influxdata

AN INFLUXDATA CASE STUDY

How Robinhood Built a Real-Time Anomaly Detection System to Monitor and Mitigate Risk

External Contributors

Allison Wang

Software Engineer, Robinhood

October 2019 (Revision 1)

Company in brief

Robinhood's story began at Stanford, where co-founders Baiju and Vlad were roommates and classmates. After graduating, they packed their bags for New York City and built two finance companies, selling their own trading software to hedge funds. There, they discovered that big Wall Street firms were paying next-to-nothing to trade stocks, while most Americans were charged commission for every single trade. So they decided to change that, and headed back to California to build a financial product that would enable everyone — not just the wealthy — access to financial markets.

A pioneer of commission-free investing, Robinhood is on a mission to democratize finance for all and believes the financial system should be built to work for everyone. That's why the company creates products that let you start investing at your own pace, on your own terms.

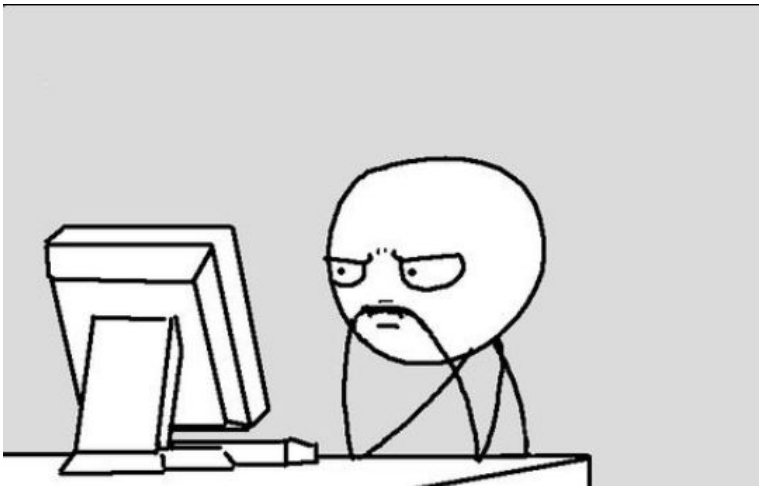
Case overview

Robinhood is democratizing the financial systems by offering commission-free investing and trading with the use of your phone or desktop. As exciting as that sounds to the outside world, internally, the team at Robinhood had to understand the different risk vectors and build engineering solutions to mitigate these risks. To build a real-time risk monitoring system, Robinhood chose InfluxDB (an open source time series database written in Go) and Faust (an open-source Python stream processing library for Kafka streams). The architecture behind the system involves both time series anomaly detection (InfluxDB) and real-time stream processing (Faust/Kafka) setups.

> *"As the number of time series grows, the amount of effort needed to understand or to detect anomalies in a time series becomes extremely costly. That is why we started to build an anomaly detection system that can intelligently alert us when something doesn't go very well."*
>
> ***Allison Wang**, Software Engineer*

