blueguava

THE DECISION MAKER'S GUIDE FOR COMPLETE CLOUD TRANSFORMATION

Taking your business to the Cloud with an efficient Go2Cloud transformation strategy

Contents

Foreword

Cloud-based solutions are both the present and future of digital businesses. The cloud is proving to be one of the best alternatives for hosting data, services, and operations. The frequent technological advances and the constant maintenance of a cloud's infrastructure ensure that it will dominate the realm of digitized businesses for years to come.

In the first half of this eBook, we will discuss **how to approach cloud transformation** by enacting a solid business strategy that lays the groundwork for the multilayered implementation of migrating and/or constructing a brand new, cloud-native microservices architecture. In the second half, **we will look at specific approaches, models, strategies, principles, and techniques** to ensure the transformation's success.

Discussing such a broad topic is easier if there is a specific example we can use to explain particular solutions. That is why we decided to describe cloud transformation through the lenses of Amazon Web Services (AWS) — our partner in cloud solutions.

In our years of overseeing and aiding organizations that migrate to the cloud, we have found AWS to be the optimal cloud choice; few other options provide the broad scope of business needs that AWS has. As one of our hallmarks at Blue Guava is to render excellence in digital product development, video and data streaming, quality assurance, and of course, cloud transformation, we have found an alignment with AWS. They seek to offer cloud services of the highest quality to their customers, and we also want to extend those services to ours. For some of the solutions described here, we refer to AWS and use it and its features as examples.

We will examine cloud transformation from different perspectives and our approach will focus on the most significant areas, which are **Platform Security** and **Operation**. Nevertheless, other considerations like Business, HR, and Governance are briefly touched on.

One last thing before we take you along for this journey: we cannot emphasize enough that the best ROI and cost-efficiency rates within the cloud can only be attained if we commit to achieving a cloud-native solution. It is no small task, but if you want the most bang for your buck, undergoing a complete transformation with rearchitecting, refactoring, and more will be worth the effort and price.

Why is there resistance against cloud transformation?

Cloud transformation is one of the most sought-after solutions in the business world today.

Unfortunately, it is sometimes a scary concept for some developers and decision-makers. While professionals are concerned that cloud transformation is an overly complicated process, they must learn something new outside their current comfort zones — or perform significant reworking of their existing and proven systems. Decision-makers fret over plenty of other concerns.

For them, it is a cloud transformation's price, validity, and efficiency that can lead to sleepless nights. Decision-makers are afraid of making mistakes, especially if those mistakes will significantly cost the organization. They will be held responsible for a failed project that did not meet the desired goals.

Consequently, they decide to involve every other stakeholder (developers, tech experts) in the decision-making process. They want the entire team to approve and bear the responsibility together. It could be a good idea; however, if those professionals have their concerns (transformation is too complicated, requires too many changes, etc.), it can lead to a dilemma that stalls the project. Some will argue that data migration is too expensive, or that they will lose ownership of their data by taking it to the cloud, or simply that it will not cut costs, but only transform their CAPEX of acquiring, upgrading, and maintaining their system's components to the OPEX of running operations in the cloud.

While these concerns are all grounded in reality, and there are some unfortunate cases and bad examples that can amplify such fears, fortunately, the vast majority of cloud transformations are implemented successfully and result in positive outcomes with significant benefits.

We have consulted, aided, and overseen enough highly successful cloud transformations to know that taking your business to the cloud is worth going through the challenges of adapting to a new infrastructure.

What are the alternatives?

Before considering cloud transformation, let's take a look at some on-premises alternatives to cloud solutions.

In this category, we chose two options that we consider potentially viable for circumventing cloud transformation. The question is, however, do they truly provide the same — or mostly the same — advantages without unnecessary drawbacks or difficulties?

First, let's take a look at on-premises enterprise software solutions designed with the specific intent of helping business organizations perform basic tasks such as workforce support, business transactions, and internal services and processes. It is a rather popular alternative with numerous advantages. It can serve a company as a secure and efficient enterprise-wide private cloud installed on the company's home turf. However, to get the most out of it, it's essential to consider the following principles at all times:

Avoiding hardware lock-in

To make use of hardware investments already implemented in their organization and manage hardware resources in the future, IT managers should prefer a hardware-agnostic solution and avoid one that specifies strict hardware requirements.

Avoiding vendor lock-in

A responsible IT manager will choose an on-prem platform that can be easily replaced in the future or bolstered by another platform without reworking all of the organization's code.

DevOps

To let an organization maintain its competitive edge and support rapid application delivery across all platforms, any on-prem platform should consistently support DevOps automation with the other platforms (whether on the public cloud or on-prem) in use by the organization.

Compute services

Computing power is the most basic service a developer needs from any cloud platform. An on-prem platform should support different compute instances to meet the developers' various needs.

Self-service provisioning

For developers, a real benefit of the cloud is the ability to "just spin up" compute instances, databases, storage, and other services on demand without having to go through a lengthy approval process with IT. To keep developers agile and innovative, any on-prem platform adopted must support the same self-service, on-demand paradigm of deploying service.

Automatic resource lifecycle management

To keep IT free of mundane, day-to-day tasks that become more burdensome as an enterprise scales, an enterprise platform must automate operations such as setup, updates, patching, and backups.

Consolidated monitoring and alerts

An on-prem cloud platform should provide a consolidated view of the different resources and services it provisions so IT can respond to bottlenecks and usage spikes quickly and easily.

Support for public cloud APIs

The private cloud is server-based. You are provisioning servers and virtual machines, configuring these environments, and then deploying and running apps on these servers and VMs.

In the public cloud, you are driving infrastructure with APIs. The public cloud's API-driven nature enables its incredible adoption and most significant benefit — near-instant and infinite capacity through developer self-service.

