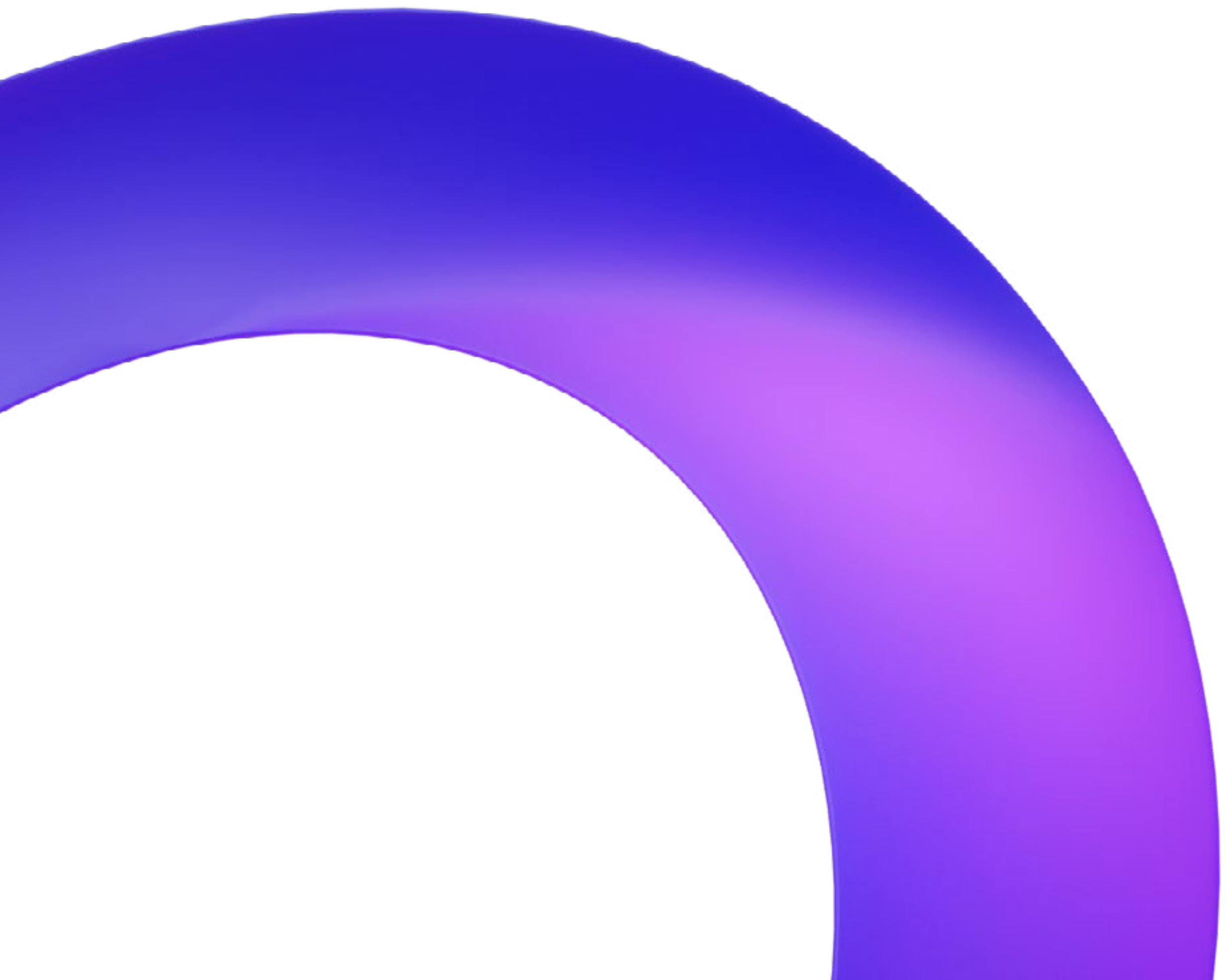
Cloud Native Applications

Architecture and Mindset

Luware
Nimbus

Luware

Contents



1. Introduction

Cloud native is a hot topic in software development. It is revolutionizing the application landscape and changing the way we think about developing and deploying software. Cloud native is more than just a set of technologies; it is a philosophical approach to building applications that fully leverage the cloud. Building cloud native applications is a big undertaking, but the payoff is equally large. Cloud native applications have fostered innovation and increased agility and flexibility. A cloud native approach allows software vendors to create scalable applications for modern and dynamic environments by fully utilizing public, private, or hybrid clouds.