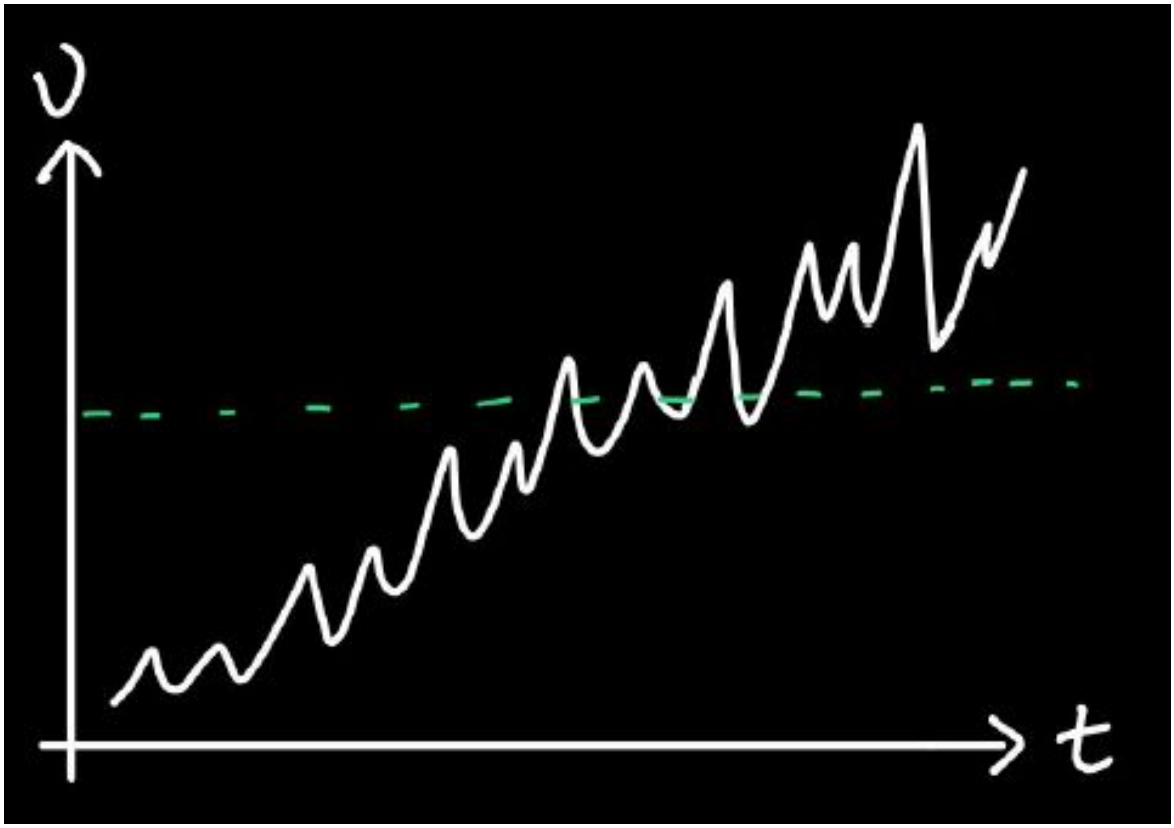
The business problem

To monitor and mitigate risk, Robinhood wanted to set up intelligent, real-time alerts on critical metrics without the need for constant manual dashboard tracking. Faced with a large volume of metrics, engineers and IT operations teams need a smart, automated way to detect anomalies in time series so that they can quickly assess risk. Since staring at a screen to track thousands of time series 24/7 to take immediate action isn't practical or scalable, Robinhood needed to build an anomaly detection system.



The technical problem

To solve their business challenge, the first anomaly detection solution that Robinhood tried was threshold-based alerting, by which an alert is triggered whenever the underlying data is over or under the threshold, as shown below.



Threshold-based alerting

Threshold-based alerting works well with simple time series (for example, to detect whether the amount of CPU utilization is over 80% or the percentage of server currently running is over 100%). But it fails to account for more complex time series involving seasonality and trends. As shown above, the time series has an upward trend, and within that upward trend, there are up-and-down patterns.

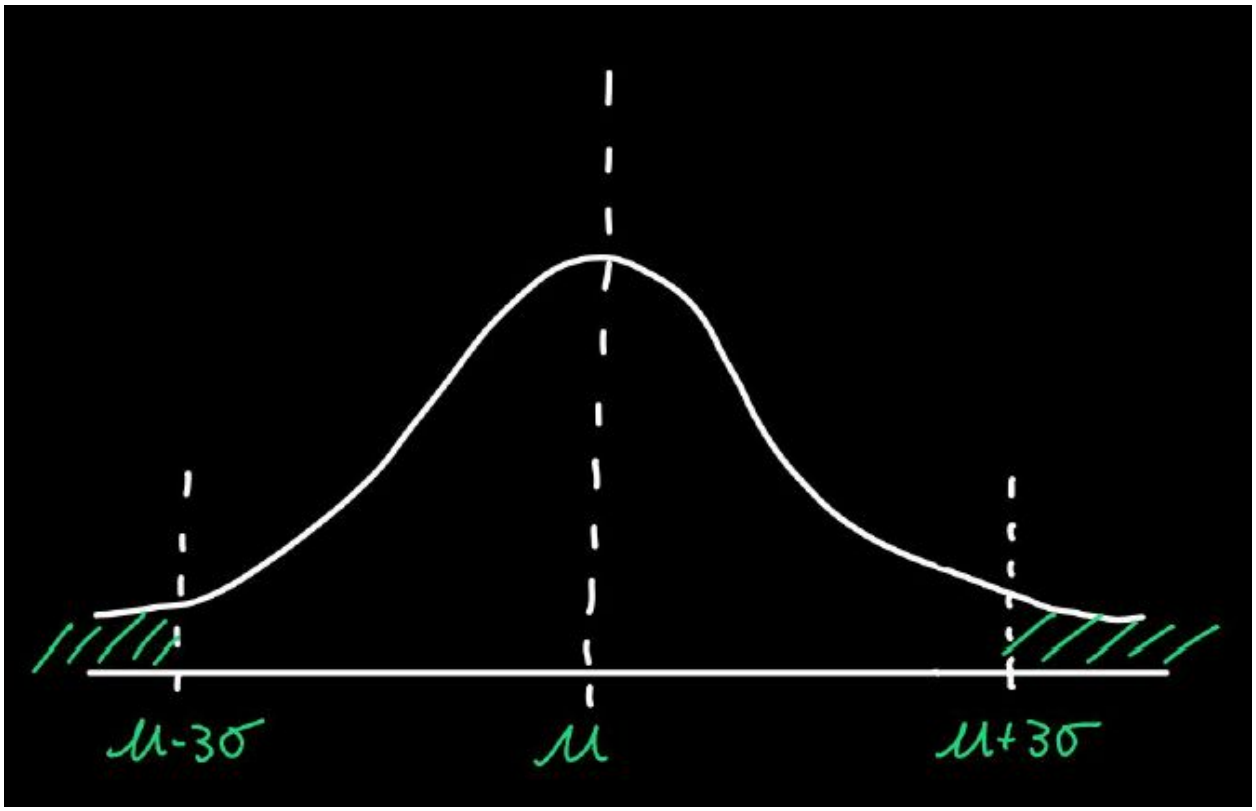
Using the fixed threshold to alert on anomalies doesn't work well because the time series will go over the threshold and trigger an alert, but will then drop down a threshold and go over a threshold again. So threshold-based alerting in the case of complex time series would require the same effort as checking the dashboard 24/7. Therefore, they needed an anomaly detection algorithm.

