

GOLANG

DEVELOPMENT

Simple. Scalable. Concurrent.

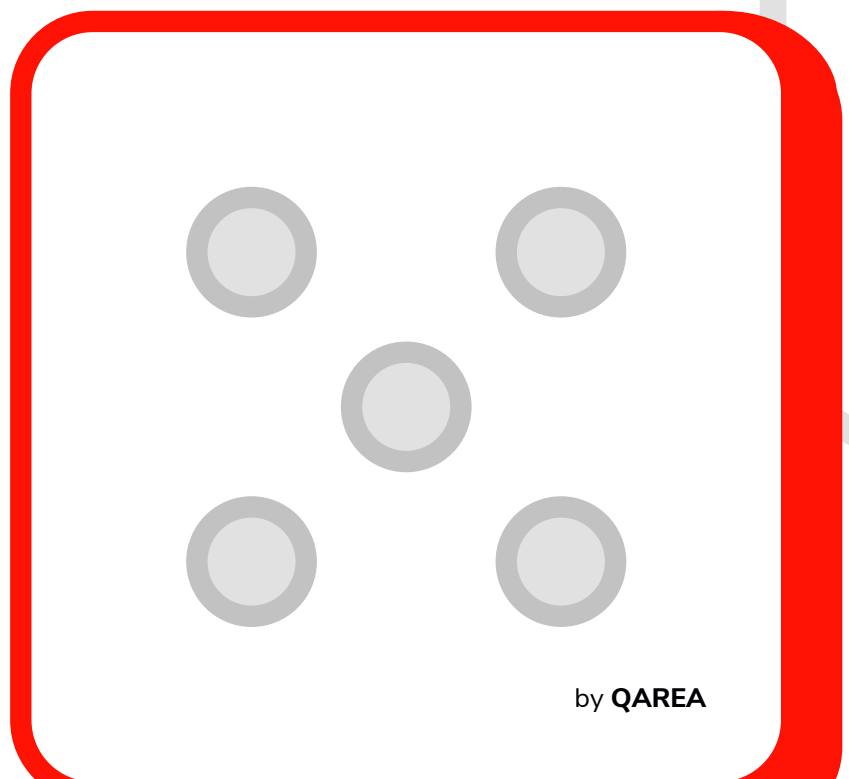
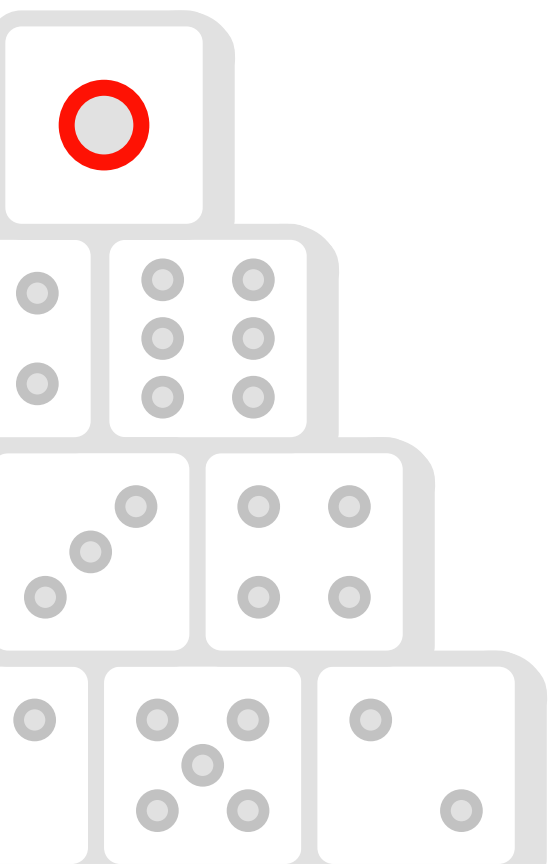


Table Of **CONTENTS**

01. Why Golang?
02. The Perfect Case
03. Reliability
04. Cross-platform systems
from the box
05. Microservices

Why GOLANG

Golang [is a growing trend](#) in the software development community. Thanks to its “from the box” concurrency, stability, and scalability, the language has become one of the core technologies for countless businesses. Some of the brightest examples of companies using Golang are:

Google

NETFLIX

 Dropbox

 Adobe

BBC

 Bitbucket

DELL

ebay

IBM

(if you are interested in other examples, the full list can be found [here](#). Who knows, perhaps you will find one of your direct competitors there.)

