The Walking Dead' secured prison — zombie free

Secure a microservices architecture

Maxime Thomas
Oct 28, 2017 · 12 min read

This article follows the talk I gave during the Paris Microservices Meetup, the 24th of October at Deezer venue.

Working on distributed architecture for such a while, I've noticed that companies, architects and developers too, are a bit confused when we talk about security and how it should be applied in the enterprise, specifically when the company is a small company and when the platform is relying on a microservices architecture.

Mainly, security is considered as unnecessary and an overkill, rightly because experience showed that security systems are very intrusive in people all days life and brings lot of complexity for everybody. But if you don't have security systems, you are vulnerable.

Let's dig in the way to secure specific distributed architectures.

Security main approach will be explained, so it will be easier to understand how, on the specific case of distributed architectures, components can be secured.

Security concern

Security is always a difficult subject to approach either by the lack of experience you may have on it ; either by the fact you should know when the level of security is right for what you have to secure.

Security is major concern for individual people and more and more for companies. As an architect, I consider that working with technical educated people (engineers, experts) and tools (systems, frameworks, IDE) should prevent the company to have issues like this.

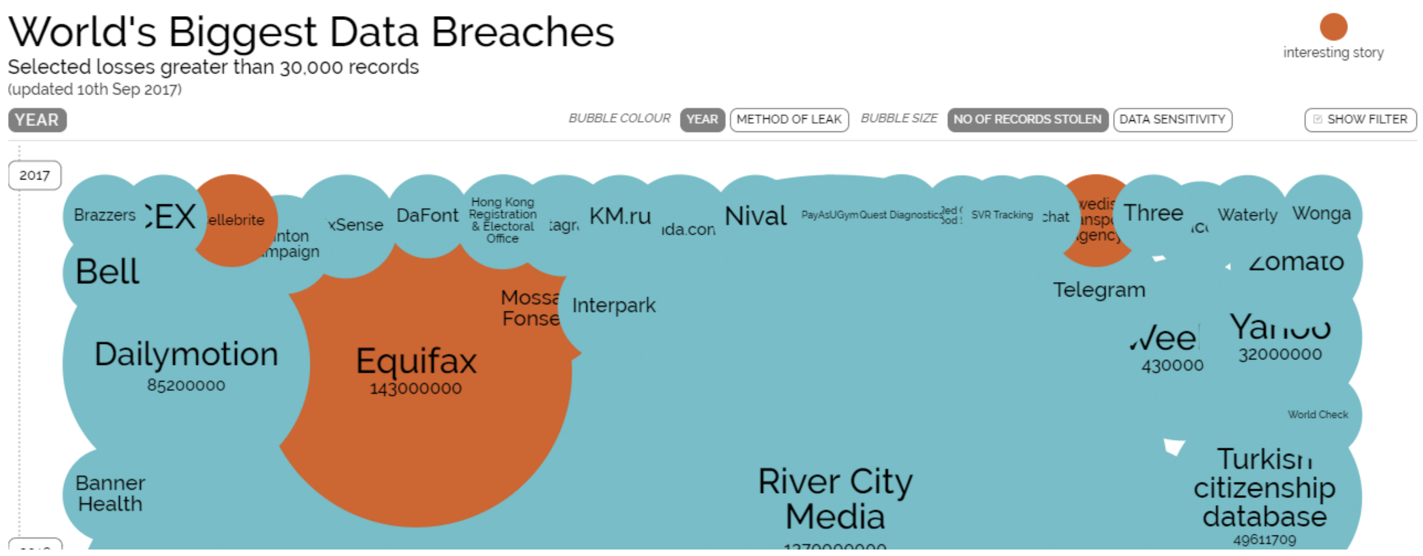
However, when your company grows, it's quite difficult to avoid, a certain infatuation from different categories of people to try to hack your platform. The fabulous website Information Is Beautiful updates quite often its visualization of World's Biggest Data Breach.

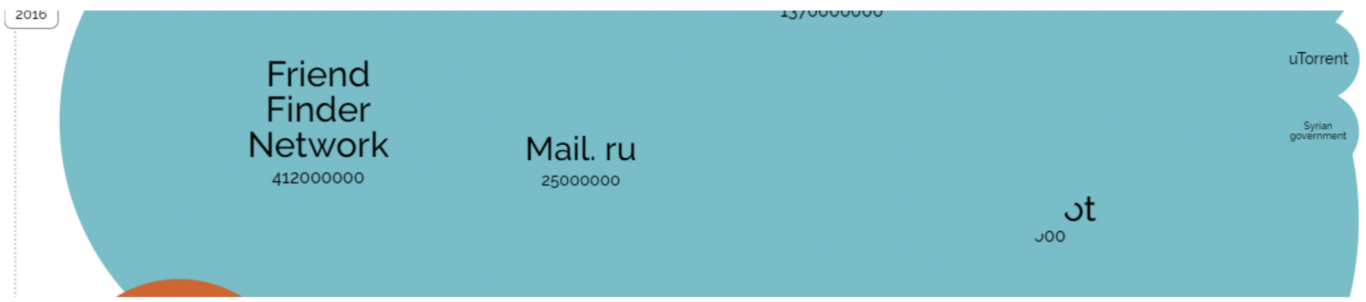
World's Biggest Data Breaches & Hacks - Information is Beautiful

Data visualization of the world biggest data breaches, leaks and hacks. Constantly updated....

