WebApp von Server zu Serverless

Das Zeitalter der Digitalisierung hat nicht nur die Wirtschaft verändert und geprägt, sondern jeden Einzelnen von uns. Es werden immer wieder neue Lösungen und Möglichkeiten erforscht und entwickelt.

Zusammenfasung

Der technische Fortschritt betrifft nicht nur neue Geräte bei ihrer Realisierung, sondern auch für alte Produkte bieten sich neue Möglichkeiten bei der Umsetzung oder ihrer Erweiterung. Eine Möglichkeit besteht darin, dass sich so genannte WebApplikationen von ihrer klassischen Realisierung mittels Server entfernen und stattdessen mittels verschiedener Cloud Services erstellt werden. Cloud Services sind vor allem unterschiedlich und vielfältig. Hierbei zahlt der Anwender nur die Services, welche zu seinen Anforderungen am besten passen und die tatsächlich benötigt werden. Dies kann von einem klassischen Server-Konstrukt bis hin zu einer WebApplikation gehen, welche ohne oder nur begrenzt mittels eines Servers realisiert wird. Die mVISE hat sich der Herausforderung des Cloud-Computing gestellt und möchte darstellen, wie man von einem klassischen Server-Konstrukt, zu einer Serverless-Variante einer WebApplikation umstellen kann.

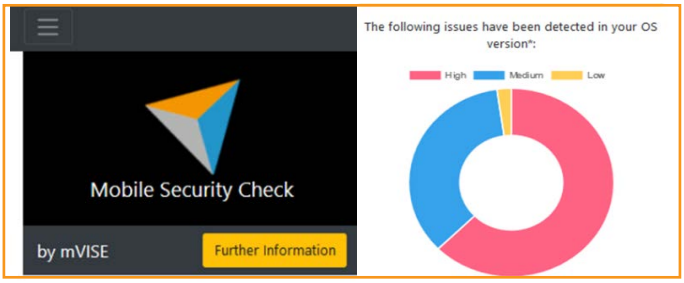
> ***"Cloud is about how you do computing, not where you do computing."***
> ***// Paul Maritz, CEO VMware***

Mobile Security Check (MSC)

Bevor es überhaupt zu der Entwicklung einer WebApplikation kam, mussten wir zunächst eine Entscheidung über ihre Funktionalität treffen. Wir entschieden uns für einen *„Mobile Security Check"* (MSC), welcher mittels *„Amazon Web Services"* (AWS) realisiert werden sollte.

UNSERE WEBAPPLIKATION SOLL DIE FOLGENDEN FUNKTIONALITÄTEN VORWEISEN: Sobald die Seite (https://msc.mvisecloud.de) von einem mobilen Android oder iOS-Gerät besucht wird, soll, abhängig von dem Betriebssystem und der zugehörigen Betriebssystemversion, die Anzahl der Sicherheitslücken aufgeführt werden. Des Weiteren sollen Informationen, wie zum Beispiel, um welche Sicherheitslücke es sich handelt (CVE Kennzeichnung), wie schwerwiegend die gefundenen Lücken sind und ähnliche relevante Informationen näher veranschaulicht werden. Eine weitere Funktionalität der WebApplikation soll die Auflistung einer Statistik zu den jeweiligen Betriebssystemen (Android oder iOS) sein. Diese Statistik soll es ermöglichen einen groben Überblick über die Anzahl von Sicherheitslücken im Hinblick zu der Betriebssystemversion zu gewährleisten.

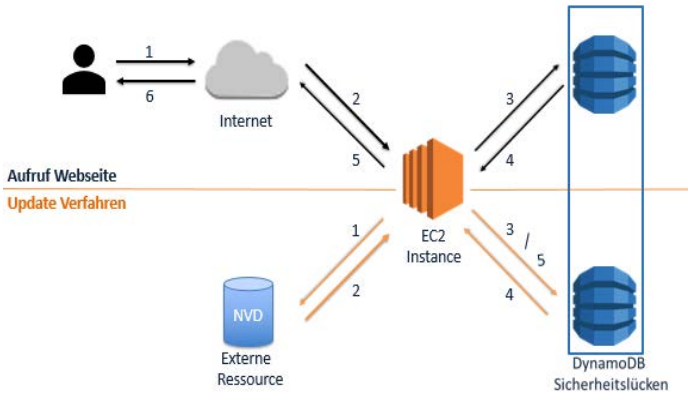
Das Betriebssystem und die Betriebssystemversion lassen sich mittels eines Programmskripts feststellen in dem der sogenannte UserAgent-String des Browsers ausgelesen wird. Er wird von dem Besucher einer Seite automatisch an die besuchte Webseite gesendet und kann dementsprechend für unsere Zwecke verwendet werden.



