Sonderdruck für QAWARE SOFTWARE ENGINEERING

Ausgabe 05|2018

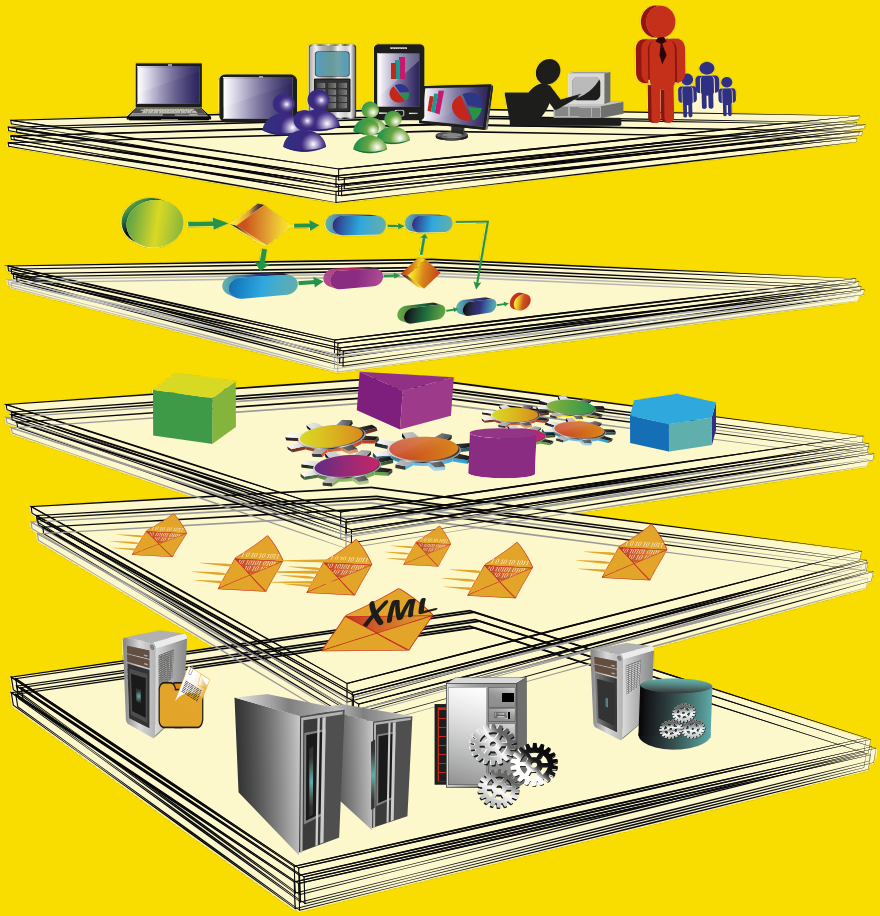
Deutschland € 9,90 Österreich € 10,90 Schweiz sfr 18,20

4 194164 109900 05

OBJEKTspektrum

IT-Management und Software-Engineering

www.OBJEKTspektrum.de



Skalierbare Architekturen

Andreas Zitzelsberger

Skalierbare Sicherheit mit SPIFFE

G 6540F

Nie wieder Zertifikate!

Skalierbare Sicherheit mit SPIFFE

Moderne Clouds bringen klassische Methoden zum Schutz von Services an und über ihre Grenzen. Das nötige Mehr an Schutz wird durch das Zero-Trust-Modell abgebildet: Vertraue niemandem, prüfe alles. SPIFFE, der neue Standard für Service-Identität, hilft, Zero-Trust-Architekturen umzusetzen. Dieser Artikel erklärt SPIFFE und zeigt, wie mit den darauf aufbauenden Werkzeugen eine skalierbare Architektur mit wenig Aufwand viel effektiver geschützt werden kann, als das bisher möglich war.

Moderne Cloud-Architekturen mischen gerade die Softwarewelt auf. Der Begriff Cloud, der früher einfach nur „anderer Leute Rechner“ bedeutete, steht jetzt für Plattformen wie Kubernetes, OpenShift, CloudFoundry oder DC/OS, die klassische Betriebsmodelle wegwischen und die Details der Infrastruktur wegabstrahieren, sodass Anwendungen die physikalische Ausprägung der Cloud herzlich egal sein kann.

Diese Plattformen bringen gegenüber dem klassischen Betrieb so immense Vorteile mit sich, dass sich derzeit auch Unternehmen der Old Economy in der Breite auf den Weg machen, ihre Anwendungen in die Cloud zu bringen.

Diese Bewegung in die Cloud führt zu Änderungen, die sowohl schwerwiegende Auswirkungen auf die Sicherheit und insbesondere die Netzwerksicherheit haben, als auch Lösungen dafür aus dem klassischen Betrieb an ihre Grenzen bringen.

SPIFFE, das „Secure Production Identity Framework For Everyone” [Gil17], bietet eine solide Grundlage für sichere, cloud-native Betriebs- und Anwendungsarchitekturen.

Der Artikel beschreibt den Problemraum, erklärt den SPIFFE-Standard und die darauf aufbauenden Werkzeuge und zeigt, wie damit ein effektiver und effizienter Schutz aufgebaut werden kann. **Kasten 1** enthält ein Verzeichnis der wichtigsten Fachbegriffe.

Das Problem

Eine moderne Cloud-Architektur verhält sich in einigen sicherheitsrelevanten Eigenschaften grundsätzlich anders als der klassische Anwendungsbetrieb im eigenen Rechenzentrum.

Vielfältige und intransparente Infrastruktur. Die Plattform als Betriebs-Abstraktion bedeutet für die Anwendungen, dass nicht mehr automatisch klar ist, wo und wie eine Anwendung betrieben wird. Es kann nicht mehr davon ausgegangen werden, dass eine Anwendung im eigenen Rechenzentrum liegt, das einfach als sicher definiert wird. Im Normalfall wird sie bei einem großen Cloud-Provider laufen und sich die Infrastruktur mit anderen teilen. Genauso ist es mit den Netzwerkstrecken.