www.informationisbeautiful.net

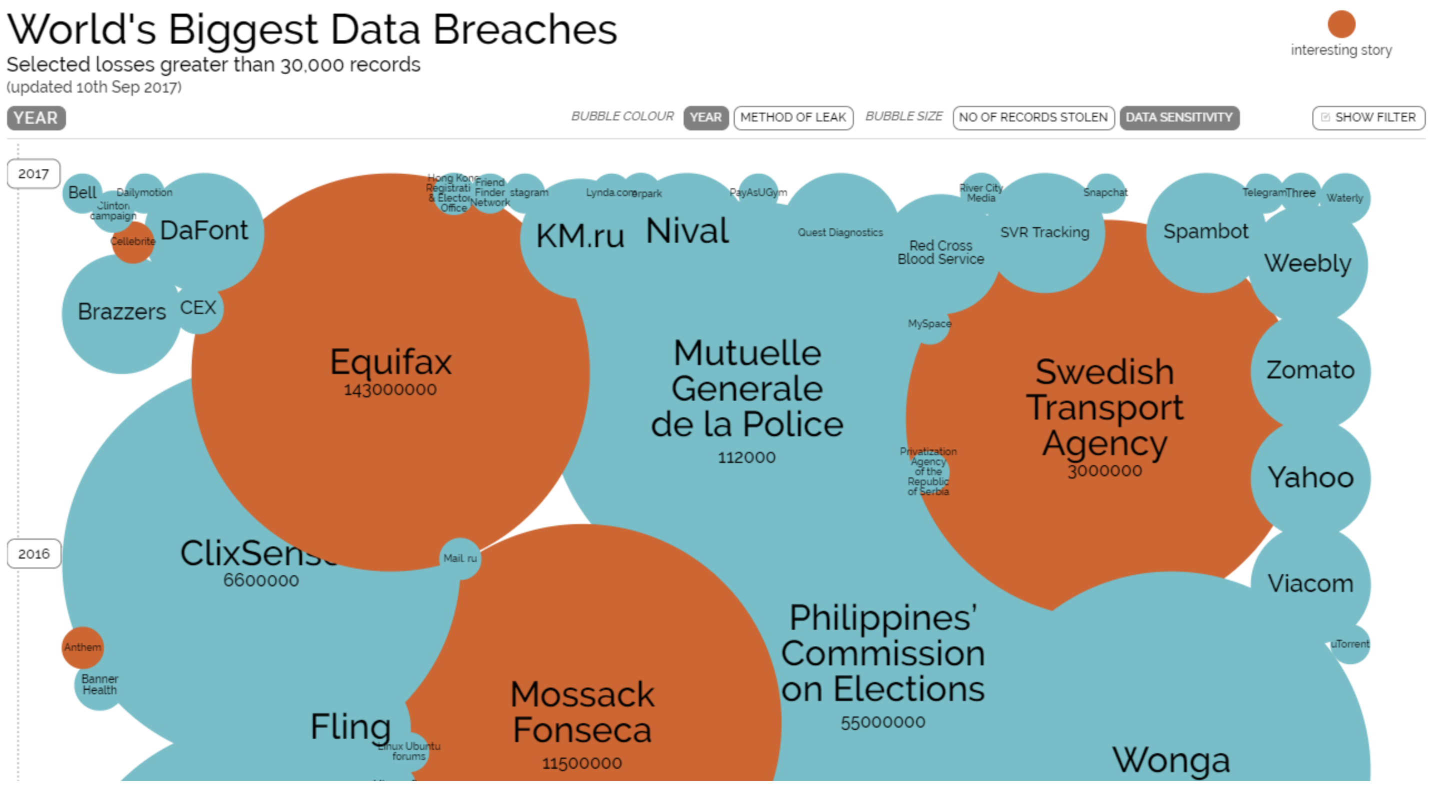
What is remarkable in this, is that you can see that great companies loose a large amount of data due too bad or weak security systems.





Information Is Beautiful — World's Biggest Data Breaches — Number of records stolen

They also have a more interesting visualization that shows us the sensitivity of data. Ok, you have lost data, but if it's interesting like banking account numbers of personal health data, it's not valuable, you cannot sell it.



Information Is Beautiful — World's Biggest Data Breaches — Data Sensitivity

As we can see, it's uplifting — Swedish Transport Agency, French Mutuelle Generale de la Police, Philippines' Commission on Elections… Pew.

Security must be taken seriously and in the same way we evaluate it for a car. The day I will send my car in a store front I will be happy to have chosen the car with the airbag included.

In IT systems, everything is virtual, so it's not really a problem when you have security issues, it's IT stuff. Actually no. Moreover, you cannot fix it by buying a single product that will magically seals all the lost and forgotten breaches in your platform.

“Security is not a product, it’s a process”
Bruce Schneier
Applied Cryptography Author

Bruce Schneier — Security is not a product, it’s a process

Bruce Schneier is so right. Imagine you have a bike and you buy a bike lock. It’s not because you have bought the lock that your bike is secured. You have to take it when you ride, you have to take the key, you have to put the lock on when you stop somewhere… You have to follow a process. For IT systems, it is the same.

Security is a process

In company, security is a main concern that impact either physical security (building, belongings, people) or virtual, such as IT.

If it’s a process, it means that is not an **one shot thing**, you have to set it up and **maintain it**. Generally, companies hire a Chief Information Security Officer — CISO — to take care of that. Maintenance can be done *a priori* by setting up standards for example, and *a posteriori* by auditing code and infrastructure (physical and virtual).

As distributed architecture are mostly based on DDD, each people is owner of domains, so **everybody is concerned** by security. For example, if you have a microservice called *Message*, the guy in charge of this microservice should be in charge of the security of its own microservice.

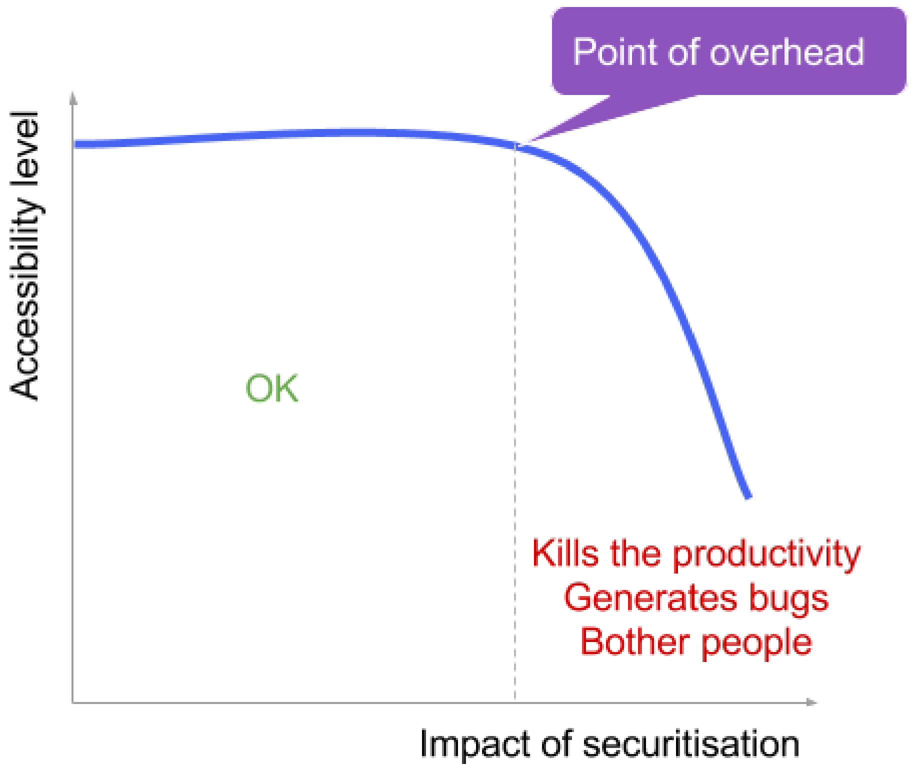
Finally, the modality of security should be **discussed** with all the stakeholders to find the point of overhead. It's just because when you are talking about security, expectations may vary from the CEO and the developer.

