Qulix

How to move to the microservices architecture?

qulix.com

Disclaimer

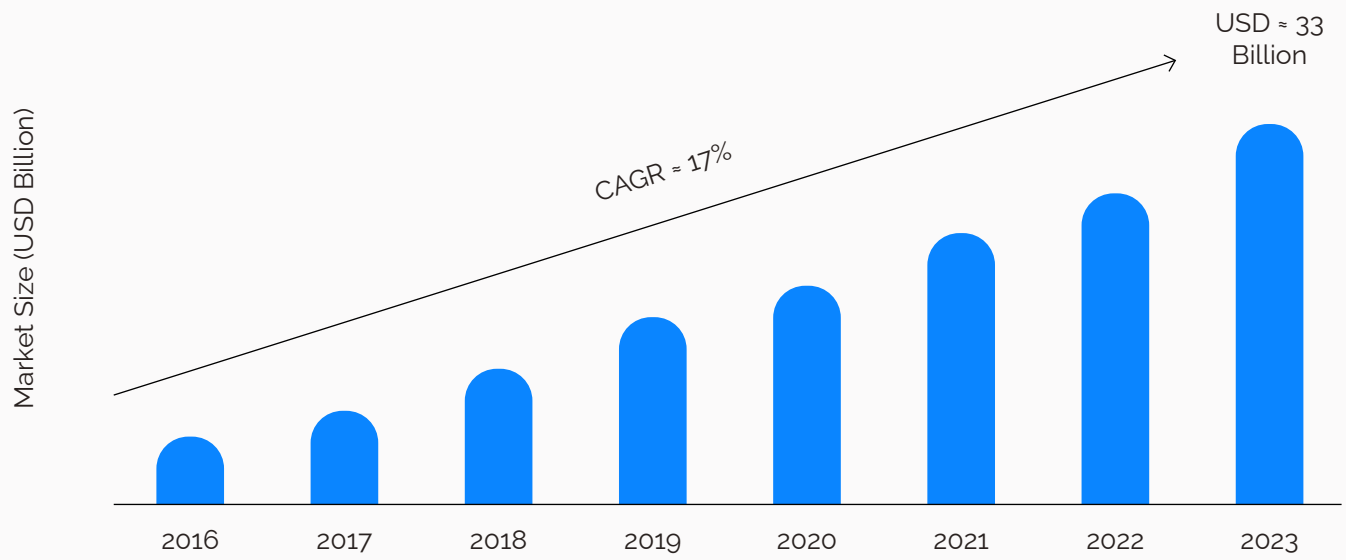
The information shared in this whitepaper does not provide exhaustive data on the topic and does not form a basis for any contract for the user of the information. The primary purpose of this document is of educational nature so that the readers can make better informed decisions regarding the subject matter of this whitepaper.

For online publications, please include a reference link to the original whitepaper.

Table of Contents

Introduction

Microservices have recently gained much momentum. It is the architectural style of choice for more than 80% of the companies worldwide, while its market share is forecast to continue growing. It comes as no surprise, as microservices-based architecture ensures faster time to market, higher system resilience, improves scalability, empowers enterprises to use the right tooling for the right task and much more.



Microservices architecture market research report global forecast 2023

The story of microservices begins in the 1970-s from object-oriented model, proceeding with actor model and, finally, service-oriented architecture (SOA). Through careful selection and preservation of the best features of its three predecessors, microservices were born and quickly won over good old monolith architecture as microservices offered a whole array of broad opportunities that monolith lacked (rapid development, scalability, quick fixes and more).

Now, as the concept has stood the test of time, why not consider implementing microservices to help your solution prosper? This whitepaper contains 6 Sections that describe in detail how to do it right.

Section 1 will help you **articulate properly** your **system's faults** that you are planning to solve using microservices architecture. In Section 2 we provide **possible pitfalls** to avoid. They vary depending on the industry and the company needs. The **decoupling methods**,

their **pros and cons** as well as **application criteria** are stated in Section 3. Section 4 offers a **decoupling roadmap** that you can follow as a guide. In Section 5, you can find helpful tips on **team structure, microservices-specific qualifications, skills and expertise**. Section 6 **summarizes** on the topic and provides a **final recap of the NTHs** for the microservices adopters.

Thus, this whitepaper answers all the typical questions that your project team may face down the road to the microservice architecture.

Why Do Companies Migrate to Microservices?

This Section provides an overview of the topic. It contains basic definitions, problem statement and the high-level solution description.

What is 'decoupling into microservices'?