Umsetzung mit der klassischen Variante

Nachdem wir uns Gedanken über die Funktionalitäten der zu entwickelnde WebApplikation machten, setzten wir uns an die Realisierung mittels der klassischen Variante. Bei der klassischen Variante nutzen wir einen Server (*EC2 Instance*) mit dessen Hilfe wir ein WebApplikation hosten. Wir haben dafür das Framework *Flask* genutzt, welches für eine Python WebApplikation genutzt werden kann. Alternativ können auch andere Frameworks wie beispielsweise *Django* genutzt werden.

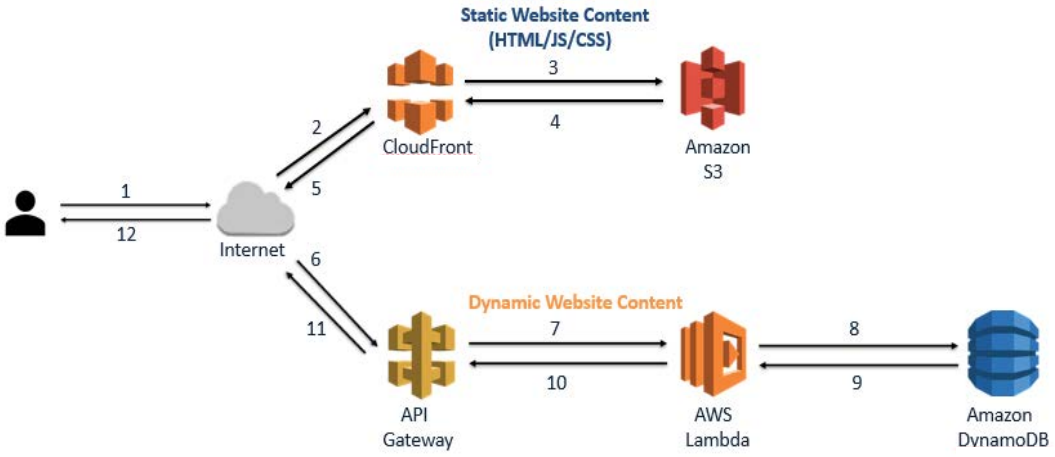
Zunächst erstellten wir mittels *EC2*-Server und *Flask* eine einfache Website, welche „*Hello*“ und im Anschluss den UserAgent des Besuchers ausgibt. Nach diesem ersten Erfolg, befassten wir uns nun mit dem Bereitstellen der Daten. Dazu erstellten wir eine nicht relationale Datenbank (*NoSQL – DynamoDB*) bei Amazon *AWS*. Wir nutzten eine NoSQL-Datenbank, da wir zu diesem Zeitpunkt noch nicht genau wussten, wie unsere Daten aussehen werden. Alternativ ist es natürlich möglich eine Datenbank direkt auf dem Server anzulegen. Wir haben uns in diesem Fall jedoch auf Grund der Einfachheit und die spätere Migration auf die Serverless-Variante für einen *AWS* Service entschieden. Nach der Erstellung der Datenbank, mussten wir diese nun mit bekannten Sicherheitslücken (*CVE*s) füllen. Dazu verwendeten wir die externe Ressource *NVD* , welche bekannte Sicherheitslücken in einem *JSON*-Format zur Verfügung stellt. Des Weiteren werden die Einträge auf der Seite regelmäßig aktualisiert. Damit wir unsere Datenbank aktualisieren können, haben wir die Daten von *NVD* auf unseren Server heruntergeladen und diese mittels verschiedener Skripte so verarbeitet, dass nur Android und iOS Sicherheitslücken auf der Datenbank gespeichert wurden. Anschließend haben wir eine weitere Datenbank angelegt, welche die Statistiken beinhaltet. Dazu haben wir ebenfalls ein Skript erstellt, welches das Vorkommen der Version mit dem jeweiligen Betriebssystem zählt und danach in die Datenbank einfügt. Da regelmäßig neue Sicherheitslücken entdeckt werden, müssen wir unsere Datenbank ebenfalls regelmäßig aktualisieren. Die verwendeten Skripte werden auf Grund dieser Tatsache mittels *Cronjobs* automatisch jede Nacht erneut ausgeführt. Nachdem wir unsere Datenbank erstellt und mit Inhalten gefüllt hatten, haben wir uns den Funktionalitäten unserer Webseite gewidmet. Der UserAgent, welchen wir zuvor nur auslasen, wurde nun verarbeitet. Dies bedeutet, dass das Betriebssystem und die Betriebssystemversion von dem UserAgent-String ausgelesen werden und die Datenbank anschließend nach möglichen Einträgen durchsucht wird. Die gefundenen Einträge werden daraufhin auf der Seite ausgegeben. Ein ähnliches Vorgehen verwenden wir bei der Statistik. Diese lässt sich per Mausklick oder Antippen auswählen. Anschließend werden, je nach ausgewählter Statistik, Daten für Android oder iOS ausgegeben.