Pillars of IT Security

To avoid misunderstanding about IT security is about, let's get back to the fundamentals, its pillars.

- **Confidentiality** : no one can access what belongs to you
- **Integrity** : information cannot be changed without your consent
- **Availability** : you can access data when you need it

There is a direct correlation between confidentiality and integrity, and obviously, a direct opposition of availability between the others pillars.



Impact of securitisation against accessibility level

As we can see, the security may affect the level of accessibility of the system in bad ways. It can kills the productivity of your platform, meaning no one will be able to use it; it will be complex and will generate bugs; or worse, it will bother people.

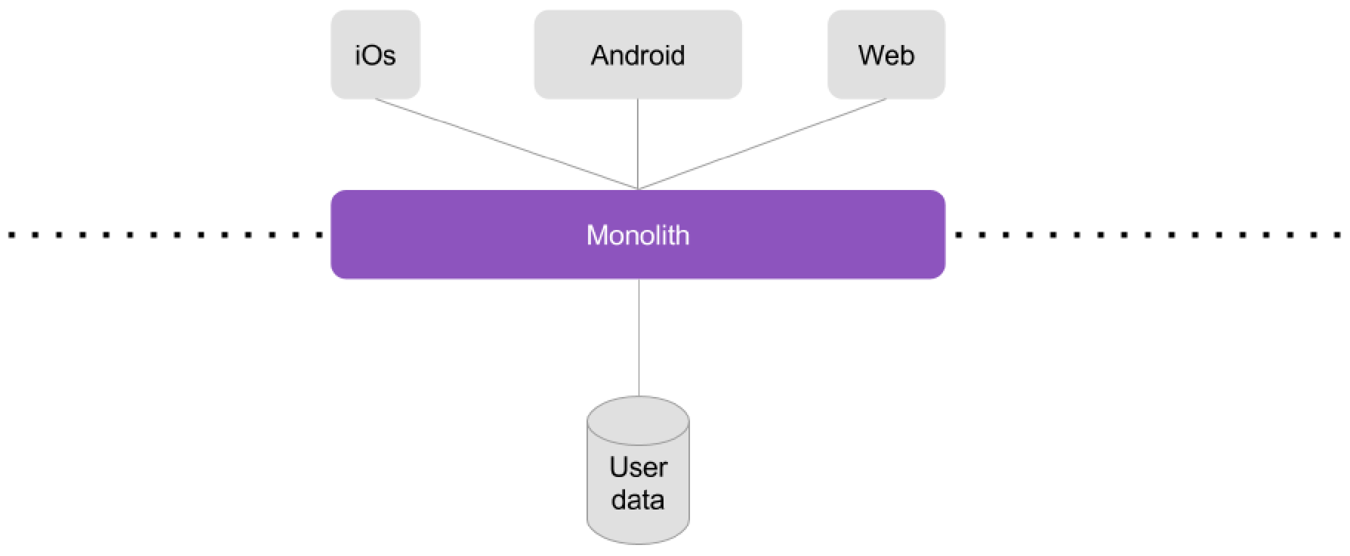
We all have in mind security systems such as *credit card number* or *3DS* that are very painful on the user side. If you are not following the nominal way of doing things, you will *loose* accessibility to your money.

In the other hand, if you reduce your requirements on security, you will maintain a high accessibility level but you will expose your company. From my window, I guess that it is what happened to all those companies that has been hacked, they reduce security to better match budgets and people availability. They pay it now.

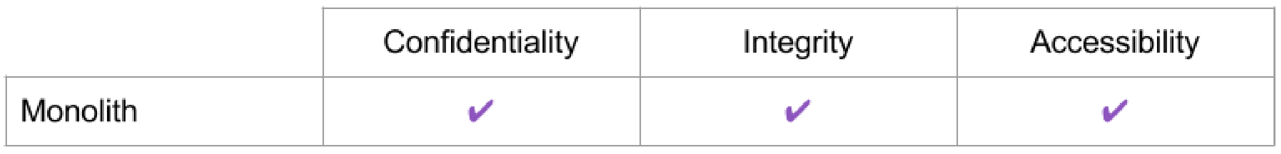
. . .

Monolith vs Microservices

We've seen pillars of IT security and that the point of overhead is important in the solution selection. This part will describe the main differences between monolithic architectures and microservices architectures regarding security pillars.



