Actor Model and how it works
  - How LinkedIn collects and processes server metrics from their fleet with an Event Driven Architecture
  - LinkedIn's monitoring system for server consoles
- **Tech Snippets**
  - How I Ship Projects at Big Tech Companies
  - How Binary Vector Embeddings work and why they're so useful
  - How Coinbase uses ML to Predict Traffic and Auto-scale Databases

How LinkedIn uses Design Patterns to Scale their Infrastructure

LinkedIn is the largest professional social networking platform in the world with over 950 million users in 200+ countries.

To serve this user base, they maintain dozens of data centers around the world with hundreds of thousands of servers globally.

In order to manage these servers, LinkedIn makes use of many tried-and-tested design patterns.

One pattern is the Producer-Consumer pattern, commonly used in event driven architectures (EDAs).

This pattern consists of three main components:

- **Producer** - generates events/messages (server metrics, status updates, data from queries, etc.)
- **Queue** - acts as a buffer to store messages until they're ready to be processed. LinkedIn uses Redis, Kafka or built-in queues for this.
- **Consumer** - reads and processes messages from the queue

Saira Khanum is a Staff Software Engineer at LinkedIn and she wrote a fantastic blog post delving into how the engineering team uses this pattern in three different systems:

1. To Collect and Maintain Data from Servers for Real-time and Analytical Queries
2. To Check Servers for Availability and Accessibility
3. To Detect and Fix any Access Policy Violations on the Servers

We'll explore these and talk about how LinkedIn implemented them.

Actor Pattern

When building event driven architectures, LinkedIn frequently uses the *Actor Pattern*. Event Driven Architectures are loosely defined so the Actor Pattern (*or Actor Model*) is a specific implementation of an EDA.

With this model, everything is represented as an *actor*.

An actor is an independent entity that can

- Send messages to other actors
- Process messages/requests
- Create new actors and designate their behavior
- Have independent state

To give you a better sense of how this might work, here's a *hypothetical* example of an Actor model at Uber for handling ride requests.

1. When a user first requests a ride, a **RequestActor** is created specifically for their request. This actor maintains the state of the request (*whether it's active or canceled*) and coordinates the entire matching process.
2. The RequestActor might first create a child **PricingActor** to figure out a reasonable price for the request based on the trip distance and time of day. The PricingActor will run internal logic based on the RequestActor's message and return the ride price.
3. Once it has the pricing figured out, the RequestActor will communicate with nearby DriverActors (*one actor per active driver on Uber*) by sending them ride offer messages.
4. The DriverActor will then handle sending a notification to the Uber driver that there's someone looking for a ride. If the driver accepts the ride then the DriverActor might create a new **TripActor** to handle the ongoing ride (tracking location updates, route changes, payment processing, etc.)

If you're looking for more details, here's a fantastic article that delves deeper on the Actor model.

Back to LinkedIn...

Event Driven Architectures at LinkedIn

LinkedIn talks about a few systems where they've found EDAs useful for managing infrastructure.

Distributed Server Queries at LinkedIn

The first system is LinkedIn's distributed server query system. This is responsible for collecting system facts (CPU/memory usage, network connections, disk space usage, etc.)

from across the server fleet and storing them so they can be queried and analyzed.

Some of the requirements are

- **Scale** - the system needs to process terabytes of data from hundreds of thousands of servers in near real-time
- **Data Refresh** - the data needs to be collected several times every hour
- **Data Maintenance** - the last known good snapshot of system facts needs to be maintained for a defined retention period. (*after the retention period is over, the system facts need to be marked as stale*)

Here's the high level architecture of the system

1. Agents (producers) are deployed across the server fleet to collect system facts
2. These facts are sent to worker processes (using the Actor Pattern) and stored on Redis
3. Different worker processes consume the data from Redis, process it and store it in different datastores