Concurrency is not the only thing Golang comes with “From The Box”. It supports a wide library of Databases that allow for realization of [RDBMS-based](#) development on the fly. What does this bring to the table? Solid, responsive [microservices that play along nicely in a given system](#).

Furthermore, Go’s from the box database support has allowed for quick development of yet another microservice, only now it’s [HTTP-based](#). Same can be said about development with any other DB.

Ok, these are just a few reasons why QArea uses Golang. But why is the entire world losing their minds over it?

Simple. The language was created with a bold intent of improving programming as it is and it has delivered exactly that.

HERE'S WHAT GOLANG IS DESIGNED TO DO:



Speed up development and eradicate slow builds



Open Source nature that allows for vast community support



Better code readability and better understanding of a program



Prevention of re-work and effort duplication



Decrease the learning curve and simplify entry for beginner programmers



Cross-platform builds from the box that support a multitude of technologies



Free updates for all as early and as often as possible



Automation-oriented technologies from the start with amazing concurrency

That looks amazing on paper. But does Golang work the way it was intended to in real life? Only a solid case may serve as proof of concept. Luckily we've got just that all wrapped and ready.

The PERFECT CASES

QArea is a large software development outsourcing company with a 17 year history. Throughout the years we have delivered complex solutions to more than 800 businesses. Also we've learned quite a few lessons.

Here's one of them: There are similarities between some products and there are specific differences in every new case. Unfortunately, there's a general rule of thumb - if we are working on a project with at least some share of legacy code in it, we are going to encounter some problems.

Here's a fine example of what is meant. One of the more recent projects we nailed had the following "features":

- 1.** 3-rd party apps integrated into the system
- 2.** Never supported before
- 3.** Original developers are long gone
- 4.** 90% of the certificates were about to expire
- 5.** Documentation is in utter chaos

Now that's what we call a Full House! And, as you may have guessed from the title of the paper, we used Golang to transform these challenges into an A+ project.

WHAT DID WE DO?

First, we had to sort out the documentation and certificate-related issues. We couldn't have done it manually, so we did what every smart developer would do under the circumstances - we used automation. Golang was friendly enough to provide us with a tool just for that - [the Log Package](#). It is designed to simplify these kinds of documentation checks. It works as follows: If the TLS certificates are loaded to the server in an incorrect order, our team would receive a notification (error message).






Then [the HTTP package](#) came to our aid. It's quite simple in implementation, and, at the same time, it has an amazing function - the ListenAndServeTSL. The function checks for the certificate's validity and authority. Plain and simple and just what we needed!

[Crypto/tls](#) was also a part of the bigger picture. This function configures usable STL/SSL versions and it implements TLS 1.2. Or, in simpler words, we had the security of the system covered in terms of connections thanks to identified elliptic curves and cipher suites.

How did Golang help us with legacy code and poor documentation?

- Simple, accessible automation that decreases the budget needed for the completion of a complex project
- Lightning-fast failure reports and error messages that are ready for analysis within moments, thus decreasing required manhours on the project
- A defined path for certificates that's both stale and well documented for future use by other teams (if need be)
- All of the benefits come from an open source technology meaning no extra investments into tools or frameworks are required

Project Code Quality Check

SQALE Rating	Technical Debt Ratio
A	<u>0.1%</u>
Debt	Issues
<u>13d</u>	<u>667</u>
 Blocker	<u>0</u>
 Critical	<u>0</u>
 Major	<u>0</u>
 Minor	<u>241</u>
 Info	<u>426</u>

As you can see, the SQUALE Rating is a solid “A”. However, the contribution is not ours to claim. Golang linters check code before it is committed and works as a quality check roadblock. Anything larger than minor code issue will not pass the check, forcing the developer to fix the issues ASAP.

This case is just the tip of an iceberg. Golang has so much more to offer. You are welcome to learn more about these offerings in the next chapters.

