influxdata®

AN INFLUXDATA CASE STUDY

How Playtech is Using InfluxDB to Monitor Their Distributed Systems

Aleksandr Tavgen

Technical Architect, Playtech

playtech SOURCE OF SUCCESS

JULY 2018

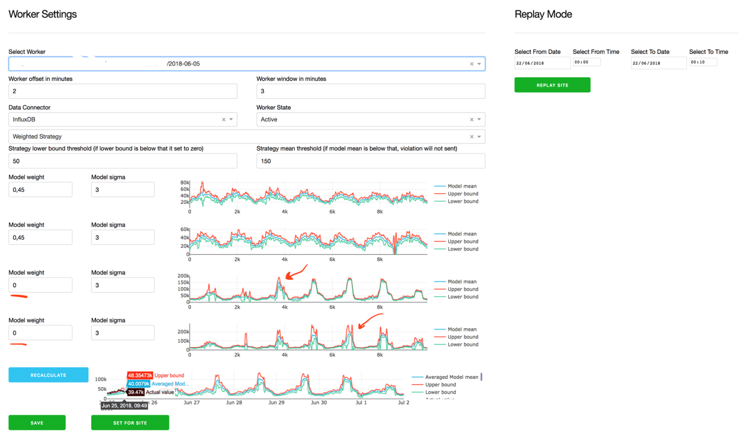
Company in brief

Playtech is the world's largest online gaming software supplier traded on the London Stock Exchange Main Market, offering cutting-edge, value added solutions to the industry's leading operators. Since Playtech's inception in 1999, its approach has been centred on the continual development of best-of-breed gaming products and content, and its success built upon strong partnerships with its licensees. Playtech has 5,000 employees, offices in 17 countries, 140 global licensees, and operates in 20 regulated jurisdictions. Playtech's product suite can be fully integrated into a complete cross-platform offering, enabling players to access online, broadcast, mobile and server- based gaming terminals through a single account. The cross- platform capability runs via a unified information management system (IMS), which provides operators with tools to maximize cross-selling opportunities, player loyalty and yield.

Case overview

Playtech needed to build a predictive monitoring and alerting system on time series data, for early detection of outages and security breaches. Their existing Hewlett Packard Service Health Analyzer solution was not delivering accurate and early predictions. They wanted a monitoring and anomaly detection solution designed for distributed systems. Monitoring distributed systems, as they found by analyzing successful and not-so-successful monitoring projects, is not a trivial task. It involves non-obvious obstacles and picking the right solution for various monitoring tasks.

Playtech decided to build Yet Another Alert System using InfluxDB, Alerta, and Grafana. The system combines monitoring and alerting for business indicators (BIs) and Key Performance Indicators (KPIs) using the regularity of economical processes. The system is taking them beyond monitoring to observability for a deeper understanding of system behavior and causes, relationships between system components, and needed action.



Worker Settings

Select Worker: /2018-06-05

Worker offset in minutes: 2

Worker window in minutes: 3

Data Connector: InfluxDB

Worker State: Active

Weighted Strategy

Strategy lower bound threshold (if lower bound is below that it set to zero): 50

Strategy mean threshold (if model mean is below that, violation will not sent): 150

Model weight: 0,45 | Model sigma: 3

Model weight: 0,45 | Model sigma: 3

Model weight: 0 | Model sigma: 3

Model weight: 0 | Model sigma: 3

Replay Mode

Select From Date: 22/06/2018

Select From Time: 00:00

Select To Date: 22/06/2018

Select To Time: 00:10

“Why InfluxDB? For Playtech, it was very important to have observability, to understand system behavior to predict possible outages and problems in the very early stages.”

Aleksandr Tavgen, *technical architect*

The business problem

Playtech was facing major issues with the early detection of outages and other problems in production systems. To understand system behavior, they realized that observability is necessary, which their existing Hewlett-Packard tool was not delivering. They had more than 50 sites with many brands located in different parts of the world; a lot of products, services, and configurations; and no location had the same mix of integrations and backends. Sometimes even minor errors or oversights in the configuration file, or some certificates problem, could disrupt a lot of services. That was the motivation for building a predictive alert system, which should be able to detect degradation before end-users do.

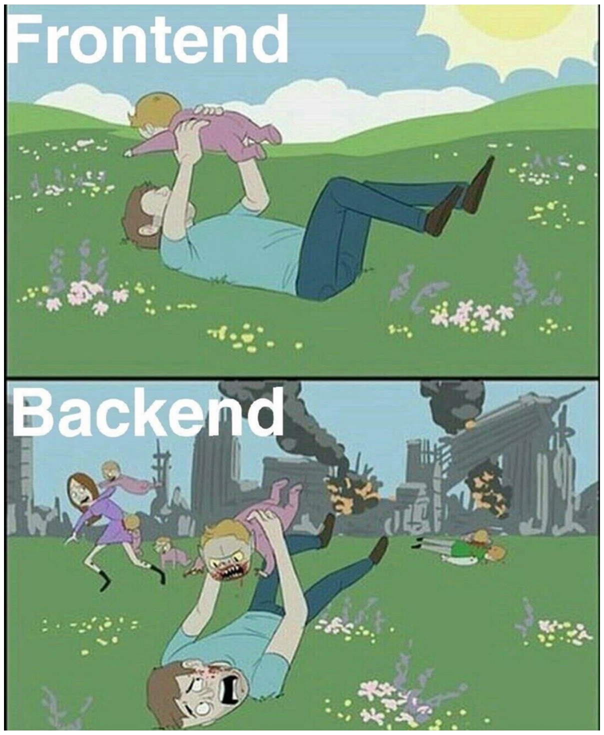
Monitoring distributed systems, however, abounds with pitfalls and choices given the huge number of possible commercial and open-source solutions for various monitoring tasks. Playtech considered the two different approaches to monitoring:

1. In low-level monitoring (monitoring of infrastructure such as CPU, disk, memory usage, networking), in most cases, applying some simple thresholds is enough. For example, in case of Java Virtual Machines, you can count garbage collection runs, allocation rates, etc. However, low-level monitoring cannot reflect business logic processes in an adequate way because there are many interdependencies between various component services. If you have some dependencies on third-party services, then low-level monitoring will not provide information about business processes.

2. In high-level monitoring, Key Performance Indicators (KPIs) are gathered — such as volumes of user sessions, number of payments, and the rate of change of transactions. These types of metrics can often identify issues that may be hard to detect with low-level monitoring. For example, the decrease in the rate of transactions can prompt a look at the number of users online, which can help determine that there is an issue with the login server (whereas just watching the login server might not have helped in cases where users stay logged in for long time periods). High-level monitoring makes it possible to indirectly monitor complex system behavior, especially in the case of distributed systems when you have different sites and data centers. Yet high-level monitoring is difficult because:

- Different problems lead to non-obvious system behaviour.
- A lot of different metrics have different correlations in time and space.
- Monitoring a complex application is a significant engineering endeavor in and of itself.
- There may be a mix of different measurements and metrics.

Playtech needed to achieve near 100% uptime to avoid losing customers but faced the challenge of monitoring distributed systems. Modern systems are moving towards distributed solutions, which have a Time Series Database, specific monitoring components, and visualization software like Grafana. For example, you can collect your business metrics from the database and perform analytics on that, but it means that your production database needs more and more resources because it's not optimized for working with times series data. What the customer sees is not what happens in the backend.