On the other hand, going into the world of microservices certainly seems like an exciting challenge. Organizations can optimize their custom code and workflow for a streamlined code execution process. However, this will require a great deal of effort and attention from all involved teams. Besides the need for increased project management, it will be challenging to cover all scenarios emerging from various business needs.

Let's see a concise rundown of the advantages and disadvantages that come with a microservices approach.

Pros

Greater agility

Improved scalability

Increased flexibility due to communication being distributed over the network

Better **fault tolerance** due to isolated services

Cloud readiness

A better **time to market** (faster and easier development cycles)

Improved debugging capabilities

Platform- and language agnostic services

Cons

Requires **DevOps culture** and heightened collaboration between development teams.

Reduced **performance** (microservices must communicate, resulting in increased network latency)

Requests traveling between modules must be handled carefully (might require extra code to avoid disruption)

More potential security issues (distributed communication can fall apart more easily)

Slower and more complicated to implement for small companies that need to create and iterate quickly

More difficult to maintain the network, requiring more load balancing

Testing and monitoring become more difficult due to the architecture's complexity

Refactoring an app across a multitude of services is much more challenging

We describe most of the specific features, benefits, and drawbacks in later sections.

In general, there are always considerations that can help determine the option that best serves our organization. What we found in the case of these two on-prem solutions boils down to three critical concerns when directly compared with the cloud:

1 | Elastic scalability[1] does not exist in on-premises solutions.

2 | Most of the currently available technology does not support horizontal scaling.

3 | Almost all of the currently available technologies used for infrastructure software or microservices have critical limitations, making them hard to extend with custom solutions.

[1] The purpose of matching the allocated resources with the overall number of resources needed at a given point in time, combined with the scalability of tracking and adapting to the needs of the application within the confines of the infrastructure, via statically adding or removing resources to meet application demands.

The important questions

Now that we have discussed potential alternatives and seen their limitations, we can talk about the essential questions decision-makers and relevant stakeholders must look at when deciding on cloud transformation.

The best question to ask when getting started is this: **how can we measure the system we currently operate and determine if it's suitable for our business goals?**

At this stage, identifying blind spots in the system and examining subcomponents can help discover just how good the overall quality and efficiency are.

The next batch of questions should revolve around the means and costs of operation, maintenance, and optimization. As you sit down together as a decision-making unit, you have to make sure a consensus is reached on the following agenda items:

- **What is the level of optimization** that we want to achieve, and when do we get there?
- What are the **viable options for increasing** the level of optimization?
- How much does it currently **cost** us to keep the system up and running (Underused on-premises servers running 24/7 can and will generate wasteful costs)?
- How much does **maintaining and supporting** the system cost us?
- Do we have **enough resources available** to support the next big scheduled event or update? If yes, how much will it cost us?
- Have we taken all **hidden, non-direct cost elements** (e.g. guards on Domain Controllers (DCs), travel costs, etc.) into account

We highly recommend coming to this discussion as prepared as you possibly can be.

What do we mean by that? The total cost of ownership (TCO) calculated forecast of running and maintaining the current system projected to the next phases of the design or product lifetime, estimated costs of planned changes or shifting to a different service model (e.g., complete cloud transformation), with cost optimization methods already on the table.

Why Choose Cloud?

A complete cloud transformation yields plenty of benefits for any organization. With a cloud-based solution, a company can turn upfront expenses into variable expenses, reduce costs by doing away with running and maintaining data centers, increase operational speed and agility, and auto-scale without additional hardware requirements.

Within a cloud solution, we don't have to worry about **guessing server capacities** anymore. Instead, our business can go global in minutes without many of the requirements and issues that arise from having to fire up on-prem hardware and configure the software that will run it.

Cloud is essentially your best tool to provide your organization with massive **economies of scale**; you will save on many costs by making both production and operation more efficient and perform at higher, optimal levels.

You will no longer have to operate server farms and pay the enormous costs of maintaining them. Instead, you will gain access to a virtual cluster of computers that is available to you 24/7/365, offering the triumvirate of cloud computing models **(IaaS — PaaS — SaaS)** in a single package.

What makes the cloud so stable and reliable compared to an in-house solution? The fact that the entire infrastructure rests on firm foundations built by a much larger body of **highly experienced professionals** who maintain and upgrade those foundations — as well as the services built upon them.

Furthermore, as the security component is always a vital element to consider in any solution, you can rest easy knowing that with high compliances and other services available in the cloud, your data and business will always be safe and secure up there.

This high level of attention and quality extends to resilience as well. With the right configuration of DCs, **Availability Zones**, and **Regions**, cloud-based services can achieve even 11 9's of durability — something that would be impossible at worst or hardly feasible at best within an on-premises environment.

In summary, these features and benefits all boil down to one major takeaway: with cloud supporting your business, you can **expand your company's coverage regionally** or even **globally in minutes**, giving the organization the means to leave global footprints across the market.

Time to Switch

How can you perform a complete cloud transformation? What are your options for it?

In general, there are six unique approaches for organizations that want to migrate their business to the cloud. These migration strategies — also called the "6 R's" by AWS — are the following:

1 | **Rehosting**
When an organization's cloud migration must be scaled quickly to meet a new business case, rehosting its applications proves to be the ideal solution. In some cases, even without implementing any cloud optimizations, organizations can save almost 30% of their costs by rehosting. It's worth keeping in mind that applications are more comfortable optimizing if they are already running in the cloud. That includes the application itself, the data, and the traffic. In AWS, rehosting can be automated with CloudEndure Migration or AWS VM Import/Export tools.

2 | **Replatforming**
When you want to change the core architecture of an application, but still seek to optimize its performance to achieve some business goals, replatforming is probably the best solution for you. Reducing the amount of time spent on database instance management can be accomplished by migrating to a database-as-a-service platform instead.