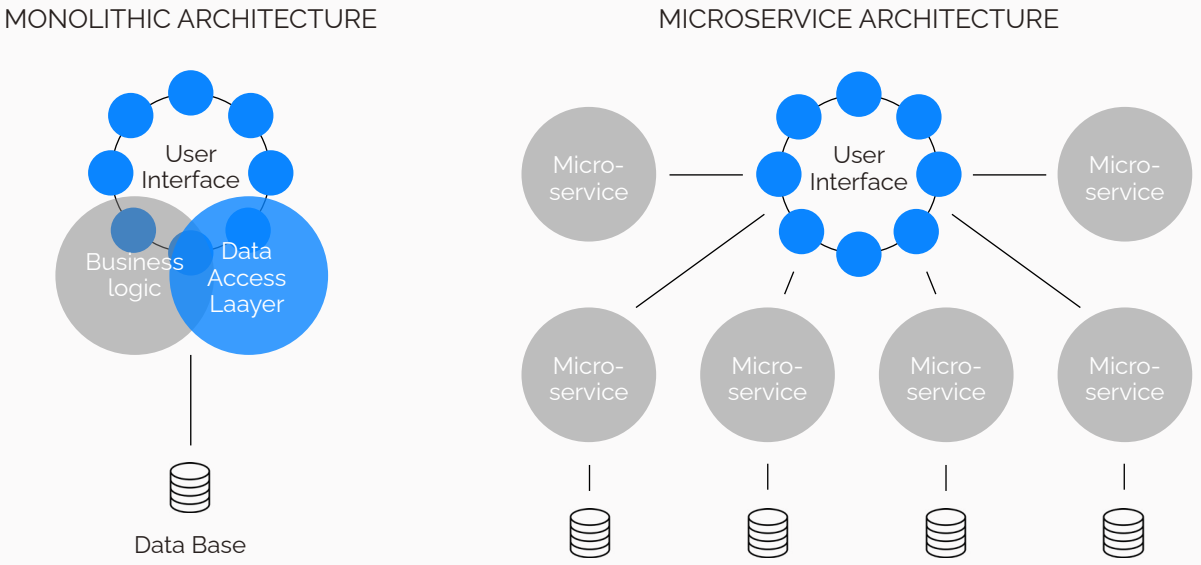
Every company's migration to microservices journey starts with the basic definitions, which are *decoupling* and *microservices*.

Decoupling is a process of dividing a single structure into several smaller ones (a monolith into microservices).

Microservices in turn is an architectural style where an application consists of an array of loosely coupled and independently deployable services usually organized around one business capability.

Through the decoupling into microservices process, a monolithic system is divided into a set of light-weighted loosely coupled microservices that run almost autonomously.

Through the decoupling process a monolithic system is divided into a set of light-weighted loosely coupled microservices that run almost autonomously.



The picture above shows a monolithic system on the left side and the same system decoupled into microservices on the right side.

A monolithic architecture is schematically shown as a structure where multiple modules share a single code base and are usually tightly interconnected. All the elements of a monolithic architecture share the same database.

On the contrary, in a microservice-based architecture the services run autonomously, have separate back-end, front-end and database layer.

Why do companies choose microservices?

Companies mainly choose the microservices architecture to tackle the below listed issues:

- **Lack of dynamic development**

Large market players turn to microservices to fight their lack of dynamic development. Often their comprehensive structure and slow internal processes (legal and security checks, long and complex budgeting) make complex system development a never-ending story.

- **Slow time-to-market**

Making changes to digital channels (and other systems) takes immense resources and time to implement, which results in slow time-to-market for new functionality and unresponsiveness to the market needs.

- **Slow technological advancement**

A lot of relatively small companies are winning quite a huge market chunk as they are simply quick to innovate (financial start-ups, for example). Large companies, unfortunately, lack this strategic advantage being tightly bound to legacy systems and technologies.

Given the problems stated above, microservices architecture is among the solutions that can help have a fresh start. This approach proves efficient for the systems that need to change dynamically in time due to loose coupling of the services, which enables high flexibility. That's why microservices fit nicely to make software systems customer-centric and responsive to the market challenges.

Microservices fit nicely to make software systems customer-centric and responsive to the market challenges.

Now, let's run through the advantages that microservices bring in more detail.

- **Shorter time to market**

In a monolithic application, new features used to come out at intervals that could be significant at times (up to 6 months). In a microservice application, new features come out independently of each other, which means the users see great updates regularly and are getting more loyal to the brand.

- **Faster regression testing**

When introducing changes into a monolithic application, often the whole system should be retested due to multiple interdependencies that are not obvious sometimes. One cannot predict how changes will affect other components of the monolithic system. In contrast to the monolithic application, microservices are autonomous components. Upon changing a microservice, only this specific component and its integration should be retested, and regression testing no longer takes ages.

- **Higher system reliability**

During the system upgrade, you do not have to shut your system down. Just turn off the selected feature for a minute for the microservice to update and then put it back into play. The rest of the system will not be affected by the change (Zero Downtime service).

- **Agile transformation of the enterprise processes**

The new trend is to establish several feature teams to deliver a new system more effectively (a feature per team). Microservices work out better with feature teams than a monolith, as this architectural style implies decentralization and autonomy. In its turn, Agile methodology is the best match for microservices-based system delivery (perfect for small teams, promotes dynamic advancement). Thus, microservices can help those companies who want to make their processes more agile.

- **Small teams**

The Agile methodology allows dividing project team into several small service-teams delivering new features independently of each other, yet sticking to the same work ethic and corporate culture.

- **Fault isolation**

Due to loose coupling between services, fault stays within the service where it was formed and comes no further, so other services stay unaffected and up and running.

- **Scalability**

Microservices architecture allows easily scaling up and down your system on demand.

- **Technology agnostic**

As services run independently, the team may try out virtually any technologies that suit the given tasks best, with few limitations.

Possible Pitfalls

Although only under 10% of companies consider their attempt to migrate to microservices a failure, microservices may be challenging for new adopters.

Below we enumerate some of the possible pitfalls, though every company may face its own unique difficulties.

---

1. Feature teams

The major pitfalls during a decoupling project may not be caused by the technologies or a novel architecture. Rather, it is the internal workflows and teamwork that may be a stumbling block. Building vertical teams or so-called feature teams is a challenge that your company must learn to cope with to run a microservice-based architecture seamlessly.

Feature teams may be a crucial factor for the success of a microservice-based system. Please contact us for some recommendations on how to build feature teams and run them efficiently.

