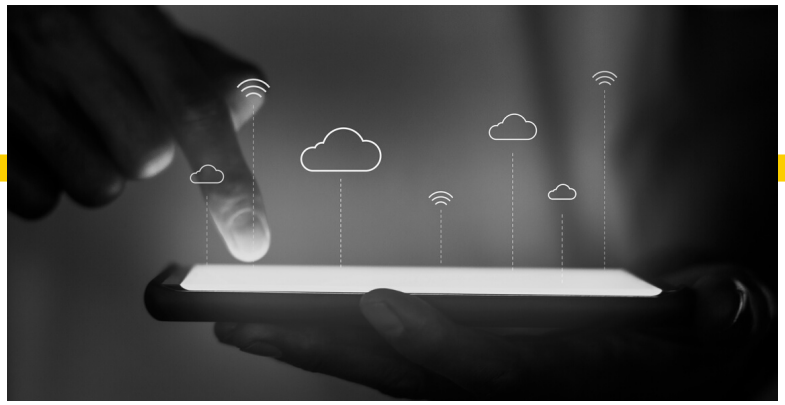
StatusNeo

Cloud Native with Microservices

Prepared by Akshay Nanda, Senior Technical Consultant at StatusNeo.

CONTENTS



INTRODUCTION

The rise of cloud providers such as AWS, Azure, and Google Cloud has resulted in a market shift in which enterprises of all sizes, large and small, are abandoning their own physical infrastructures in favour of using space within a specialist cloud provider's environment.

The cloud is allowing businesses to link people, data, and processes in new ways in order to take advantage of the opportunities provided by contemporary technology. Business leaders are bringing business and IT closer together and improving operations to provide new value for customers to succeed in a digital-first environment.
The Cloud notion has progressed from an appealing potential source of cost savings to just the way that enterprises must function.Today, as firms continue their digital transformation initiatives, leveraging the cloud is a critical strategy.

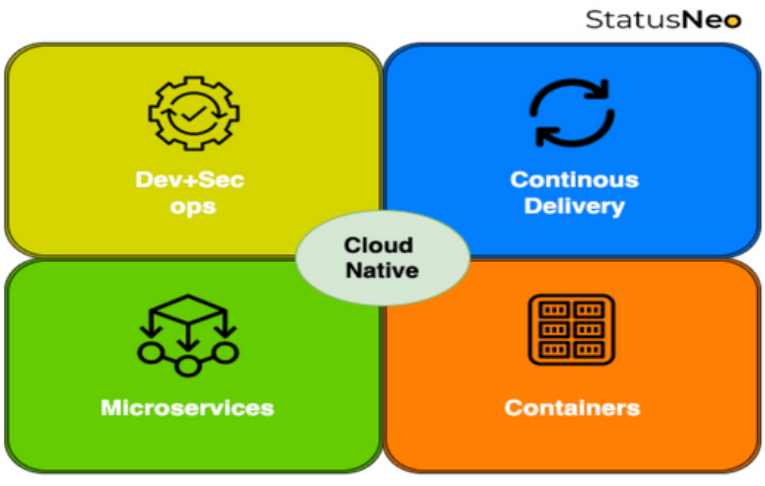
The requirement for agility and flexibility in the face of growing innovation and disruption from competitors is the key driver for cloud adoption across enterprises of all sizes. However, an increasing number of businesses are discovering that just relocating and migrating their legacy systems to the cloud is insufficient to meet their demands. Their monolithic architecture systems are preventing them from achieving their objectives.

In this whitepaper we explore the Cloud native approach to development, the challenges with monoliths, and why we need a different architecture to get the most out of our cloud investments. We'll also talk about how microservice architecture is better suited to cloud native programming. Finally, we go over a design approach for modernising applications with microservices.

WHAT IS CLOUD-NATIVE AND WHY CLOUD-NATIVE?

The term cloud native refers to the concept of building and running applications to take advantage of the distributed computing offered by the cloud delivery model. Cloud native apps are designed and built to exploit the scale, elasticity, resiliency, and flexibility the cloud provides.
Cloud native technologies empower organizations to build and run scalable applications in public, private, and hybrid clouds. Features such as containers, service meshes, microservices, immutable infrastructure, and declarative application programming interfaces (APIs) best illustrate this approach.
Going cloud-native means abstracting away many layers of infrastructure—networks, servers, operating systems etc.—allowing them to be defined in code.