3 | **Repurchasing**
Simply replace the product: let go of the old and buy a new one. In most cases where product replacement occurs, organizations transition to a SaaS platform (e.g., moving a CRM to salesforce.com).

4 | **Refactoring / Re-architecting**
In case there's an emerging business need to expand features, add scale, and improve performance, achieving such objectives will become more and more difficult in the application's existing environment. When reimagining how the application is architected and developed, using cloud-native features will lead to an optimal outcome. If you're considering migrating from a monolithic architecture to a service-oriented or serverless one, then refactoring is your best option.

5 | **Retiring**

After exploring an environment to its fullest extent, it is viable to ask each functional area about application ownership. It could reveal some surprising things. According to AWS, about 10–20% of organizations' IT portfolios were found to be no longer useful and could simply be turned off. Retiring obsolete systems will cut costs, decrease the area that must be secured, and let you direct your team's attention to more critical tasks.

6 | **Retaining**

A golden rule in cloud transformation is only to migrate what makes sense for your business. As time passes and more of your data and systems are migrated to the cloud, the retained software percentage will shrink.

The table below summarizes the efficiency of the strategy from different viewpoints.

Strategy	Cost Savings	Operation Stability Improvement	Development Impact	Organization Impact
Rehosting				
Replatforming				
Retire				
Retain				
Repurchase				
Refactoring				

Low impact in cost optimization and operation.

Moderated impact in operation but give more cost!

High impact in operation, require more changed but it brings more cost saving and flexibility.

Not Available

Consideration Phase

The First Rule

The first rule of cloud transformation that you must know by heart is this: **understand your business logic**, and when fleshing out the strategy, refer to it as many times as possible. This will help throughout the transformation, as your organization will have enough self-awareness to know where and what should be improved due to this journey.

Furthermore, **knowing the environmental limitations** of that business logic is also a crucial requirement. Some areas simply cannot be changed or improved right away and must be revisited later on after the initial transformation is completed (more about that in the next section).

After the business logic and the environmental limitations have been built into it, we can create the rest of the strategy. Learning from industry best practices — and applying them straight away — is a good start. However, in the end, the most critical step is to **have a flexible, future-proof strategy** in hand that is aligned with the organization's goals and initiatives.

A crucial step in creating any cloud strategy is to **find the correct focus for implementing the transformation**. And what we suggest every time is to choose the core business as the center point the changes will bloom around as the implementation begins. This means that aside from your business's core functions, no other side project or gig or its related material should be involved in the strategizing. No additional analytics, logging, or reporting should be part of the new system's core if the transformation's goal has little to do with those areas.

As such, the best practice we recommend is to find one or two workloads that are relevant but not mission-critical to your business. Migrate those first and gain expertise along the way; at the end you will see the benefits of being in the cloud so you can replicate this on a higher migration scale. This is also a great way to secure the necessary management sponsorship and attention for subsequent, major transformation projects.

There's Always Room to Improve

In case you feel the strategy still has some blind spots, worry not! Letting perfection be the enemy of good only leads to unnecessary delays, preventing your organization from launching the implementation. Not committing ourselves to a rigorous, inflexible strategy that leaves little room for improvement will also make for a better transformation in general, giving us the means to adapt to unforeseen trends and handle challenges gingerly.

Below, we have collected some areas that must be considered and included in the overarching strategy. Still, if the specifics are fleshed out during the transformation, you will not lose a single day to delay while still retaining the capacity to work them in mid-implementation:

Reducing code complexity

A never-ending process that will always result in fewer issues and a more streamlined process.

Simplifying functionality

The purpose of this iterative improvement is the same as with the reduction of code complexity. It is done with the intent of avoiding the overcomplication of system functionalities.

Heavy computing and data processing

Ensure that data preparation2 is handled and streamlined on a code level. This iteration should be combined with improving heavy data processing, which requires the construction of proper databases.

Data Processing

IT education today is unfortunately all too focused on teaching only real-time data processing. However, with the cloud being capable of both asynchronous and batch processing, your IT professionals will also have to learn these processes. Cloud transformation makes batch and asynchronous processing more viable. Adapting your system to workflows with new types of processing also takes time and iteration. Still, in the end, that extra effort will pay off as these new data processing types can lead to significant cost reductions.

Asynchronous vs. synchronous

Taking the ability to process asynchronously one step further will help streamline your system processes if you do away with synchronous operation during the transformation. Instead, focus on using asynchronous resource access and logging wherever possible.

---

[2] Data preparation is the manipulation (usually preprocessing) of raw data into a form that can readily and accurately be analyzed for business purposes.

Data storage location

During the transformation, you will have the opportunity to decide whether you want to store data in a local or a distributed cache. However, what's also essential is to let go of useless databases and API calls in the process.

Unused resources

The transformation yields an unparalleled opportunity to get rid of any unused resource that clutters your system as quickly as possible. Use it!

The Essentials

Over the years, we have overseen and aided many cloud transformation projects, and as you would expect, we had to solve some significant challenges and unforeseen issues. These lessons have been great for gaining experience and acquiring true mastery over the entire process.

Here are a few tips and tricks, as well as important considerations to keep in mind for the finish line:

1 | **System warmup**
If nothing happens at the system start, you are probably facing a cold start issue: a problem where the system launched or restarted and is not working at its regular operational settings. It usually lacks — or goes through a slow initializing — of internal objects, cache populating, or other booting subsystems. Furthermore, if multiple requests come in from the same resource, calling on a CPU-intensive function (network access, calculation, file reading), sync/single flight can solve it. With it, you can reduce the number of individual, simultaneous requests by consolidating them into a single one. After the process is complete, all those individual requests will share the result. Just keep in mind that it only helps with in-progress requests, and won't cache the result after the function call is complete.

2 | **Resource usage**
Any operation will cost resources such as time and CPU usage. If you encounter a problem where you are waiting for a resource response at an overly high CPU consumption of 99%, the chances are that an unlimited limit issue is the culprit. Services must be protected by introducing basic settings to limit functions.