Dynamik und Skalierung. Früher wurde die Uptime guter Software in Monaten und Jahren gemessen. Jetzt ist es normal, dass Software im laufenden Betrieb ständig rauf- und runtergefahren, skaliert und verschoben wird, vielleicht sogar über Rechenzentrumsgrenzen hinweg. Da sich IP-Adressen laufend ändern, ist klassische IP-basierte Sicherheitstechnik nutzlos.

Dekomposition von Anwendungen. Es gibt gute Gründe, Microservice-Architekturen zu bauen. Im Vergleich zum klassischen Betrieb explodiert jedoch die Zahl der Kollaborateure und Netzwerkverbindungen. Was früher tief im Inneren eines Monolithen sicher versteckt war oder in einem dicken JEE-Cluster lief, ist jetzt über das Netzwerk exponiert und angreifbar. Das Problem potenziert sich, da wenig Kontrolle über das Netzwerk besteht.

Hybrid Cloud. Was früher nur das eigene Rechenzentrum war, ist jetzt Amazon. Und Google. Und Azure. Und vielleicht auch DigitalOcean, denn man will sich natürlich nicht an einen Lieferanten binden. In vielen Fällen gibt es auch das eigene Rechenzentrum weiterhin, sei es um klassische Anwendungen weiter zu betreiben, sei es, weil es Anwendungen oder Daten

Glossar

- *Authentifizierung:* Prüfung der Identität eines Aufrufers
- *Autorisierung:* Prüfung, ob ein Aufrufer berechtigt ist, eine bestimmte Aktion durchzuführen. Authentifizierung ist notwendige Vorbedingung für Autorisierung
- *Cloud-nativ:* Anwendungen, Technologien und Architekturen, die für Cloud-Computing konzipiert sind und die Möglichkeiten moderner Cloud-Plattformen nutzen können
- *Control Group:* UNIX Control Group, auch cgroup genannt – Linux-Feature zur Isolation von Prozessen, Grundlage für Container
- *JWT:* JSON Web Tokens – Standard für Zugriffs-Token
- *Pod:* Gruppe von Containern, die zusammen ausgeführt werden und geteilte Ressourcen haben
- *Service:* Bezeichnet eigentlich ein Dienstprogramm, im Kontext dieses Artikels Synonym für das weniger gebräuchliche „Workload“
- *Unix Domain Socket:* Mittel zur Kommunikation zwischen Prozessen auf Unix-Betriebssystemen
- *Workload:* Überbegriff für „laufende Software“, beinhaltet unter anderem Dienste und Batch-Jobs
- *X.509:* Standard für eine auf asymmetrischer Kryptografie aufbauende Zertifikats-Infrastruktur, Grundlage der Transportsicherheit im Internet mit HTTPs/TLS
- *Zero Trust:* Sicherheitsmodell, bei dem Aufrufern per se kein Vertrauen entgegengebracht wird, Kommunikation wird abgesichert, als fände sie im Internet statt

Kasten 1

gibt, die nicht in der Cloud landen sollten. Alle Anwendungen sollen sicher miteinander sprechen können, ohne sich um diese Verteilung Gedanken zu machen.
Diese Eigenschaften vergrößern die Angriffsfläche deutlich, worauf nur mit besserer Sicherheit reagiert werden kann.

Zero Trust als Ziel

Das Zero-Trust-Modell ist die Antwort auf die vergrößerte Angriffsfläche in der Cloud und die wichtigste Grundlage einer cloud-nativen Sicherheitsarchitektur: Das ganze Netzwerk wird als unsicher betrachtet, jeder Service ist abgesichert, als ob er ungeschützt im Internet stünde. Das bedeutet unter anderem flächendeckende Verschlüsselung und Service-Authentifizierung.
Eine tief gehende Behandlung des Themas Zero Trust sprengt den Rahmen des Artikels. Zur Vertiefung ist das Buch „Zero Trust Networks“ [GiBa17] empfehlenswert, in dem Evan Gilman und Doug Barth die Konzepte und den Stand der Technik ausführlich darlegen.
Auf Ebene der Verbindungen bedeutet Zero Trust, dass einzelne Verbindungen geschützt werden müssen. Eine klassische Zonierung mit Firewalls, Proxies, Gateways oder Overlay-Netzen ist für eine Zero-Trust-Architektur unzureichend, da sie eine Ost-West-Bewegung eines Angreifers nicht verhindern kann. Wenn ein Service in einer Zone übernommen wird, so steht dem Angreifer die gesamte Zone offen.
Der Schutz der Verbindungen kann technisch mit m-TLS (mutual Transport Layer Security) umgesetzt werden, das heißt, TLS sowohl mit Server- als auch mit Client-Authentifizierung. Damit ist die Verbindung verschlüsselt und Server und Client sind sicher authentifiziert.
Alternativ ist auch Micro-Zoning mit einem cloud-nativen Overlay Network wie Calcio [Cas16] möglich. Micro-Zoning bedeutet, dass kleinste Netzwerk-Zellen gebildet werden, die nur die zulässigen Kommunikationspartner beinhalten. In den meisten Fällen ist das ein Client-Server-Paar.
Im Folgendem wird TLS als Beispiel für eine Absicherung auf Verbindungsebene benutzt, da TLS sehr weit verbreitet ist und die Konzepte im Rahmen dieses Artikels auf andere Methoden übertragbar sind.
Wenn zwei Services miteinander reden wollen, müssen zwei Vorbedingungen erfüllt sein:

- Beide Services brauchen eine Identität und eine technische Abbildung dieser Identität.
- Beide Services müssen in der Lage sein, die Identitäten ihrer Kommunikationspartner zu prüfen (Authentifizierung).