These features enable loosely coupled systems that are scalable, resilient, manageable, and observable. They allow engineers to make high-impact changes frequently and with minimal effort.



Cloud Native Architecture

They have many benefits, like

- Independence
- Scalability
- Resiliency
- Standards-based
- Business agility
- Automation
- No downtime

Cloud native applications have become a key way to increase business strategy and value, because they can provide a consistent experience across private, public, and hybrid clouds. They allow your organization to take full advantage of cloud computing by running responsive and reliable cloud native apps that are scalable and reduce risk.

Cloud-native with Microservices based development

For most applications running in production today, monolithic architectures are the norm, but they are not the best fit for complex cloud-based systems. Enterprise applications rely on large, monolithic codebases, making it difficult to quickly introduce new services, while distributed development and operations teams make alignment difficult. Moreover, users are more demanding than ever before - enterprises need to scale efficiently and monitor deployments to ensure high performance and consistency for their customers.

Companies like Amazon and Netflix popularized the use of microservices and use them to build massive scalability and decrease time between releases. While such companies are digital first, large enterprises with legacy systems to support can also benefit from a microservice architecture. And though the challenges—such as monolithic systems, legacy technology, skills gaps and cultural issues—might be great, the rewards can be far greater.

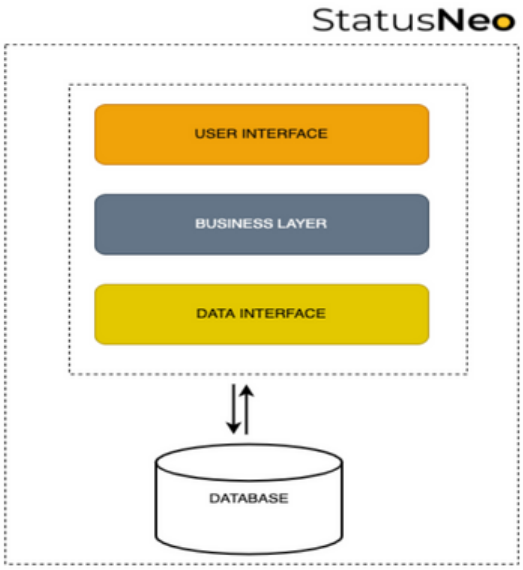
Why are Monoliths not the best architecture for the cloud?

In the past, an application was usually designed using a monolithic architecture. In this mode of development, the application is built, tested, and packaged, as well as deployed as a single unit. Codebases are also compiled together, and the application is deployed as a single entity. Scaling required copying instances of the application binaries and the required libraries to different servers, and the application code typically ran as a single process. Continuous delivery — an approach that involves fast, iterative software development and safe updates to the deployed application — was challenging since the full monolithic application stack needed to be recompiled, relinked, and tested for even the smallest incremental release.

Before we discuss more around the problems with monolithic application, lets first understand what they are.

A monolithic application is built as a single unit. Enterprise applications are built in three parts (3-Tier):

- A database — consisting of many tables usually in a relational database management system.
- A client-side user interface — consisting of HTML pages and/or JavaScript running in a browser).
- A server-side application — which will handle HTTP requests, execute domain-specific logic, retrieve and update data from the database, and populate the HTML views to be sent to the browser.



Monolithic Architecture

Problems presented by monoliths include:

- As application functionality and fixes grow, a monolithic code base grows, putting more strain on the technological stack and the development environment's capabilities.
- Changing the technological stack of a monolith can be difficult because it consists of a single executable.
- As the monolith becomes more sophisticated, the deployment and testing effort (even if automated) grows exponentially.
- Refactoring code becomes more complex since it becomes more difficult to foresee how it will affect application functionality.
- If any single function or component fails, the entire application will go down.
- Issues with one module's performance can have a negative impact on the entire application.
- When a single function consumes more resources than expected, overall performance suffers.
- Scaling a monolithic application is commonly done by deploying it to additional servers (i.e., horizontal scaling). Each parallel monolith then places an equal load on the underlying resources, which is a highly inefficient design.

What are microservices and how are they better?

Microservices is a software architecture that divides programmes into small, self-contained services. Services are often disconnected along corporate boundaries and focused on a single, distinct purpose or function. Separating services along business lines allows teams to focus on the relevant goals while also ensuring service autonomy. Each service is built, tested, and deployed separately, and services are normally split as different processes that communicate across a network via agreed-upon APIs, though the network may be local to the system in some situations.