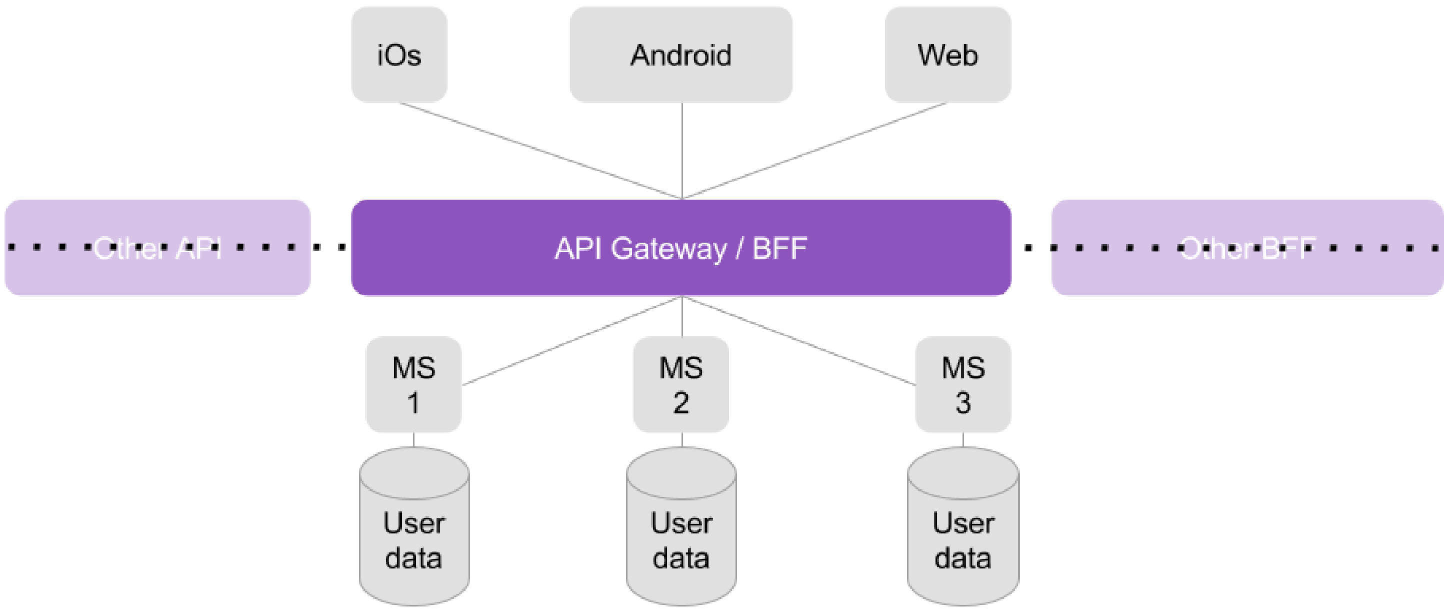
Monolithic architectures are set in one block (sic) and data is generally stored in one storage system.



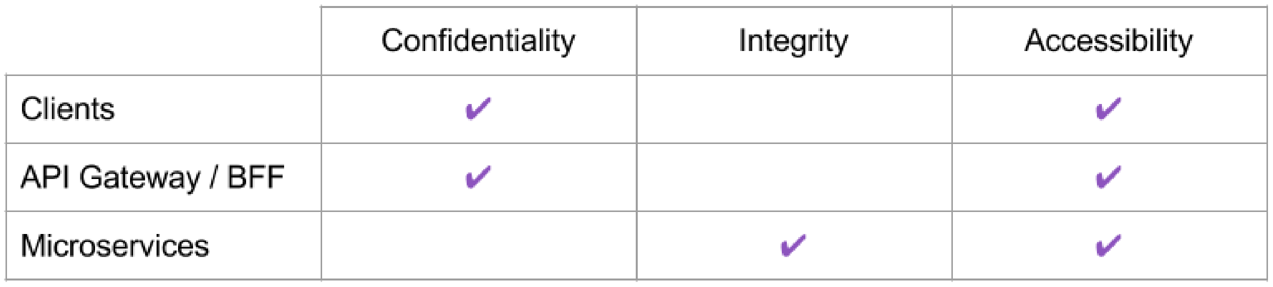
	Confidentiality	Integrity	Accessibility
Monolith	✓	✓	✓

If we break down pillars, well, we can see that the *monolith* has to handle each pillars.

Now, if we are with a distributed architecture, such as :



Where we can considerate 3 layers : *clients* (android, ios, web, …), *API Gateways* (or Backend For Frontend, BFF), and *microservices* and their data.



	Confidentiality	Integrity	Accessibility
Clients	✓		✓
API Gateway / BFF	✓		✓
Microservices		✓	✓

There's a **responsibility segregation** through the different layers. Mechanically, exposed layer and front layers are taking care of **confidentiality** meanwhile microservices are handling **integrity**. **Accessibility** is also shared between all the layers as if they're not accessible, the platform is not working.

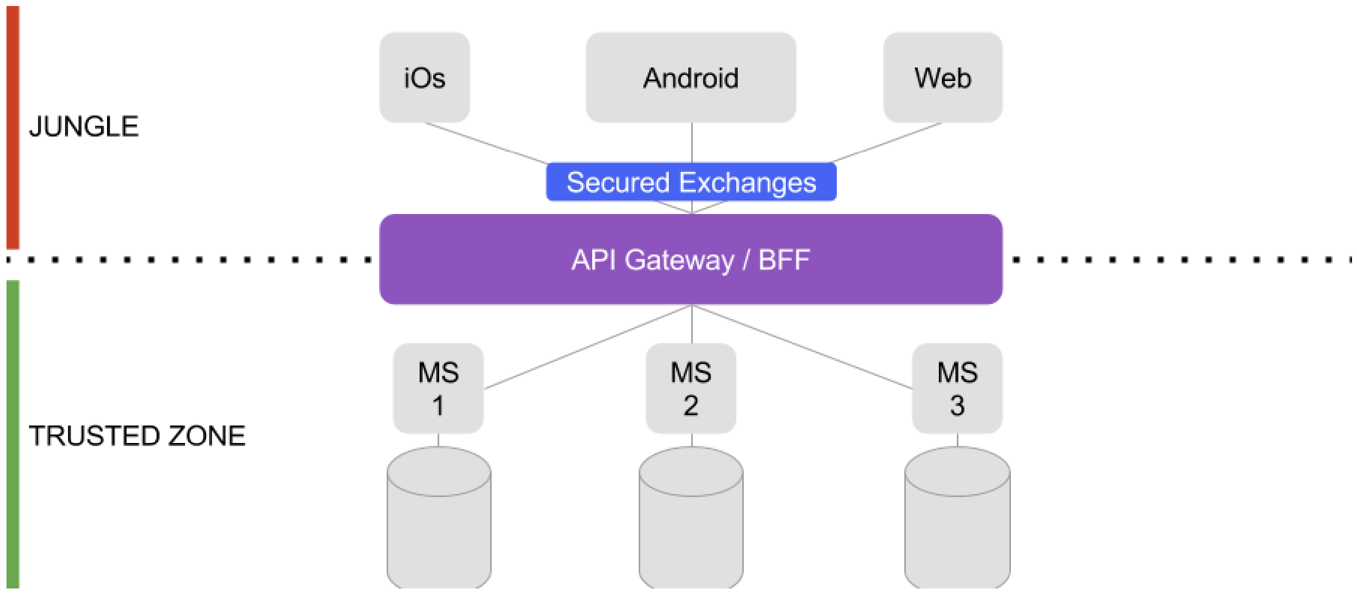
Thus, the main difference here is that we split security requirements in the platform to ease accessibility and reach the best point of overhead. The main takeaway is here :

> Distributed software does not allow you to create security breaches

So we simplify security requirements but in the other way, we have to be at the point of overhead for each components.

