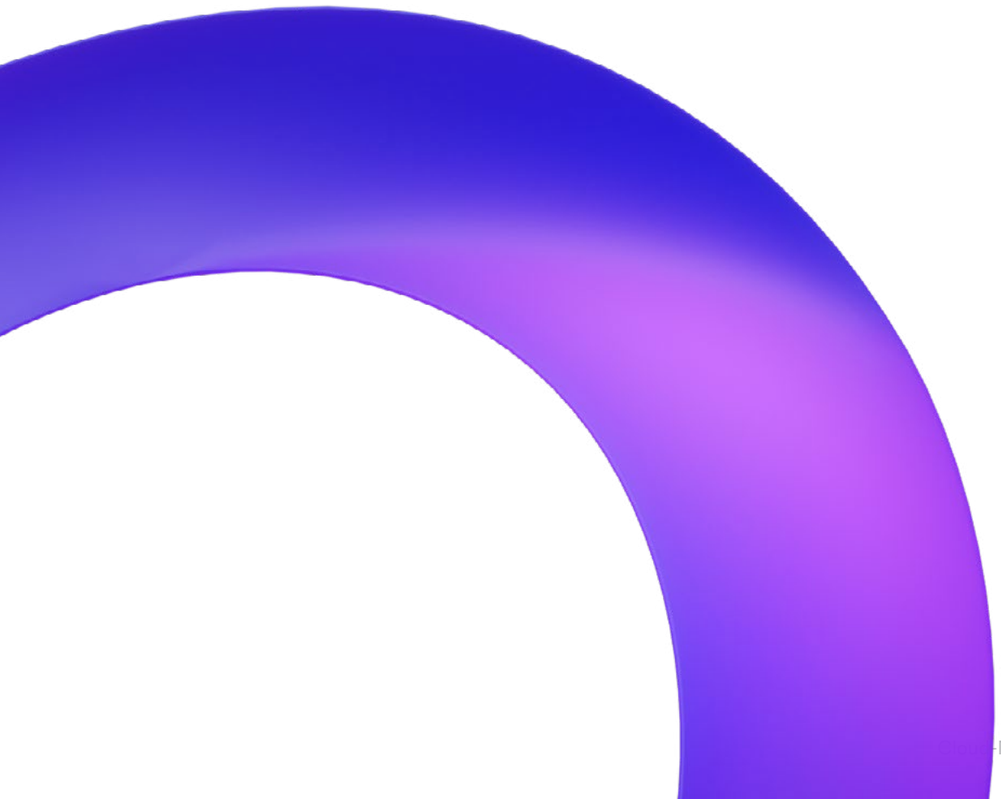
Cloud-Native Applikationen

Architektur und Mentalität

Luware
Nimbus

Luware

Inhaltsverzeichnis



1. Einleitung

Cloud-Native ist gerade ein heisses Thema in der Softwareentwicklung. Es revolutioniert die Applikationslandschaft und verändert die Art und Weise, wie wir über die Entwicklung und Bereitstellung von Software denken. Cloud-Native ist mehr als nur eine Reihe von Technologien; es ist ein philosophischer Ansatz für die Entwicklung von Applikationen, welche die Cloud voll ausschöpfen. Die Entwicklung von Cloud-Nativer Applikationen ist ein grosses Unterfangen, welches aber auch gross belohnt wird. Cloud-Native Applikationen fördern Innovation und steigern die Agilität und Flexibilität. Ein Cloud-Nativer Ansatz ermöglicht es Softwareanbietern, skalierbare Applikationen für moderne und dynamische Umgebungen zu erstellen, indem sie öffentliche, private oder hybride Clouds vollständig nutzen.

1.1 Cloud-Native definiert

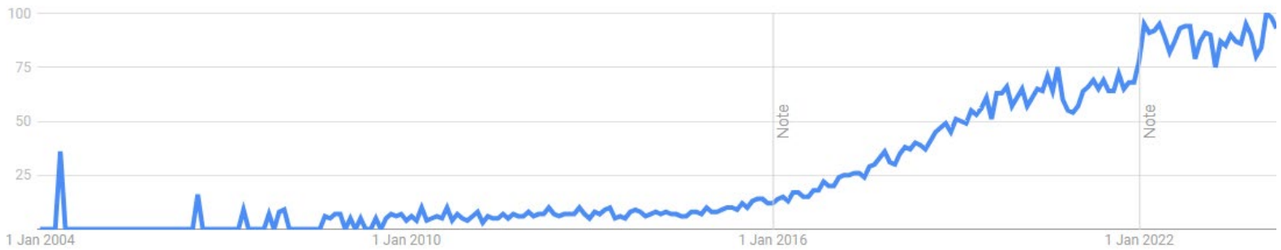
Der Begriff Cloud-Native führt oft zu Verwirrung, da er zweideutig verwendet wird. Einerseits benutzt man Cloud-Native um eine Applikation zu beschreiben, die explizit für eine Cloud-Umgebung gebaut ist. Andererseits wird der Begriff auch dazu verwendet, um die Eigenschaften und die Architektur solcher Applikationen zu beschreiben. Da sich die Technologie und die Cloud ständig weiterentwickeln, verändert sich auch der Begriff Cloud-Native.