2. Monolithic back-end

According to our experience as a custom software development company, a back-end of almost every relatively large company is usually a set of interdependent monoliths. Those are often legacy systems. As a rule, to implement a new service/feature requires making changes to all of its systems. Even if the team builds microservices on top of them, this will not make the resulting system dynamic: a microservice-based front-end will be held back by the monolithic back-end. In the end the company cannot reduce time-to-market and still has to wait for months to deliver a new feature to the users.

3. Slow approval process

The feature may be ready to be run only to stumble into routine operations that precede the launch (e.g. testing, security checks, legal consultations, etc.). If these routine procedures aren't approached consistently beforehand, the feature may result deadlocked for weeks or even months instead of bringing revenue and clients.

4. Security issues

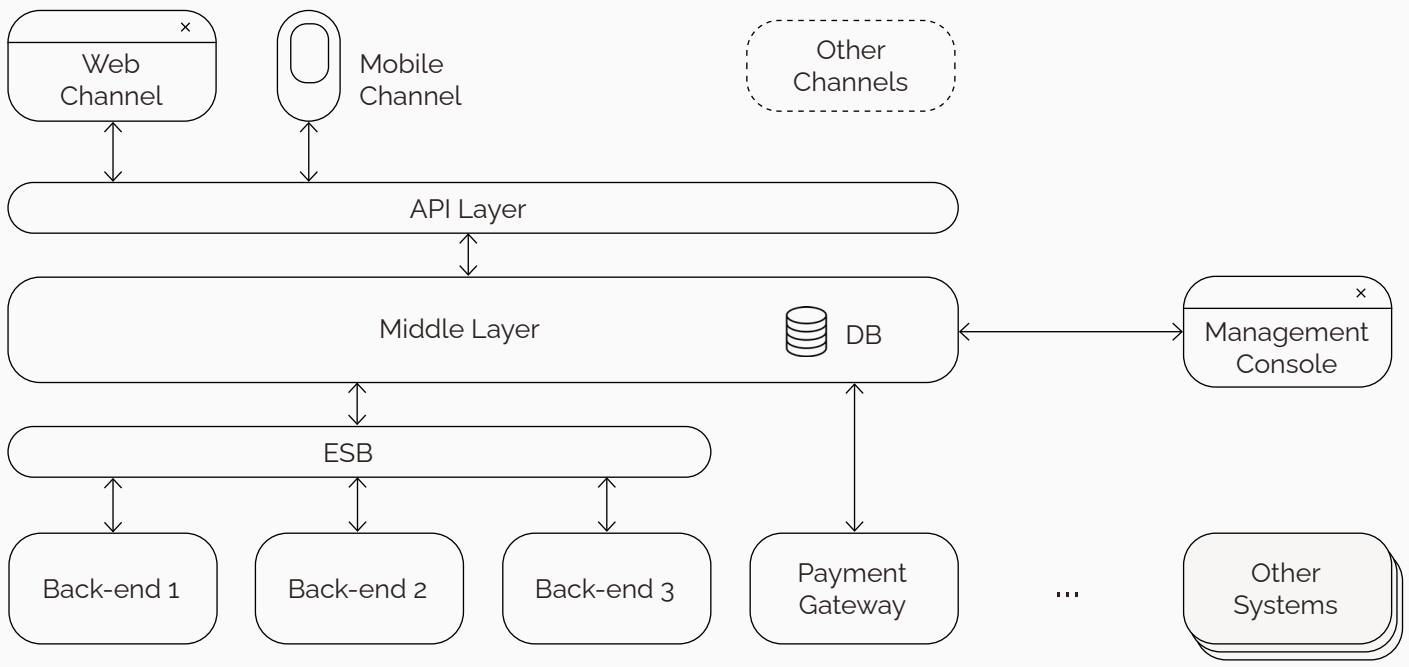
Security unit may also be a blocker on your way to a microservice-based architecture. They may have their own viewpoint regarding Kubernetes and CI/CD processes and whether they are appropriate for your system. Without commitment from the start, security issues will bring the project to a halt.

3 WHAT's of Decoupling into Microservices

The next step on migration to microservices journey will be identifying the best candidates for migration. Besides, the proper migration scenario matters along with selecting the first migrant(s).

1. What to decouple?

Before diving into decoupling details, let's revise a typical architecture of a complex legacy system.



Legacy monolithic architecture has several distinguishing features.

- Monolithic middle layer;
- Several integrated backend systems;
- Monolithic front-end;
- Single database.

As you can see from the list above, this is a rather comprehensive system, so what to decouple as well as what to decouple first should be carefully considered. Each project is unique and each component of a system has its own unique decoupling strategy.

However, as a rule the system decoupling master schedule will include the following parts:

- **Web App decoupling**

For web apps (as well as mobile apps) consider microfrontends. This is a good option to maintain app flexibility and speed up development, testing and deployments.

Web applications are quite friendly to microfrontends since dynamic code loading is supported natively.

- **Mobile App decoupling**

Native applications are monoliths by their nature because delivery implies creation of single executable file (ipa, apk). Dynamic code loading is not widely supported or prohibited explicitly.

Mobile app decoupling may imply some difficulties for the teams that are not well-versed in the microservices-based architecture. Contact us for useful tips on how to do it right.

For both web apps and mobile apps there is a wide array of technologies allowing microfrontends implementation. Consider Angular, React, and Vue.js, just to name a few.