Klassische Variante: Vorteile & Nachteile

Die klassische Variante, welche einen einzigen Server und Datenbanken nutzt, hat eine einfache und intuitive Struktur. Prozesse werden größtenteils weder zerlegt, noch ausgelagert. Dadurch, dass ein eigener Server verwendet wurde, hat man eine gewisse Kontrolle, wie etwas umgesetzt oder abgespeichert wird. Des Weiteren ist es möglich auf dem eigenen Server mit *Root*-Rechten zu agieren, sodass eine größtmögliche Kontrolle, unter der Berücksichtigung von etwaigen Sicherheitsaspekten, besteht. Zu beachten ist jedoch, dass man für diese Variante immer mehr zahlen muss, als man im Durchschnitt benötigt. Dies geschieht aus dem folgenden Grund: Da man seine Seite auch zu Peak-Zeiten funktionstüchtig halten möchte, muss man dafür Sorge tragen, dass genügend Kapazitäten zu diesem Zeitpunkt bereitstehen. Falls nun die Zugriffe auf der Webseite sinken, da die Peak-Zeiten vorbei sind, bleiben zu viele Kapazitäten übrig, welche nicht genutzt

werden. Diese müssen jedoch immer noch bezahlt werden. Eine weitere Problematik sind die Sicherheitskonzepte, welche man eigenständig entwickeln muss. Dies beinhaltet Zugangsberechtigungen, als auch Erteilung verschiedener anderer Berechtigungen. Beispielsweise kann man auf den *EC2*-Server von Amazon Credentials abspeichern, so dass man keinerlei Zugriffsmanagement betreiben muss. Dies ist jedoch nicht der Optimalfall – zu viele Berechtigungen können ausgenutzt werden, vor allem bei einer Kompromittierung des Systems bietet dies eine größere Angriffsfläche. Mittels *IAM*-Rollen (Identity & Access Management) kann man einzelne Berechtigungen für Services freigeben, ohne, dass Credentials auf dem *EC2*-Server abgespeichert werden müssen.



Umsetzung der Serverless-Variante

Die Serverless-Variante verzichtet auf einen *EC2*- Server bei der Darstellung der Webseite. Lediglich verschiedene *„Software as a Service"*-Lösungen von Amazon *AWS* werden verwendet. Bei dieser Umsetzung unterscheiden wir zwischen der statischen- und der dynamischen Darstellung einzelner Inhalte. Für die statische Darstellung der Webseite nutzen wir *Amazon S3* und *CloudFront*. *Amazon S3* ist im eigentlichen Sinne zwar ein *Storage* (Lager), jedoch ist es ebenfalls möglich eine statische Webseite zu hosten und darzustellen. Dies bedeutet, dass es uns möglich ist, mittels *Amazon S3* unsere Webapplikation, in Kombination aus HTML, JavaScript und CSS darzustellen. Wir haben uns für eine Darstellung durch *React* entschieden, da *React* eine JavaScript-Applikation ist. Anschließend verknüpfen wir *Amazon S3* mit dem Dienst *CloudFront*. Dieser Service ermöglicht es uns ein eigenes SSL-Zertifikat zu nutzen, so dass wir den Domainnamen: *msc.mvisecloud.de* verwenden können. Besuchen wir die Webseite *msc.mvisecloud.de* so werden die Inhalte von unser *React*-Applikation des *Amazon S3* Buckets angezeigt. Des Weiteren leiten wir HTTP auf HTTPS um, damit eine sicherere Verbindung gewährleistet werden kann.