Bei TLS ist die technische Identität ein X.509-Zertifikat mit privatem Schlüssel. Die fachliche Identität dagegen ist meist nicht sauber definiert. Ist es ein Hostname, eine IP-Adresse, ein Distinguished Name, eine E-Mail-Adresse oder etwas völlig anderes? Wird ein technischer Nutzer verwendet, der vielleicht von mehreren Services verwendet wird oder sogar an eine Person gebunden ist?
Die Authentifizierung erfolgt, indem bei den Services die Zertifikate ihrer Kommunikationspartner in einem Trust Store hinterlegt werden.
Bei X.509 gibt es das Konzept der Delegation von Vertrauen. CAs (Certificate Authority) signieren Zertifikate sowie untergeordnete CAs und beglaubigen damit, dass der Besitzer des privaten Schlüssels auch die im Zertifikat enthaltene Identität besitzt. Anstatt zahllosen einzelnen Zertifikaten zu vertrauen, kann nun einer CA und damit implizit allen ausgestellten Zertifikaten und untergeordneten CAs vertraut werden. Im Internet ist dieses Vorgehen üblich. Betriebssysteme und Browser vertrauen einem Satz an übergeordneten CAs, was sie von der Bürde entbindet, das Zertifikat jeder aufgerufenen Website zu prüfen.
Für Zero Trust reicht dieses Vorgehen nicht aus, da ein Angreifer mit einem gültigen Zertifikat alle Services erreichen kann, die der CA vertrauen, die dieses Zertifikat signiert hat. Wird CAs vertraut, so ist eine zusätzliche Prüfung des Kommunikationspartners notwendig, zum Beispiel durch eine Access Control List, also einer Liste der vertrauenswürdigen Kommunikationspartner.
Darüber hinaus muss ein konzeptionelles Problem beachtet werden: Revocation, das heißt, für ungültig erklären einmal ausgestellter Zertifikate, kann prinzipiell nicht zuverlässig sein, da ein Angreifer mit Kontrolle über das Netzwerk einfach die Revocation-Prüfungen unterdrücken kann. Deshalb sollen Zertifikate in kurzen Zeiträumen getauscht beziehungsweise rotiert werden. Ein Angreifer hat nur so lange Zugriff, wie das Zertifikat gültig bleibt.
Mit diesen Überlegungen können wir nun die wichtigsten Eigenschaften einer cloud-nativen Zero-Trust-Architektur definieren. Diese sind in **Kasten 2** zusammengefasst.
Für einige dieser Eigenschaften gibt es schon seit Längerem gute Lösungen. So kann man zum Beispiel Kommunikationsbeziehungen über einen zentralen Secret Store zentral verwalten und bereitstellen. Kubernetes bietet mit Service Accounts eine Möglichkeit, Service-Identitäten zu verwalten, aber nur auf der Plattform Kubernetes. Wir haben also bisher eine Sammlung von Insellösungen.
Die Dreh- und Angelpunkte für fortschrittliche plattformübergreifende Lösungen sind die Service-Identität und die Lösung des Bootstrap-Problems zur automatischen Provisionierung von Service-Identitäten.

Eigenschaften einer cloud-native Zero-Trust-Architektur

- *Service-Identität:* Wohldefinierte, plattformübergreifende Identität eines Service und eine prüfbare technische Abbildung (z. B. als Zertifikat oder JWT)
- *Automatische Provisionierung der Identität:* Services sollen automatisch mit ihrer Identität versorgt werden, ohne dass sich die Services explizit authentifizieren müssen. Diese Eigenschaft wird auch Bootstrap-Problem genannt
- *Automatische Provisionierung von erlaubten Kommunikationsbeziehungen:* Services sollen automatisch mit den Identitäten der für sie zulässigen Kommunikationspartner versorgt werden. Minimal sind das die erlaubten Aufrufer eines Service
- *Zentrale Verwaltung:* Identitäten und Kommunikationsbeziehungen müssen zentral verwaltbar sein
- *Rotation der technischen Identitäten:* Die technischen Identitäten (z. B. Zertifikate) sollen in kurzen Abständen ausgetauscht werden, damit das Zeitfenster für einen Angriff auf die Laufzeit einer technischen Identität begrenzt ist

Kasten 2

SPIFFE, der Standard

Hier kommt SPIFFE ins Spiel. Die Grundideen von SPIFFE sind, einen Standard für fachliche und technische Identität von Services und eine Schnittstelle zum Abruf der Identität festzulegen und diese Identität