- **Middle layer decoupling**
- **Database decoupling**

2. What decoupling scenarios are out there?

When planning their microservices migration strategy, companies have two scenarios to follow. It can be either a one-time migration scenario a progressive approach (one-by-one migration), depending on your system needs.

During a decoupling project, companies can follow either a one-time migration scenario or choose a progressive approach (one-by-one migration).

Basically, the result is the same – the whole system runs in a new environment, under new rules. Yet, it is the project initiation that makes the two scenarios so different from each other.

a. One-time migration (SADT)

Under this scenario, a newly built system will fully replace the old one only when it's 100% complete.

Pros	Cons
✓ Deep system analysis	✗ Time-consuming
✓ Resources optimization (1-2 engineers at the project start)	
✓ Requires lower engineering skills	

Pros and cons of the one-time migration scenario

How the project starts:

1. A Solution Architect singles out the subsystems to be decoupled and their interfaces.
2. The SA creates an infrastructure plan for microservices and the skeleton for the future system.
3. The rest of the team joins in.

b. One-by-one migration (Agile)

This is a solution for the systems under active development. The system parts are replaced one by one.

How the project starts:

1. Quick analysis (SA, BA);
2. Infrastructural project;
3. Decoupling iteration (1-2 loosely coupled services);
4. Next iteration and next migrant till the whole system migrates to the new infrastructure.

Pros	Cons
✓ "Quick win" approach	✗ Costly (the whole team joins almost from the start)
✓ New clients and revenues come faster	✗ Requires higher engineering skills

Pros and cons of the one-by-one migration scenario

3. What system parts can be the first migrants?

As a rule, every decoupling project starts from the best candidates for decoupling. This can be one or more relatively independent subsystems.

- **Stateless components decoupling**

Every system includes both stateless and stateful components.

Stateless components do not have brittle dependencies with other system parts. They are system parts that render the same data, unlike stateful components which keep track of the changing data.

Stateless components do not have brittle dependencies and render the same data. They are good candidates to migrate first.

Thus, stateless components can migrate easily without affecting negatively the whole system and are ideal candidates to start decoupling with.

- **Functional decoupling**

Under the functional decoupling scenario, first migrants are those parts of the system which work with external services. These are relatively autonomous modules that cannot affect the application logic (gateways, payment modules, authentication server, notification module, etc.).

Gateways, payment modules, authentication server, notification module, etc. suit greatly for the migration purposes as they cannot affect the application logic.

Decoupling Roadmap

Down the road to the microservices architecture, companies should follow a well-defined roadmap with clear milestones to enable consistent and seamless system migration.

Below is the time-tested roadmap with basic milestones that Qulix recommends for the microservices migration projects.

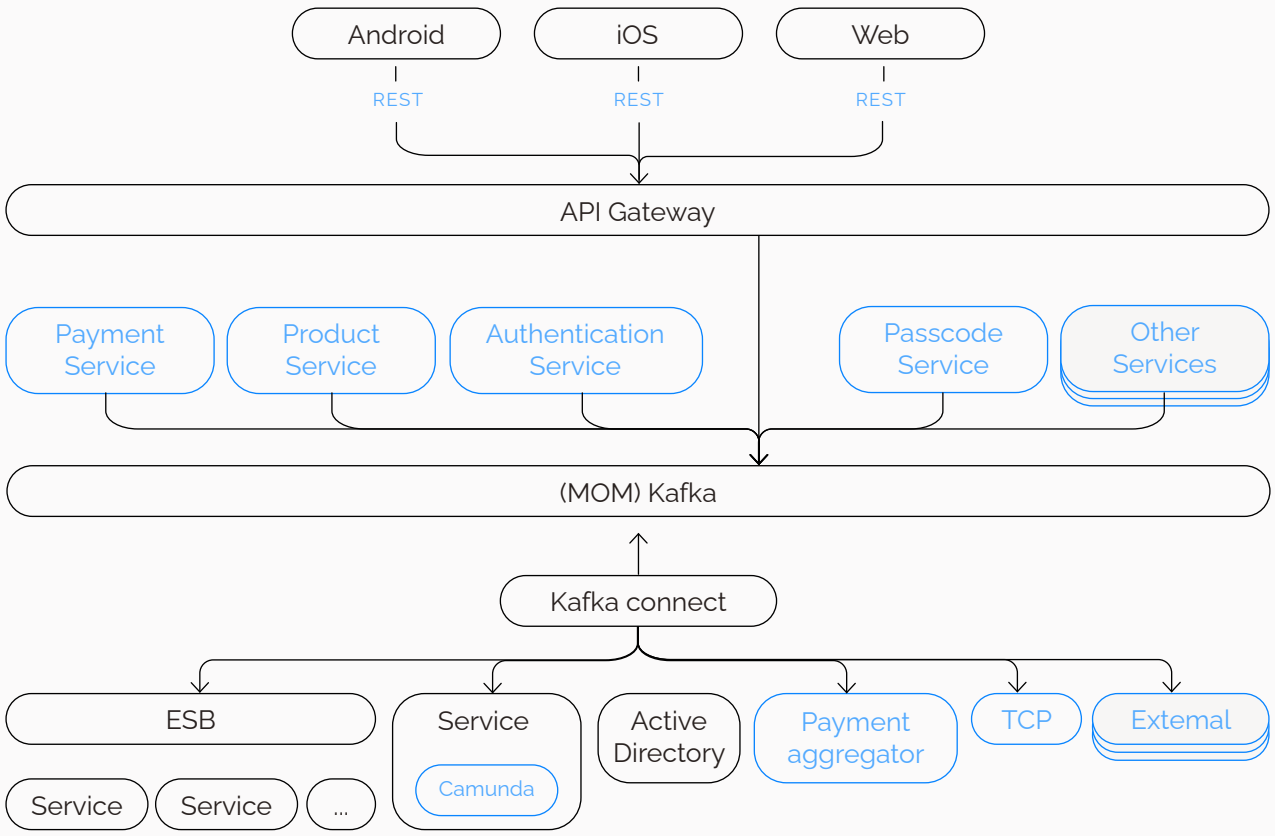
---

1. Run system audit and revise your architecture

Run the system audit or hire an IT-vendor to do it for you. Identify the candidate(s) to migrate first (stateless services or functional decoupling). Revise your current architecture.