The nature of Playtech's operations — involving financial transactions in highly regulated markets — demanded high-level monitoring. High-level monitoring of their distributed system required analyzing financial data, which is time series data. The architecture of the system is service-oriented, the components have complex business logic, and there are different backend variants for different B2B clients (companies).

Low-level monitoring and tests do not cover a number of problems:

- Firstly, because of the high complexity of products and a huge number of settings, there are situations where incorrect settings lead to a degradation of financial indicators, or hidden bugs in logic are reflected in the overall functionality of the entire system.
- Secondly, there are specific third-party integrations for different countries, and the problems that arise with partners were beginning to flow to Playtech. Problems of this kind are not caught by low-level monitoring; to solve them, Playtech needed high-level monitoring to monitor KPIs, compare them with the statistics on the system, and look for correlations.

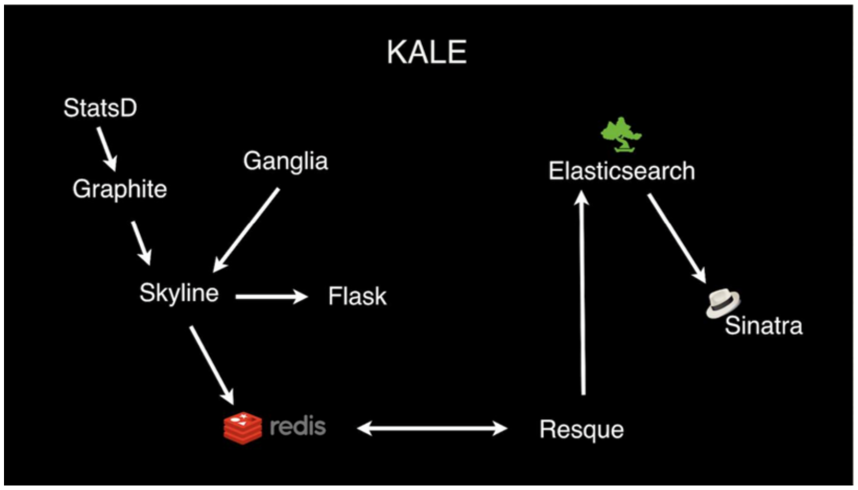
The technical problem

Playtech previously implemented the Hewlett Packard Service Health Analyzer (SHA) solution, which was unsatisfactory. Based on its marketing prospectus, it is a system that trains itself and provides early detection of problems (it can take data streams from multiple sources and apply advanced, predictive algorithms in order to alert to and diagnose potential problems before they occur). Yet Playtech found SHA to be a "black box" that cannot be configured, faced long support response times and unsatisfactory problem resolution. Most importantly, they found this existing "one-fit-for-all" solution to have:

- Low efficiency and side effects
- False Positives and missed incidents
- False Negatives
- Poor operability

The granularity of SHA was 15 minutes — this meant that even in case of problems, the minimum reaction time is 15 minutes, while Playtech needed to be able to react in real time. They decided to build a monitoring system from scratch to gain observability into their distributed system. Before doing so, they analyzed successful (and unsuccessful) cases.

1 - The first use case examined was **Etsy's Kale System**. Etsy, an online retailer for handmade goods with headquarters in New York, collected about 250000 different metrics from all their servers. They tried to find anomalies in measurements and generate alerts on them, and they used complex math algorithms. Their system was built using different stacks and frameworks (the picture below shows four different stacks plus two frameworks). As a result, highly qualified engineers with experience in working with different languages and frameworks were required for maintaining and developing such a system as well as for fixing any bugs that were found.



Kale's approach to monitoring was also problematic. It searches for anomalies or outliers. But if you have hundreds of thousands of metrics, every heartbeat of the system will inevitably result in a lot of outliers due to statistical deviations. Etsy engineers made a futile attempt to combat the false positives detected, and finally the project was closed. Andrew Clark, a data scientist from Etsy, concluded:

- Firstly, anomaly detection is more just outlier detection. There will always be outliers in any real production data, in the case even of any normal system behavior. Therefore, not all outliers should be flagged as anomalous incidents at any level.
- Secondly, a one size-fits-all approach will probably not fit anything at all.
- Finally, alerts should only be sent out when anomalies are detected in business and user metrics, and anomalies in other metrics should be used for root cause analysis.

2 - The second case examined, which is more successful than Etsy Kale, is **Google SRE teams' BorgMon**. According to the system's authors:

- Google has trended towards simpler and faster monitoring systems, with better tools for post hoc analysis (this makes it possible for the system to detect very simple, specific, and severe anomalies very quickly)
- They avoid "magic" systems that automatically learn thresholds or automatically look for correlations.
- The rules for problem-solving systems were kept as simple as possible. They gave a very fast response time to simple, localized, and serious anomalies.

Based on these principles, Google built a successful BorgMon system.

Solution

> *"We've chosen InfluxDB. Why? Because InfluxDB is widely adopted by the community. It is fast, it is reliable, and comes with great support. We have almost no problems with InfluxDB even with large, parallel tasks running and fetching huge chunks of data."*

Why InfluxDB?

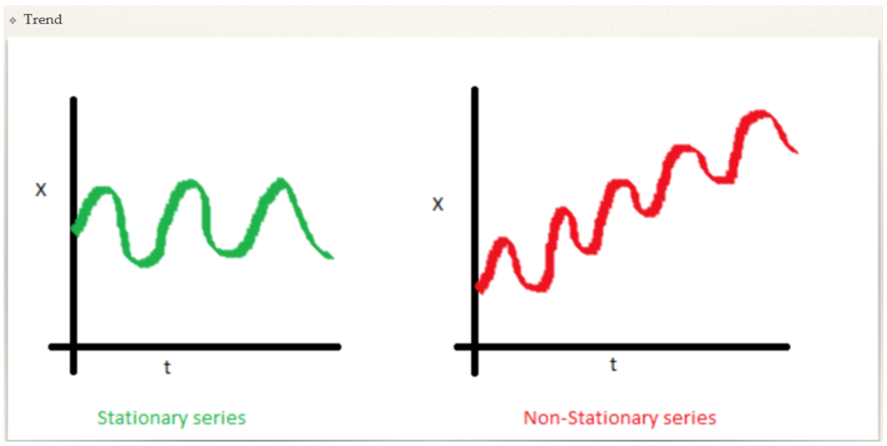
Playtech chose InfluxDB because it is purpose-built for time series data, provides fast time to value, ease of use, and support. They approached their distributed monitoring solution from their perspective of what constitutes time series data: a series of data points indexed (or listed or graphed) in time order. They used the regular structure of economic processes, which simplifies analysis.

