Building Pitaya, Wildlife's Own Scalable Game Server Framework

For several reasons, many of our backends were written in node.js. It is scalable and has excellent performance, especially when compared to languages like Python and Java. It was the language that our software engineers were most proficient in and, most importantly, the most popular programming language at the moment.

We were able to use several open-source projects to build our systems. One of these projects was Pomelo, "a fast, scalable game server framework for node.js" as written in their readme on Github.

But, why do we need a game server framework? In a very simplified manner, for a complex game, we need different types of servers.

These services could be called "not-so-micro" services. Each of these services (or server types) has different responsibilities and needs to scale in an independent way. We have connectors that are the servers to where players (or their phone) are connected. These connectors handle basic operations, such as authorization and authentication. We have the game servers where the matches (and the magic) actually happen. We have metagame servers where the other operations in the game happen, such as card upgrades, fetching the leaderboards, etc.

The demands for a game server are quite different from a metagame server and we need all of them to scale up and down horizontally, given that we operate with a very large demand which oscillates with special dates, feature releases and at different times of the day.

The framework has one major responsibility: to ensure that all the servers can be managed in a transparent and efficient manner. Servers need information about other servers regardless of their type. They also need the ability to communicate with one another, often performing operations in another server. These features are called RPCs (remote procedures calls) and service discovery.

Another important responsibility of the framework is the need for clients to connect to the servers. These connections are often long-lasting and require low latency. It's also very important to be able to broadcast messages to several players that are playing the same match.

Pomelo really simplifies both responsibilities and was a great tool for us to launch our first synchronous multiplayer games. Unfortunately, the framework's maintainers did not keep it up-to-date and used an older version of node.js. We discovered bugs and sent PRs.

Unfortunately, because the code was not maintained it took a very long while for these changes to be merged.

Most importantly, eventually, some of its bugs or design choices didn't allow it to meet our demands at the scale we needed. Gaming Due to too many servers, broadcasting ping messages that indicated whether a server was online or not was enough for overload. RPCs were also difficult if the servers were located in different areas.

Pomelo was reaching its limits, so we began to think about what next. We are a firm believer in innovation through research. This means that we spend a lot of time researching what works and what doesn’t before we commit to creating something new. Because we used Pomelo as the framework for our game servers for several years we knew it extensively: its strengths and its flaws. This enabled us to keep the good, change the bad, and improve what could be better.

Go takes place

In early 2018 we had already been using Go as the main language for the backend systems for quite a while.

The switch from building services with node.js into Go in mid-2016 was largely based upon Go's incredible concurrency mechanisms. These are essential for building scalable platforms for our millions of users. We also have many servers so low resource usage is an important feature. This reduces infrastructure costs and complexity.

It was then a perfect choice for us to build Pitaya, Wildlife's own scalable game server framework.

Pomelo kept the distributed design and protocol to allow client-server communication. The most significant changes were made to the service discovery and RPCs approach. These were the main production issues that we faced in Pomelo-based games.

Pitaya should have observability features built in. We made Pitaya compatible with OpenTracing frameworks like Jaeger. It also supports Statsd as well as Prometheus. This is one of the greatest things about Go. We were so involved with the community and its amazing open-source programs that it not only helped to create a more effective framework but also encouraged us to use other open -source Go projects like etcd and NATS.

About Pitaya's success and adoption inside Wildlife, the framework is being used in production in our main games for a few years now and has helped us successfully launch

some of the most amazing games we've done so far like Tennis Clash and Zooba.

Our Game Engineers are happier, as we also wanted related projects that would make Pitaya's development process easier. pitayabot, a framework that allows for integration and stress test bots, is an example. Go's low-memory and CPU footprint and the amazing tooling pprof provides for profiling make our SRE and Infrastructure Engineers happier.

Pitaya is an open-source platform that allows anyone to build amazing projects. It was the logical choice after we have benefited so much from another amazing open-source project in the past. As an added bonus, we can have others to help us improve it by contributing to the projects.

This post should have explained why Pitaya was built and why Go was the best choice. In the next post, we'll explain in-depth some of Pitaya's features and technical decisions.

If you love building challenging new projects and Go as much as we do, you could be a part of our team! We hope to see you soon and invite you to take a look at the available positions!