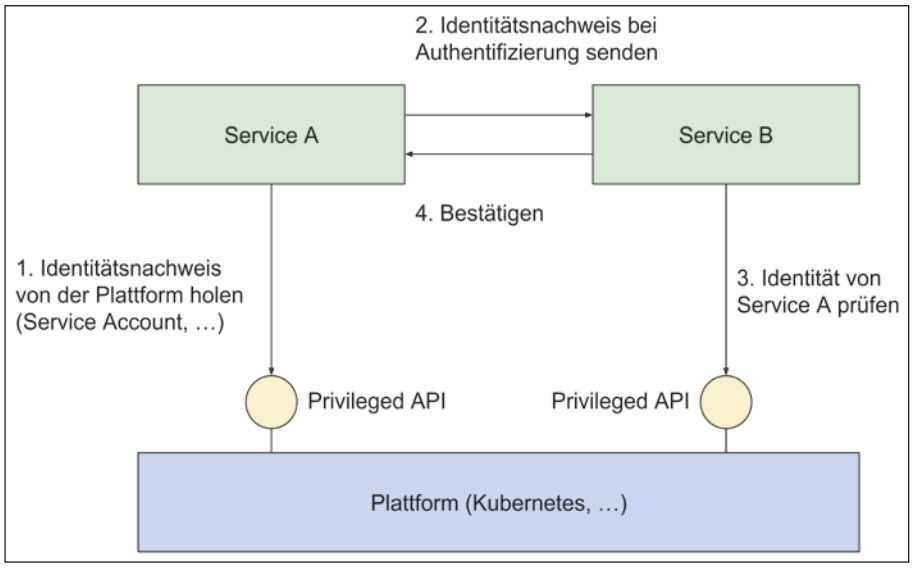


Abb. 1: Plattform-vermittelte Identität

direkt aus der Betriebsumgebung durch einen „Attestierung" genannten Prozess abzuleiten, also eine Plattform-vermittelte Identität anzubieten (**siehe Abbildung 1**).
Tatsächlich ist diese Idee nicht neu. Ähnliche Konzepte fanden sich 2005 in Googles internem LOAS (Low-Overhead Authentication System), zu dem sich allerdings nur spärlich öffentlich verfügbare Informationen finden. Identität als Infrastruktur-Dienst war ebenso bereits 2002 in dem Authentifizierungs-Protokoll Factotum von Plan 9 enthalten [Cox02]. Einige Cloud-Plattformen bieten ebenfalls Plattform-vermittelte Identitäten, die allerdings nur für die jeweilige Plattform nutzbar sind. Ein Beispiel sind die Kubernetes Service Accounts.
SPIFFE selbst wurde 2016 von Joe Beda auf der GlueCon vorgeschlagen [Bed16], 2017 auf der KubeCon NA vorgestellt und im April 2018 in die CNCF (Cloud Native Computing Foundation) als Sandbox-Projekt aufgenommen.
Was SPIFFE entscheidend von früheren Ansätzen abhebt, ist der Anspruch, eine plattform- und infrastrukturübergreifende Lösung zu schaffen, die in jedem Kontext funktioniert.
Der SPIFFE-Standard besteht aus mehreren Komponenten:
Die fachliche Identität wird in Form einer *SPIFFE ID* festgelegt (**siehe Abbildung 2**). Die SPIFFE ID ist eine URI mit vorgegebenen Format, bestehend aus Trust Domain und Pfad. Die Trust Domain beschreibt, zu welchem Authentifizierungskreis eine Identität gehört, vergleichbar dem Realm bei Kerberos. Der Pfad kann frei gewählt werden. In den meisten Fällen wird man eine Pfadhierarchie definieren, zum Beispiel entlang der Organisation.

Die technische Identität wird als *SPIFFE Verifiable Identity Document* (SVID) dargestellt. Die SVID ist quasi der Personalausweis für einen Service mit kryptografisch abgesicherter Identität. Aktuell gibt es eine technische Definition als X.509-Zertifikat, eine weitere mit JWT (JSON Web Token) ist aktuell in Arbeit. Im Zertifikat ist die SPIFFE ID als URI Subject Alternative Name (SAN) enthalten. Darüber hinaus gibt es Festlegungen, wie genau das Zertifikat auszusehen hat und welche Felder belegt sein müssen. In Summe handelt es sich um ein standardkonformes Zertifikat, das mit bestehenden Frameworks ausgewertet werden kann.
Eine SVID gilt innerhalb eines Authentifizierungs-Kreises, im SPIFFE-Kontext Trust Domain genannt. Das Konzept der *Trust Domain* ist mit den Realms aus anderen Authentifizierungs-Standards vergleichbar. Die Trust Domain ist technisch als CA-Zertifikat abgebildet. SPIFFE-Implementierungen können auch anderen Trust Domains als der eigenen vertrauen, womit hybride Szenarien über mehrere Clouds hinweg oder in einer Cloud zusammen mit On-Premises (vor Ort) umgesetzt werden können.
SPIFFE Workload API (**siehe Abbildung 3**) ist die Schnittstelle, mit der die Identität abgerufen werden kann. Weiter bietet diese Programmierschnittstelle die Informationen, die notwendig sind, um Identitäten zu prüfen. Im SPIFFE-Kontext wird der Begriff „Workload" als Bezeichnung für laufende Software verwendet.
Technisch liefert die Workload-Schnittstelle die SVID und den privaten Schlüssel für den anfragenden Service, die CA-Zertifikatskette für die aktuelle Trust Domain und CA-Zertifikatsketten für die Trust Domains, denen ebenfalls vertraut wird. Die Möglichkeit, zusätzliche CAs für weitere Trust Domains zu beziehen, ist der grundlegende Baustein für die Unterstützung von hybriden Clouds.

SPIRE, die Implementierung

SPIFFE ist der Standard für Service-Identität. SPIRE, das SPIFFE Runtime Environment (**https://spiffe.io/spire/**), ist die Referenzimplementierung der SPIFFE Workload API.
SPIRE löst das oben vorgestellte Bootstrap-Problem: Wie bekommt ein Service Identität, ohne sich authentifizieren zu müssen? Die Antwort ist überraschend banal: Frag die Plattform. SPIRE unterstützt dazu Attestator-Plug-ins, die mit Plattformmitteln die Identität von Aufrufern der Workload-Schnittstelle prüfen können.
Das Ganze läuft in einem zweistufigen Prozess (**siehe Abbildung 4**). Auf jedem Node läuft ein SPIRE-Agent und zentral für eine Trust Domain ein SPIRE-Server. Der SPIRE-Server attestiert die Nodes, auf denen die Agents laufen, die Agents die eigentlichen Workloads.
Bei Betrieb auf Kubernetes etwa funktioniert das wie folgt: Der SPIRE-Agent bietet die Workload-Schnittstelle über einem Unix Domain Socket an. Wenn sich ein Service verbindet, kann über das Betriebssystem die Prozess-Id abgerufen werden. Daraus kann die Control Group und damit der Pod abgeleitet werden, in dem die Workload läuft. Die Kubernetes-Schnittstelle gibt Auskunft, welchem Namespace und welchem Service-Account der Pod zugeordnet ist. Die Socket-Verbindung blockiert, bis der Prozess abgeschlossen ist.