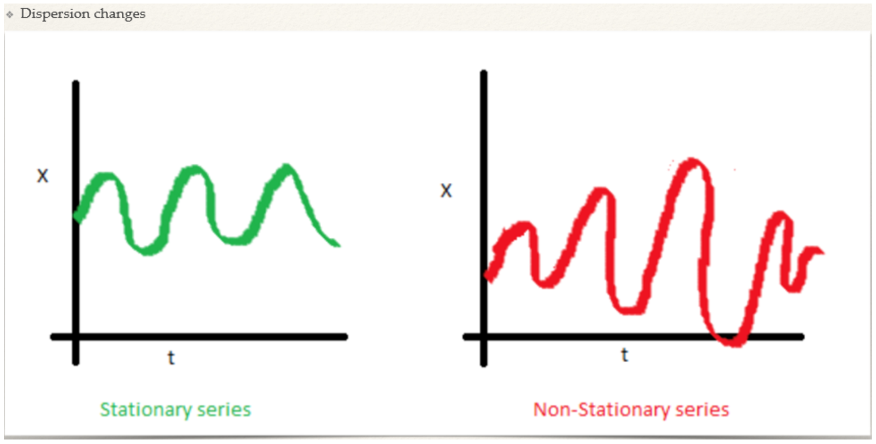
Playtech classify time series data into:

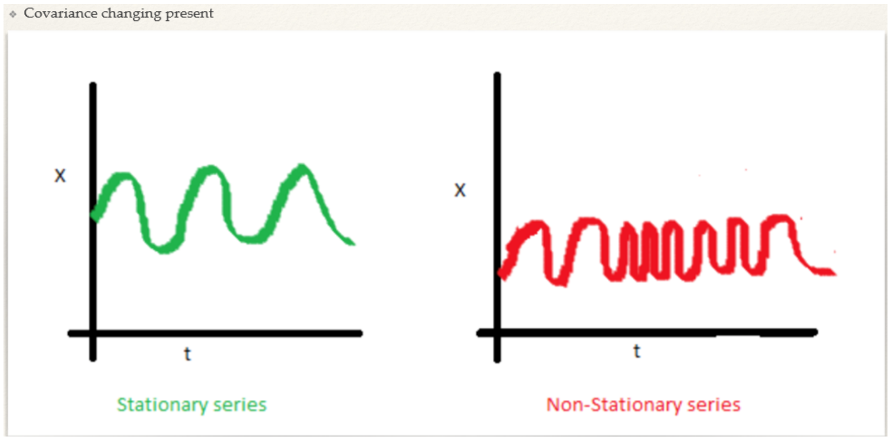
- **Stationary** (a stochastic process, whose characteristics do not change) which can be described with the two parameters: mean and standard deviation.
- **Non-stationary** (such data has a trend line, a dispersion change, or some covariance changing present)

At InfluxData, we refer to these as Metrics (regular time series) and Events (irregular time series).

Stationary vs. Non-Stationary Time Series



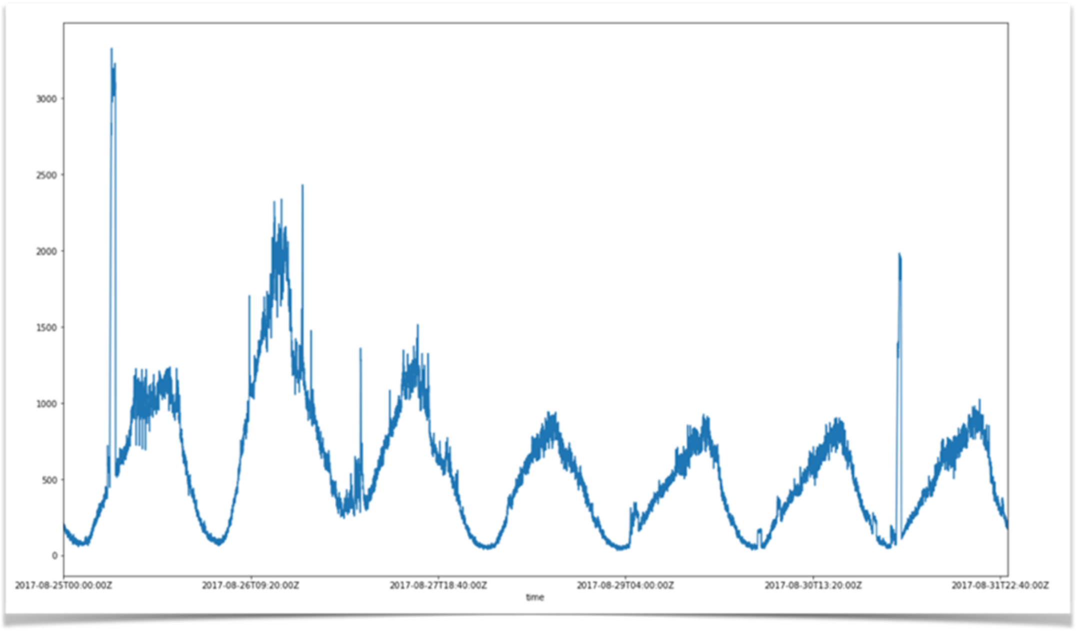




A weekly series of real data on the number of transactions shows pronounced regularity (two peaks are performance tests). There is daily regularity, plus an increase in activity on Friday and at the weekend, after which the activity begins to subside until the next weekend.

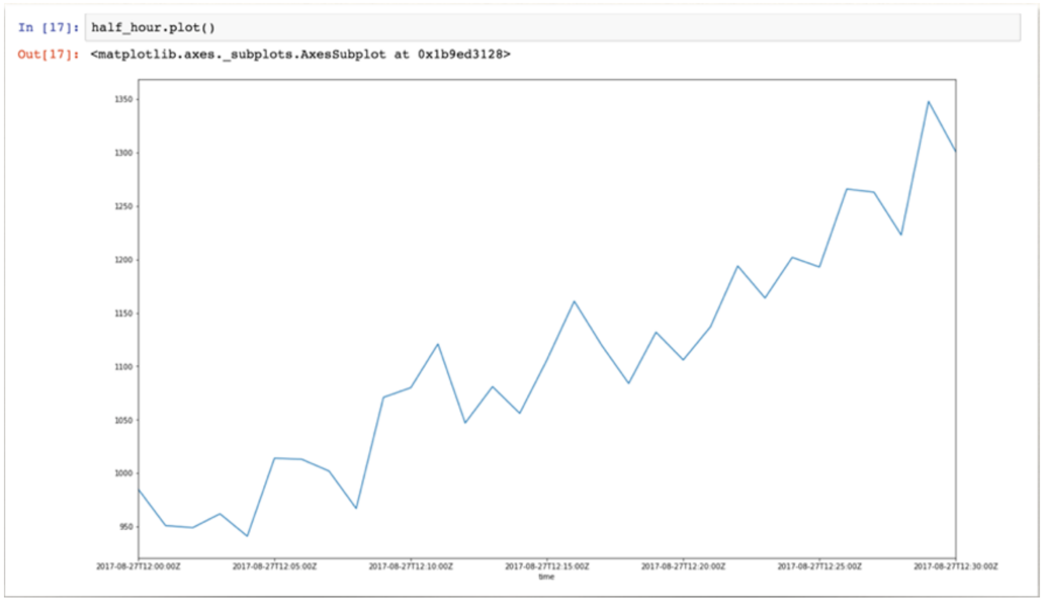
Each measurement contains a signal component and a measurement / noise error (**Point = Sig + err**). There are many factors that influence both the processes themselves and the collected metrics. Playtech have a model that describes the signal. If the model is subtracted from the measurement, the better the model will grasp the signal, the more the result of subtraction will tend to stationarity or white noise.

Playtech’s Sample Data: One Week of One Measurement



The half-hour data plot clearly shows a trend line. It means that this data is non-stationary.