RELIABILITY

DropBox is a behemoth when it comes to cloud services. Don't take just our word for it though, take a look at the stats that support the bold claim:

- They have more than 500 million users
- DropBox's servers store more than 500 petabytes of user data
- There is a multi-exabyte go storage system

With colossal volumes of data like that DropBox chose Go as the primary technology to support some of their largest systems. Why? According to [Tammy Butow](#), the Site Reliability Engineering Manager at DropBox, her company had quite the expectations. Tammy's team was tasked with development of a reliable and secure system. Not only that, but DropBox also needed those security and reliability factors built into the core design.

Additional expectations were as follows: DropBox needed to create a system that provides annual data with 99.9999999999% durability and a system with availability over 99.99%.

Now that's what we call a challenge! How did everything work out?



The lion's share of DropBox's entire infrastructure is written in Go



Over 150 developers from the open source community contributed to the server repository



Over 1.3 million lines of code were written in Go

Here's something even more interesting - according to Tammy, despite the colossal volumes of work, the development team became more productive and generally they've gotten happier. It's just so easy to be productive with Go, especially today when there's a solid standards library backing important decisions. Standards allow for clarity. If a change was made to the system, everyone involved will know exactly how and why it was done.

A heavy emphasis on the microservice architecture that Golang is known for allows teams to build tools for other teams. All in one system. All with the same coding standards that prevent legacy code from entering into the system.

And, for the cherry on top - Go has a stunning array of debugging tools from the box. The combination is like Christmas in July to a professional developer who works in a team and with other team involved in a large-scale project.

DropBox is known for its fast release cycles. They rely heavily on automated tests that cover the code in order to keep up the speed that's so much needed in today's world of cloud and SaaS. Go's own unit testing library helps greatly. Testing is as simple as using pre-made solutions from the library and running them. Such an approach saves a lot of time and helps greatly with deployment.

Sounds great on paper, doesn't it? But did Go work for DropBox?

If you ever used the app in your life you probably know that Tammy's mission ended up with a jaw-dropping success. That's just what Go - while in the right hands - does to large-scale projects.

Cross-platform systems

FROM THE BOX

Go has a library that provides string formatting that also provides the crypt line function. So, in other words, you are simply saying to the system - “here’s your string, love. Please print it out.”

What about actually building the program? Just say the following:

```
$ go build main.go.
```

```
$ file main
```

Voila, your build is ready. Here’s what you’ve gotten out of it: you have an ELF 64-bit LSB executable that is not stripped, but statically linked. This means that the program will run on any 64-bit (or 86bit) you wish to put it on. **No external runtime environments are needed!**

Here’s when the fun begins. Few businesses today have the luxury of running everything on a single system. Your developers, marketing and sales teams might be using different tools on different platforms or even OS versions. This makes it challenging for you to build a solution like a time management app or an ERP system or whatever else you wish for all of them to use.

What usually happens in businesses is as follows. You have a program designed for administering a task or a series of tasks. Your dev team is trying to keep the same runtime on every environment that’s in use, obviously. But you have way too many of them. They may be even the same OS, but in different versions.

Go changes the rules, of the game, though! If you need to adjust the code to run on any OS, you only need to adjust ONE variable!

That's it. Use an updated build with the Hello World program and run it on Windows - it will work flawlessly.

You get your MacOS executable. Same program, same code with 1 change in 1 line is all you need for development of amazing cross-platform apps!

Same can be said about any other OS.

Sounds nice and peachy, right? But we've only discussed current, relevant systems. What if there's a 3-rd party solution you have purchased ages ago for your business and it has since become an integral element of your business processes? The firm that sold the app to you is long gone, there's no support or updates and shifting to something new is too expensive or too complicated because it affects countless processes in various departments?

You've already invested a fortune into keeping that app alive with fixes, patches and duct tape. Is there a way out? Sure.