Abb. 2: Die SPIFFE ID

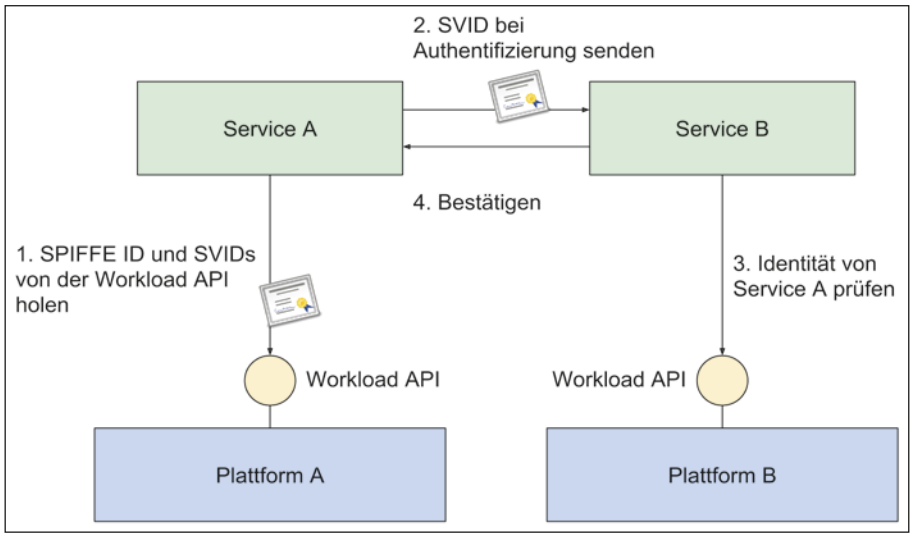


Abb. 3: Die Workload API ist die Schnittstelle zwischen SPIFFE-Implementierung (SPIRE) und nutzenden Services

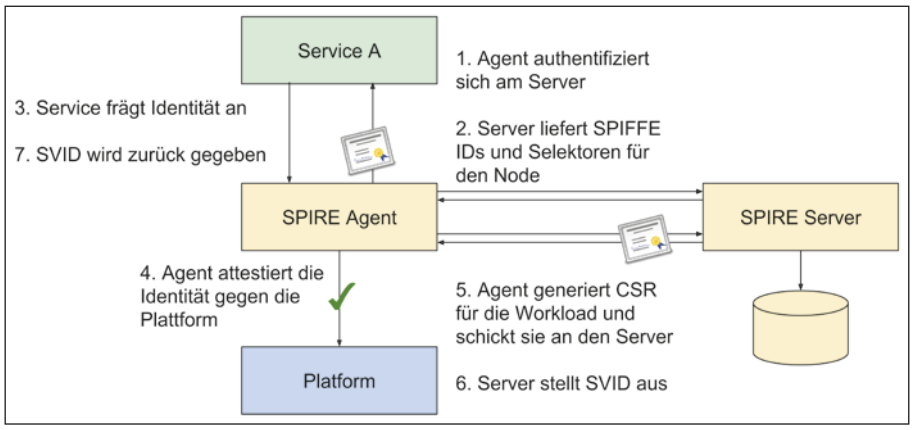


Abb. 4: Ablauf der Node- und Workload-Attestierung in SPIRE

Mit diesem Zirkelschluss kann der SPIRE-Agent auf Kubernetes Identität basierend auf dem Namespace und/oder dem Service-Account zuweisen, ohne dass der Service oder der Administrator irgendeine zusätzliche Aktion durchführen müsste.
Derzeit unterstützt SPIRE out-of-the-box verschiedene Attestierungs-Verfahren (**siehe Abbildung 5**). Für Node-Attestierung:

- Join Token (d. h. ein Shared Secret),
- AWS Instance Identity Documents (Identität einer Instanz eines Amazon Web Service).

Für Workload-Attestierung:

- AWS Instance Identity Documents,
- Unix mit Nutzer- und Gruppen-Id (UID, GID),
- Kubernetes mit auf Service Account oder Namespace.

Service Meshing

Das Fundament ist nun gelegt; mit SPIFFE steht ein übergreifender Standard für Service-Identität zur Verfügung, anhand von SPIRE wird klar, wie eine Implementierung des Standards aussieht. Zeit zu betrachten, wie auf dieses Fundament gebaut werden kann.
Für die Infrastruktur, die Querschnittsaspekte verbunden mit Kommunikation löst, hat sich im Cloud-Umfeld der Begriff „Service Mesh“ eingebürgert. Der Begriff