2. Build new architecture

Together with your vendor build and approve new architecture. See below for an example:



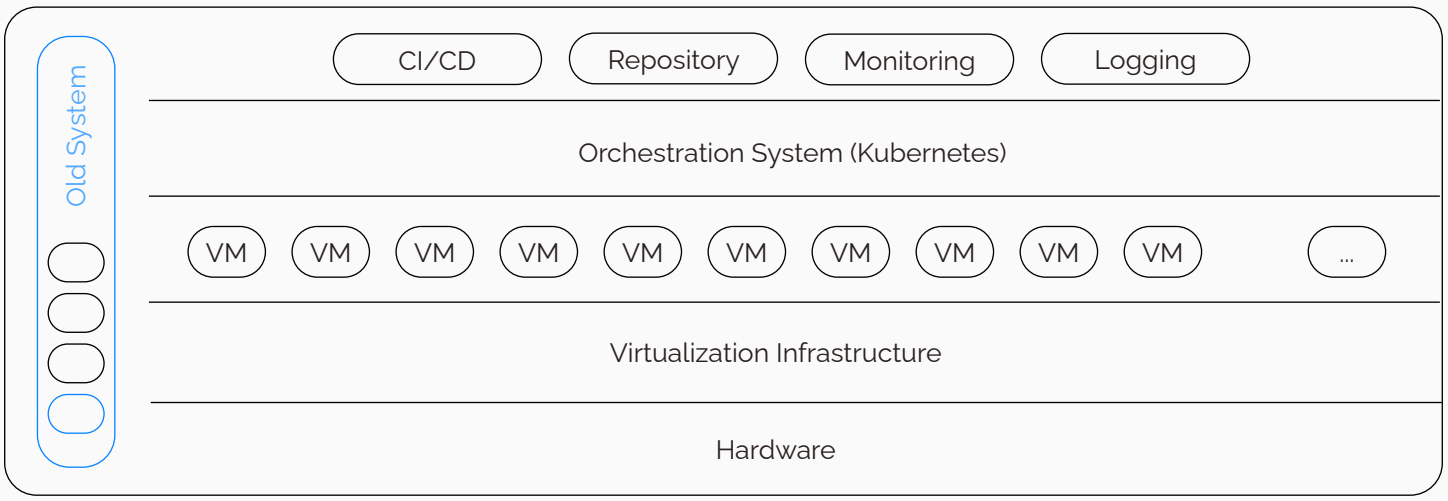
3. Build the project infrastructure

a. Virtualization infrastructure – adopt it to your system needs and prepare for the launch;

b. Container orchestration system – choose the one that suits you best (Kubernetes, Azure Kubernetes Service, Google Kubernetes Engine, etc.);

c. CI/CD tooling – select the one for you from a wide array of CI/CD tooling available out there (Jenkins, CircleCI, AWS CodeBuild, Azure DevOps, Atlassian Bamboo, Travis CI, etc.);

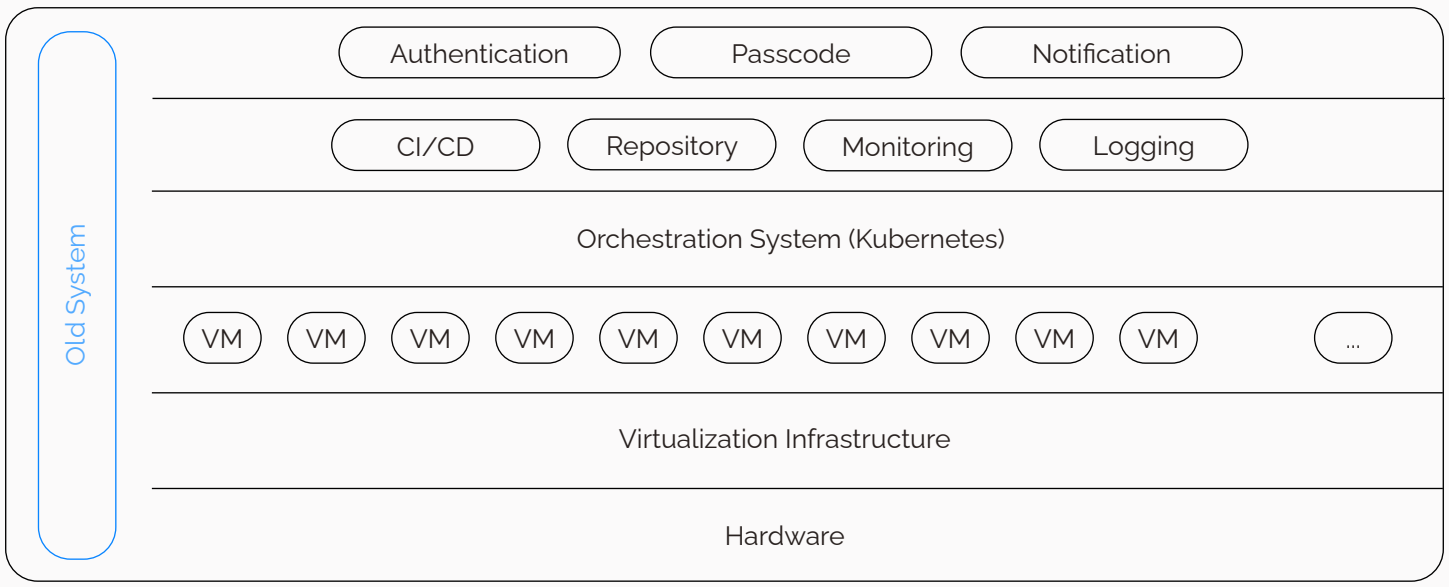
d. Prepare internal artifact repository.