Half an Hour: Trend Line Showing Non-Stationary Data

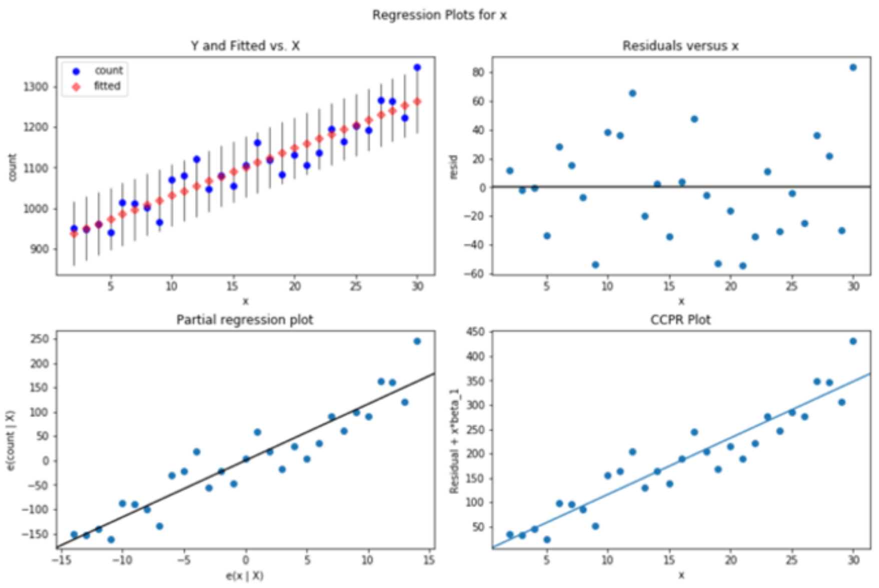


```
In [17]: half_hour.plot()
Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x1b9ed3128>
```

Playtech uses linear regression to interpolate this data by line. They consider it one of the clearer approaches to modeling data and proving that their model captures the signal.

Linear Regression: Interpolate Data by Line

```
In [22]: fig = plt.figure(figsize=(12,8))
         fig=sm.graphics.plot_regress_exog(ols_test,'x',fig=fig)
```

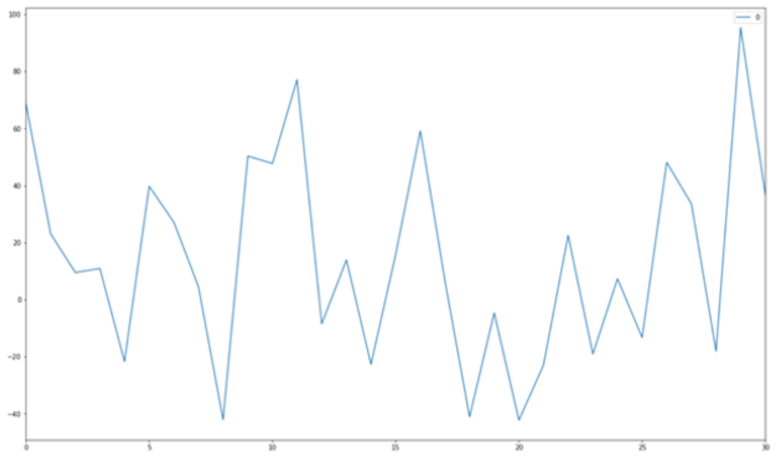


If Playtech subtracts this trend line from their initial data, then it looks like stationary time series. It means that they captured the signal and that they can prove that this time series is stationary.

After Subtraction: Trend Line Removed

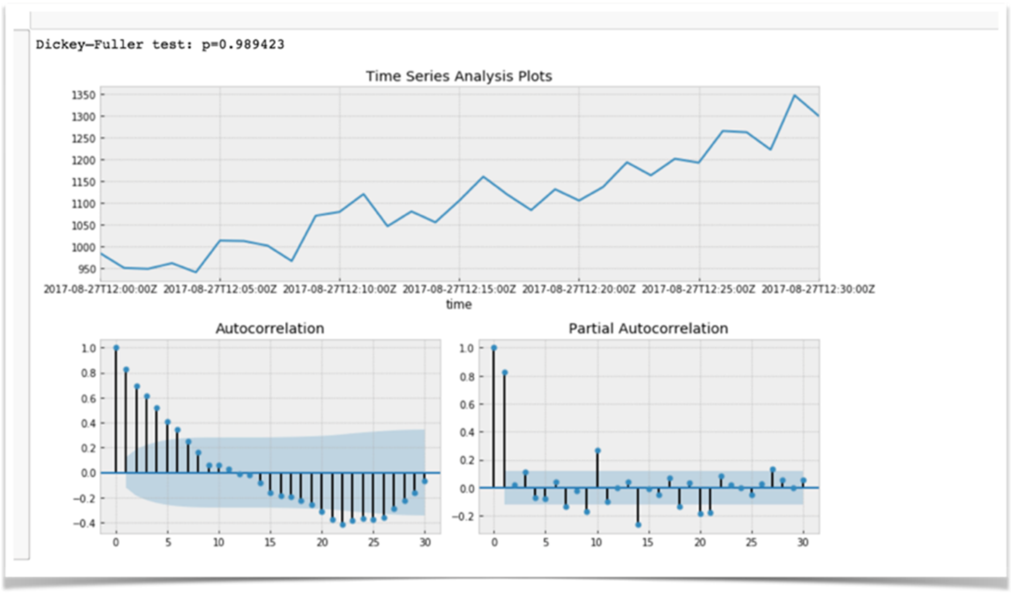
```
In [25]: rows_list = []
         step = 0
         for index, row in df_ols.iterrows():
             rows_list.append(row['count'] - (level + slope * step))
             step = step + 1
         df_norm = pd.DataFrame(rows_list)
         df_norm.plot()

Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x1913dc160>
```



Playtech uses the Dickey-Fuller method to test time series data stationarity.

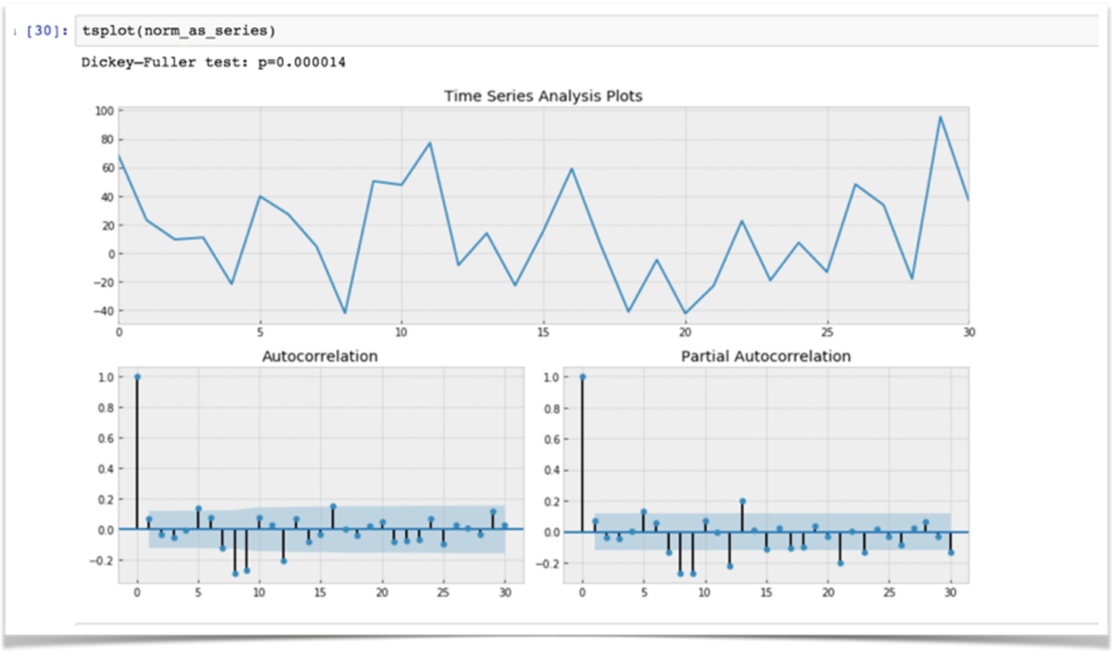
The Dickey-Fuller Test: Testing Time Series Data on Stationarity



```
Dickey-Fuller test: p=0.989423
```

Upon testing their initial data, they can see that autocorrelation is very high and the test result is near 1, which means that this test rejects their stationary hypothesis. Upon running this test against their results, the test outcome is near 0, which means that their hypothesis about stationarity is acceptable.