Creating an anomaly detection algorithm

They sought to leverage historical data to determine — given an incoming data point — what a reasonable threshold would be, in order to define the anomaly state. So, they resorted to a concept derived from statistics called normal distribution. The idea behind normal distribution is that given a list of data, you can compute the mean, and the standard deviation of the data points. To put the normal

distribution concept to work for their purpose, Robinhood alerted on data outside of three standard deviations:

- 1σ - 68.27%
- 2σ - 95.45%
- 3σ - 99.73%



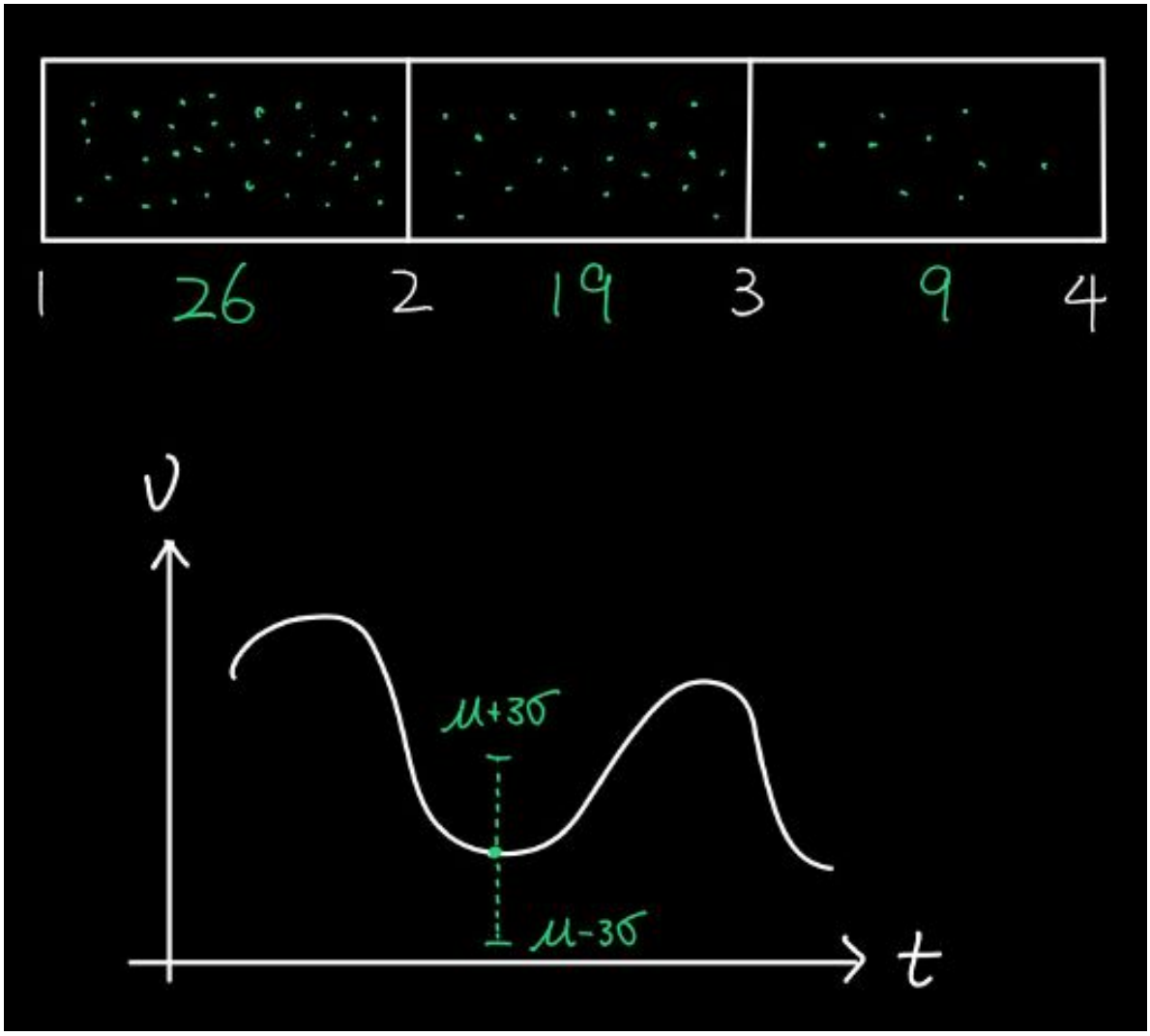
An example of data with a normal distribution. Data that is outside of three standard deviations away from the mean (shaded with green lines) accounts for only 0.03% of all of the data.