3 | **Technological limitations**
Even though you will be using a virtual cluster of unrelated computers with your hardware, that hardware will still be the source of most of the interactions with the cloud. Knowing the technical capabilities and limitations of your hardware is the key to optimizing the best performance. For example, knowing the number of threads your hardware can perform processing on and reducing the size of the payload and responses will lead to a more balanced and smoother operation.

4 | **Debug and operation strategy**
Coming up with an efficient debug and operation strategy is crucial. It can help identify and describe a whole set of issues without involving a single developer or requiring code change

5 | **Automation**
If a system is properly automated in the cloud, there is less chance for (human) failures, enabling you to develop and operate at a faster speed.

What is the Ultimate Approach for Cloud Transformation?

Choosing the specific approach to cloud transformation is probably the most critical part of creating the strategy.

So, which one to choose? Which is the ultimate approach?

Out of the above "6 R's," we found refactoring to provide the best cost-benefit ratio.

Code refactoring refers to the process of restructuring existing computer code (changing its factoring) without changing how it behaves externally. The purpose of refactoring is to improve the software's design, structure, and implementation while preserving its functionality.

Within the scope of cloud transformation, refactoring also requires a mindset change to fully utilize the advantages it can provide for your organization. In a cloud environment, you can develop more than a single set of resources once a more flexible architecture is adopted through refactoring.

Also keep in mind that the serverless approach is equally (if not more) important than microservices. You can focus better on key business challenges and development if more IT services are used in a fully managed, serverless architecture. It also costs less than traditional IT architecture in the cloud.

We have collected some of the same options and feature sets that become available through refactoring, using AWS cloud transformation as the example:

Microservices

With this direction, we can decouple the domain functionality into smaller, more manageable groups.

Chained microservices

Communication with other services is also a type of resource usage.

Database per service

A recommended solution if we want to separate data traffic and data processing from other services.

Caching / Data distribution

Data does not necessarily have to be read from the database. Within AWS, several other cache patterns exist that are worth using.

Micro frontends

Another code level to make our application fast and service independent.

Queuing

It's not always necessary to process data immediately. If the data can wait for processing, then creating a queue for prioritizing can be the right solution.

Monitoring

Monitoring the infrastructure's default KPI (CPU, memory) is not enough sometimes. We recommend adding service-level KPI to measure the app behavior as well. For example, Success / No Success responses, where not only the exceptions count.

Dependency management:

It is also worth considering how our application collaborates with other services. If we change anything within one service, then we might unintentionally affect another one.

Cost management

Measuring costs during operation is crucial. This feature will help you control the unoptimized environment so that it won't drain your wallet!

Batch processing

It is a simple task to process a single data record, but with batch processing, groups of several data record streams can be processed at once. This approach can also serve as a cost and performance optimization option.

Analytics, Business Intelligence

Analytics and KPI must be part of the modern solution development. It is not a privilege anymore to add more intelligence to the solution, which helps us understand how our customers and the application behave and optimize the solution for even better customer experiences.

Microservice Principles

Microservice architecture is a variant of service-oriented architecture that organizes an application to collect loosely coupled services. Within its framework, code is broken into independent services. Although these services run as separate processes, their output is used as an input to another in a system of independent, communicating services.

Moving to a microservice architecture can be a huge turning point in your organization's life. As always, business goals should be the determining factor in whether to adopt a microservice architecture.

To help see the bigger picture, below we have collected the main reasons this kind of architecture is essential and why it pays off to adopt it:

1 | **Continuous delivery**
If there's an ideal architecture for ensuring continuous delivery, it is through microservices where each application is situated in a separate container along with the environment it needs to run. As such, we can avoid accidental meddling with other applications when editing an app within its container. With a microservice architecture, you can quickly update your software, meaning simplified troubleshooting and no downtime for users.

2 | **Faster deployment**
Microservice architecture can also accelerate deployment while simultaneously increasing app reliability. Because of the containerized environments that the apps are running in, they can be moved anywhere without affecting their environments. This leads to a faster time-to-market.

3 | **Increased resilience**
In a monolithic architecture, a single failure's impact can cascade out to all other services or functions. But thanks to the decentralized, decoupled nature of microservices, if an update to one container breaks something, it won't spiral out of control, affecting all the other separate containers. Even when several systems malfunction or are brought down for maintenance, the users won't notice anything.

4 | **Adaptation to changing market conditions**
Since the microservices approach leads to quicker update deployment and testing, your organization will be able to follow new market trends better and adapt its products faster. There's also the benefit of enabling innovation. Developers no longer have to fear to experiment, since the apps are isolated so that making changes in one won't affect all the others. With the ability to innovate more freely, your organization is bound to get ahead of competitors and create new revenue streams.

5 | **Enabling developers**

Developers will have access to the tools to build higher quality software products. With the containerized approach, developers can build apps using multiple components and then code each component in the language that is best suited to its function. In a monolithic architecture, they would have to make sacrifices and choose a single language for all apps, leading to suboptimal results. The depth of optimization available within microservices can significantly increase software quality.

6 | **Reduce costs**

Changing an application in a monolithic architecture is bound to be expensive; as all parts interact in a monolith, a change cascades throughout the architecture. Consequently, it leads to more and more time, money, and energy invested into just a single update. And with each update that builds on the previous one, the monolith becomes more and more bloated, increasing the number of resources needed to maintain it. With microservices, you can laser-focus on a single change without having to deal with — and pay for — modifications spilling over to other applications.

7 | **Considerable ROI, reduced TCO**

Your teams can work on separate services concurrently, leading to quicker deployment. Simultaneously, as the developers' code generally becomes more reusable, development time will also decrease. Furthermore, the decoupled nature of microservices means that you will no longer have to resort to expensive machines just to operate your systems — even basic x86 machines can do the trick. Besides the reduced infrastructure costs, the increase you gain in efficiency results in less downtime, too.