Some of the choices LinkedIn made were

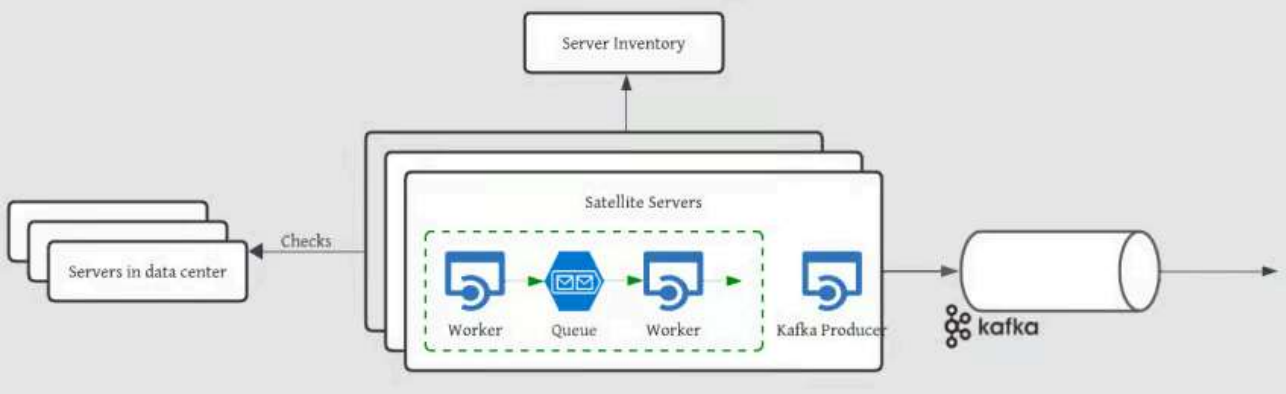
- **Redis** - LinkedIn picked Redis as the queue since they were looking for low latency. The messages are short-lived and introducing a tool like Kafka would introduce too much overhead.
- **Actor Pattern** - Workers that collect and process server metrics use the Actor pattern. They're implemented with Gunicorn.

Server Console Monitoring

The second system is LinkedIn's distributed system to monitor the server console for their infrastructure. Server consoles (*often called service processors*) allow administrators to manage and monitor physical servers remotely (*even when the server is powered off or unresponsive*). They're essential for troubleshooting, rebooting and maintaining servers.

LinkedIn's monitoring system checks that these server management consoles are available and accessible.

Here's the architecture for how they do that.

