OBLO

To 1M
in a nutshell

Your partner for IoT products and services

Introduction

Building an IoT solution assumes dealing with many different technologies, developing heterogeneous software and hardware components, and facing integration challenges.

Today, every IoT solution is centralized to a certain point. Usually IoT devices are constrained in terms of resources and they rely on a cloud as a central, extendible resource and service pool. The cloud can be either remote or local (on-premises), depending on the infrastructure location. There are use cases when it makes sense to have the cloud on-premises, e.g. due to performance or security reasons. When IoT devices are spread in a wider geographical area, the cloud has to be remote and publicly accessible.

Depending on the business strategy and available technical expertise, a company can either partner with an IoT solution provider or develop the solution internally. Taking into account our longstanding experience in product development and expertise in diverse software technologies, ten years ago we decided to develop end-to-end IoT platform (cloud service, hub/edge software, mobile clients and SDKs). With such solution we remove dependency of other technology vendors and provide flexibility to our customers with respect to customization and integration.

IoT and the business around it was (and still is) evolving; our first deployments were rather small – up to a thousand devices. Over the years the traction was growing and we had to look into the future and support massive deployments of millions of connected devices in a system!

Due to live deployments, we had to make some hard decisions on how to move on.

This article gives you some insights about our journey on reaching a production-quality, performant, reliable, and scalable cloud. The goal is to share experience we collected on this road, and to point out pitfalls developers or customers may face in the IoT domain.

The journey
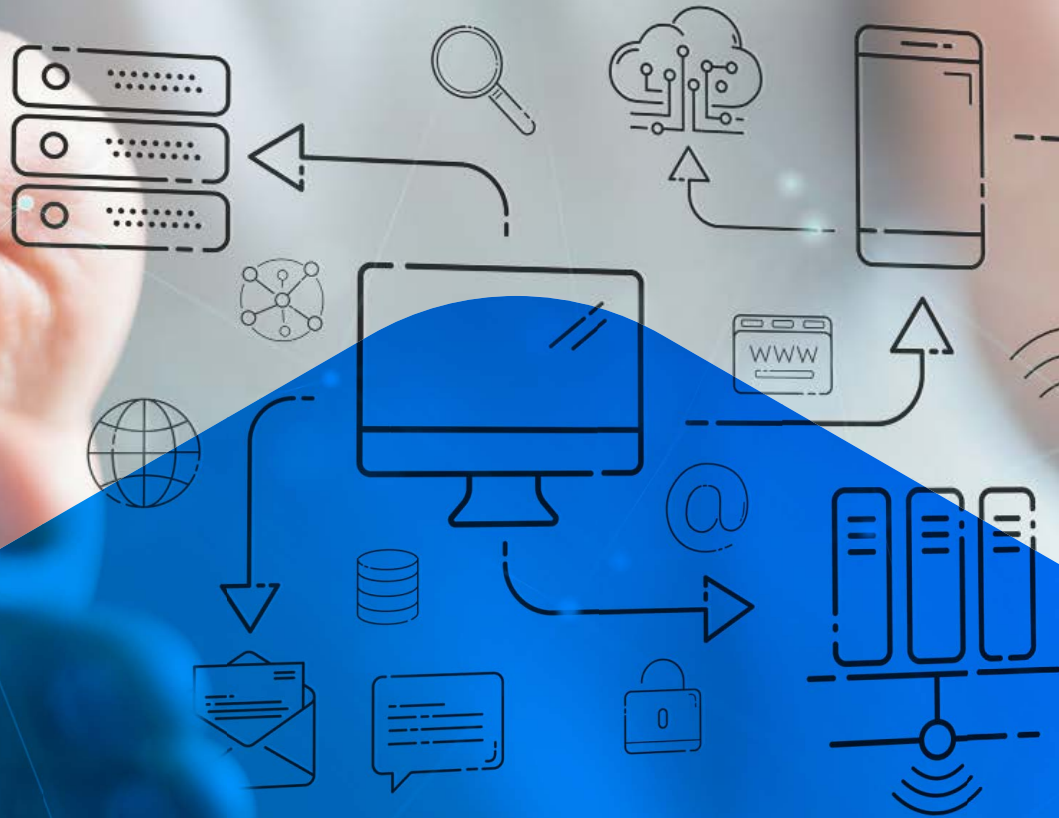
Early days

At the very beginning of our involvement in IoT, the technology selection was slightly biased by the technical know-how, as well as by our intention to cover several business cases with a single solution.

As a result, we built the first version of the cloud based on Java, Hibernate, and other typical technologies often bundled with Java. We believed TR-069 can be reused for IoT as well. That involved HTTPS/XMPP messaging and XML formatted payloads.