As you can see, a great microservices architecture can be a total game changer for your business. Nevertheless, it's worth keeping in mind that not every business can benefit from this, as some organizations cannot pull it off. We have collected some crucial considerations to check if your organization is prepared to be working in a microservices architecture:

You need to be equipped

This architecture requires rapid provisioning and app deployment to capitalize on its advantages. Developers must be able to provision resources instantly, and that is quite draining in the long run. You need to keep both human and IT resources fresh and capable of managing fast and frequent deployments — and bearing the pressure that comes with it.

Robust monitoring is crucial

As your teams work simultaneously on separate services using their language, platform, and APIs, you will need an efficient monitoring system to oversee and manage the entire infrastructure.

Embracing DevOps culture

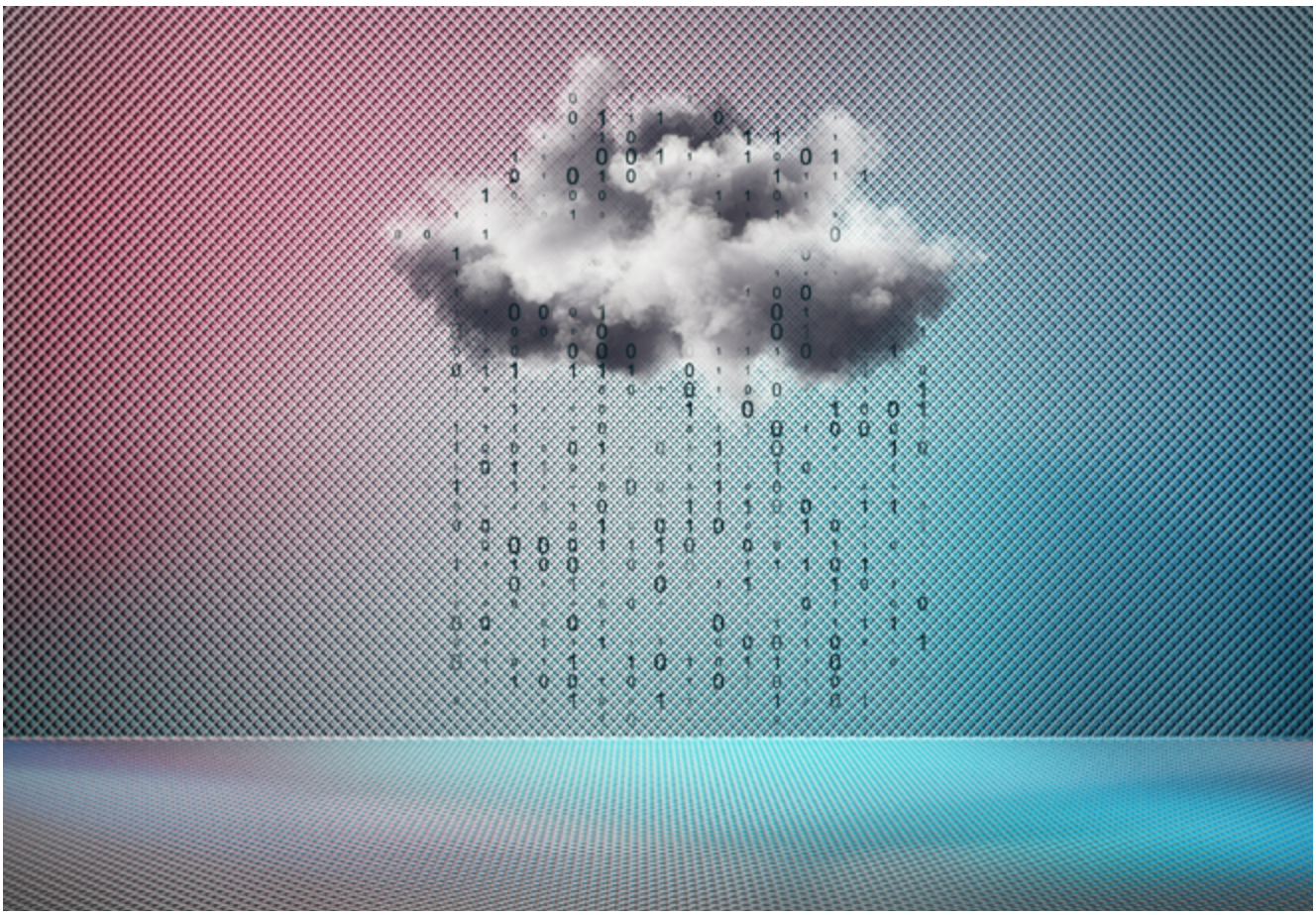
Learning and adopting DevOps practices is vital if you want cross-functional teams to work and cooperate effectively. In DevOps, the development and operations teams are no longer focused on their tasks; their work becomes intertwined, and both groups are responsible for service provisioning.

More complex testing

The dependencies of services are more widespread in a microservices architecture. As new features are added, not only does the complexity of services increase, but the number of dependencies also goes up. As such, resilience testing and fault injection become essential tools for your testers, as they must make sure that service unavailability, database errors, caching issues, or network latency do not take down an entire service.

Be prepared for failure

Every organization must be ready to handle failure issues such as system downtime, unexpected responses, or slow service. In case of failures, the impacted service should still be capable of running in a degraded functionality without crashing the entire system.