Pragmatic Secured Microservices Architecture

I purpose this pragmatic architecture as it may reach all the intakes of small companies, have the best of security without stopping product development or platform usage.



Let’s define that inside our platform, this is a **trusted zone**, where exchanges can be done easily with few restrictions. Outside, however, it’s the **jungle**, we don’t know what can happen and we have to do more efforts on security and particularly on exchanges between clients and gateways.

Security References

Before digging in the set of rules for this architecture, we should consider security references as it is easy to find reliable sources of how to implement security in your system.

I guess OWASP is the more known security reference as they collect a lot of best practices and help you to prioritize rules to implement. I particularly point on the so named Cheat Sheet that will give you clues on how to hack your system :

- REST Cheat Sheet
- Mobile Cheat Sheet
- Network Cheat Sheet

Another good reference system is CVE that gives you a score — CVSS-referencing the severity of the breach for your company. It may help you to find the best implementation order to maximize the impact on security.

. . .

Rules

Now we have the main approach of security, we can dig in detail each rules that will allow us to reach the point of overhead of our security system on our distributed platform.

Global infrastructure

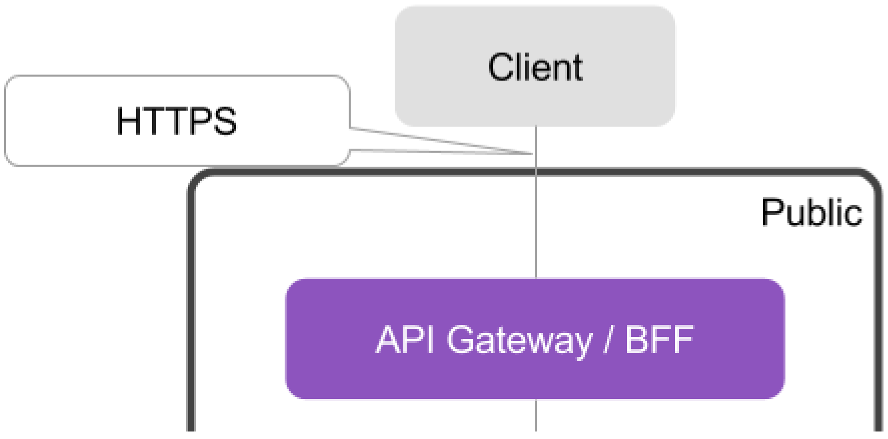
Certificate rotation

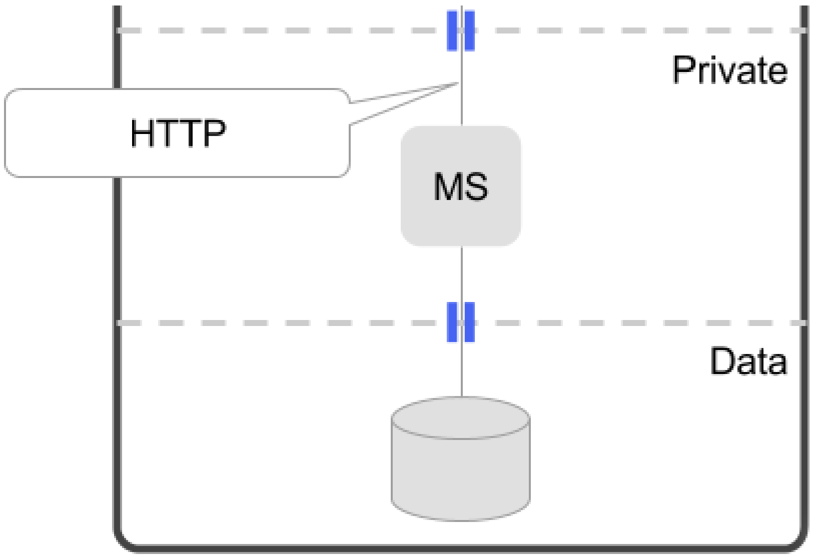
Client requests a server. Server is providing official information and specifically a secured certificate proving that client asks to the good server and not a fake one. Rotating the certificate will reduce the opportunity for a hacker to access a copy of this certificate. If it is updated regularly, the probability that the hacker gets an outdated certificate is stronger.

Solutions : Let’s encrypt, AWS ACM

SSL/TLS Termination

When a request arrives on the server, it’s encrypted through a secured protocol. API Gateway is in charge to decipher a request and transmit it to microservices on a deciphered way. We do that because it’s a pain for developers to set up SSL in their application servers and it’s a pain for ops to maintain certificate rotation on all the components.





Global infrastructure rules

Isolated Subnets Layers

Global network is locked, there is only one entrance. It should be the same inside. You should apply bulking, compartmentalization of your layers, to avoid the opportunity for a hacker to progress from layers to layers.

The more sensitive the component is, the deeper it should be in the infrastructure.

Isolated Inbound and Outbound Rules

Network rules should define exactly what can be received and what can be let out. This is the next granularity level from the previous rules. If you master that, you will then be able to define specific exceptions on it. For debug purpose, it should be handy.