BENEFITS OF MICROSERVICES

Many organizations can better meet the needs of modern application development by implementing microservices. The benefits include:

01

FASTER TIME TO MARKET

Monolithic applications require redeploying the entire stack of the application for even small changes, resulting in higher risk and complexity. This leads to longer release cycles. With microservices, small changes to services can be committed, tested, and deployed immediately since they are isolated from other components of the system.

02

EASIER TO BUILD AND ENHANCE

Microservices enable higher quality code by encouraging more focus. Since individual microservices, by definition, are smaller than monolithic applications, they have less scope and less code. This makes experimentation and testing with incremental code updates much easier.

03

EASIER TO DEPLOY

Microservices are easier to deploy than monolithic applications because they are smaller and thus have fewer environmental dependencies. The reduction of dependency discrepancies from development to production is also a key advantage of a containerized architecture.

04

EASIER TO MAINTAIN, TROUBLESHOOT, AND EXTEND

Microservices more seamlessly enable ongoing maintenance and fault tolerance, which are major challenges in any large-scale software environment. The distributed nature of microservices across multiple computer servers and resources helps with fault tolerance strategies to enable 24x7 deployments. Any given microservice can be deployed redundantly so that there is no single source of failure.

05

SIMPLIFY CROSS-TEAM COORDINATION

One of the challenges of any large-scale software development effort is the risk of over-complicating the integration points. In a microservice, the internal workflow can be as simple as reading data from a source, performing an action on the data, and then sending outputs to a destination. Microservices typically deal with small amounts of data at a time, so it is easier to manage and share the output once the data is processed. This simplicity in the processing and the data scope leads to simplicity in the handoff.

06

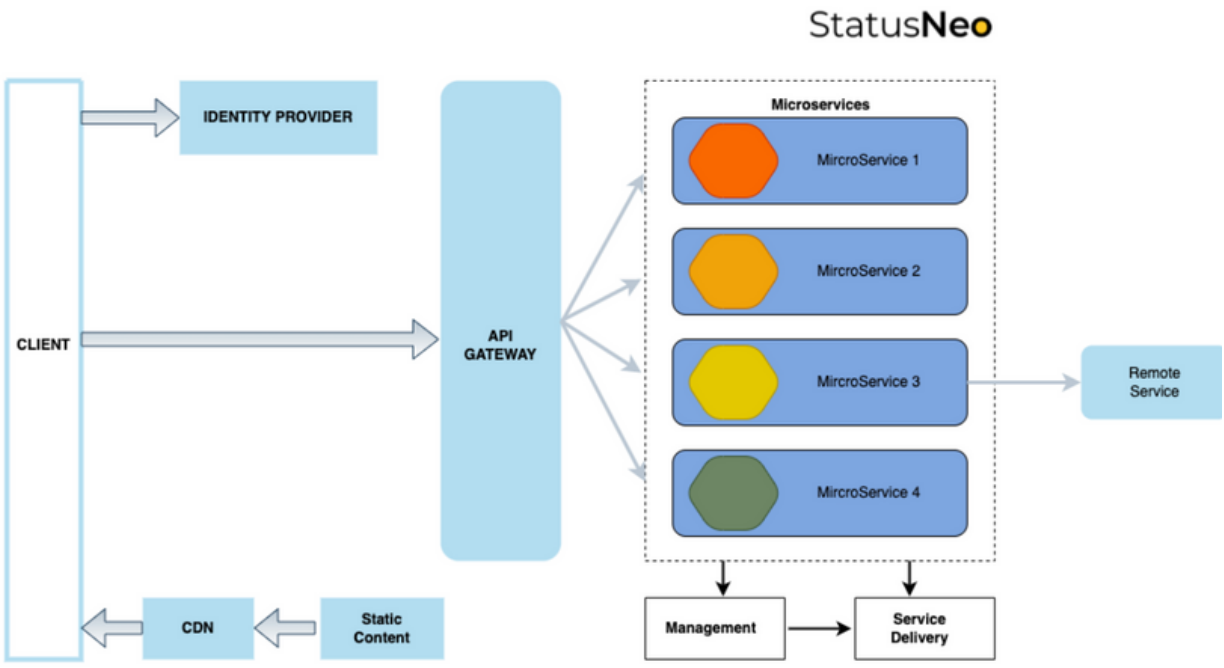
MICROSERVICES DELIVER PERFORMANCE AND SCALE

The distributed architecture of microservices opens opportunities for increasing performance and scaling out. Just as each microservice can be run redundantly for fault tolerance, this redundancy also enables greater parallelism that adds performance and scale.