VORTEILE
- Nur zahlen, was man nutzt
- Viele Möglichkeiten

NACHTEIL
- Bindung an den Cloud Provider

Der dynamische Inhalt unserer Webapplikation lässt sich mittels JavaScript-Anfragen an das Backend, welches den dynamischen Part unserer Webseite übernimmt, realisieren. Dazu verwenden wir ein API Gateway. Bei Verwendung des API Gateways sei jedoch die *„Cross-Origin Resource Sharing"* (CORS) Einstellungen zu beachten aufgrund der *„Same Origin Policy"* (SOP). Diese muss auf unseren Domainnamen gesetzt werden, damit unsere Webseite die Berechtigung erhält um die angefragten Inhalte auch darzustellen zu können. Wir verknüpfen außerdem verschiedene *AWS* Lambda-Funktionen mit dem API Gateway. Wird nun mittels JavaScript eine Anfrage an das API Gateway gesendet, so wird die AWS Lambda-Funktion ausgeführt und übergibt ihre Ergebnisse an das API Gateway. Die verschiedene Lambda-Funktionen ersetzen die Skripte, welche wir zuvor auf unseren Server genutzt hatten. Mittels AWS Lambda lassen sich die Daten in der Datenbank

(*DynamoDB*) auslesen und anschließend zurückgeben, so dass diese auf der WebApplikation graphisch dargestellt werden können. Einige der Update-Verfahren für die DynamoDB können ebenfalls mittels *AWS* Lambda-Funktionen durchgeführt werden. Jedoch muss beachtet werden, dass die *AWS* Lambda-Funktionen ein zeitliches Limit bei ihrer Ausführung besitzen (5 Minuten). Werden die 5 Minuten überschritten, so wird die Funktion abgebrochen. Alternativ kann man einen speziellen *EC2*-Server erstellen, welcher mittels Lambda-Funktion gestartet wird und nur die notwendigen Berechtigungen hat um Updates für die Datenbank automatisch durchzuführen. Anschließend fährt sich der Server selbstständig herunter. Sowohl diese Lambda-Funktion als auch andere Lambda-Funktionen können mittels eines zeitlichen Events (*Cronjob*) ausgeführt werden. Um dies zu erreichen, wird *CloudWatch* genutzt.

Serverless-Variante: Vorteile & Nachteile

Im Vergleich zu der klassischen Variante bietet die Serverless-Variante viele neue Möglichkeiten. Das Anbinden neuer Funktionen und Services wird von Amazon AWS im Allgemeinen gut unterstützt und lässt sich dementsprechend auch nachträglich schnell ändern, um neue Funktionalitäten zu integrieren. Einen Service, welchen man zum Beispiel einfügen kann um weitere Möglichkeiten zu erhalten, ist Amazon Cognito. Amazon Cognito kann zur Registrierung und Anmeldung von Nutzern, also einer Art Zugriffskontrolle, genutzt werden. Ein weiterer Vorteil im Vergleich zu der klassischen Variante ist, dass man nur das zahlt, was man wirklich nutzt. Dies bedeutet in der Regel, dass man keine überflüssigen Kapazitäten besitzen muss damit die Applikation sowohl zu Peak-Zeiten als auch zu Zeiten mit minimalen Besuchern einwandfrei laufen kann. Dementsprechend ist die Applikation leicht zu skalieren. Dies funktioniert, da sowohl Amazon Lambda-Funktionen, als auch Lese- und Schreib-Zugriffe bei einem S3 Bucket einzeln berechnet werden. Dieses Konzept lässt sich ebenfalls auf viele andere Services von Amazon AWS übertragen.
Das Konzept Serverless mag zwar etwas eigensinnig und vor allem ungewohnt auf den ersten Blick scheinen. Im Vergleich zu der klassischen Server-Variante bietet es jedoch neue Möglichkeiten der Realisierung von verschiedenen Webapplikationen, welche dem eigenem Bedarf entsprechen.

Security ist eine Entscheidung - Sie beginnt schon bei den Berechtigugen und Einstellungen
- IAM Rollen
- Security Groups
- CORS
- S3 Bucket (öffentlich oder nicht?)