For example, in the AWS-based microservices architecture, we can secure the following benefits:

Elasticity	The ability to acquire resources as you need them and release resources when you no longer need them. In the cloud, you want to do this automatically.
Scalability	Successful, growing systems often see an increase in demand over time. A scalable system can adapt to meet this new level of need.
High Availability	It describes systems that are dependable enough to operate continuously without fail. They are well-tested and sometimes equipped with redundant components.
Resilience	The ability for a system to recover from a failure induced by load, attacks, and failures.
Flexibility	By using a small, template-based approach at the code level, we can ensure more efficient version handling and flow separation.
Autonomous Services	By achieving a full separation at the service level, the system can avoid the consumption of other services' memory.
Decentralized Governance	Each microservice would be governed independently, allowing each team to choose the best tool for the job.
Failure Isolation	By allowing each service to be responsible only for its own failures, we can minimize dependencies.
Auto-Provisioning	By allowing predefined / automatic sizing for each microservice based on load.
Continuous delivery through DevOps	By leveraging Terraform templates, automated test scripts, load tests, and improved quality assurance cycles for deployments.

Patterns of Transformation

Now that we've established the basic principles and benefits of the microservices architecture, let's discuss some of the standard microservice designs that lay down the path to a complete transformation.

These designs or transformation patterns are generally built on the six essential principles summarized below:

Reuse

Although reuse as a microservice design principle has been sidelined in recent years, it is still a valid and potentially viable solution. Today, its framework has changed to a merit-based reuse approach, where teams can configure communication models for determining how to adapt microservices for use outside the environments they were designed in. However, if an existing microservice API does not match your business goals, it's more feasible to build one that does.

Loose coupling

This principle can minimize dependencies between services and customers. Through standardization of contracts expressed via business-oriented APIs, you can ensure that the service's implementation — or any updates made to it — will no longer impact customers. As mentioned in previous sections, this provides the organization with the capacity to modify or replace parts of the system without affecting it as a whole and inconveniencing users with downtimes.

Autonomy

The service's control measures over runtime environment and database schema during implementation. Since it leads to an uptick in the service's performance and reliability, customers are provided high-quality services that they expect from a microservice solution. Overall availability and scalability can also be improved by coupling autonomy with statelessness.

Fault tolerance

Because the services within a microservices architecture are independent and containerized, we can cut off communication to a failed service through a technique called "circuit breaker." This principle enforces the idea to create a system where we can prevent failures from a single service cascading through the distributed system, thus ensuring the fault has minimal impact on collaborating services.

Composability

Through this principle, we can ensure that the services deliver value in various environments. Such compositions can form new service aggregates that are often seen as a new type of application development.

Discoverability

A simple principle that communicates the understanding of relevant business goals and the microservice's technical interface to all interested parties. This means that the service must function according to customers' exact needs, providing them with the feature set that helps their operations and realize their business purpose.

The patterns can be categorized into four distinct types: decomposition, integration, observability, and crosscutting concern patterns. Each pattern comes with its subsets of transformation models, but here we will focus only on those that are present within AWS:

Decomposition Patterns

1 | **Decompose by Subdomain**
Domain-Driven Design (DDD) subdomains form the basis of services in this pattern. Within DDD, the application's problem space is the business itself. It includes a domain consisting of multiple subdomains, where each subdomain is matched to a different part of the business. There are three types of subdomains here: **core** (key differentiator of the business, most valuable part of the application), **supporting** (side services of the business, can be outsourced), and **generic** (not a dedicated business component, usually implemented using off-the-shelf software).

2 | **Decompose by Business Capability**
In this approach, services are defined as business capabilities — activities that a business does to generate value, like a business object (e.g., order management, customer management). Within microservices, these business capabilities can be organized according to a multilevel hierarchy. This way, an enterprise application can have top-level categories matched to business objects/departments such as product development, product delivery, quality assurance, etc.

3 | **Sidecar pattern**
The segregation of an application's functionalities into a separate process. This approach is viable if you want to expand the application with new capabilities without including additional configuration code for third-party components. Ultimately, this pattern provides the opportunity to establish loose coupling between the application code and the underlying platform. Furthermore, it can help reduce both the complexity in the microservice code and the amount of code duplication — you won't have to write unique configuration code for each service.

Integration Patterns

1 | **Composite Aggregators**
A multi-bucket aggregation that creates composite buckets from different sources. It can be used to paginate all buckets from a multilevel aggregation efficiently, providing a way to stream all buckets of a specific aggregation. A composite bucket is a combination of the values extracted for each aggregated item.

2 | **Public/Private APIs, Proxy — API Gateway**
An API management tool situated between a client and a collection of back-end services. It serves as a reverse proxy that accepts all API calls, aggregates the various services needed to fulfill them, and returns the appropriate result.

3 | **CQRS — Event Sourcing**
CQRS stands for Command Query Responsibility Segregation; it is an application architecture pattern often used with event sourcing — a collection of patterns based on persisting the full history of a domain as a sequence of "events." CQRS splits an application into two internal parts: the command side ordering the system to update the state and the query side that acquires information without changing it.

4 | **Chained Microservices**
With this pattern, we can produce a single, consolidated response to any request. For example, Service #1 receives the client's request, initiates communication with Service #2, and starts communicating with Service #3. At the end of the chain, all services return with a unified response through Service #1. Throughout the communication, the services are most likely to use synchronous HTTP request/response messaging.

Another important aspect to understand here is not to make the chain too long. This is important, because the chain's synchronous nature will appear as a long wait on the client's side, especially if it's a web page waiting for the display's response. There are workarounds to this blocking request/response, which are discussed in a subsequent design pattern.

5 | **Client-Side UI Composition — Micro Frontend**
Using this pattern, we can decompose the frontend into separate, semi-independent "micro-apps" that work loosely together. If you involve several independent development labs to work on your app, choosing a micro frontend can make it easier to collaborate.

6 | **Database per Service**
Here, we can differentiate between **private-tables-per-service, schema-per-service, and database-server-per-service** approaches. In the former, each service owns a set of tables within the database. A service's access is restricted to that particular set of tables and no others. In schema-per-service, each service possesses a private database schema. And in the last approach, each service has its database server.