1.1 Cloud Native defined

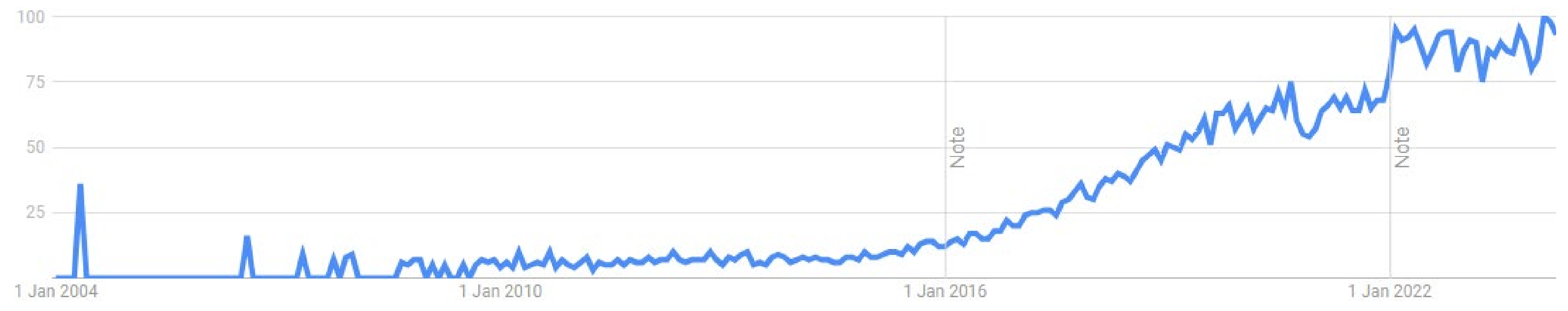
The term cloud native often invokes a lot of confusion, in part because it is used twofold. One the one hand, cloud native refers to the approach of building an application that is intentionally designed for a cloud environment. On the other hand, the term is also used to describe the characteristics and architecture of such applications. As technology and the cloud are constantly developing, so too is the term cloud native.

According to Google Trends, the term was frequently used in the early 2000s during the starting days of cloud computing and gained momentum as an industrial buzzword around 2015. Everyone seems to have a different idea of what exactly cloud native means, so in an attempt to streamline the term, the **Cloud Native Computing Foundation defined cloud native** as “technologies [that] empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach. These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil.”

The difference between Cloud native and Cloud computing

Cloud native should not be confused with cloud computing. Cloud computing, often simply called ‘the cloud’, refers to the delivery of infrastructure and services through the internet. Cloud native on the other hand refers to the architecture and mindset for assembling cloud-based services in a way that optimizes them for the cloud environment.

Figure 1 | Google searches for the term "cloud native" worldwide over time



2. Cloud Native Architecture

Just like there are many ways to design a building, different techniques to design and code an application exist. This is what we call application architecture. The different architecture types have various strength and weaknesses and lend themselves to different types of applications. Some are useful for highly scalable applications, whereas others might be more suitable if you want to build a highly agile application. For complex applications that have multiple subsystems that are frequently updated, such as contact center software, a microservice architecture is the architecture of choice.

2.1 Microservices

APIs are Application Programming Interfaces. APIs let different applications communicate with one another and enable data transmission. They are like carrier pigeons that deliver a message or requests to the relevant provider and return with the appropriate response.

Microservices are an integral component of cloud native applications. In a microservice architecture, the functionality is divided into loosely coupled autonomous subsystems, where each subsystem is assigned a certain business functionality – for the contact center software these could be call handling and steering, task routing, service availability calculation or reporting. These systems, which we call Microservices, can vary in size and contain a single module or a large chunk of the application. Because the systems are separate from each other and do not share any code, the most appropriate technologies can be used for the different services. Microservices communicate through APIs to make up the fully functioning application.