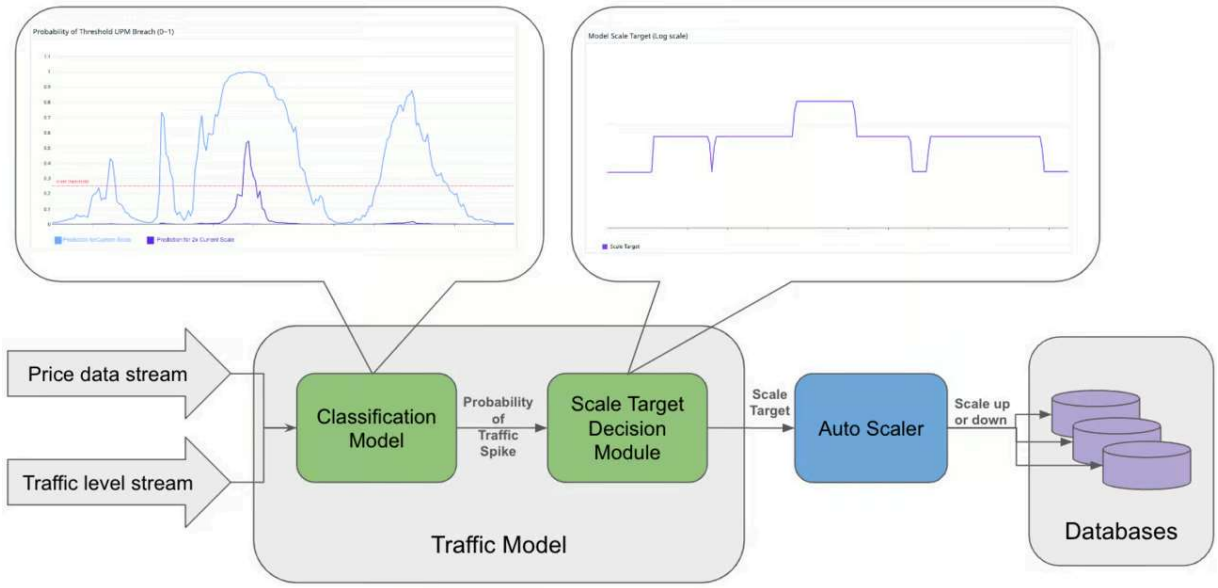


1. Satellite servers run checks across the servers in the data center. Each check is handled by a separate actor.
2. Messages from each check are passed through RabbitMQ. The result of each check determines if the next check should be run (*if the next actor should be created*)
3. Final results are sent to Kafka. Consumer applications can read results for storage/analysis from the various Kafka streams.

Some of the tech choices LinkedIn made were

- **Actor Pattern -** Each check that LinkedIn has to do is an actor. The checks are done sequentially so they pass messages to each other to send results and status updates.
- **Kafka and RabbitMQ -** RabbitMQ is used for communication between the actors whereas Kafka is used for forwarding the final results down to the consumer applications for further processing and storage

Tech Snippets



How Coinbase uses Machine Learning to Predict Traffic and Autoscale Infrastructure

With crypto markets, sudden price movements can cause massive traffic surges on Coinbase.

Instead of scaling reactively based on CPU usage (which is often too late), Coinbase built an ML model that predicts traffic spikes in advance by analyzing:
- price volatility in major cryptocurrencies
- current traffic patterns and growth rates
- historical seasonal trends
- load testing data

This has helped them prevent system outages and reduce costs from over-provisioning resources.

www.coinbase.com/blog/how-coinbase-is-using-machine-learning-to-predict

How I ship projects at big tech companies

Sean Goedecke is a Staff Engineer at GitHub. He wrote a terrific article on what it actually means to “ship“ projects at big tech companies.

Here’s some insights

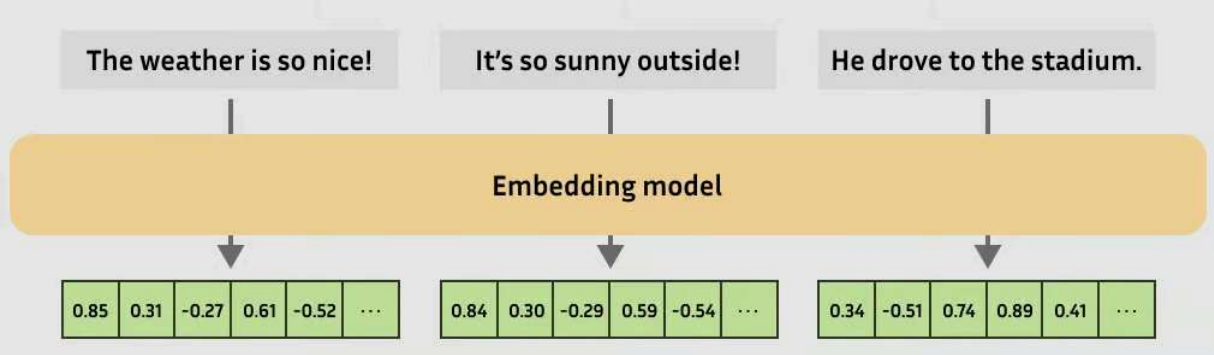
- Shipping isn't automatic - the default state of most projects is to get delayed indefinitely. Someone needs to take ownership and make sure it gets launched.

- Shipping is more than just deploying code - A project hasn't truly shipped until important stakeholders acknowledge it.