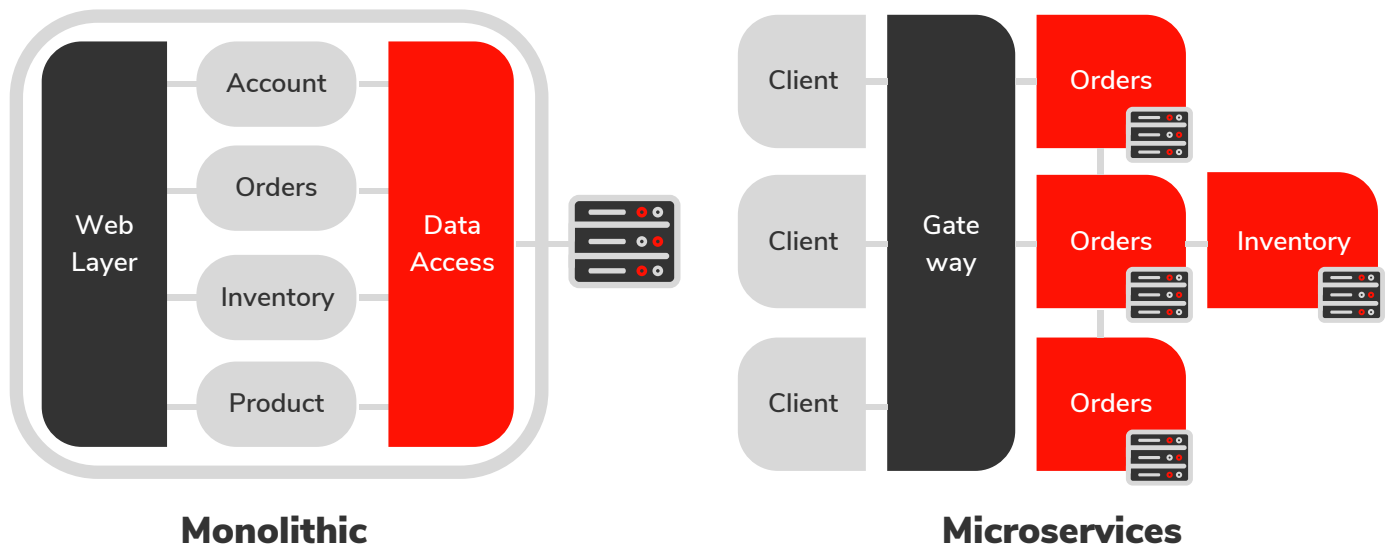
As we said before - all it takes is one iteration!

MICROSERVICES

Microservice-based architecture is the future and that future is already here. Amazing systems like Netflix, Twitter or Hailo are based on these tiny bad boys. That's because the benefits are colossal!

In a monolithic system even the slightest change in the code can lead to the entire system collapsing. That's why we see sites and servers down all over the web after the company released a patch or something. A monolith is not a scalable solution for an ongoing, evolving business for that exact reason.

What are microservices then? They are a system that's made of tiny clusters of code, each responsible for a particular feature. The system runs when all of them are together and even when one or several of them are removed. A patch or a change in the cluster will never lead to the entirety of your app crashing and it's easier to find the soft spot that causes issues in a hundred lines of code than in several thousands.



Ok, so far so good, But why's Go great for microservice-based architectures?

Concurrency!

Scalability is a major issue in modern software development. Having several microservices isn't such a big deal. But what will happen when your system will consist of hundreds of them? Concurrency becomes essential.

Following good coding practices, you can build a server that is highly scalable in-terms of concurrency, CPU and memory utilization.

"Go's big contribution was the native concurrency story, and arguably the implicit interface stuff. With these features you get a lot of value, and you get a lot of expressiveness that wasn't available in languages like C." - Peter Bourgon, an engineering rockstar and a QConn speaker.

Would you like to know more about Go?

Contact QArea and submit your questions into [this simple form](#). It won't take longer than a minute. Our team of analysts will come back to you with a working solution ASAP. Allow us to guide you through everything there is to know for your next big, scalable project!

About QArea




QArea is a custom software development company with 17 years experience in the industry. It provides a full cycle of the desktop, web, and mobile application development, testing and QA services.


QArea's offices are located in the US and Europe with 4 R&D centers in Eastern Europe. QArea provides a full cycle of the desktop, web, and mobile application development, testing and QA services.

The Company stands for client-oriented services with a well-rounded approach. The team of 250+ qualified engineers specializing in Golang, Python, Java, JavaScript, C++ and other most demanded programming languages stimulate the development of truly unique customized products. QArea is a winner of numerous industry awards, named as a TOP outsourcing company in 2017. World's biggest brands like Microsoft, Skype, HuffPost and more intrust QArea to build software solutions united by the high standards of quality.

Contacts

 qarea.com
 contact@qarea.com

 facebook.com/qarea.inc
 twitter.com/QArea
 linkedin.com/company/qarea/

 +1 310 388 93 34
+41 43 508 07 94
+38 057 763 6024