Defining your threshold from a standard deviation for anomaly detection is advantageous because it can help you detect anomalies on data that is non-stationary (such as in the example above). The threshold defined by a standard deviation will follow your data's trend. Since Robinhood defined an anomaly as anything outside of three standard deviations away from the mean, this meant 99.7% of the data lies within this range.

Bucketing data points

Here's how the normal distribution concept can be applied to anomaly detection. If you have a stream of data coming in, you can:

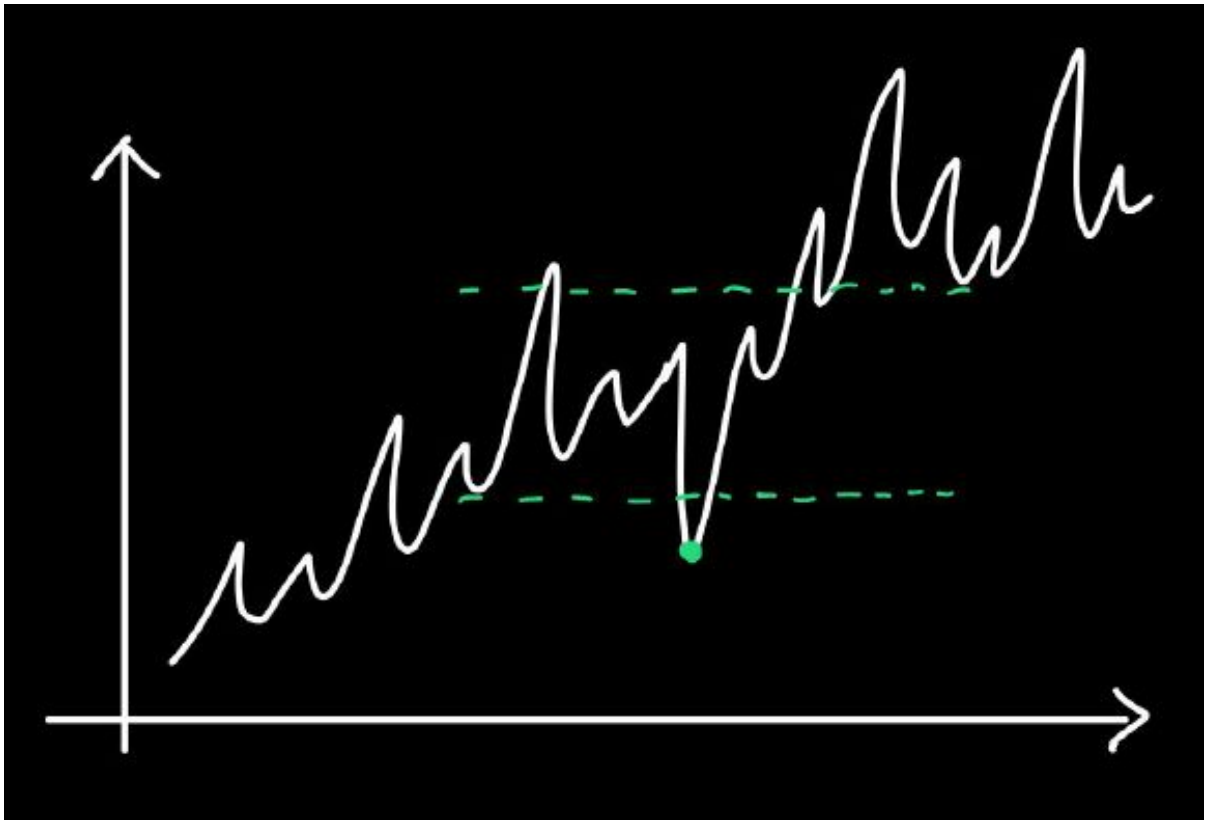
1. Bucket those data points within a very small time interval, such as a minute
2. Construct normal distribution for every minute in the day for the past 30 days



For example, they count the number of data points coming in within the first minute and the second minute, which is 26, for example. Then they count the second and third minute — and the third and fourth and so on. Once they have an aggregation, they apply it to the data point that is over the past 30 days. This is how they get a list of aggregated data points for every minute in the day for the past 30 days.

They utilize the time series that they aggregate to compute the mean and the standard deviation over the past 30 days and use that as a boundary for a time series threshold. After computing the threshold, whenever there's a new data point coming in, they can compare the number with the threshold. If the incoming data point is over the threshold or under the threshold, then they alert. This is the idea behind anomaly detection using data in the past.

Now that they set an anomaly detection algorithm, the next step was to productionize the algorithm in their system so that they can automate alerting.