Observability patterns

Patterns also exist to ensure the reliable operation of applications. There are three main patterns used in the maintenance of applications: **logging, monitoring,** and **alerts**.

While having access to logs, metrics, and traces doesn't necessarily make systems more observable, these are powerful tools that, if understood well, can unlock the ability to build better systems.

Crosscutting Concern patterns

These pattern types are applicable throughout the application and may affect all parts of the application. **Logging, security**, and **data transfer** are present in almost every app module, making these areas crosscutting concerns.

Data Management

It is an umbrella term used to describe the processes of acquiring, validating, storing, protecting, and processing data to ensure the accessibility, reliability, and timeliness of that data for relevant users.

With a complete cloud transformation, all data management tasks and subtasks become smoother. However, if we go for a significant increase in handling data's efficiency, we must lay down a proper set of goals.

1 | Separate database queries and scans from primary operations.
2 | Provide the ability to scale the service load against databases.
3 | Decrease application requests.
4 | Increase maintainability.

With the right goals set, it becomes easier to implement the right solutions as well. For instance, through AWS's DynamoDB, an organization will be boosted by high-performance data access, data integration support via Dynamo Streaming, and enhanced data access performance using Global Secondary Indexes (GSI), and last but not least, reduced response time for queries. On top of it all, all maintenance overhead is owned by AWS under its Managed Services package.

Consider the benefit of using purpose-built DBs. At AWS, we have more than 10 purpose-built databases, including relational and non-relational DBs.

Decoupling the code

Using traditional coding techniques for developing and upgrading larger applications can easily lead to a bloated, monolithic architecture. However, through the concept of coupling, an organization can make considerable gains while reducing dependencies in their architecture at the same time.

But what exactly is coupling? It measures the interconnectedness of two functions, modules, or applications. The lower the coupling "score" is, the better; it is usually credited to a great architectural design system. From a business point of view, when the coupling is low, you can also reduce maintenance costs and achieve high code readability.

In microservices, a technique called **decoupling** is used to attain low coupling. It can ensure that the system maintains flexibility even as the number of features grows. By decoupling services and functional areas of applications, you can create a new architecture that offers greater extensibility and simplicity for developers as newer functionalities are added.

Using a **serverless event bus** (e.g., Amazon EventBridge), we can decouple services within an application, resulting in a simplified architecture and allowing each service to operate independently with no dependence on event consumers.

There are two main approaches to achieve decoupling: **synchronous** and **asynchronous**. Both services must always be available with the former, but they don't need to recognize the other's presence. In the latter case, communication occurs and is maintained even if the receiver is not available.

Synchronous

Synchronous decoupling with load balancers. Exposing a single web server to the outside world introduces a dependency: the public IP address of the EC2 instance. From this point on, you can't change the public IP address again because it's used by many clients sending requests to your server. You are now faced with the following issues:

- Changing the public IP address is no longer possible because clients rely on it.
- If you add a server (and IP address) to handle the increasing load, it will be ignored by all current clients, as they're still sending all requests to the public IP address of the first server.

Asynchronous

A distributed message queuing service (e.g., Amazon's Simple Queue Service) is needed to establish a messaging queue infrastructure. This allows you to move data between your application's distributed components that execute different tasks without message loss or the need to make every component always available.

You can solve these issues with a DNS name pointing to your server. But DNS isn't fully under your control. DNS servers cache entries, and sometimes they don't respect your time to live (TTL) settings. A better solution is to use a load balancer.

Through AWS, once you decouple your services either synchronously or asynchronously, you also gain access to the following features:

Push Provisioning	Updates immediately as the change occurs.
Querying vs. Awaiting Update	Allows asynchronous data processing at the database level.
Delayed Answer Queuing	Allows asynchronous data exchange.
Grace Period	Provides temporary or forecast value or decision management.
Notifications	Sends notifications about data processing to partner services.
Cache Management	Allows you to keep the value in the cache as long as it is needed.

Security

As mentioned at the beginning, one of the most common reasons for resistance against cloud transformation — and preference for on-prem solutions — is the need to have data protected, secure, and close to home.

But gone are the days when the cloud was less secure than internal servers. If anything, with the amount of attention and constant improvements that cloud solutions receive, your data is probably safer there than on an internal server.

These are a few considerations from the security aspect:

1 | Internal and external penetration testing

2 | Dynamic and static application security testing for OWASP / CORS

3 | Secure SDLC with bug reporting built into the pipeline

4 | Code Quality / Integrate the Code Security check into the CI/CD pipeline

5 | Unit tests (test against business functionality in the code without using infrastructure or a persistence layer)

6 | And further Compliance Services

The systema is also protected by stateful firewalls and domain-based DDoS, Rate Limiting, and Web Application Firewall (WAF) capabilities. And, of course, all of this is **GDPR-compliant.**

Deployment

When it comes to the new architecture's long-awaited deployment, it is critical to ensure that deployment times and human error are reduced to a bare minimum throughout the entire process.

This is best achieved through cloud deployment. Within the framework of cloud deployment, we mean creating a virtual computing environment that typically involves the setup of a SaaS, PaaS, or IaaS platform. At the end of the deployment, you will finally have a set of flexible and scalable virtual computing resources at your disposal.

However, to implement it, you will have to choose one of the **five cloud computing deployment models**. A model describes the environment where cloud applications and services can be installed readily available to users, with each model offering different management, ownership, access control, and security protocol options.

The five models are the following:

Public cloud

Using the standard cloud computing model, a public cloud makes resources such as applications, storage, and virtual machines available to users remotely. It is a viable option for web applications, file sharing, and nonsensitive data storage. Public cloud services are either free, subscription-based, or pay-per-use.