User sessions

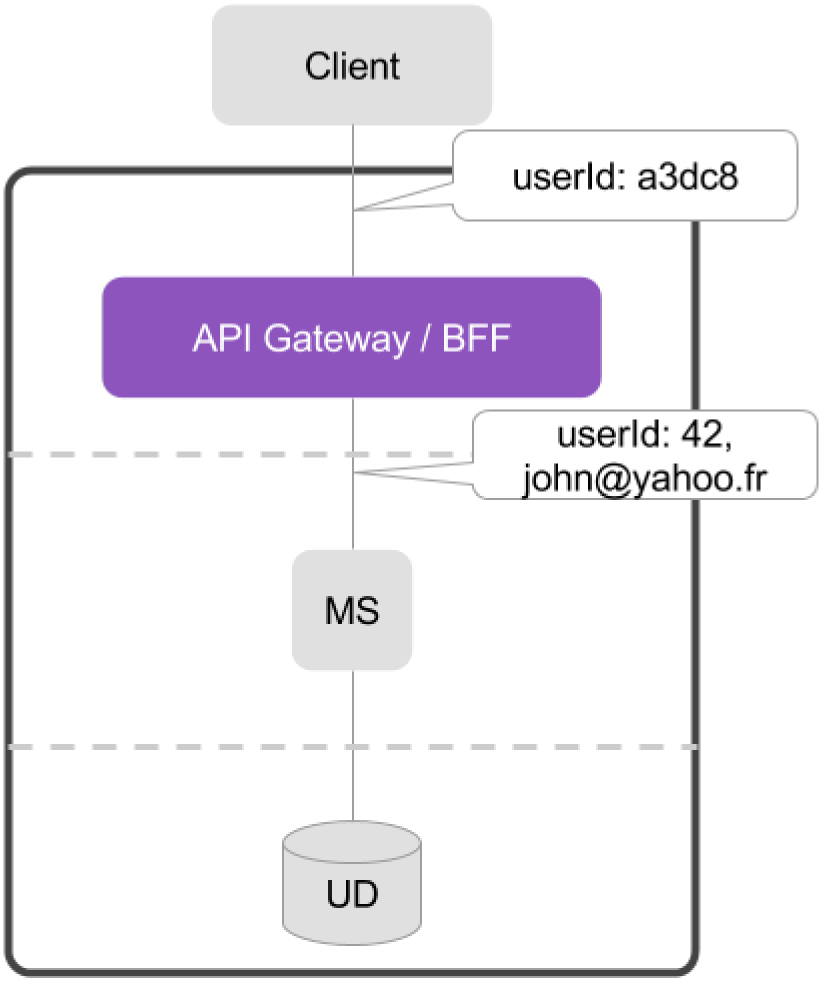
The most sensitive data in our platform is the user session, we should then handle it with secured rules.

Outside session

We encrypt an authentication token that should be ideally a totally random and reproducible token. It can be as simple as a UUID. We do that for two reasons :

- **we don't care** about maintaining a specific coherency outside our platform, client is logged in with direct user input so it gets back a token and then use it;

- **random** is the new defense against hacking, we don't give information about how we handle user information inside our platform;



User session rules

Inside session

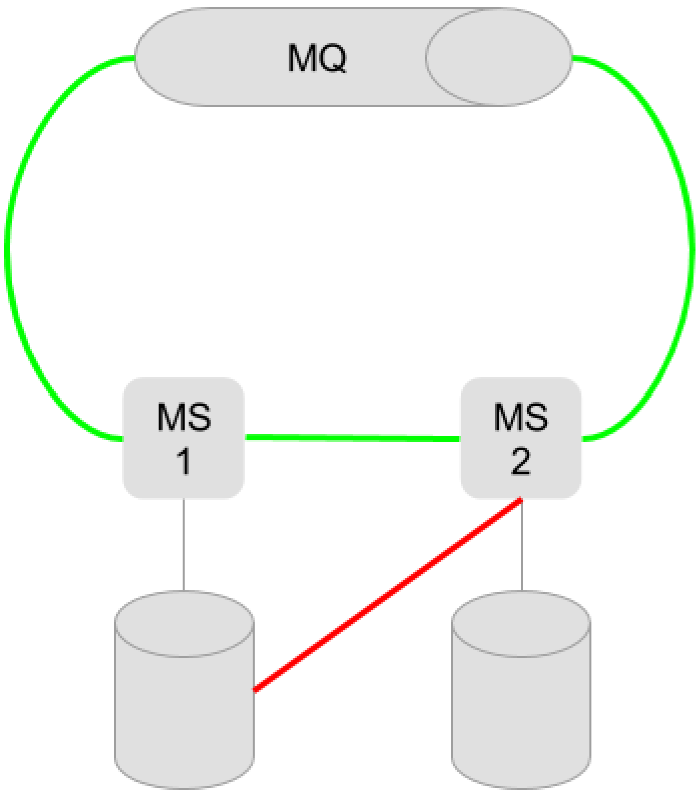
The user data that is useful must be in the global context, ie in each request header. The **API Gateway** is in charge to do the mapping between inside data and outside token. The logic of handling data must be **externalized** in a specific *account* microservice that will store this user data (not in the api gateway).

We do like this because :

- **user data is centralized**, it can be also be optimized for performance ;
- **other microservices** receive the user data that is in context as they don't own this data and they **don't have to request** it each time;

Microservices Data Isolation

The data belongs to the microservice so it should not let other microservices read or write inside the data system storage **directly**.