Checking if the incoming data point (aggregated) is in the range ($\mu - 3\sigma$, $\mu + 3\sigma$)

As they wanted to monitor time series data, they realized they needed a time series database that would meet their system requirements:

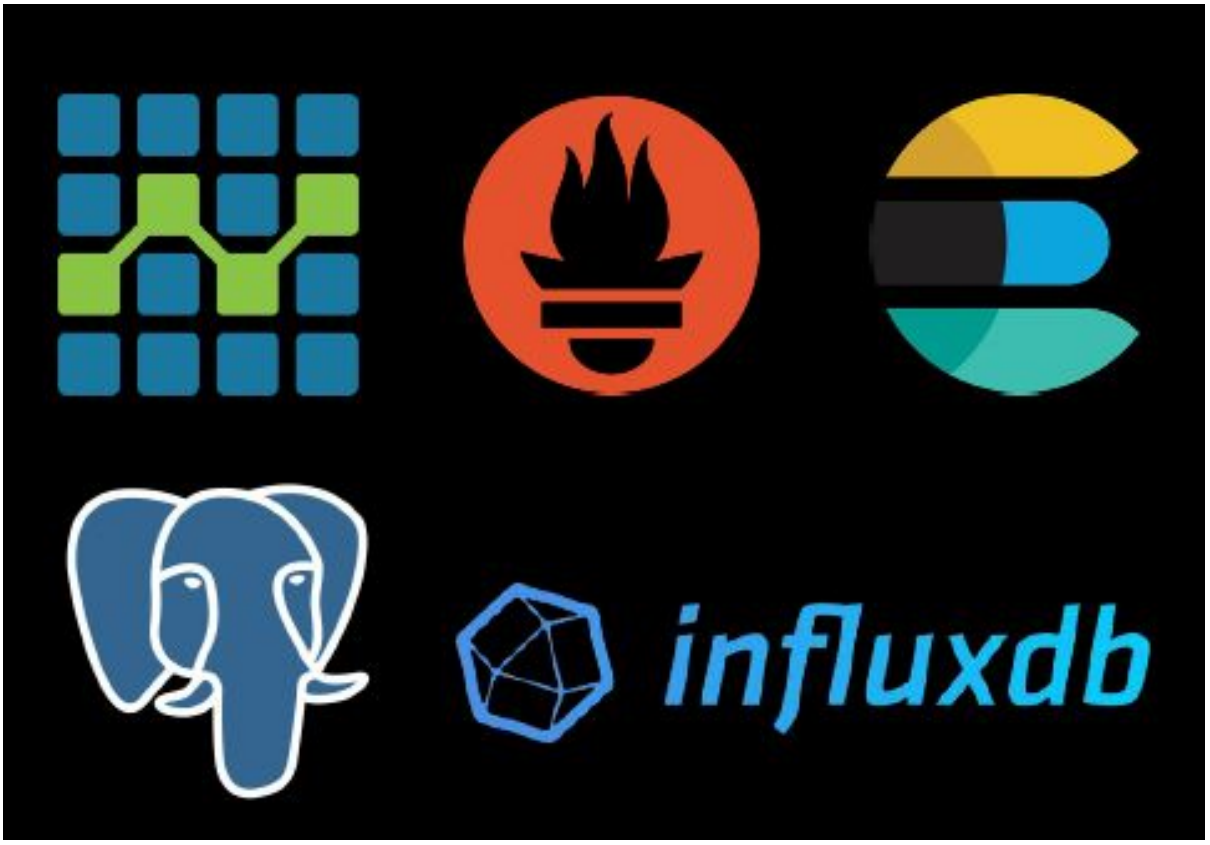
- A database for fast time series data ingestion and aggregation
- A system for querying and computing anomaly in real time
- Visualization and alerting capabilities

The solution

> *"InfluxDB has a very awesome stack, which gave us everything we needed when constructing the system."*

Why InfluxDB?

After considering Prometheus, Elasticsearch, OpenTSDB, Postgres, they chose InfluxDB as their time series database. Robinhood Engineer Allison Wang had first encountered InfluxDB as a student at Carnegie Mellon University (CMU), where InfluxDB creator Paul Dix gave a talk titled “InfluxDB Storage Engine Internals” discussing time series databases, the Time-Structured Merge Tree (the InfluxDB storage engine written from scratch), InfluxDB 2.0 and Flux. Two years following that talk, everything he mentioned during it has become part of the actual Robinhood system.



Below are the attributes that led Robinhood to choose InfluxDB:

- **Lightweight:** InfluxDB is very lightweight. Compared to OpenTSDB, a database built on top of Hbase (which requires the Hadoop ecosystem to be set up), InfluxDB doesn't require any third party or other system to set up in order to run in production. You can download InfluxDB from GitHub or run it in a Docker container, spin up an instance and see how it operates.

- **Schemaless:** InfluxDB is schemaless. Whereas Postgres has a very strong schema (where you need to define the schema before you can build a table, ingest the data, and update a table),

InfluxDB doesn't require having a schema beforehand and therefore reduces the overhead of doing a schema migration whenever the fields change.

- **Allows indexing via a specific field in the data:** In Robinhood's system, as time series data streams into the system, there are multiple fields (for example, a field called createdAt or updatedAt). These time series need to be indexed using different timestamps that are presented in the time series or in the original data point. They tried using Prometheus as a push-based system with exporters to fetch metrics from non-Prometheus systems. This didn't serve their purpose because when exporters detect or export the data, it gets indexed into Prometheus the time that it is ingested into the database. So it's very difficult to use one of the fields that are originally inside of the time series to index the data into the time series database. In contrast, InfluxDB allows doing that by changing the time column, making it easy to select which field to use in order to index the data into the time series database.