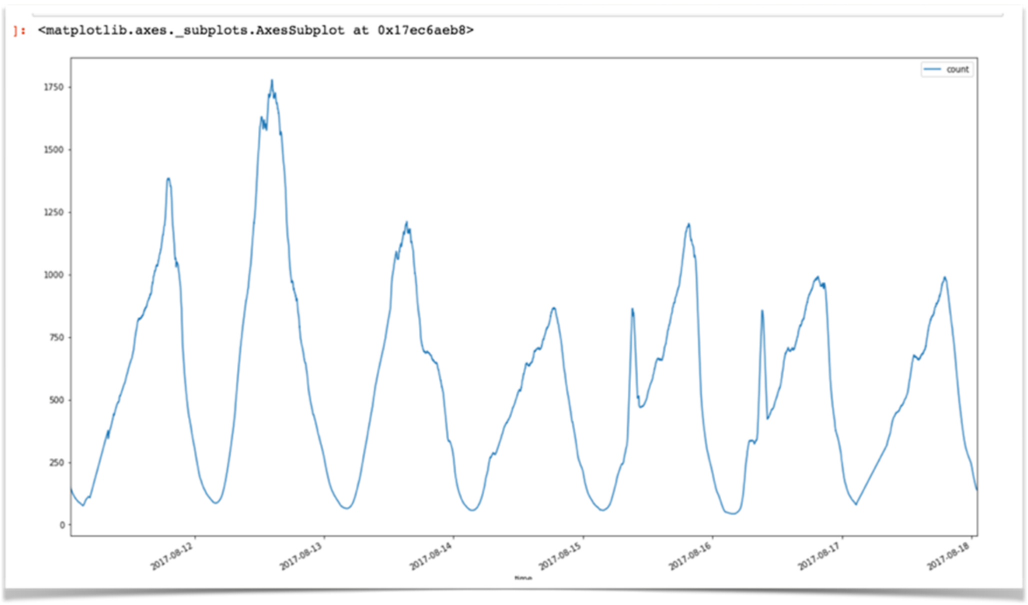
The Dickey-Fuller Test: Final Result



```
[30]: tsplot(norm_as_series)
Dickey-Fuller test: p=0.000014
```

For the monitoring system, Playtech chose modeling using moving statistics for the average and variance. The moving average is essentially a low-pass filter that smooths out the noise and leaves the main trend lines. Moving statistics made it easy to detect outliers, and they can build, from that, their model. Here is their data on the number of transactions in the next week, after the passage of the moving average with a window of 60 minutes (peaks removed):

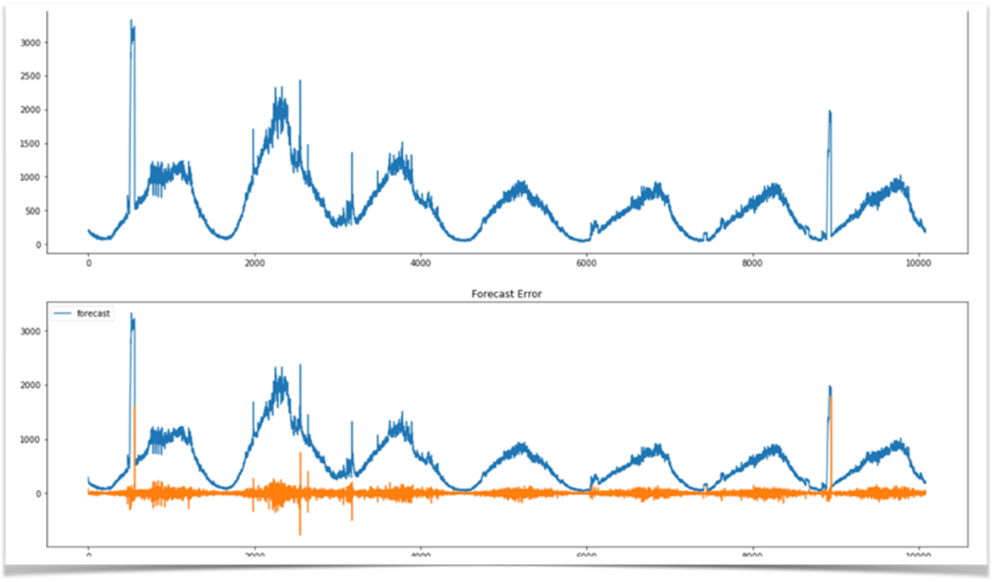
Moving Statistics



```
]: <matplotlib.axes._subplots.AxesSubplot at 0x17ec6aeb8>
```

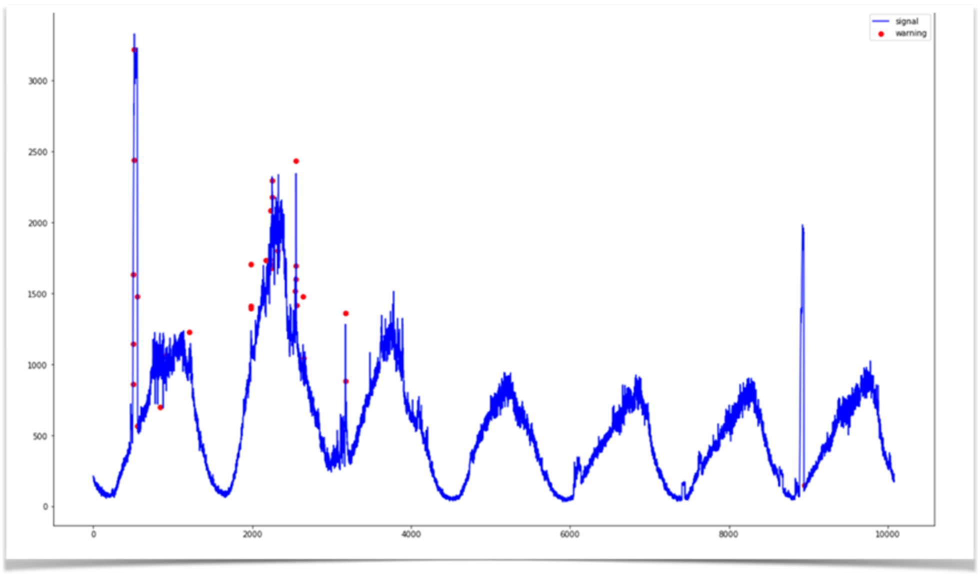
Playtech tested other algorithms, among them Twitter S-ESD (which tries to detect peaks and anomalies on almost any of the time series). That did not fit their needs because it was more oriented to low-level monitoring.

Twitter S-ESD



Playtech realized that simple is better than complex, and that it's best to drop everything you don't need. When they made the initial data analysis on their actual metrics, they started with a simple architecture, which is based on InfluxData technology. For Playtech, it was very important to have observability, to understand system behavior to predict possible outages and problems in the very early stages.