Laut Google Trends wurde der Begriff in den frühen 2000er-Jahren, den Anfangstagen des Cloud Computing, häufig verwendet und gewann im Jahr 2015 Momentum als industrielles Modewort. Jeder scheint eine andere Vorstellung davon zu haben, was genau Cloud-Native bedeutet. In einem Versuch, den Begriff zu vereinheitlichen, **definierte die Cloud-Native Computing Foundation Cloud-Native** als «Technologien, [die] Unternehmen erlauben, skalierbare Anwendungen in modernen, dynamischen Umgebungen wie öffentlichen, privaten und hybriden Clouds zu erstellen und auszuführen. Container, Service Meshes, Microservices, unveränderliche Infrastruktur und deklarative APIs sind Beispiele für diesen Ansatz. Diese Techniken produzieren lose gekoppelte Systeme, die widerstandsfähig, verwaltbar und beobachtbar sind. In Kombination mit einer robusten Automatisierung ermöglichen sie Entwicklern, Änderungen mit hoher Auswirkung häufig und vorhersehbar mit minimalem Aufwand durchzuführen.»

Der Unterschied zwischen Cloud-Native und Cloud-Computing

Cloud-Native sollte nicht mit Cloud-Computing verwechselt werden. Cloud Computing, oft einfach «Cloud» genannt, bezeichnet die Bereitstellung von Infrastruktur und Diensten über das Internet. Cloud-Native hingegen bezieht sich auf die Architektur und Denkweise für die Entwicklung von Cloud-basierten Diensten, um diese für die Cloud-Umgebung zu optimieren.

Abbildung 1 | Google-Suchanfragen für den Begriff ‹Cloud Native›



2. Cloud-Native Architektur

Genauso wie es verschiedene Stile gibt, um ein Gebäude zu entwerfen, gibt es auch verschiedene Techniken, um eine Applikation zu entwerfen und zu codieren. Dies wird als Applikationsarchitektur bezeichnet. Die unterschiedlichen Architekturtypen haben verschiedene Stärken und Schwächen und eignen sich für verschiedene Arten von Applikationen. Hoch skalierbare Applikationen haben eine andere Architektur als speziell agile Applikation. Für komplexe Applikationen, die mehrere Subsysteme haben, die häufig aktualisiert werden, wie z. B. Contact-Center-Software, ist eine Microservice-Architektur die Architektur der Wahl.

2.1 Microservices

Microservices sind ein integraler Bestandteil von Cloud-Nativen Applikationen Eine Microservices-Architektur ist nützlich für komplexe Applikationen, die aus mehreren Subsystemen bestehen und häufig aktualisiert werden, wie z.B. eine Contact Center Software. In einer monolithischen Architektur ist die Software in sich geschlossen und alle Komponenten der Applikation miteinander verbunden. Bei einer Microservice-Architektur wird die Funktionalität in lose gekoppelte, autonome Microservices (Subsysteme) aufgeteilt und jedem Microservice wird eine bestimmte Zuständigkeit zugewiesen. Bei einer Contact-Center-Software könnten das die Anrufbearbeitung und -steuerung, Aufgaben-Routing, Berechnung der Serviceverfügbarkeit oder Reporting sein. Diese verschiedenen Subsysteme teilen sich keinen Code, so dass für die verschiedenen Dienste die jeweils am besten geeigneten Technologie verwendet werden kann. Microservices kommunizieren über APIs, um eine funktionsfähige Applikation zu bilden.

Microservices können unabhängig von anderen Services entwickelt, getestet, bereitgestellt und skaliert werden – die Wartung der Applikation wird dadurch stark vereinfacht. Anstatt eine ganze Applikation zu aktualisieren, müssen nur bestimmte Komponente angefasst werden.

APIs steht für Application Programming Interfaces. APIs ermöglichen es, dass verschiedene Applikationen miteinander kommunizieren und Daten übertragen können. Sie sind wie Brieftauben, die eine Nachricht oder Anfrage an den jeweiligen Anbieter liefern und mit der Antwort zurückkehren.