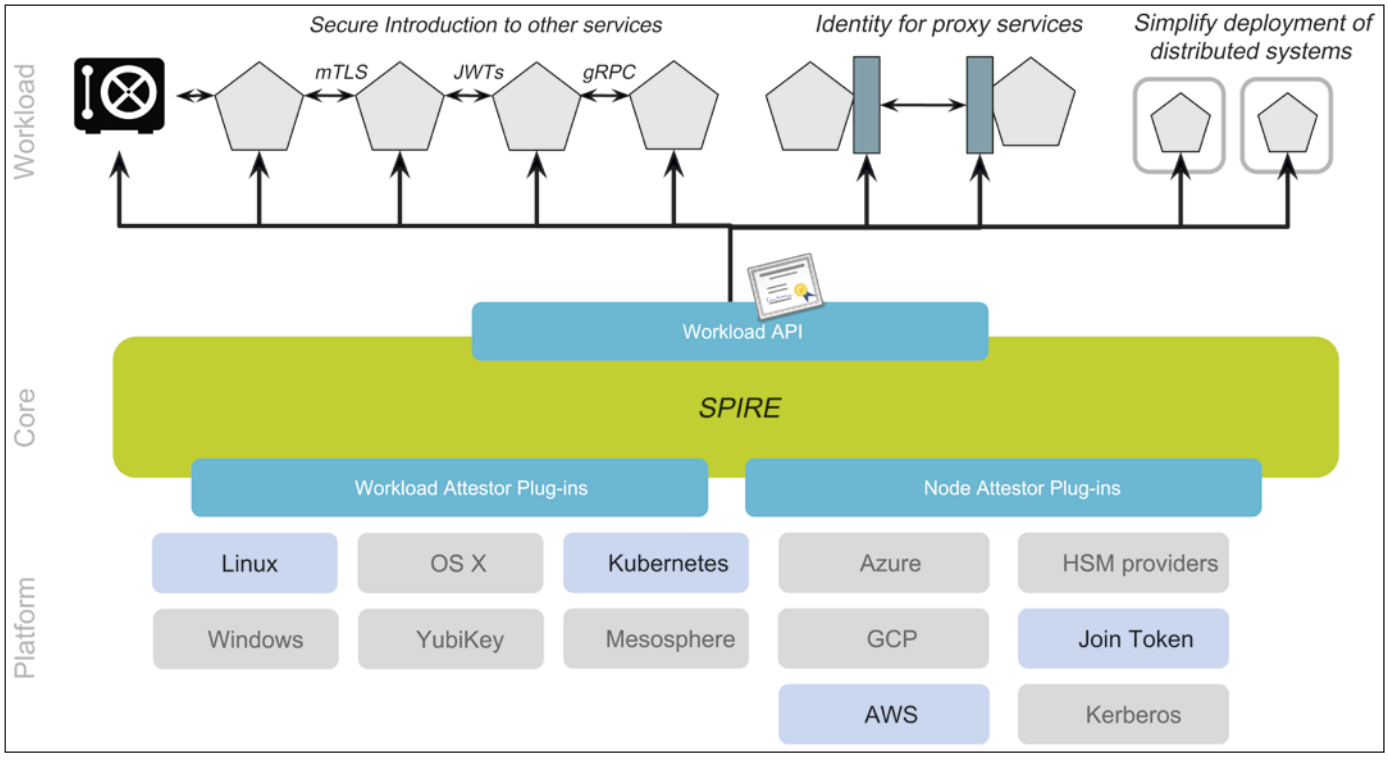


Abb. 5: SPIRE vermittelt Identität zwischen Plattform und Services (Quelle: [CNI18])

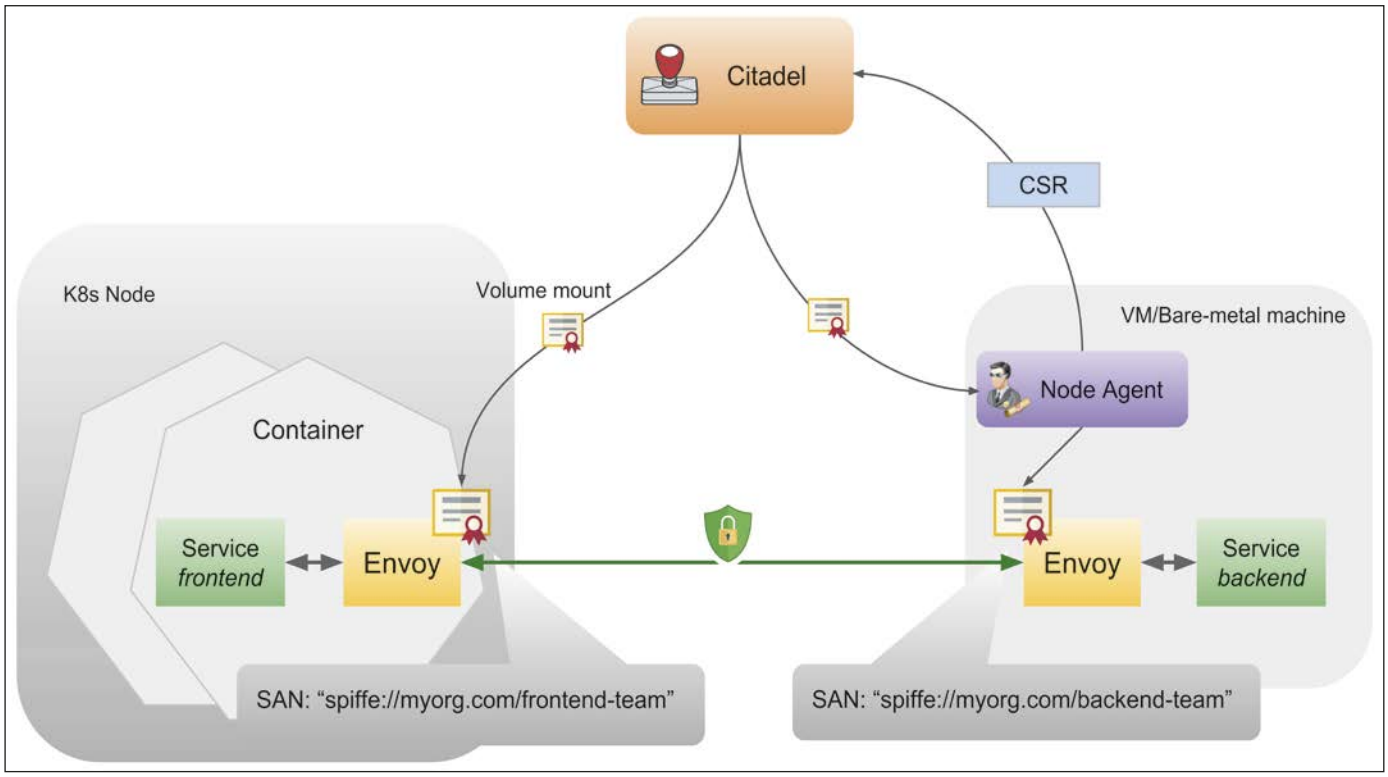


Abb. 6: Überblick über Authentifizierung in Istio (Quelle: [IsTLS])

ist nur unscharf definiert und bezeichnet sowohl ein Netz von Microservices und ihre Beziehungen als auch die Infrastruktur, die dieses Netz ermöglicht.