4. First migrants

a. Put them into the new infrastructure;

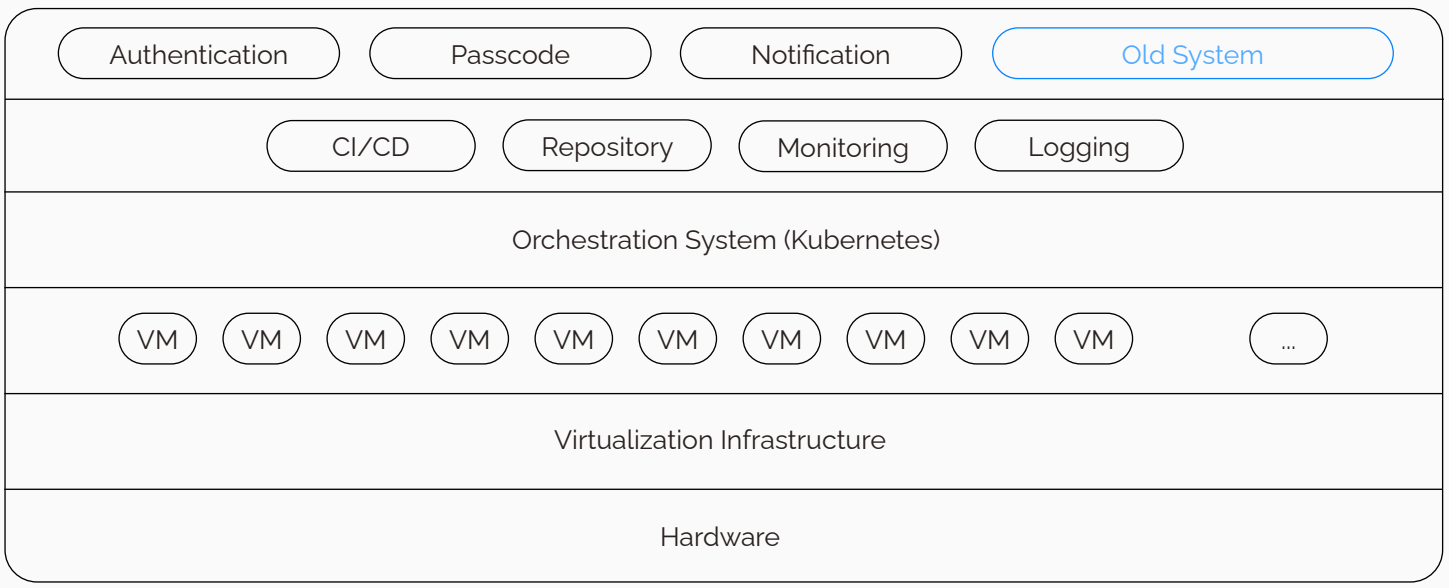
b. Establish the CI/CD processes for them (sometimes unique for each microservice).



5. Put the existing system into the new infrastructure

a. Develop provisioning scripts for the new architecture (message broker, etc.);

b. Instrument first migrants with messaging communication channel (message broker).



6. Proceed with other services

a. Approve the set of services that qualify for functional decoupling;

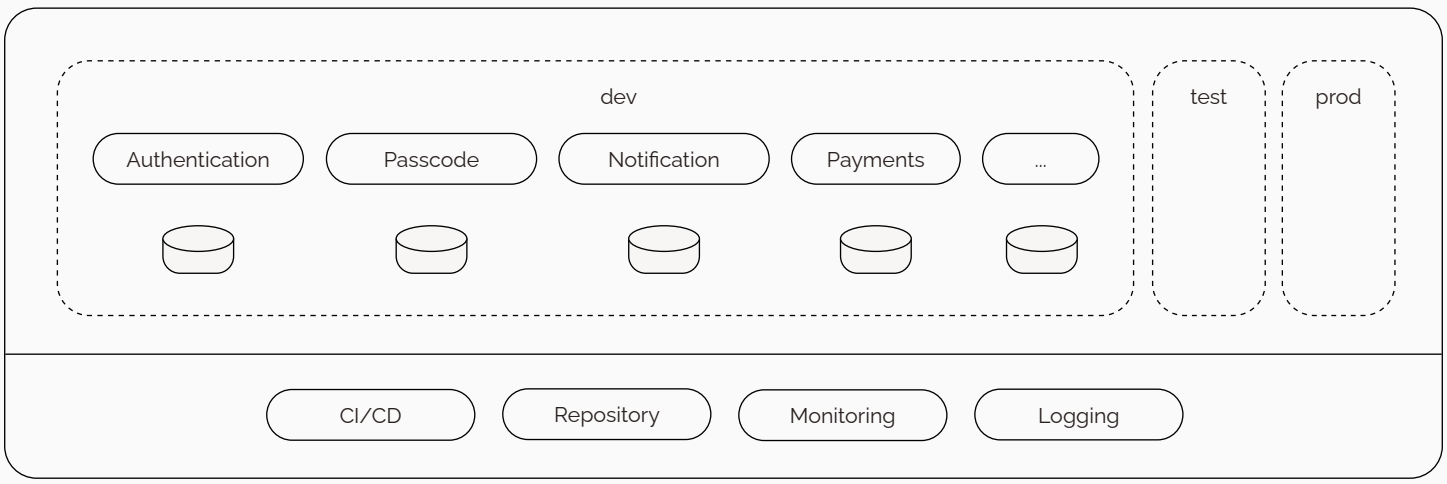
b. Build services dependency graph;