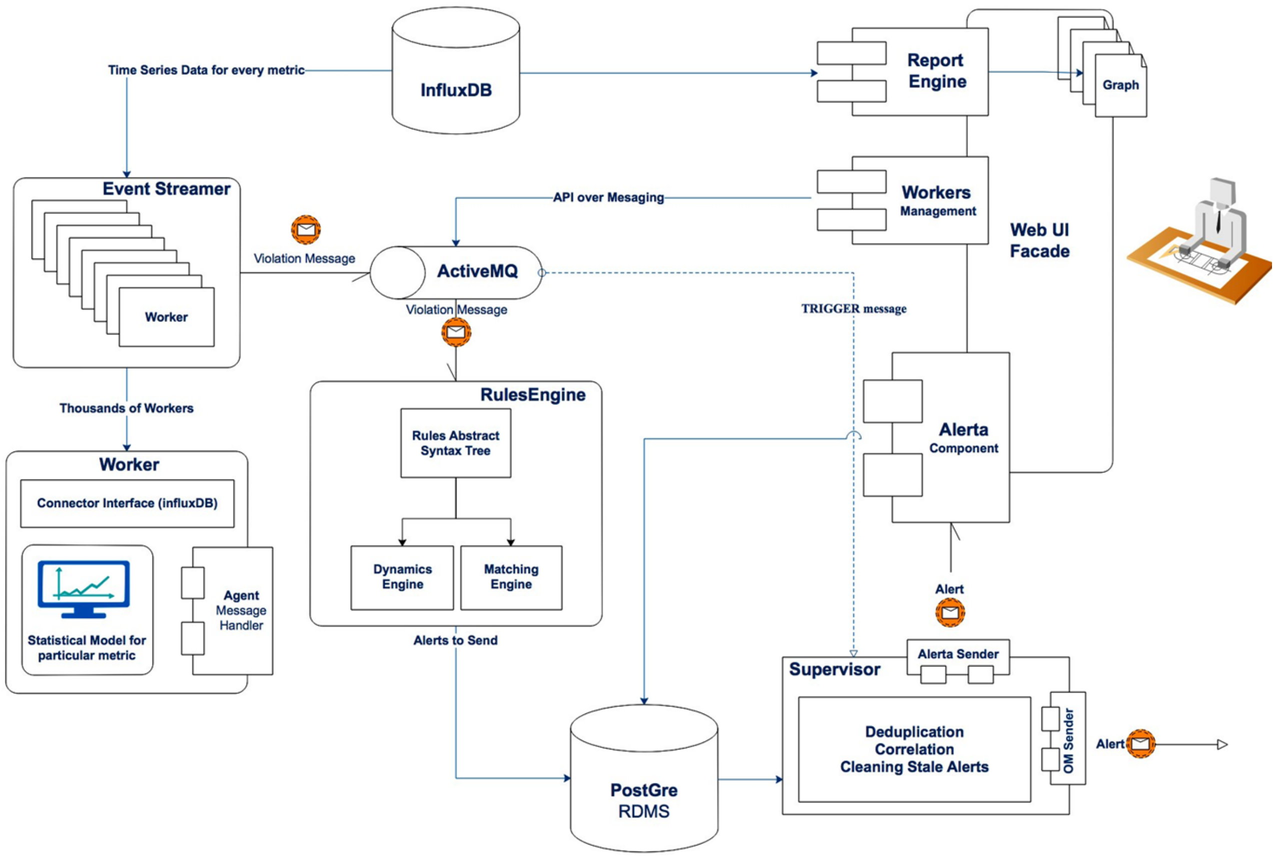
Anomalies



Technical architecture

> *"We can store those type of events in InfluxDB, and it means that we can work with this data without increasing the load on our production database. Our system also has the ability to replay all incidents and tune models."*

InfluxDB for Central Storage to Enhance Observability



Playtech's monitoring system is distributed by design. It is built as a set of loosely coupled components or microservices that rotate in their processes on their Python VMs. This design is a natural way of defining the boundaries and responsibilities of each component, which makes it easier to extend the system and add features independently and on the fly without affecting the main logic or risking breaking it. It also makes it possible to perform distributed deployment or implement a scalable solution with some configuration changes.

System design overview

- Each component does a limited number of specific things. Between the components, a sufficiently clear boundary is drawn.
- Distributed deployment is convenient if you need to place the agent locally closer to the site it is monitoring — or you can aggregate a large number of different systems together.
- Python was chosen as the main stack because there is an excellent ecosystem for data analysis, fast libraries, and excellent support for web solutions.

- InfluxDB serves as the central storage engine for Key Performance Indicators (such as logins, bets, signups, online players) to increase observability and provide understanding of what is happening on their different sites. Data is fetched and stored in a regular way using Telegraf and InfluxDB.
- This distributed design is also event-driven: it is natural for a general logical event (emitter / rules engine).
- Communication is built on the basis of messages since the entire system must be asynchronous. They chose ActiveMQ for implementing the message queue because it's a stable and well-known solution. They could easily switch to, for example, RabbitMQ since all components communicate with the standard STOMP protocol.

System functionalities

To collect the right data, the system has the following functionalities:

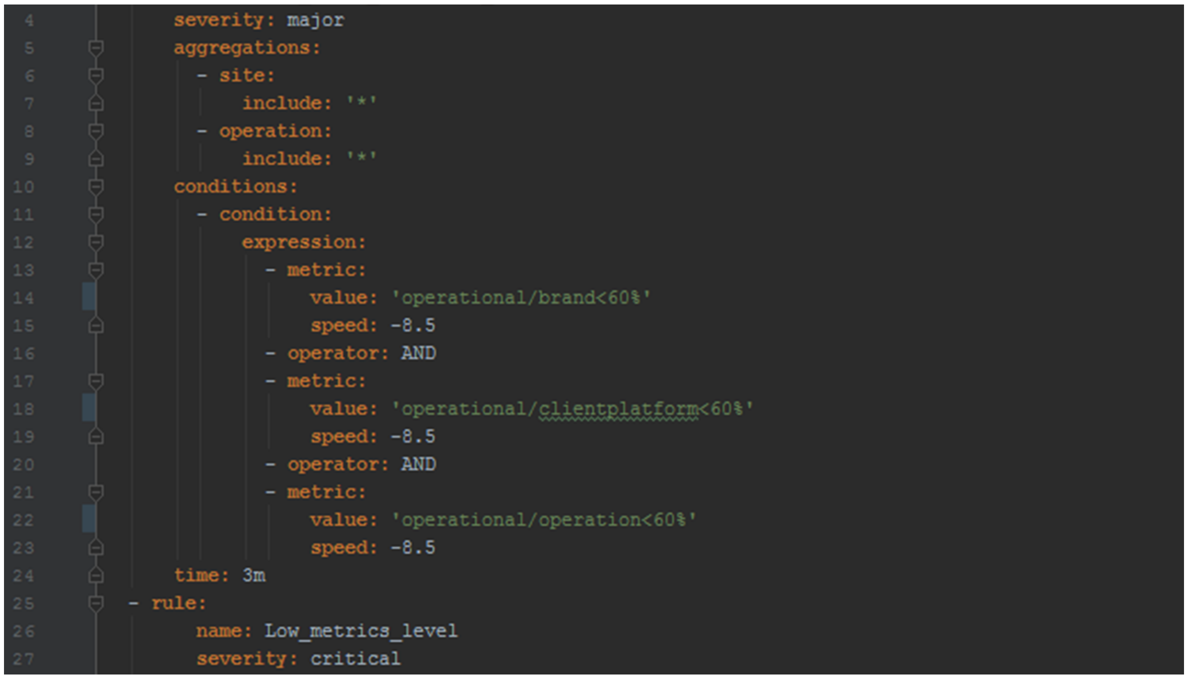
Event streamer (the information emission layer) is a component that stores statistical models, selects data periodically, and finds outliers. Playtech collect, for every metric, an ensemble of models. This component fetches data from InfluxDB and tests it against statistical models. If a violation is found, then it sends a message with all information about this violation. (Humans similarly solve problems in this manner: If they notice something strange happening in one part of the system, they check the other parts, propose some hypothesis, and try to disprove it.)