In monolithischen Applikationen verheddern sich mit der Zeit die Code-Abhängigkeiten. Da Microservices keinen Code teilen, können einfacher neue Funktionen ausgerollt werden. Sofern die Architektur gut konzipiert ist, hat ein Update oder ein Fehler in einem Service keinen Einfluss auf den Rest des Systems und die Applikation kann unterbrechungsfrei weiterlaufen. Bei monolithischen Applikationen hingegen kann ein Fehler in einem Teil der Applikation einen ganzen Release aufhalten, und somit die Veröffentlichung neuer Funktionen verzögern. Microservices ermöglichen ein unterbrechungsfreies und schnelles Ausrollen von Updates und neuen Funktionen.

Eine Microservices-Architektur erhöht die Agilität. Obwohl das System als Ganzes in einer Microservice-Architektur komplexer ist, ist jeder einzelne Service sehr einfach aufgebaut. Das bedeutet, dass einzelne Services von einem kleinen Team entwickelt und gewartet werden können. Dies erfordert weniger Koordination und ermöglicht ein schnelleres Ausrollen von Funktionen. Das erhöht die Produktivität und senkt die Entwicklungskosten. Einer der grössten Vorteile einer Microservice-Architektur ist die Skalierbarkeit. Bei steigender Nachfrage können Services unabhängig voneinander skaliert werden, ohne die gesamte Applikation zu skalieren. So können Ressourcen an die am meisten benötigten Service zugewiesen werden oder neue Service einfach hinzugefügt werden. Wenn zum Beispiel die Anzahl der Anrufe steigt, können automatisch neue Call-Handler gestartet werden, um die erhöhte Last zu bewältigen. Microservices ermöglichen somit eine optimale Ausnutzung der Rechenkapazität.

2.2 Modernisierungsstufen

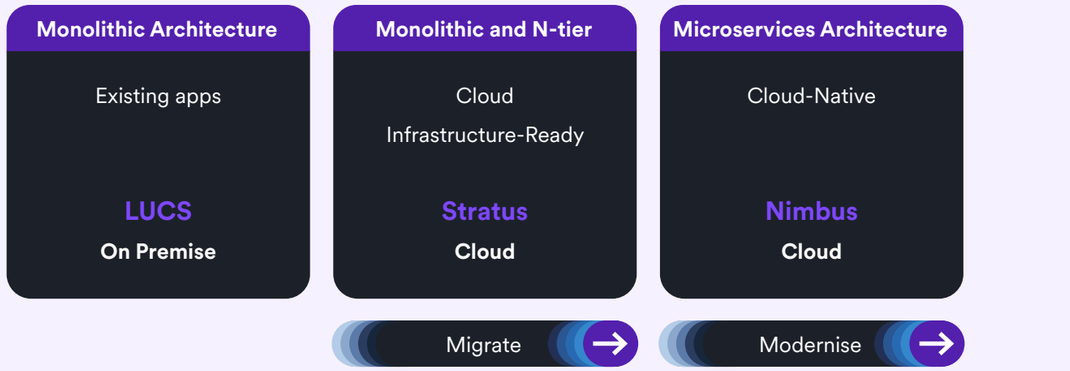


Figure 2 | Unterschiedliche Modernisierungsstufen am Beispiel der Luware-Produkte [Grafik von Microsoft adaptiert]

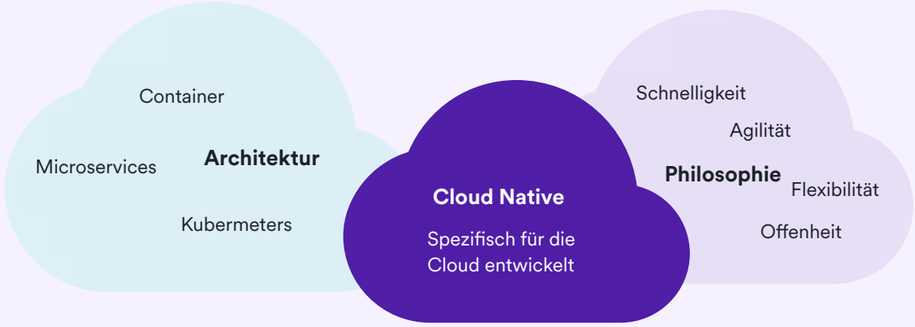
Die Software Architektur hat sich im Laufe der Zeit verändert und Applikationen werden durch **verschiede Modernisierungsstufen differenziert**. Traditionelle Applikationen wie Luware LUCS werden on-premise mit einer monolithischen Architektur aufgebaut, d. h. sie benutzen eine singuläre Codebasis für alle Funktionen. In einer monolithischen Architektur sind die Entwicklungsteams auf eine oder zwei Programmiersprachen beschränkt, Codeänderungen müssen sorgfältig koordiniert werden und die Bereitstellung neuer Funktionen erfordert einen hohen Testaufwand im Vorfeld. Der nächste Schritt im Modernisierungsprozess besteht darin, die Applikation in die Cloud zu migrieren. Das bedeutet, dass die Applikation nicht mehr auf dem Server eines Unternehmens vor Ort läuft, sondern in ein externes Rechenzentrum verlagert wird, worauf Unternehmen über das Internet Zugriff haben. Luware Stratus war ein Beispiel für eine solche Infrastructure-Ready-Applikation. Der Höhepunkt der Modernisierung ist eine Cloud-Native Applikation wie Luware Nimbus, die explizit mit einer Microservice-Architektur entwickelt wurde, um in einer Cloud-Umgebung optimal zu funktionieren und zu skalieren.