07

MICROSERVICES SIMPLIFY REAL-TIME PROCESSING

Real-time processing introduces many challenges around performance, scale, reliability, and maintainability, and the microservices approach can help alleviate the difficulty.



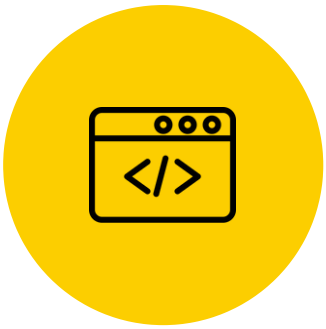
MicroServices Architecture

Considerations while building a microservices stack.

It's critical to discover the correct ingredients in a microservices-enabled technology stack or platform after the decision to pursue a microservices approach has been taken. Look for the following items:

LIGHTWEIGHT CONTAINERS

Does the platform provide a lightweight, dynamic container? This is most likely the most important criterion.



PROGRAMMING ENVIRONMENT WITH MANY LANGUAGES

Is the platform capable of supporting a polyglot environment—that is, more than one programming model? The platform should never impose a specific programming paradigm, style, or language on its users. Instead, it should be "agnostic" and provide multi-channel, multi-container support, allowing a single instance to host native code, python, Java, and node, among other things.

OUT-OF-THE-BOX CAPABILITIES

What out-of-the-box capabilities does this platform have to reduce overhead during implementation? How soon, for example, can a messaging infrastructure be put in place? How much does it cost to run? How well are the framework's capabilities integrated?

SECURITY INTEROPERABILITY

How much standardisation is there for security interoperability? Are there any tools for managing certificates? Is it possible to use different types of credentials inside a single implementation, and, more importantly, is that capacity abstracted away from the service implementation? Selecting a system where security, interconnectivity, and interoperability are baked into the service implementation is a significant error. They have to be kept apart.

HOT "SWAP-ABILITY"

Is hot-swapping of microservices possible on the platform? Getting a container provisioned at runtime is only one part of the process. How dynamic the changes are when microservices are replaced, deprecated, or new services are added is another factor to consider.

MANAGEABILITY AND MONITORING

Despite all of the advantages of microservices, one of the drawbacks is that due to the widely distributed nature of the architectural style, microservices can be difficult to manage, control, and isolate errors. How will IT operations determine the source of any issues? How will they monitor the system to figure out what's broken, what's functioning, and what's not? What methods will they use to carry out upgrades and corrections? Any technological platform should include the ability to monitor a microservice-based end-to-end implementation without requiring any instrumentation in the microservice itself.

Transitioning from monolithic architecture to microservice architecture

When looking at how to approach the introduction of microservices to an enterprise, taking a design-based approach is very helpful. This approach can be broken down into **five different steps of design**:

01

OUTCOME DESIGN

Look at your goals and ask, "Why are you doing this?" It's not enough to say: "Everybody's on the microservice bandwagon, let's jump on too." It's important that you understand the value points you're going after.

02

SYSTEM DESIGN

Examine how you identify the scope of the system that you're going to be architecting. This is about decomposing the domain.

03

SERVICE DESIGN

Once you have a picture of what your domain is and what all its services are going to be, look at the design of all those individual services to make sure they're built in the right way, so that they can evolve and interact in the correct way for the system you want to build.

04

FOUNDATION DESIGN

The previous steps have been very technology-agnostic. So now, you need to look at the underlying capabilities - the technological tools and platforms that will be required to build out the system best-suited to the needs of your organization.

05

ORGANIZATIONAL DESIGN

Look at the organization itself, the people side. How do you make sure that the culture and methodologies you're using and even the organizational structure, match what you hope to achieve?

Conclusion

Microservices architecture embodies the failures of monolithic applications and can be seen as a logical response in this age of frequent functionality changes and constant operational churn. Unlike the monolithic architecture, this is a sustainable architecture as with it, technical debt is contained, and rapidly changing business requirements are met by adding new microservices, not modifying (and breaking) old ones.

> **Almost all the successful microservice stories have started with a monolith that got too big and was broken up.**
>
> **-Martin Fowler**