Worker is the main working unit, which stores one model of one metric together with meta-information. It consists of the date of the connector and the handler to which the data is transferred. The Handler tests them on a statistical model, and if it detects violations, it passes them to the agent that sends the event to the queue. Workers are independent of each other — each cycle is performed through a thread pool. Since most of the time is spent on I / O operations, Global Interpreter Lock Python does not significantly affect the result. The number of threads is set in the config; on the current configuration, the optimal number was 8 threads.

RulesEngine (the information consumption layer) is where every message with violation information is collected, matched and analyzed against different rules. Correlations can be set up between different metrics as well as different thresholds and different speeds of degrading. They have logical operators (precision on every type of their rules for alerting). They have a parameter of the speed (how fast their metrics are degrading). For example, for serious outages, in which there is an almost immediate drop in some critical metrics, the speed is higher than in some slowly degrading metrics.

Each message is sent to the RulesEngine. Playtech had to recall the Abstract Syntax Trees and define a simple language for describing the rules. Rules are described in YAML format and specified in the form of regular expressions for metric names or simply metric prefixes.

RulesEngine



```
        severity: major
        aggregations:
          - site:
              include: '*'
          - operation:
              include: '*'
        conditions:
          - condition:
              expression:
                - metric:
                    value: 'operational/brand<60%'
                    speed: -8.5
                - operator: AND
                - metric:
                    value: 'operational/clientplatform<60%'
                    speed: -8.5
                - operator: AND
                - metric:
                    value: 'operational/operation<60%'
                    speed: -8.5
        time: 3m
    - rule:
        name: Low_metrics_level
        severity: critical
```

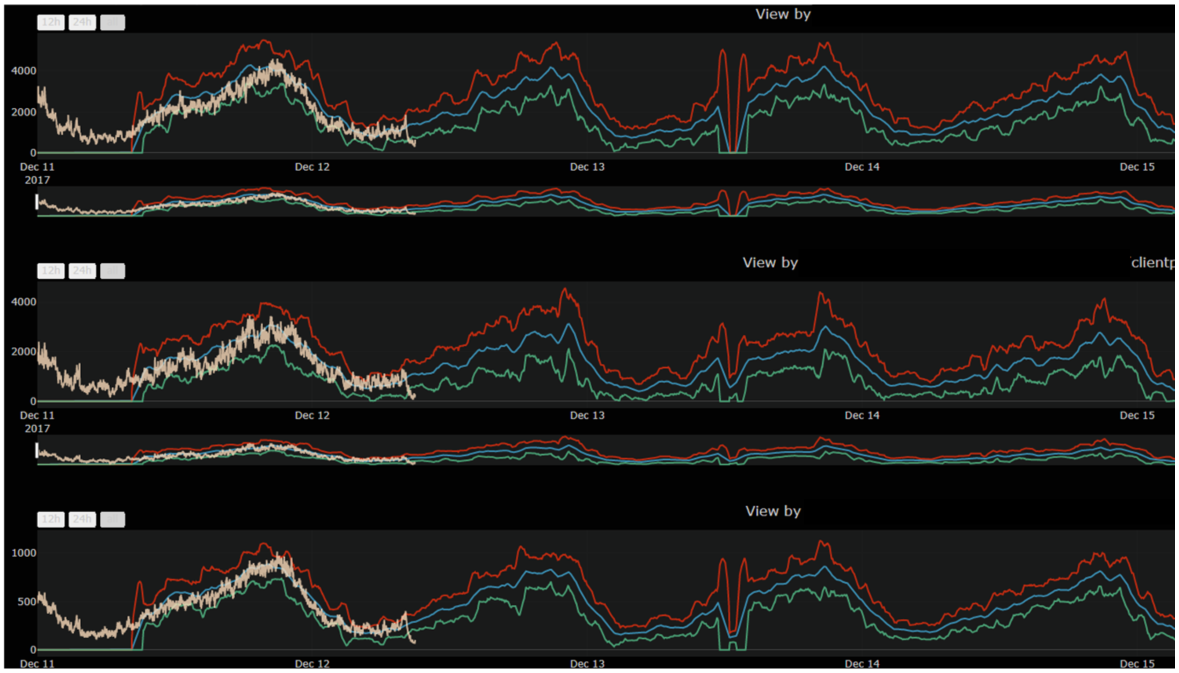
When the component starts, the rules are read and the syntax tree is built. Upon receipt of each message, all events from this site are checked for one tick according to the specified rules, and if the rule is triggered, an alert is generated. There can be several rules.

By considering the dynamics of incidents developing in time, Playtech can also take into account the rate of fall (severity level) and the change in velocity (severity forecast).

Speed is an angular coefficient or discrete derivative that is counted for each violation. The same goes for accelerations, a second-order discrete derivative. These values can be specified in the rules. The cumulative derivatives of the first and second orders can be taken into account in overall assessment of the incident.

Report Engine generates a report on-demand. After an alert is emitted and rules are matched, a URL with the alert description is created, and the report is generated showing which metrics came down and how it looks like. Grafana is used for visualization.

Reports

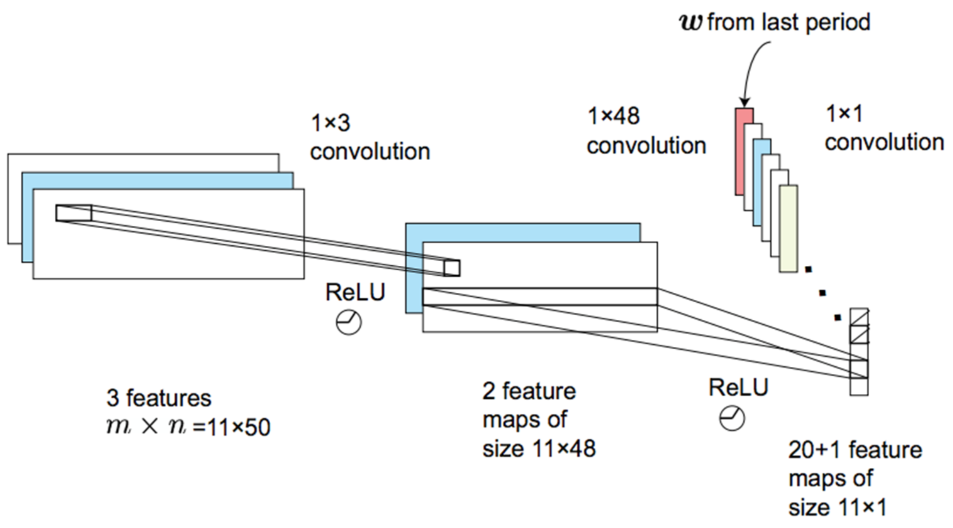


Though Playtech had in place a system of accounting and deduplication, they decided on an alternative solution: Alerta. A tool used to consolidate and de-duplicate alerts from multiple sources for quick 'at-a-glance' visualization, Alerta was very easy to integrate, including the user interface that they built into the WebUI system. Alerta has multiple integrations and can write Alerts back to InfluxDB for further visualization in Grafana.