Microservices werden containerisiert, was im Grunde bedeutet, dass der Code und alles, was er zur Ausführung benötigt, gebündelt und von der Betriebsumgebung isoliert wird. So gibt es keinen Konflikt zwischen verschiedenen Programmiersprachen oder Frameworks und man kann Container einfach in verschiedene Umgebungen verschieben. Container können hinzugefügt, ersetzt oder aktualisiert werden, ohne die Applikation zu beeinträchtigen, wodurch die Applikation agil und hyper-skalierbar wird. Orchestratoren, wie z.B. Kubernetes, werden eingesetzt, um Container bedarfsgerecht zu verwalten und eine optimale Ressourcenverteilung zu gewährleisten. Solche Plattformen erleichtern auch die Identifizierung eines Containers mit einem Fehler und ermöglichen so eine unterbrechungsfreie und einfache Fehlersuche. Dies erleichtert die kontinuierliche Aktualisierung der Software. Während in einer monolithischen Anwendung wie LUCS Updates halbjährlich bereitgestellt werden, werden neue Funktionen in einer Cloud-Nativen Anwendung wie Nimbus wöchentlich oder sogar noch häufiger ausgerollt.

3. Cloud-Native Entwicklung

Cloud-Native ist mehr als nur eine Ansammlung von Technologien. Die Umstellung auf Cloud-Native Software ist mit erheblichen Entwicklungskosten sowie kulturellen und organisatorischen Veränderungen verbunden. Software-Anbieter müssen sich komplett neu organisieren und eine Cloud-Native Denkweise adaptieren. Der traditionelle monolithische Ansatz fördert eine konservative und vorsichtige Arbeitsweise, da ein fehlendes Semikolon ein ganzes System ausser Gefecht setzen kann. Eine Cloud-Native Philosophie hingegen unterstützt eine konstruktive Fehlerkultur, erhöht die Risikobereitschaft und bietet Entwicklern die Chance, mit neuen Funktionen zu experimentieren. Prozesse werden beschleunigt und für eine kontinuierliche Bereitstellung optimiert. Schnelle Innovationen sind daher erforderlich, um mit den sich schnell ändernden Kundenbedürfnissen Schritt zu halten. Eine Cloud-Native Denkweise umfasst alle architektonischen Möglichkeiten der Cloud in Bezug auf Flexibilität, Agilität und Skalierbarkeit.

4. Zusammenfassung



- Cloud-Native kann sich auf die Tatsache beziehen, dass eine Applikation in einer Cloud-Umgebung optimal skaliert und funktioniert. Ausserdem kann sich Cloud-Native auf die Hauptmerkmale von Cloud-Native-Applikation wie eine Microservice-Architektur beziehen.
- Cloud-Native Applikationen sind auf Microservices aufgebaut. Microservices werden in Container verpackt und mit Kubernetes dynamisch orchestriert, um die Ressourcen optimal zu verteilen.
- Zu den Vorteilen einer Microservice-Architektur gehören die praktisch unbegrenzte Skalierbarkeit sowie die erhöhte Agilität, um auf Marktanforderungen zu reagieren.
- Microservices erleichtern die Wartbarkeit von Applikationen. Updates können durchgeführt werden, ohne die Anwendung herunterzufahren. So wird die Resilienz gegenüber Ausfällen verbessert.
- Die Umstellung zur Cloud-Nativen Softwareentwicklung erfordert eine andere Denkweise, um alle architektonischen Möglichkeiten der Cloud zu nutzen.

Luware