Private cloud

Here, the computing services are available through the Internet or a private internal network, but only to select users. Organizations opt to use this deployment model, since private cloud computing provides nearly the same benefits of a public cloud but with additional layers of control, customization, and a larger set of dedicated resources than on-prem hosting solutions. Private cloud services also provide a higher level of security and privacy via firewalls and internal hosting.

Virtual private cloud

With regard to security, privacy, and exclusivity, in concept, it is similar to the private cloud, but it also provides a configurable pool of shared resources that are stored within a public cloud environment. In a way, it is a compromise between public and private where a user has exclusive access to a segment of a public cloud, but access to the VPC itself is restricted and requires a secure connection.

Hybrid cloud

A combination of two or more infrastructures. It is a mixed computing, storage, and services environment composed of a private cloud (either regular or virtual), dedicated on-prem servers, and a public cloud like AWS, with orchestration between the platforms. For example, when an organization stores critical data on a private cloud and less sensitive information on a public cloud, they have a hybrid cloud infrastructure.

Community cloud

A collaborative effort to share the infrastructure between several organizations from a specific community with common concerns. It can be hosted on-premises, at a peer organization, or by a third party.

Within AWS, the chosen cloud deployment model allows for:

Use of Terraform Templates — Infrastructure as Code

Infrastructure as code is the process of provisioning and managing your cloud resources by writing a template file that is both human-readable and machine consumable. It also provides extra reliability and security (e.g. in case of DR) for the customer.

Continuous Delivery

Continuous delivery is a lean practice. Its purpose is to keep the production fresh by achieving the shortest path from the availability of new code in version control — or new components in package management. It can also minimize the time to deploy and mitigate (or time to remediate production incidents — TTM and TTR) if it is automated. Continuous delivery can help an organization optimize process time and eliminate downtime, and if coupled with the complementary practices of IaaC and monitoring, it can achieve even better results.

Automatic Database Setup and Change Management

Managing database changes in a production environment was always tricky. Data Schema, Add, Remove Index Changes, or just changing the database settings can lead to temporary outages or performance degradation.

Preconfigurable Scalability Options

With infrastructure as code, you can write it once and reuse it many times. As such, one well-written template can serve as the basis for multiple services in numerous regions around the world, making it much easier to scale horizontally. Within AWS, you can put together a custom set of options to streamline and accelerate the process by which you create these reusable templates.

Canary Release Implementation via Weighted Router Settings

The continuous delivery approach can sequence multiple deployment rings for progressive exposure (also known as “controlling the blast radius”). Users of progressive exposure groups then get to try new releases to monitor their experience in “rings.” The first deployment ring is often a canary that tests new versions in production before a broader rollout, such as a beta test, while weighted router settings let you associate multiple resources with a single domain name or subdomain name and choose how much traffic is routed to each resource.

The established approval process for deployments

The deployment from one ring to the next can be automated, but it requires an optional approval step first — the changes are usually signed off on electronically by a decision-maker. You can create an auditable record of approvals that satisfies regulatory procedures and other control objectives as you get ready for deployment.

Retrospective

What are the general benefits that an organization can gain from a complete cloud transformation with AWS?

We have looked at the metrics and KPIs of clients who chose AWS, and we've seen vast improvements across all measured areas:

DOWNTIME

- Dramatically improved

DEPLOYMENT

- From monthly to daily deployment
- Automated QA

COST

- The microservices approach brought a three-fourths cost reduction through the Rehost strategy
- Microservices cost optimization increases cost savings to 50% from the previous 20%
- Configuring to the right size: 20% cost cut

DELIVERY

- Go-to-market time improved to 200%

PERFORMANCE

- Dramatically improved

HIGH AVAILABILITY

- Dramatically improved

CODE COMPLEXITY

- Managed service/microservice
- Improved by 50%

Afterword

“

Many companies and organizations dream about enterprise-level IT or a highly scalable software solution. It is easy to express this goal and envision the results, but it is tough to achieve it. During my professional career, I faced and was fully involved in many related challenges and witnessed numerous conversions that only partly succeeded. We gathered these experiences and incorporated the lessons into each software and DevOps process we have built.

Today, AWS provides all ingredients for applying an enterprise-level IT approach to anyone, including small organizations. We believe that their services are the best in the market because their methodology and frameworks helped us approach this thorny, labyrinthine issue from the right direction, allowing us to create a significantly easier system to develop, deploy, and maintain. Although it required more and more considerations for every subsequent decision, the process remained easy throughout, making our transformation completely worth it!

”

Péter Dikházi — Founder and CEO of Blue Guava

About Blue Guava

Our goal is to become the best long-term partner that any of our clients could wish for. With more than ten years of state-of-the-art software development, streaming, and testing solutions, we have helped market-leader partners increase their revenue and the efficiency of their IT operations while cutting costs and time. Simultaneously, the software products we developed for them streamlined and optimized the streaming experience for millions of their customers across more than 50 countries on three continents.

At Blue Guava, we believe in exceptional customer service. Our passion is to provide our clients with nothing but the highest quality services that are guaranteed to meet their needs and help them in their quest to produce excellent software solutions.Our content delivery, content management software solutions, and quality assurance services will help you maximize customer engagement, ultimately empowering your business's customer adoption and retention capabilities.

HU-MSZT-ISMS/040-40

Contact Us

About Amazon Web Services

Amazon Web Services is a subsidiary of Amazon providing on-demand cloud computing platforms and APIs to individuals, companies, and governments on a metered pay-as-you-go basis

AWS products and services provide state-of-the-art solutions for cloud computing, storage, networking, database, analytics, application services, deployment, management, mobile, developer tools, tools for the Internet of Things, and more.

With AWS, your company can go global in minutes. By adopting AWS as the center of your digital business operation, your work gains momentum, your data will be safe and secure, your costs go down and your revenues up, allowing you the flexibility to scale your business.

As trusted AWS partners, we can guide you on a journey toward complete AWS cloud transformation. Let us show you the way!