Die bekannteste Service-Mesh-Architektur ist Istio ([Ist18], **siehe Abbildung 6**). Istio löst Querschnittsaspekte wie Routing, Discovery, Load Balancing, Fault Tolerance, Authentifizierung, Autorisierung, Verschlüsselung und Policy, wofür natürlich eine robuste Service-Identität als Grundlage notwendig ist.

Istio verwendet im Mesh SPIFFE als Service-Identität, setzt aber nicht auf SPIRE auf, sondern implementiert SPIFFE selbst. Leider exponiert Istio weder die Workload-Schnittstelle noch die SPIFFE-Identitäten außerhalb des Service Mesh, was das Zusammenspiel mit anderen Komponenten erschwert.

Proxying

Service Meshes verwenden normalerweise das Architekturmuster des Sidecar Proxy. Das bedeutet, dass ein lokaler Proxy neben dem eigentlichen Service läuft und die Verbindungsabwicklung inklusive TLS-Terminierung übernimmt.

Dieses Architekturmuster lässt sich auch direkt verwenden, ohne gleich ein komplettes Mesh-Produkt wie Istio einzusetzen. So können Authentifizierung, Autorisierung und Verschlüsselungen aus den Anwendungen ausgelagert werden.

Envoy [Env17] ist ein cloud-nativer Proxy, der auch in Istio verwendet wird und SVIDs verarbeiten kann. Envoy kann gleichzeitig als L3-Proxy (Netzwerkschicht) und als L7-Proxy (Anwendungsschicht) für einige Protokolle wie HTTP dienen. Envoy kann also genutzt werden, beliebige IP-Protokolle abzusichern, und bietet für bestimmte Protokolle sehr viel weiter gehende Möglichkeiten. Envoy und SPIFFE können über eine kleine Hilfs-Applikation integriert werden, die Envoy mit SPIFFE-Ids versorgt.

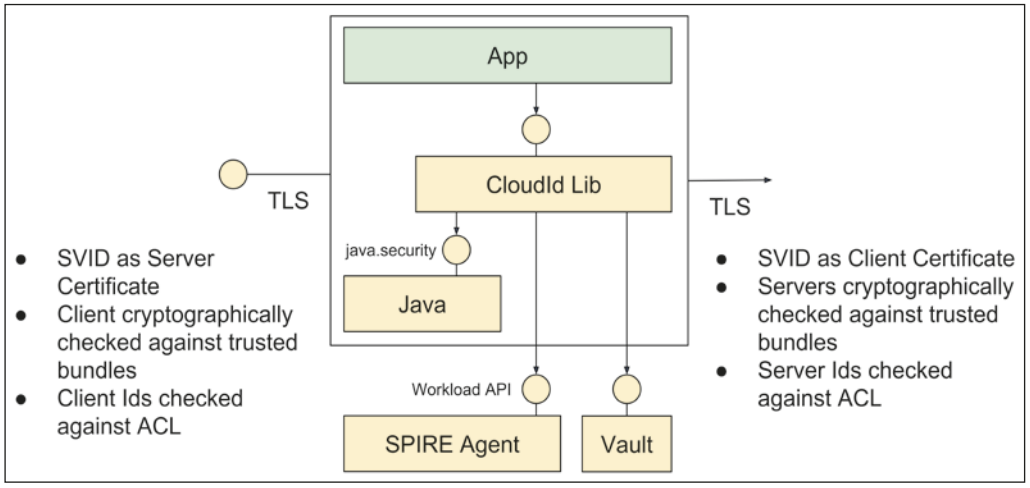


Abb. 7: Integration von SPIFFE in eine Java-Anwendung (Quelle: [CNI18])

Nginx [Ngi17] ist ein weit verbreiteter HTTP-Proxy und Webserver. Nginx lässt sich sowohl als Sidecar Proxy als auch als klassischer Reverse Proxy einsetzen und kann so helfen, Anwendungen im klassischen Betrieb an ein SPIFFE-basiertes Mesh anzuschließen. Für Nginx gibt es prototypisch direkte Unterstützung von SPIFFE durch ein eigenes SPIFFE-Modul, das die Workload-API anbindet [SpiNg].

Secret-Provisionierung

Die SPIFFE-Identität kann benutzt werden, um weitere Sec-

rets zu beziehen, zum Beispiel Credentials oder API Keys für Dienste, die sich nicht direkt in einer SPIFFE-Infrastruktur integrieren lassen. Damit kann ein SPIFFE-Mesh nahezu beliebig erweitert werden.
Vault [Vau17] ist ein prominentes Beispiel für einen zentralen, besonders sicheren Secret Store. Vault bringt einige spannende Eigenschaften mit:

- Multi-Key-Verschlüsselung der Daten mit Shamir Secret Sharing: Damit lässt sich ein Mehraugen-Prinzip erzwingen und Daten können vor Abfluss durch gezielte Angriffe auf einzelne Admins geschützt werden.
- Automatische Erzeugung von rotierenden Credentials für Datenbanken.
- Eine eigene PKI (Public Key Infrastructure), die rotierende Zertifikate erzeugen kann.

Vault unterstützt SPIFFE seit Version 0.10.2 direkt. Das Plug-in Certificate Auth kann die im URI-SAN-Feld der SVIDs enthaltenen SPIFFE IDs zur Zugriffskontrolle verwenden [VAuth].