- **Fast data ingestion (write) and aggregation (read):** A real-time system requires ingestion of upstream data in real time, as well as querying data out of that database in real time. So they needed a database that can do the ingestion and aggregation simultaneously. Robinhood found that though Elasticsearch has a high read speed, it has a larger overhead than InfluxDB when you want to ingest and aggregate data simultaneously.

- **High availability (InfluxDB Enterprise):** Robinhood found the Enterprise version of InfluxDB (InfluxDB Enterprise) to be fault-tolerant. Using the Enterprise version was absolutely a requirement for them to avoid the risk of having one server crash and thereby cause the entire database to crash.

- **InfluxDB Stack (Kapacitor, Chronograf, Telegraf):** They use Kapacitor for alerting, Chronograf to explore the data and to plot the different time series on top of InfluxDB, and Telegraf to send time series data from different sources to InfluxDB.

- **Community:** InfluxDB has a large and active community that can answer their questions.

Building an end-to-end anomaly detection system

Next, Robinhood needed a mechanism to query data out of InfluxDB to compute the mean and standard deviation.

Real-time stream processing

For this purpose, they use a real-time stream processing system open-sourced by Robinhood and called Faust.

Faust, a Python version of Kafka Stream, sends queries and runs the anomaly detection algorithm for Robinhood. Faust is:

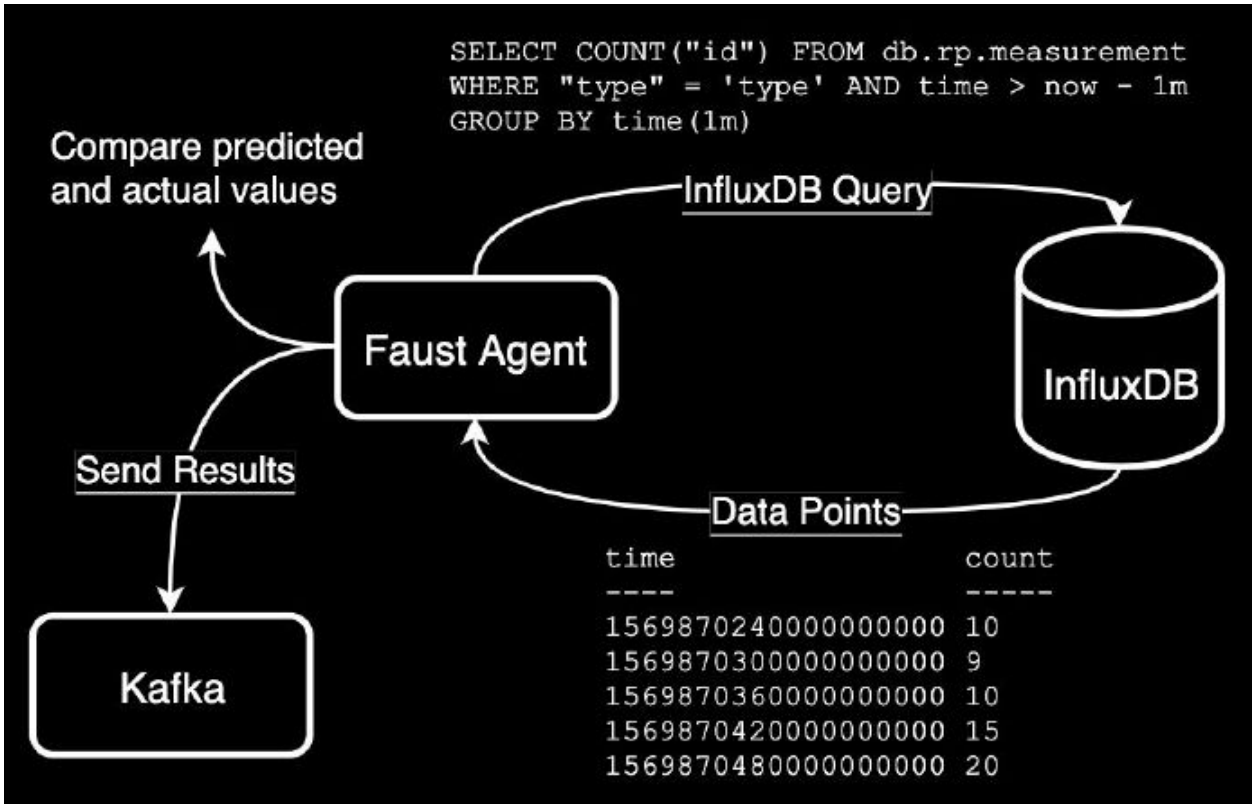
- Performant (utilizes the Python 3 Asyncio format)
- Fault-tolerant (RocksDB, a high-performance embedded database for key-value data)
- Scalable (has Kafka as its underlying structure)

```
import faust

class Greeting(faust.Record):
    from_name: str
    to_name: str

app = faust.App('hello-app', broker='kafka://localhost')
topic = app.topic('hello-topic', value_type=Greeting)

@app.agent(topic)
async def hello(greetings):
    async for greeting in greetings:
        print(f'Hello from {greeting.from_name} to {greeting.to_name}')

@app.timer(interval=1.0)
async def example_sender(app):
    await hello.send(
        value=Greeting(from_name='Faust', to_name='you'),
    )

if __name__ == '__main__':
    app.main()
```

The Faust function that they leverage to send the query to InfluxDB to retrieve data is called a timer (see @app.timer above). They set an interval that represents the frequency for the timer task to run, and schedule a lot of tasks. Whenever the scheduled interval is reached, the timer will execute this function. For example, `hello.send`, `influxdb.query` will query InfluxDB, construct the query, and render the time series data that they need to be able to detect anomalies.