Microservices can be developed, tested, deployed, and scaled independent of other services, which makes application maintenance a lot easier. Instead of updating a whole application, only specific components need to be touched. In monolithic application, code dependencies would become tangled over time, but because microservices don't share any code, it is easier to roll out new features. If designed well, an update or bug in one service will not disrupt the rest of the system

and the application can continue to function without any downtime. Meanwhile, in traditional applications, a bug in one part of the application could hold back an entire release cycle, meaning new features would be delayed. Microservices facilitate a non-interruptive and quick roll out of updates and new features.

A Microservices architecture increases agility. Whilst the system at large is more complex in a Microservice architecture, each individual service is very simple. This means individual services can be developed and maintained by a small team. This requires less coordination and allows for a quicker roll-out of features. This increases productivity and lowers development costs.
One of the biggest advantages of a microservice architecture is the scalability. When demand increases, services can be scaled independently as needed without scaling out the whole application. New resources can be allocated to the most needed services or new services can easily be added to the system. For example, if the number of calls increases, new call handlers can automatically be started to deal with the increased load. Microservices consequently allows for an optimal use of computing capacity.

2.2 Levels of modernisation

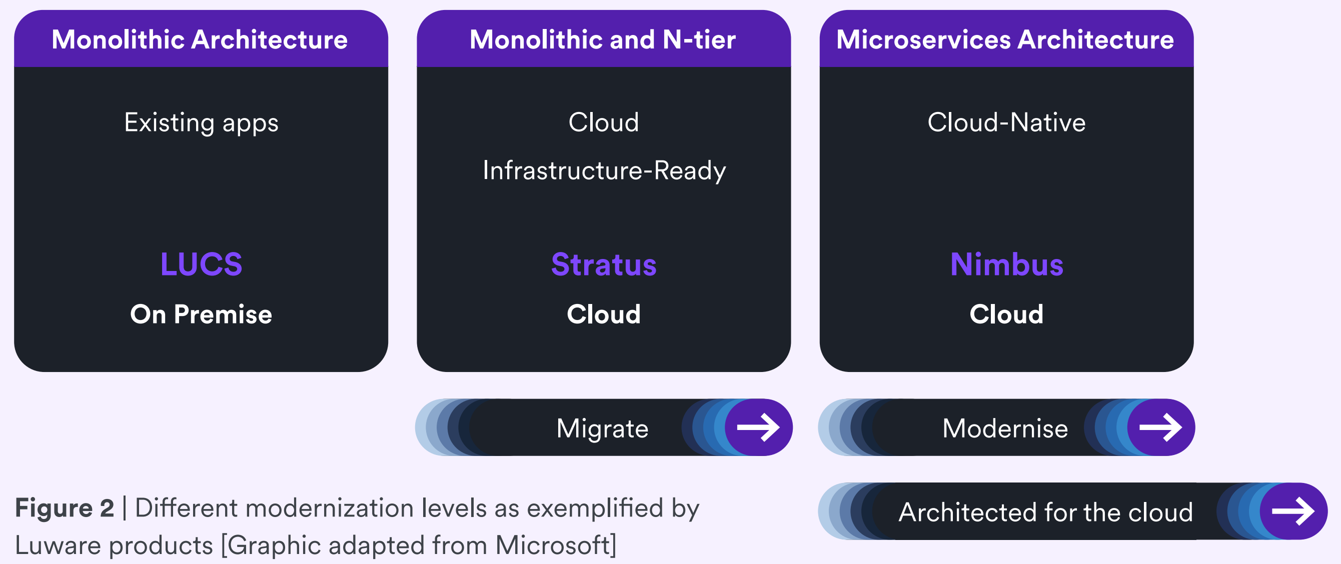


Figure 2 | Different modernization levels as exemplified by Luware products [Graphic adapted from Microsoft]

Software architecture has changed over time and applications can be **distinguished by various levels of modernisation**. Traditional applications such as Luware LUCS are built on-premise using a monolithic architecture, i.e. they use a single codebase for all functionality. In a monolithic architecture, development teams are restricted to one or two coding languages, code changes have to be carefully coordinated and deploying new features requires a lot of upfront testing.