What’s next for Playtech?

Playtech is constantly expanding their system and adding new functionality in order to build business intelligence in near real time. Now they are working on statistics in the field to fine-tune the rules. There are further plans for system development, such as looking at metrics through Convolution Networks (a third-party project).

Deep Neural Networks Engine



w from last period
1×3 convolution
1×48 convolution
1×1 convolution
ReLU
ReLU
3 features $m \times n$ =11×50
2 feature maps of size 11×48
20+1 feature maps of size 11×1

By seeing the full picture of their time series data, they have had good results training and testing their neural network, but more months of measurements and results analysis are needed. Playtech intend to release this monitoring system in open source in the next few months.

Results

> *“As I saw, different teams try to reinvent the wheel. They put a lot of effort into that, which could be turned to solving their problems but not with how they fetch, store information and so on. Now it's very easy. With Docker, you can just take images and run Grafana, InfluxDB, and Telegraf within minutes.”*

Playtech are able to observe, in an automated way, BI’s and KPI’s stored in InfluxDB and visualized in Grafana, to gain meaningful information for all parties who needed it. They now catch most of the incidents a few minutes earlier (on average 12-15 minutes) than their previous HP alerting system. In addition, the new system detects and signals problems that slip through other monitoring systems.

Their monitoring system's benefits include:

- Very low footprint: The system uses Vanilla Python distributed (Vanilla allows you to build concurrent software in Python). Everything runs on one virtual machine, which has 8 GB of RAM and two virtual cores.
- Ability to replay historical data and tune the system: Playtech started with one model but realized they needed to have an ensemble of models. They collect four different models, and can easily turn on, off, or increase the weight of some models for their system.

The system took less than 9 months of Research and Development to build and implement.

Early detection of malicious attacks and outages has proved critical to Playtech's business as they are now able to:

- Detect, earlier than previously, attempted security attacks to guess passwords or usernames (attacks characterized by a very clear pattern on data, such as amount of false login attempts).
- Detect service-start problems earlier because they see activities which are critical — and have special rules for that — begin degrading (some services didn't start well previously when they made a deployment and had some mistakes in their configuration). Playtech can quickly catch numbers of incidents within a week, enabling very quick response.

Beyond monitoring to observability

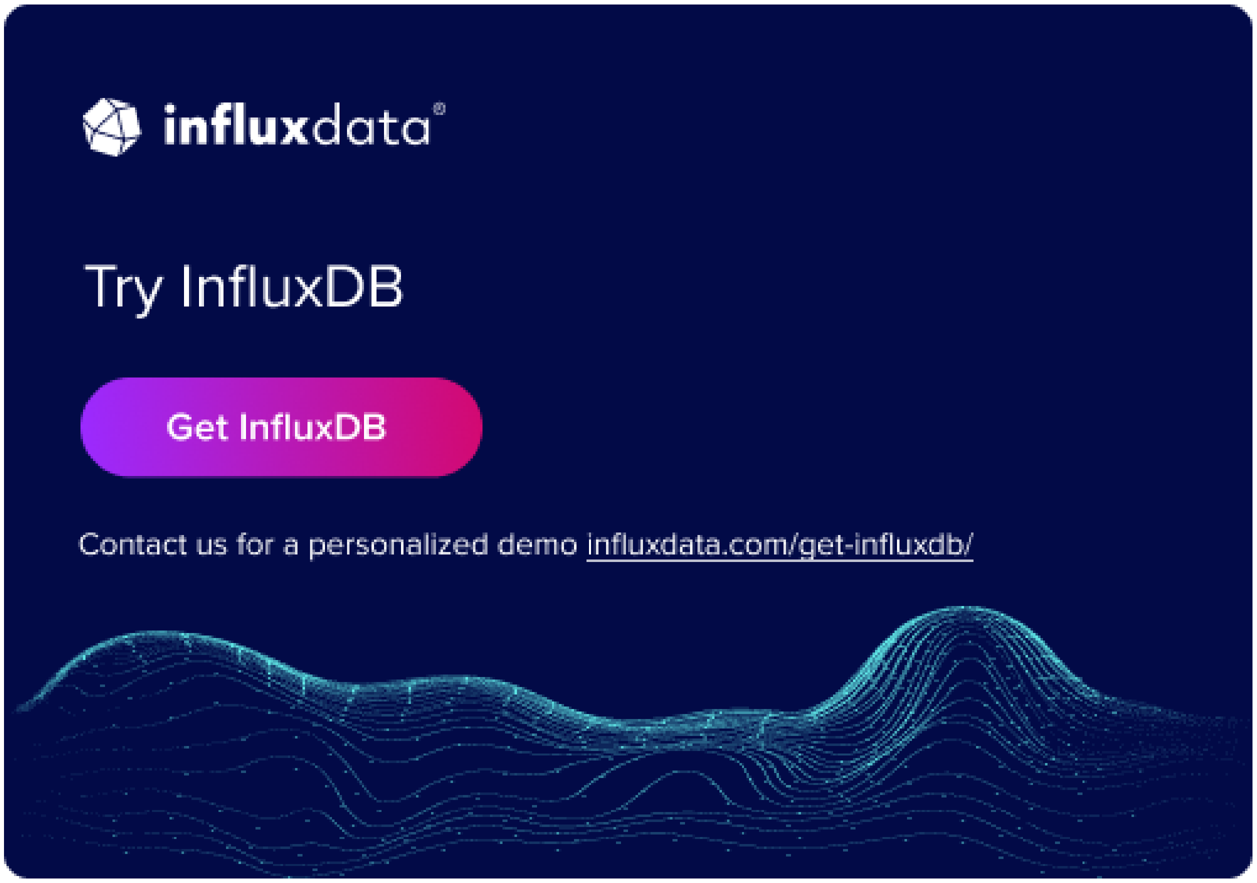
Playtech have concluded that low-level monitoring alone will not address a large set of specific issues as it is insufficient to explain what is happening with the business, and they have also concluded that observability:

- Is wider than monitoring and leads to a deeper understanding of how the system behaves.
- Means that you can't prove your hypothesis, but you can disprove it. You can get more information, which gives you insights about what can be proved and solved.
- Requires building an analytical system which creates correlation between different parts of different systems.

Therefore, a combination of different types of monitoring (low-level, high-level, mid-level) and instrumentation of your code will lead to observability of your systems. With observability, you will be able to understand your systems and safeguard the performance of your content and sources of revenue.

About InfluxData

InfluxData is the creator of InfluxDB, the leading time series platform. We empower developers and organizations, such as Cisco, IBM, Lego, Siemens, and Tesla, to build transformative IoT, analytics and monitoring applications. Our technology is purpose-built to handle the massive volumes of time-stamped data produced by sensors, applications and computer infrastructure. Easy to start and scale, InfluxDB gives developers time to focus on the features and functionalities that give their apps a competitive edge. InfluxData is headquartered in San Francisco, with a workforce distributed throughout the U.S. and across Europe. For more information, visit influxdata.com and follow us @InfluxDB.


influxdata®
Try InfluxDB
Get InfluxDB
Contact us for a personalized demo influxdata.com/get-influxdb/

548 Market St. PMB 77953, San Francisco, CA 94104