c. Get ready with the decoupling plan. Priority is based on service business value and its external dependencies;

d. Consider decoupling of leaf services first;

e. Develop new features with new paradigm in mind.

7. Get to the target microservices-based architecture



Team Expertise

As long as companies are done with the preliminary stages described above, they set out looking for the contractors/vendors to carry out the endeavor. Yet, the proper expertise on the contractor's part does not guarantee the project success. What else should there be?

Expertise on the customer's part

An important prerequisite for a successful migration is **deep vendor's expertise** and a **substantial understanding of an issue on the customer's part**. Why is the latter important?

The project race does not end at the production phase, as a long and demanding maintenance phase is waiting ahead after the microservices-based system goes live. To put it simply, now the customer has a fully-featured in-house IT project that requires maintenance and support 24/7/365.

In such cases often the customer is working closely with the vendor on the project to support the system. So, the proper in-house maintenance expertise is a must-have. This is especially true, if the customer is planning to run the project on their own in future without the vendor's involvement.

Customers should grow their in-house microservices expertise, especially if they are planning to run the project on their own in future.

The tools and technologies that customers have to master to be able to support their systems include:

Cloud	MS Azure, AWS (Amazon Web Services), GCP (Google Cloud Platform)
Containers (Orchestration)	Kubernetes, Docker Swarm, MS Service Fabric; Cloud analogs: Azure AKS, Azure Service Fabric, AWS EKS, Google GKE etc.
Cloud	Apache Kafka, Rabbit MQ, Active MQ, MS SQL SSBS; Cloud Analogs: AWS SQS, Azure Service Bus, GCP Pub/Sub etc.
API Gateways	Ocelot, Tyk, Nginx; Cloud: AWS API Gateway, Azure API Management, GCP API Gateway
Health monitoring, Logging	Prometheus / Grafana, Elastic stack (Elasticsearch+Kebana), Zabbix, Logstash
DevOps tools	Ansible, Helm
CI/CD tools	Jenkins, Thoughtworks GoCD, TeamCity; Cloud: Azure DevOps

Tools and technologies to support microservices-based systems

Expertise on the vendor's part

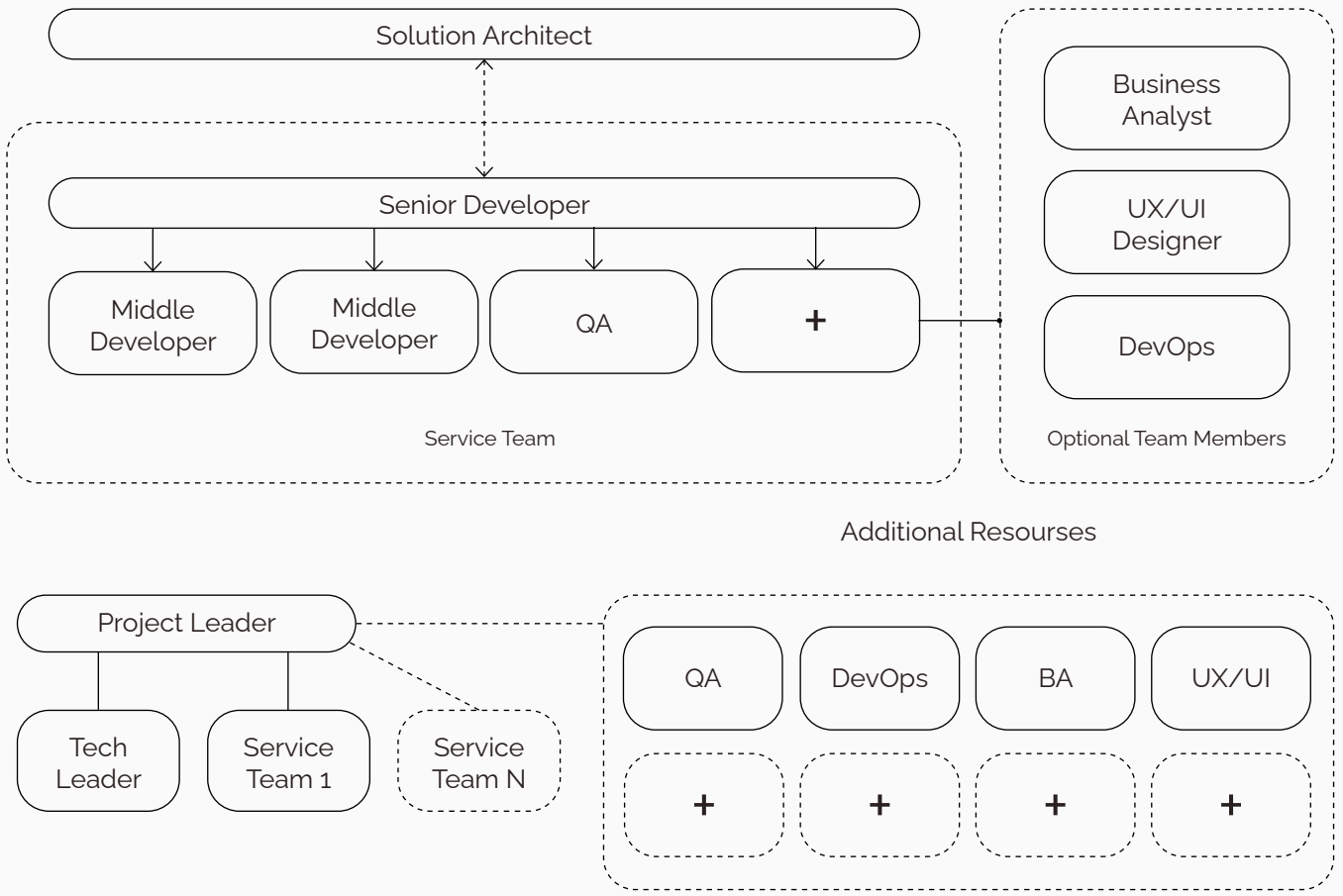
The vendor's team should be well-versed in your domain specifics and have microservices skills and qualifications.