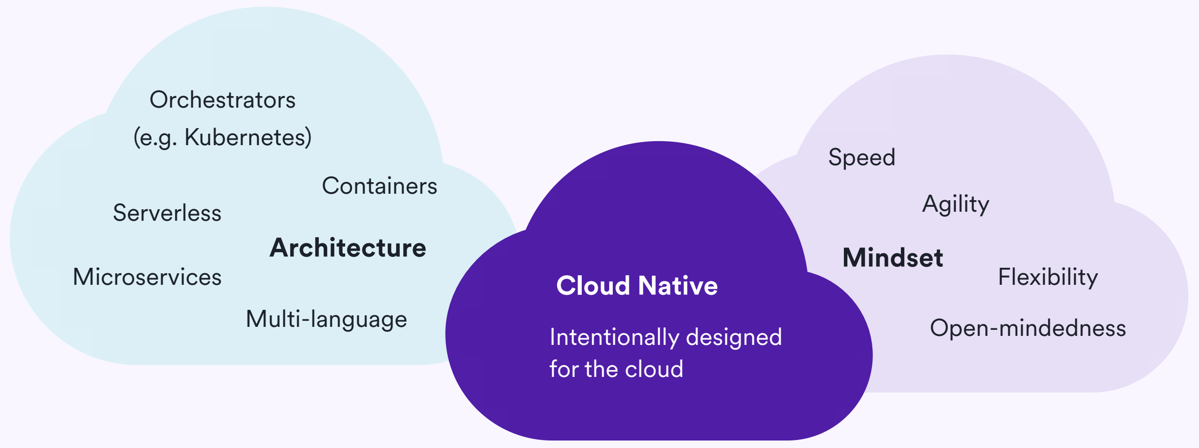
The next step in the modernization process is to migrate the application into the cloud. That means that instead of running on an enterprise's on-site server, the application is moved to an off-site data center where it can be accesses by the enterprise through the internet. Luware Stratus was an example of such an infrastructure-ready application. The peak of modernization is a cloud native application such as Luware Nimbus, which is explicitly built using a microservice architecture to optimally perform and scale in a cloud environment.

Microservices are containerized, which basically means that the code and everything that it needs to run is bundled up and isolated from the operating environment. This removes any risk of conflict between different coding languages or frameworks and makes it easy to move containers into different environments. Containers can be added, replaced, or updated without disrupting the application at large, making the application agile and hyper-scalable. Orchestrators, such as Kubernetes, are used to manage containers based on demand to warrant an optimal resource distribution. Such platforms also make it easier to identify a container with a bug, thereby facilitating a non-interruptive and simple troubleshooting. This facilitates continuous updates in the software. Whereas in a monolithic application like LUCS updates are deployed bi-annually, new features are rolled out weekly or even more frequently in a cloud native application like Nimbus.

3. Cloud Native Development

Cloud native is more than just a set of technologies. Migrating to cloud native software comes with significant development costs as well as **cultural and organizational changes**. Software vendors have to completely reorganize themselves and adapt a cloud native mindset. Whilst the traditional monolithic approach lends itself to a conservative and cautious mindset, as **one missing semicolon can shut down an entire system**, a cloud native culture should embrace failure and promote risk-taking and allow developers to experiment with new features. Processes are sped up and optimized for continuous delivery and fast innovation is required to keep up with quickly changing consumer needs. A cloud native mindset embraces all the architectural possibilities of the cloud in terms of flexibility, agility, and scalability.

4. Conclusion



- Cloud native can refer to the fact that an application optimally scales and performs in a cloud environment. Additionally, cloud native may refer to the key characteristics of cloud native applications such as a microservice architecture.
- Cloud native applications are built on microservices. Microservices are packaged into containers and dynamically orchestrated using for e.g. Kubernetes to optimally distribute resources.
- Benefits of a microservice architecture include practically unlimited scalability as well as increased agility to respond to market demands.
- Microservices facilitate maintainability of applications and updates can be undertaken without shutting down the application. Resiliency to failures is improved.
- Transitioning to cloud native development requires a change in mindset to embrace all the architectural possibilities of the cloud.

Luware