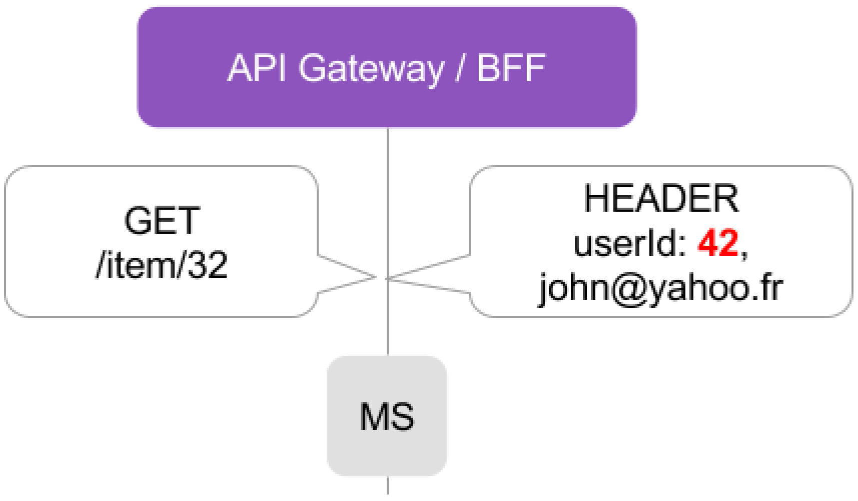
Microservices Data Isolation

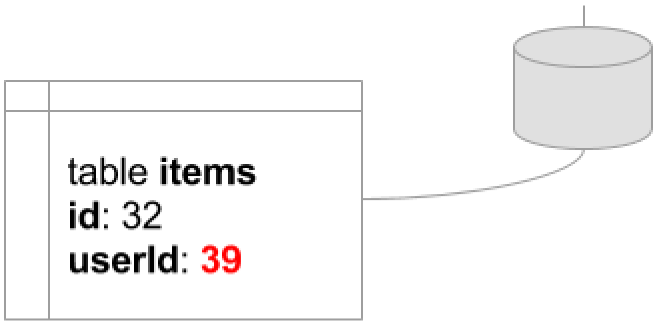
The microservices must access to it via an **API call or asynchronously**.

And obviously, set up **different users** / **password** to your data storage systems.

Microservice ownership check

This one is about applying confidentiality : when the microservice is receiving a user id, it has to **verify** that the data which is sent effectively belongs to the right user.



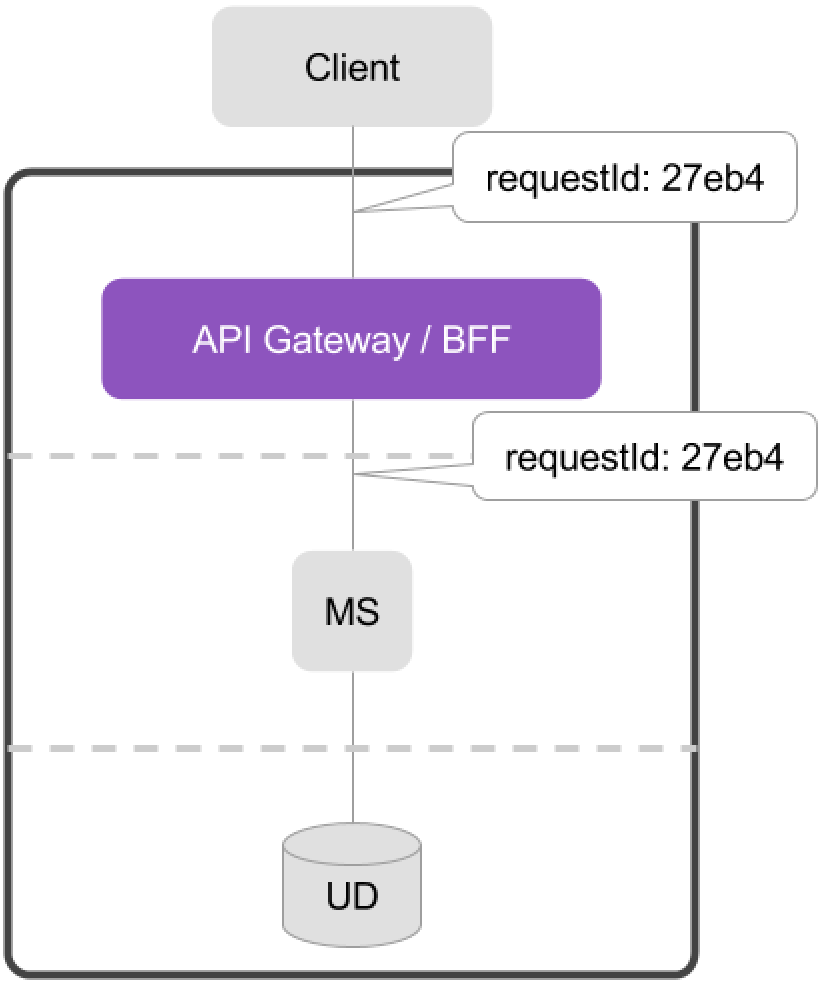


Microservice ownership rules

This mechanism may be extended to **roles** if you have any.

Correlation Id

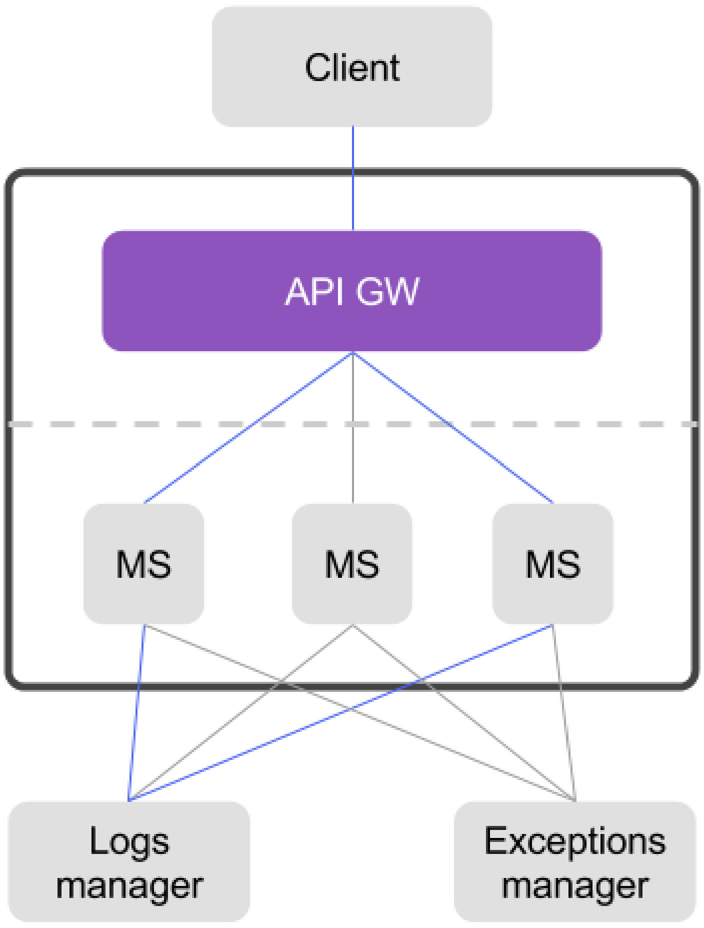
Your API Gateway / BFF must transmit a **correlation ID** to all the requests, which is set by the clients. UUID v4 is enough. It should be transmitted in the headers of your request as it's a meta information.