Microservices skills and qualifications may be as follows:

- Microservice architecture (common architectural patterns);
- Domain Driven Design;
- DevOps tooling and practices;
- Containers and Container Orchestration;
- Security in a microservices-based architecture;
- Microservices testing (unit testing, API testing, end-to-end testing);
- CI/CD tooling and practices;
- Soft skills (great teamwork, commitment to result, individual/shared responsibility, etc.).

Vendor's team structure

Decoupling projects usually require quite typical team structure (however, some positions may vary from project to project).



Vendor's team structure

Solution Architect

Software architecture design, technical assessment and improvement of the existing architecture, technical mentoring for the team.

Developers

The team may include both back-end and front-end developers, in some cases - full-stack developers.

- **Front-end developers**

App front layer implementation, participation in API design, micro-frontends implementation, responsive and mobile design, cross-browser development.

- **Back-end/Full-stack developers**

Essential front-end (for full-stack developers), databases, caching, integration APIs, hosting environment management.

QA engineer

Manual software testing, Test Automation, thorough system analysis and errors/bugs reporting.

System Analyst

Functional and non-functional requirements, identification of integration points, specification of integration part of business features, description of business processes and business logic.

UI/UX designer

CX design, user interaction flows design, navigation design (user journeys), Target Audience analysis and segmentation, page structure development, emotional bond between a brand and the TA.

DevOps Engineer

System build and deployment, maintenance, hosting management, implementation of CI/CD processes and their automation, software monitoring and emergency response.

Conclusion

A quick summary on the topic and Nice-To-Have's before you go.

Microservices-based architecture brings enormous benefits to the businesses. Along with **faster time to market** and **customer-centric mindset**, microservices adopters enjoy **agility, system reliability** and **resistance**, and more. Your business can join the game, too!

Yet, before you set off with your first microservices endeavor, go through a final recap of the decoupling highlights that Qulix Systems outlined in this whitepaper. These NTHs will help you safeguard your system from possible setbacks.

- **Clearly articulated problem**

Remember to run a detailed analysis to find what your current architecture faults are and how you are going to fix them with microservices.

- **A list of possible pitfalls**

From vertical teams to approval workflows, every possible detail that may set you back must be captured and debated on.

- **A list of the first migrants and a decoupling scenario**

Run a thorough structural analysis of the system to identify the components that qualify for migration. Choose the scenario that matches your system needs.

- **A well-defined roadmap**

Come up with your own unique step-by-step guidance or use ours to map your decoupling route and set the milestones.

- **In-house expertise**

Consistently grow your relevant in-house expertise to be able to support and maintain the microservice-based system after it goes live.

- **Long-standing partnerships**

It's worth mentioning that this architectural style has gained its well-deserved prominence to a great extent due to the deep expertise and commitment of the industry experts. That is why our last recommendation for you will be to look for the long-standing partnerships with the vendors who consider every risk and benefit that such transition may imply for your system.

With all the above NTHs at hand, backed by the rich experience of your technology partner, your migration to the microservices-based architecture will be smooth and seamless.

About Qulix

Custom Software Development Company Operating Globally

Qulix is an international custom software development company delivering high-quality software solutions. Since 2000, we have been rendering top-notch development and QA services to our 200+ clients from all over the world. Our expertise covers all the stages of SDLC, which includes concept design, architecture design, code development, quality assurance, support and more.

Qulix delivers turn-key and custom software projects for banking, finance, insurance, multimedia, IoT and other areas. The list of the services that we offer includes backend and web apps development, mobile and cloud apps development, QA services, UI/UX design, and DevOps.

Find more of our best microservices practices by visiting our blog or website.

For any further questions, please contact us. We'll be happy to share with you our insights on the topic to help your business grow.

tel: +44 151 528 8015 **email:** request@qulix.com

UK:
Oakwood, Dunstan Lane, Burton, Neston,
Cheshire, United Kingdom, CH64 8TQ

Poland:
Braniborska 40, 53–680
Wrocław, Poland

Uzbekistan:
130 Mustakillik Avenue, Mirzo-Ulugbek
District, Tashkent, Uzbekistan, 100077