Technical architecture

Below is a diagram of how Robinhood combines Faust with InfluxDB.



- Faust, through its timer function, continuously sends queries to InfluxDB, which returns data points.
- They can compute the mean and standard deviation, and compare the pre-computed threshold with the actual data points that they ingest.
- If either is outside the range, they can send that result back to Kafka, which can in turn be ingested back into InfluxDB and alerted with Kapacitor.

Data ingestion

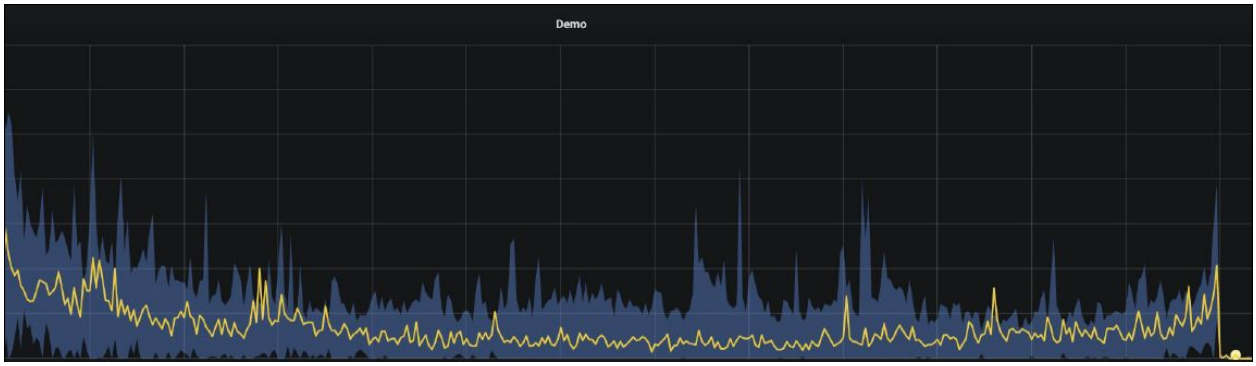
They have a large volume of time series in Kafka, a pub/sub stream processing engine. To move this time series data to InfluxDB, they ingest it through Logstash (a piece of infrastructure they already had set up) and Telegraf, which together can be thought of as the connector piece in their architecture. Telegraf has an input plugin where you can specify which Kafka topic to ingest or consume from; it can do certain transformations, output to InfluxDB, and specify which build you want to be indexed (for example, `updatedAt` or `createdAt` depending on the use case).

Visualization

In their production system, they use Grafana (to build the dashboard that combines the data that they query from multiple databases) and Chronograf (to quickly construct a time series).

The aggregated data (yellow) is bounded by upper and lower limits (blue):

- The yellow line is the actual data that they aggregated.
- The blue shades represent the upper and lower bounds of the time series.



Boundary visualization in Grafana: Example of infrastructure telemetry collected with InfluxDB by Robinhood

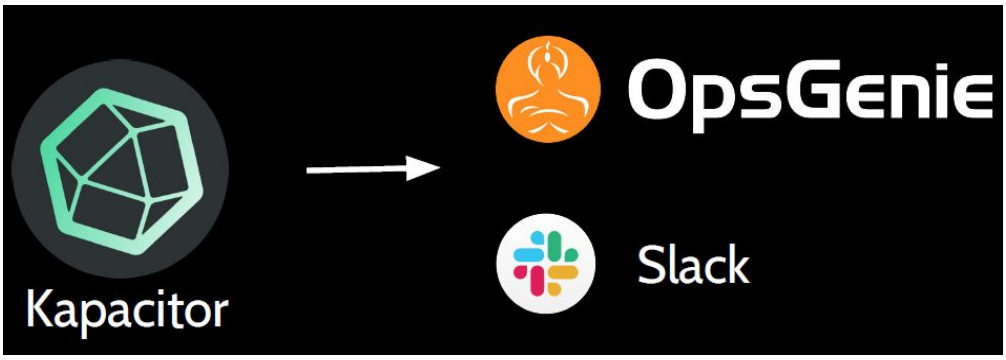
The graph above shows the aggregation by minute — each individual point over is the amount of data that they have over the past minute. On the very right-hand side, the yellow dot represents an anomaly that occured and that needs to be alerted on. The above graph is created through Grafana's boundary graph feature, which allows drawing the shape within a boundary. By shipping data to Kafka and including the boundary points in Grafana, they can visualize their metrics.