- Deploy Early - Sean recommends deploying features behind flags as early as possible so you can catch issues early.

Read the full article for more details.

www.seangoedecke.com/how-to-ship



How Binary Vector Embeddings work and why they're so useful

Vector Embeddings allow you to convert text into numbers that represent meaning. This is super useful for semantic search and similarity matching.

Traditional embeddings use 32-bit floating point numbers but binary quantization converts each number to a single bit.

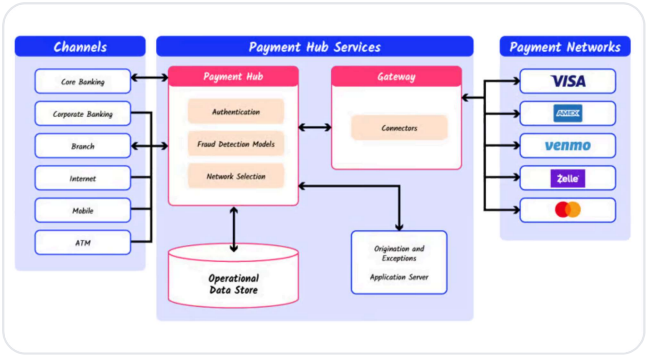
With this approach, you can compress embeddings to just 3% of their original size while retaining 95%+ of the original accuracy.

Evan Schwartz wrote a fantastic article on this technique.

emschwartz.me/binary-vector-embeddings-are-so-cool

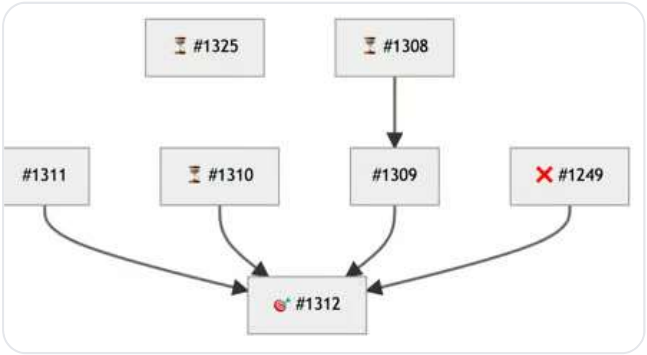
Essential Reading For Engineering Leaders

Keep reading



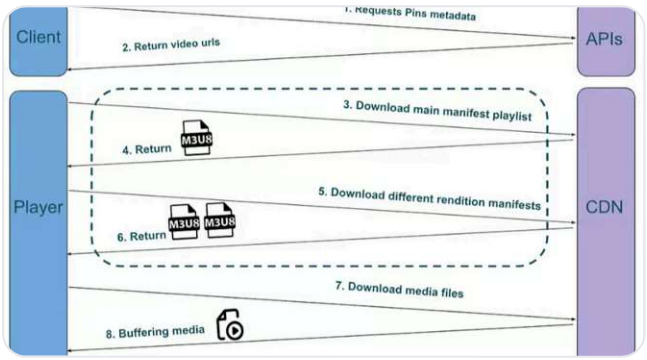
How Stripe synchronizes time across their distributed…

How Stripe uses physical and logical clocks for keeping time. Plus, a deep dive on caching in system design, how eBay uses LLMs…



How Reddit built a Metadata Store that Handles 100k Reads…

We'll talk about the design of Reddit's Metadata Store and the tech behind it. Plus, non-llm software trends to be excited…



How Pinterest Optimized Video Playback

An introduction to Adaptive Bitrate Streaming and how Pinterest was able to reduce startup latency for videos. Plus,…

View more ›

Quastor

Get Summaries of Big Tech Engineering Blog Posts on Frontend, Backend, Machine Learning, Data Engineering and more!

Home
Posts

Account
Upgrade

Premium Articles
Premium Articles

E... Join Free

Privacy Policy Terms of Use

Powered by beehiiv