Developed cloud was functional, working for tens of thousands of devices. But... our simulations showed that we would hit the ceiling at ~50k devices, due to connection handling and the massive amount of data generated by devices. The other concern was the timeliness of data. Our use cases assume a certain level of interaction with the system; therefore the communication between the device and the cloud had to be more frequent, bringing the protocol overhead issue to the surface. So we decided to change our platform architecture completely – it was not an easy decision.

Hello Cloud v2.0!

New technologies

During the conception of the new version of the platform, we have formed a team of experts and architects from all domains. After rounds of analysis and de-risking, we decided to go for MQTT. Other contenders were CoAP, AMQP, but we ruled them out in favor of overall platform benefits. MQTT is a good fit for purpose, secure, robust, and scalable. It is an open standard, actively developed, and has the great support of the IoT community.

The selection of the basic concept of the platform had a great impact on the cloud codebase. We found JavaScript and Node.js based technologies to be more suitable for an IoT cloud. As a result, we had to completely re-write the cloud and integrate it with other platform components.

At the end of the day, this huge work paid off! Without that tough decision, we wouldn't have been able to offer a production-quality end-to-end platform. The selection of more appropriate technologies helped us to increase the scaling capabilities and speed up the evolution of our software.

Challenges and design considerations

Multiple codebases. The more performant cloud is great, but one thing must not be forgotten: maintaining two IoT cloud codebases is not what we envisioned or something we could do over a longer period. Yet, the existing deployments needed some attention, maintenance, and introduced some inertia in software development. So we gradually migrated all deployments from V1 to V2, including devices and related services. Luckily, support for that was built into the new architecture, so with great attention, we managed to perform this transition in 6 months. But still, it consumed manpower and energy.

The selection of the right technologies is just a prerequisite for success. Beyond that, a lot of work needs to be invested in feature development, robustness, security, and scaling.

IoT evolution. IoT rapidly evolves. As of today, 5G and NB-IoT are emerging, IPv6 is getting its moment due to the jump in a number of connected devices, especially in a populated country like India. New features are generating more and more data and someone has to take care of that. Therefore, the software evolution is inevitable and has to be built into the software architecture and deployment procedures.

Robustness and scaling. In the real world, things can and will fail – a software process can go wild, the network may be down, hardware can fail, power can break. But once you get involved in offering services, from the launch date, the service must be functional.

To ensure robustness and scaling, we opted for clustered services. Clustering means spreading the service across several computers. The benefits are twofold:

- It is unlikely that all instances of a service will crash at the same moment; some of them will be always operational and they take over the load until failed ones are restarted and back to cluster.
- The service throughput is increased by adding new nodes to the cluster.

The following main services are clustered:

- One of the key components is MQTT broker that takes a significant part of the load. Almost all ingress data goes through the broker. It is tightly oupled with advanced Access Control (ACL). To avoid having a single point of failure, MQTT brokers and ACL services run in a cluster.
- MongoDB is used as a database. There are dedicated DB clusters for configuration (carrying low-speed data) and device-generated data (medium speed data). Data replication and sharding are ensured in the cluster, so the data remains safe.
- Redis cluster is used for services (like ACL) that require very frequent and fast data lookups.
- For internal messaging and RPC RabbitMQ is used.

When the system scales to 100k devices, 500k devices, 1M and beyond, underlying cloud infrastructure is getting more-and-more influential: connections may become unstable and dropped, delays appear, inherent thresholds cause unexpected effects. For huge installations, the infrastructure becomes a part of the development, deployment, testing, and maintenance. Major IaaS providers we work with (Google, Amazon, and Microsoft) have their own specifics and operational quirks. For a big infrastructure, DevOps needs to take care of front load-balancers, internal load balancers, proxies, network interface configurations and limits, and many more. It might turn out that a server could handle 100k devices from a resource point of view, but devices might not be able to connect to the server due to network interface limits set by the IaaS provider or the operating system.

To summarize, the impact of cloud software and infrastructure on each other becomes more significant as the scale raises, introducing another variable into the ecosystem.

Our cloud supports both vertical and horizontal scaling. Vertical scaling assumes the extension of already used server nodes with more CPU power and RAM memory. Horizontal scaling involves adding new server nodes to the cluster, without modifying the existing nodes. The strategy of the scaling may be constrained by underlying infrastructure and related costs. Mixed scaling strategies are also an option.

Testing

System testing before deployment is crucial and covers as much as possible test cases. Tests should push the system to the limits and beyond. The goal of the testing is not to confirm that the system is working correctly but to break the system and find its limits and flaws.