Security in AWS

Sowohl bei der klassischen, als auch bei der Serverless Variante sollten Sicherheitskonzepte beachtet werden. Diese sind dazu da, das Gerät, beziehungsweise die Applikation, vor Angreifern zu schützen.
Generell gilt jedoch, dass Sicherheit eine Entscheidung ist, welche schon bei den einzelnen Berechtigungen der Nutzer, Entwicklern, Services und Ausführung verschiedener Programme beginnt, als auch bei deren Einstellungen und Wartungen. Beispielsweise können *IAM* (Identity & Access Management) Rollen bei Amazon AWS verwendet werden. Es gilt: Nur Berechtigungen die absolut notwendig sind, sollten gegeben werden (Principle of least privilege). Eine Lambda-Funktion, welche nur dazu da ist, in der Datenbank für Statistiken zu lesen und diese anschließend auszugeben, braucht keine Lese- oder gar Schreibberechtigung für eine andere Tabelle der Datenbank. Auch Security Groups, welche mit einem *EC2*-Server verknüpft werden, können, bei richtiger Einstellung, zur Sicherheit des Systems beitragen. Sie dienen als eine Art Firewall, welche Protokoll und Port beinhalten, damit man sich mit dem Server verbinden kann. Ein weiterer Aspekt welcher häufig ein Sicherheitsrisiko darstellt ist es einen *S3* Bucket öffentlich zu stellen, so dass jeder Zugriff auf die Dateien im *S3* Bucket hat. Dies mag zwar das Zugreifen auf die Dateien erleichtern, sicher ist es jedoch nicht. Da jeder alle

Daten die sich Ort befinden lesen kann. Ein Ende letzten Jahres eingetretener Fall sind öffentlich zugängliche Kreditkartendaten. Es gilt: Daten welche schützenswert sind (Stichwort: DSGVO) sollten nicht öffentlich zugänglich sein. Auch hier kommt es auf den Use-Case an. Dies sind nur einige wenige Beispiele, auf welche Dinge es zu achten gilt. Weitere Services, Wartungen und Einstellungen können natürlich die Möglichkeiten der Sicherheit erweitern.
Es gilt jedoch, dass Systeme und Programme immer auf dem aktuellsten Stand sein sollten (regelmäßiges Patchen), eine defensive Programmierung benutzt werden soll und unsichere Programmierbefehle vermieden werden sollte. Penetrations-Test sind ebenfalls eine wertvolle Ergänzung: Sicherheitsexperten schauen sich ihr System beziehungsweise Applikation genauer an und durchsuchen sie nach möglichen Sicherheitslücken und Schwachstellen.
Es ist sicherlich ein erster Schritt nachzudenken wozu man etwas braucht und nicht mehr als die absolut notwendigen Berechtigungen zu ersteilen. Damit lassen sich Fehler und vor allem Fehlerauswirkungen minimieren.

Profitieren Sie von qualifizierten Lösungen

Die mVISE AG bietet ein umfassendes Leistungsspektrum rund um das Thema Informationssicherheit, Cloud-Computing und Penetrations-Tests. Kunden profitieren von einem ganzheitlichen Lösungsansatz. mVISE unterstützt, berät und realisiert IT-Security Projekte von der Konzeption bis zur Umsetzung in allen Phasen. Die Herausforderungen der Digitalisierung löst mVISE gemeinsam mit den Kunden.

ÜBER DEN AUTOR

Bernhard Borsch ist seit 2014 für die mVISE AG tätig. Als Manager für das IT-Security Consulting beraten sein Team und er Kunden zum Thema IT-Security im Zuge der Herausforderungen der Digitalisierung. Zu seinen persönlichen Themenfeldern gehört neben den klassischen Themen, wie Enterprise- & Cloud-Security, PKI und Kryptographie, auch Zukunftsthemen der IT Security, wie Blockchain und Deception Technology. Professionelle und fachkompetente Beratung sind sein Schlüssel zum Erfolg.

mVISE

Wir unterstützen mittelständische und große Unternehmen aller Branchen dabei, von der digitalen Revolution zu profitieren. Die besondere Kombination aus firmeneigenen Software-Lösungen mit ausgewählten Experten-Teams in den relevanten und aktuellen IT-Themengebieten schafft nachhaltige Wettbewerbsvorteile für unsere Kunden.

Unsere Experten bestimmen, gestalten, kreieren und steuern IT-Infrastrukturen und Software-Lösungen für Datenintegrations- und Enterprise-Data-Management-Projekte, mit dem Ziel, die aktuellen Geschäftsmodelle unserer Kunden zukunftssicher zu machen und gleichzeitig neue Geschäftsmodelle zu identifizieren.

Sprechen Sie uns an – gerne stellen wir Ihnen unser Angebot
in einem persönlichen Gespräch näher vor.
service@mvise.de | www.mvise.de

mVISE AG
Wahler Straße 2
40472 Düsseldorf
Fon: +49 211 78 17 80 – 0