Correlation ID transmission rule

We do that so then we will able to track precisely the origin of a request and the consequences of a request in your platform. It gives you the knowledge of what is happening in your platform.

Microservices — Diagnose and recover



Logs and exceptions managers

Once you have a correlation id, you should also extend your system to quickly diagnose issues and recover from those.

- **Tag** everything in your infrastructure, following a naming convention to identify every elements in it;
- **Log** all the request to a central log system that can be either in your infrastructure or as a third party service, omit headers and post information;
- Use an **exceptions** platform to handle exceptions and get technical context on it, ie with what you did not log;

We do that to split information for tracking (log) from technical incidents (exceptions). Logging trace is a static way to collect information meanwhile exceptions are a dynamic way to do it. Some exception managers also handle the resolution state and can tell you when you have close or open again an exception.

API Gateway Detect traffic patterns

Now that the platform is collecting a lot of tracking information, we are going to analyze it in order to detect traffic patterns. We are following here a statistical approach against the collected log.

Automated traffic

Half of the internet traffic is due to automated scripts, such as robots, spiders and crawlers, but also to marketing companies that scan your website for example in order to provide automated marketing analysis to its customers.

Legitimate robots are following the **robots.txt** and will follow your directives about what they are allowed to scan or not.

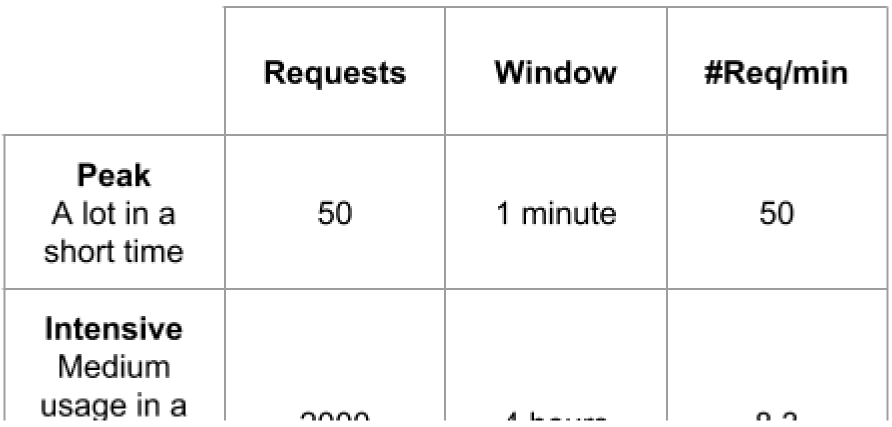
You should **exclude** those logs from your statistics as it is **noise**.

Real traffic

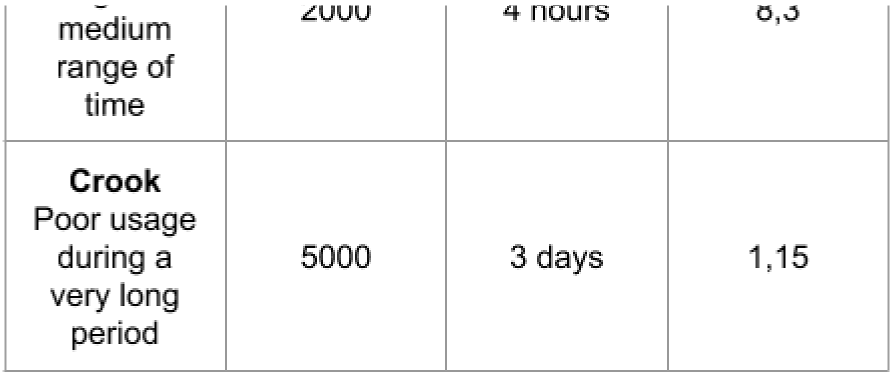
You may also want to **exclude** real traffic of your statistics as a user should have more or less the same profile and the same statistical behavior. For example, you can consider that a user generates between 2 or 3 hits by min on our api gateway, with a range of maximum half an hour.

Accumulated traffic

Last, but no the least, you should identify clients that generates a large amount of requests on different frames. I set here a common example that should match for a lot situations.



	Requests	**Window**	**#Req/min**
Peak A lot in a short time	50	1 minute	50
Intensive Medium usage in a	2000	4 hours	8.3



medium range of time	2000	4 hours	8,3
Crook Poor usage during a very long period	5000	3 days	1,15

Accumulated traffic for edge traffic usage

I define here three categories of users that are spotted statically because the frequency of requests or the length of usage is **abnormal**.

Prevent and block suspicious behaviour

Now you have spotted people or companies, you can speculate about their intention :

- **DDoS** attacks → Put your platform down
- Attacks through **DAST** → Find backdoors to hack you
- **Brute force login** → Scammers
- **Enumeration** of urls → Steal data through scraping

In order to block those people, you may use a Web Application Firewall (WAF) that include specific rules, either on rate limiting, geographic belonging or IP origin (useful for TOR endpoints). This should be set at the entrance of your platform, prior to your API Gateway. My opinion is that a WAF is pretty dumb and intelligence can be difficult to give it.

You may also complete this with a third party solution giving you complementary advice, in real time, about the request you received. You should integrate this inside each of your api gateways. If interested, you should look at Sqreen or Distill Network.

Mobile

Finally, advice for mobile security, a bit at the edge of the subject of this article but that is worth it.

- Master all the **interfaces** of your app : in its **environment** : file system, identity system, http layer, … and what you **expose** : share / publication system;

- Be careful of **what is left** on the device after uninstall;
- **Certificate pinning** : a user can usurp client identity through self signed certificate;

Conclusion

We have seen in overview and in detail that security is important and should be not neglected by companies or developers. Security is a process that has to be **integrated** in the main **development process** and **platform life cycle**.

Microservices are leveraging complexity in code by adding complexity in interaction. My approach is to leverage extra security layers that are not needed inside the platform. Security has also to be tempered to reach **the overhead point** and applied **only when needed**.

Don’t hesitate to react and share, I will be happy to discuss alongside this subject.

Microservices Security Architecture