Functional testing

Functional correctness of the system is verified by using physical devices, according to the verification test plan, and followed by FUT (friendly user tests).

When a high number of devices is required for testing and verification (e.g. massive firmware upgrades, big data analytics), we use device models. Device models are software components that abstract the most important functionalities of physical devices and can be executed at a massive scale (tens of thousands on a single computer).

Security testing

Integral part of testing process is checking the system against known vulnerabilities and flaws. We do track the evolution of open-source software components used in the system and upgrade them periodically.

However, final security check shall be done by professional companies and we are proud that our IoT platform already passed security audits by leading security companies.

Robustness testing

As mentioned, software and hardware can break. The platform is designed to cope with such failures and this is verified through robustness testing before deployment. The testing is controlled by DevOps team that observes system behavior in case of VMs failure and system services failure.

Scale and stress testing

The goal of scale and stress testing is to find the limits of the platform. It is done using a powerful test server park running software device models, each attacking the cloud simultaneously. The cloud is running in a staging environment that is the same as the production environment. Such setup allows us to play with different failure scenarios, observe the system behavior, key performance indicators, resource usage, recovery procedure, and timings.
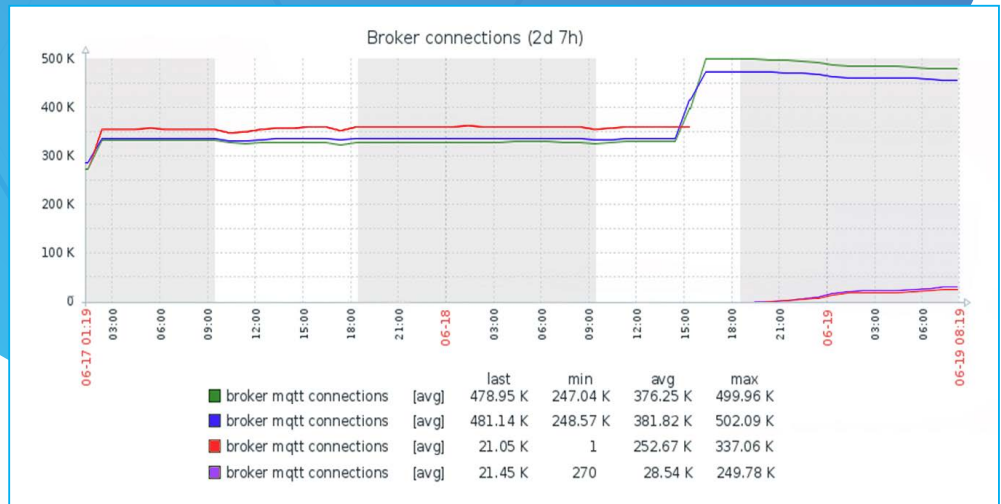
1M connection tests including recovery after broker failure

Test runs may take considerable time, here are a few examples:

- 4 hours for load verification tests (e.g. bug fixes validation)
- 2 days for robustness and stress tests (e.g. server down, network down)
- 1+ weeks for stability tests (e.g. resource utilization consumption)

Some interesting numbers from testing are:

- During 48h tests, 1M edge devices exchange ~5TB of data via the broker.
- In the same test the device history database grew for 500GB
- The average number of MQTT messages was 20k messages/second.
- To generate the substantial load of 1M devices, 20 test servers were used.

After the testing is successfully finished, we know exactly which infrastructure is required for a given load and QoS. This allows us to tune the scaling strategy, appropriately dimension the cloud infrastructure, manage costs, and meet the objectives of our customers, today and tomorrow. And we are sure our platform will not be limiting our customer's business!

Key takeaways

OBLO IoT platform has gone through an evolution in terms of technologies and capabilities. This paper briefly depicts the journey, challenges, our solution and the benefits for our customers.

Having a scalable, production-ready cloud is not just about technology selection and development. There are other important aspects like use cases-driven architecture, infrastructure knowledge, testing methodology, feedback from the field, and a skilled team promoting product-oriented mindset.

Thanks to the effort and knowledge our team invested, we are proud today to offer our customers end-to-end IoT platform ready for millions of connected devices. Usage of the platform extremely shortens time to market; Our platform can be deployed on IaaS or private infrastructure within weeks and from that point further customized and integrated with Customer's IT services.

And course, we are not sitting idle. We are constantly working on further improvements and new features – stay tuned!

Narodnog Fronta 23a
21000 Novi Sad
Serbia

Phone: +381(0)21 480 1213

info@obloliving.com