Alerting

After they identify the anomaly, and in order to alert on the downstreams, they leverage Kapacitor. Kapacitor allows:

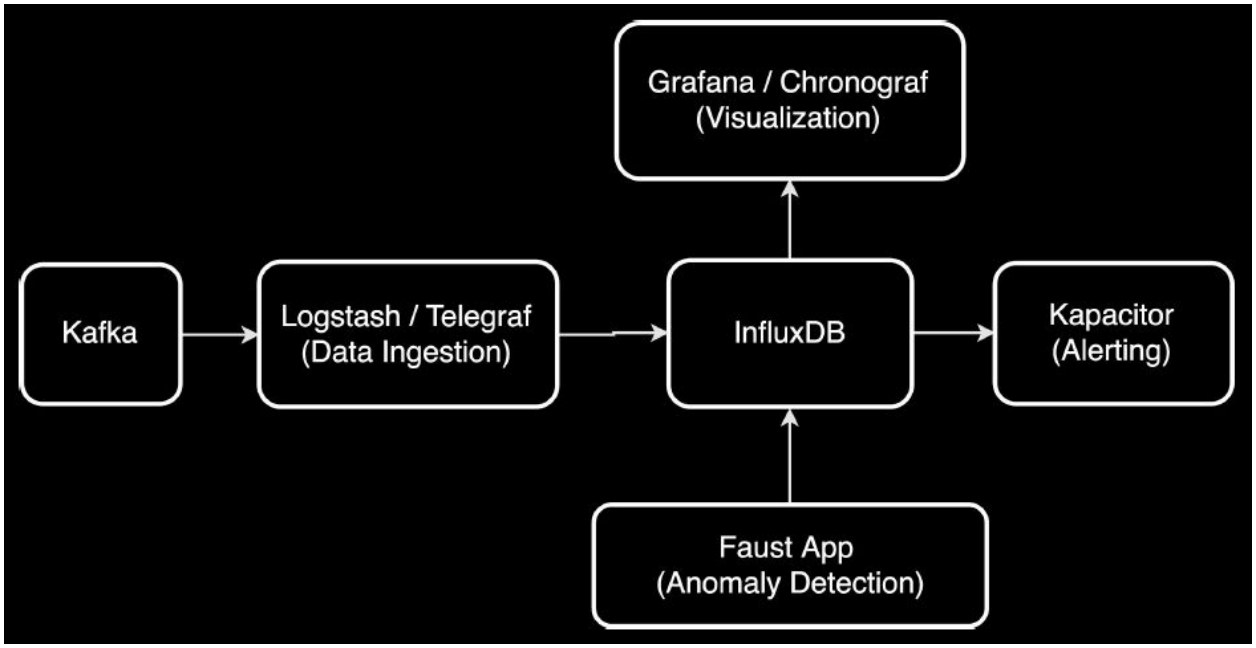
- Defining the rule based on TICKscript (an invocation chaining language used to define data processing pipelines)
- Alerting and configuring which downstream services should receive the alert.

They use Opsgenie for serious alerts and Slack for a status update.



High-level system architecture

The diagram below shows a high-level view of Robinhood's architecture:



- Time series data in Kafka gets ingested through Logstash into InfluxDB.
- Grafana is connected to InfluxDB to visualize the time series.
- Kapacitor is used as an alerting service.
- Faust computes and determines whether the incoming data point is abnormal.

The anomaly detection algorithm covered above is just one of many they can use to detect anomalies. Others include algorithms in Kapacitor's anomaly detection library as well as the Holt-Winters function included in InfluxDB. Since they use Faust, which is built on top of Python, they can easily send their time series data into machine learning libraries to detect anomalies. They can import Scikit-learn, Numpy, Pandas, PyTorch, and TensorFlow. They can also deploy deep learning algorithms using LSTM.

Results

> *"Without InfluxDB, we wouldn't have been able to build a system in such a short period of time and to have the system running in production so well."*

- **Lightweight** - they were able to build a system in a very short period
- **Extensible** - you can use different kinds of anomaly detection based on your business needs
- **Horizontally scalable** - each piece in their architecture is horizontally scalable, which means if the system load increases, you can add more servers, and it can handle the workload even if it involves a hundred times more time series.

This use case was first presented by Allison Wang at InfluxDays San Francisco 2019.

About InfluxData

InfluxData is the creator of InfluxDB, the open source time series database. Our technology is purpose-built to handle the massive volumes of time-stamped data produced by IoT devices, applications, networks, containers and computers. We are on a mission to help developers and organizations, such as Cisco, IBM, PayPal, and Tesla, store and analyze real-time data, empowering them to build transformative monitoring, analytics, and IoT applications quicker and to scale. InfluxData is headquartered in San Francisco with a workforce distributed throughout the U.S. and across Europe.

Learn more.

InfluxDB documentation, downloads & guides

Download InfluxDB
Get documentation

Additional case studies

Join the InfluxDB community

548 Market St
PMB 77953
San Francisco, California 94104
www.InfluxData.com
Twitter: @InfluxDB
Facebook: @InfluxDB