Diagnosability

Ein nützlicher Anwendungsfall für SPIFFE ist die Korrelierung von Logs, Metriken und Traces. Der große Vorteil von SPIFFE an dieser Stelle ist das einheitliche und konsistente Format der Identitäten.
Dazu müssen Log-Einträge und Metrics-Endpoints um die SPIFFE Id angereichert werden. Mit Tools wie Elasticsearch, Kibana, Prometheus, Jaeger oder Zipkin lassen sich die Daten anschließend nach SPIFFE Ids korrelieren und damit lässt sich die Diagnostizierbarkeit einer Architektur verbessern.
Stand heute gibt es dafür keine Produkt-Lösung, allerdings auch keine Gründe, die eine eigene Implementierung mit den oben genannten Werkzeugen behindern würden.

Integration in Anwendungen

SPIFFE kann auch direkt in Anwendungen integriert werden. Auf der KubeCon EU 2018 wurde ein leichtgewichtiger Ansatz für SPIFFE-basierte Authentifizierung und Autorisierung auf Anwendungsebene vorgestellt ([CNI18], **siehe Abbildung 7**).
Im Gegensatz zu einem Service Mesh gibt es hier keine Proxies, die Authentifizierung und Autorisierung mit m-TLS löst und dazu SPIRE als Identitätsschicht und Vault als Secret Store nutzt.
Die SPIFFE-Identitäten sind für die Anwendung und nach außen jederzeit transparent. Diese Architektur wurde entwickelt, um in einem großen Brownfield-Umfeld inkrementell ausgerollt werden zu können. Dazu gibt es einen ausführlichen Showcase auf GitHub [CNISH].
Für individuelle Lösungen auf SPIFFE gibt es offizielle Clients für die SPIFFE Workload API. Aktuell sind Go, C/C++ unterstützt, ein Client für Java ist in Arbeit und bereits weit fortgeschritten. Die Workload-Schnittstelle ist aber einfach genug, um auch ohne Client direkt angebunden zu werden.

Fazit

Durch die zunehmende Verbreitung von Cloud-Technologien brauchen wir ein Mehr an Sicherheit. Die konzeptionelle Antwort darauf liefert das Zero-Trust-Modell: Jeder Service wird so behandelt, als ob er im Internet stünde. Mit klassischen Methoden sind Zero-Trust-Architekturen aber nur aufwendig zu verwalten, und jede Cloud-Plattform bietet jeweils nur Lösungen innerhalb genau dieser einen Plattform.
SPIFFE und SPIRE schließen die Lücken, um plattformübergreifende Zero-Trust-Architekturen mit geringem Verwaltungsaufwand zu bauen. Der SPIFFE-Standard bietet einen plattform- und technikübergreifenden Begriff von Service-Identität, SPIRE liefert die Referenzimplementierung dazu und macht es möglich, Service-Identität aus den unterliegenden Plattformen zu beziehen und durch einen cleveren Zirkelschluss automatisch ohne weitere Authentifizierung zu vergeben.
Damit ist SPIFFE besonders für Service Meshes wichtig. SPIFFE hilft darüber hinaus, Service-Authentifizierung an sich zu vereinfachen, egal ob in einem Infrastruktur-Produkt oder in einer individuell gebauten Anwendung.
Der SPIFFE-Standard wird in der Community gut angenommen und von vielen Projekten adaptiert. Wir dürfen also in Zukunft auf weniger Kopfschmerzen bei Umsetzung und Betrieb sicherer Architekturen hoffen. ||

Literatur & Links

[Bed16] J. Beda, Who's Calling? Production Identity in a Microservices World, GlueCon 2016

[Cas16] F. Castelli, Networking Approaches in a Container World - Flavio Castelli, LinuxCon+ContainerCon Europe 2016

[CNI18] A. Zitzelsberger, A. Jessup, Cloud Native Identity Management, KubeCon EU 2018

[CNISH] Cloud Native Identity Management Showcase, siehe: **https://github.com/qaware/cloudid-showcase**

[Cox02] R. Cox, E. Grosse, R. Pike, D. Presotto, S. Quinlan, Security in Plan 9, in: Proc. of the 11th USENIX Security Symposium, 2002

[Env17] M. Klein, Lyft's Envoy: Experiences Operating a Large Service Mesh, SRECON 2017

[GiBa17] E. Gilman, D. Barth, Zero Trust Networks, O'Reilly, 2017

[Gil17] E. Gilman, Introducing SPIFFE: An Open Standard for Identity in Cloud Native Environments, in: KubeCon + CloudNativeCon North America 2017

[Ist18] S. Ray, T. Li, M. Ahmad, Istio: Zero-trust communication security for production services, OSCON 2018

[IsTLS] Istio 0.8, Mutual TLS Authentication, siehe: **https://istio.io/docs/concepts/security/mutual-tls/**

[Ngi17] D. DeJonghe, The Complete NGINX Cookbook, O'Reilly, 2017

[SpiNg] NGINX with SPIFFE support, siehe: **https://github.com/spiffe/spiffe-nginx**

[Vau17] C. Stevens, Using Hashicorp Vault for Secrets Management, GlueCon 2017

[VAuth] TLS Certificate Auth Method (API), siehe: **https://www.vaultproject.io/api/auth/cert/index.html**

Der Autor

Andreas Zitzelsberger
(andreas.zitzelsberger@qaware.de)
ist Principal Software Architect bei QAware. Sein aktueller Schwerpunkt ist das Thema Cloud-Computing in all seinen Aspekten. Davor hat er unter anderem Big-Data-, IoT- und SOA-Architekturen entworfen und umgesetzt.

IT-Probleme lösen.
Digitale Zukunft gestalten.

QAware GmbH München
Aschauer Straße 32
81549 München
Tel.: +49 89 232315-0
Fax: +49 89 232315-129
E-Mail: info@qaware.de

QAware GmbH Mainz
Rheinstraße 4 D
55116 Mainz
Tel.: +49 6131 21569-0
Fax: +49 6131 21569-68
E-Mail: info@qaware.de

QAWARE
SOFTWARE ENGINEERING