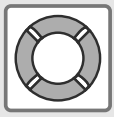
15 Service Meshes

Direkt nach Containern ist der Begriff *Service Mesh* zu einem Synonym für Cloud-native Entwicklung geworden. Aber wie Container ist auch ein Service Mesh ein allgemeinerer Begriff, der eine Reihe von Open-Source-Projekten und auch kommerzielle Produkte umfasst. Es ist hilfreich, die allgemeine Rolle eines Service Meshs in einer Cloud-nativen Architektur zu verstehen. Dieses Kapitel wird Ihnen zeigen, was ein Service Mesh ist, wie es von verschiedenen Software-Projekten implementiert wird, und schließlich (und am wichtigsten), ob es sinnvoll ist, statt einer komplexen Architektur in Ihrer Anwendung ein Service Mesh einzusetzen.



Tipp

In vielen abstrakten Diagrammen zur Cloud-nativen Architektur scheint der Einsatz eines Service Meshs zwingend dazuzugehören. Das ist aber definitiv nicht so. Wenn Sie sich überlegen, ein Service Mesh zu nutzen, müssen Sie die Komplexität berücksichtigen, die durch das Hinzufügen einer neuen Komponente (die im Allgemeinen von dritter Seite kommt) zu Ihrer Liste von Abhängigkeiten entsteht. In vielen Fällen ist es einfacher und zuverlässiger, sich schlicht auf die bestehenden Kubernetes-Ressourcen zu verlassen, wenn diese die Anforderungen Ihrer Anwendung erfüllen.

Wir haben weiter oben andere Networking-Primitive in Kubernetes beschrieben – wie zum Beispiel Services oder Ingress. Angesichts dieser Netzwerk-Fähigkeiten im Kern von Kubernetes stellt sich die Frage, warum man noch zusätzliche Fähigkeiten (und Komplexitäten) in den Networking-Layer einbauen muss. Letztendlich geht es um die Anforderungen der Software-Anwendung, die diese Networking-Primitive verwendet.

Networking im Kubernetes-Kern ist sich nur der Anwendung als Ziel bewusst. Sowohl Service- wie auch Ingress-Ressourcen haben Label-Selektoren, die den Traffic an einen bestimmten Satz von Pods leiten, aber darüber hinaus gibt es durch diese Ressourcen recht wenig zusätzliche Möglichkeiten. Als HTTP Load Balancer geht Ingress darüber ein wenig hinaus, aber die Herausforderung, eine allgemeine API zu definieren, die viele verschiedene bestehende Implementierungen abdeckt, schränkt die Fähigkeiten der Ingress-API ein. Wie kann eine wirklich »Cloud-native« HTTP-

Routing-API mit Load Balancern und Proxies kompatibel sein, die von Bare-Metal Networking Devices bis hin zu Public-Cloud-APIs reichen, die entstanden sind, ohne Cloud-native Entwicklung im Hinterkopf zu haben?

Die Entwicklung von Service-Mesh-APIs außerhalb des Kubernetes-Kerns ist eine direkte Folge dieser Herausforderung. Die Ingress-APIs bringen HTTP(S)-Traffic der Außenwelt in eine Cloud-native Anwendung. Innerhalb einer Cloud-nativen Anwendung in Kubernetes – frei von der Notwendigkeit, mit bestehender Infrastruktur kompatibel sein zu müssen – sorgen die Service-Mesh-APIs für zusätzliche Cloud-native Networking-Fähigkeiten. Worum handelt es sich dabei? Die meisten Service-Mesh-Implementierungen bieten drei verschiedene an: Netzwerk-Verschlüsselung und -Autorisierung, Traffic Shaping und Observability. Die folgenden Abschnitte werden sich mit jeder dieser Fähigkeiten befassen.

15.1 Verschlüsselung und Authentifizierung mit Mutual TLS

Das Verschlüsseln des Netzwerk-Traffics zwischen Pods ist für die Sicherheit in einer Microservice-Architektur entscheidend. Ein Verschlüsseln durch Mutual Transport Layer Security oder mTLS ist einer der verbreitetsten Anwendungsfälle für ein Service Mesh. Es ist beim Entwickeln zwar möglich, diese Verschlüsselung selbst zu implementieren, aber der Umgang mit den Zertifikaten und das Verschlüsseln des Traffics ist knifflig und nur schwer richtig zu machen. Überlässt man das Implementieren der Verschlüsselung den Entwicklungsteams, führt das dazu, dass die Verschlüsselung ganz vergessen oder nur schlecht umgesetzt wird. Bei einer schlechten Umsetzung kann die Verschlüsselung die Zuverlässigkeit beeinträchtigen und im schlimmsten Fall auch zu keinem tatsächlichen Sicherheitsgewinn beitragen. Durch das Installieren eines Service Meshs auf Ihrem Kubernetes-Cluster wird hingegen automatisch eine Verschlüsselung des Netzwerk-Traffics zwischen jedem Pod im Cluster aktiviert. Das Service Mesh ergänzt jeden Pod um einen Sidecar-Container, der transparent in die gesamte Netzwerk-Kommunikation eingreift. Neben dem Absichern der Kommunikation sorgt mTLS auch für eine Identität beim Verschlüsseln durch Client-Zertifikate, sodass Ihre Anwendung die Identität jedes Netzwerk-Clients sicher annehmen kann.

15.2 Traffic Shaping

Wenn Sie erstmals über Ihr Anwendungsdesign nachdenken, führt das im Allgemeinen zu einem übersichtlichen Diagramm, in dem es eine einzelne Box für jeden Microservice und jede Schicht im System gibt (zum Beispiel den Frontend Service, den User Preferences Service und so weiter). Bei der Implementierung in der Praxis gibt es oft mehrere Instanzen jedes Microservice der Anwendung. Führen Sie beispielsweise einen Rollout von Version X Ihres Service auf Version Y aus, gibt es einen Zeitpunkt im Rollout, bei dem Sie gleichzeitig zwei verschiedene Versionen dieses Service laufen haben. Der Übergang eines Rollouts mag zwar ein vorübergehender Zustand sein,

aber es gibt viele Situationen, in denen Sie ein länger laufendes Experiment erstellen müssen. Ein gebräuchliches Modell in der Softwarebranche ist, ein »Dog-Fooding« Ihrer eigenen Software durchzuführen – also eine neue Version der Software erst intern auszuprobieren, bevor sie auf die große, weite Welt losgelassen wird. In solch einem Modell läuft vielleicht Version Y Ihres Service einen Tag oder eine Woche (oder länger) für eine Untermenge der Anwender, bevor Sie sie allgemein allen zur Verfügung stellen.

Solche Experimente erfordern die Fähigkeit zum *Traffic Shaping* – dem Routen von Requests abhängig von ihren Eigenschaften an unterschiedliche Service-Implementierungen. In diesem Beispiel würden Sie ein Experiment aufsetzen, bei dem der gesamte Traffic der internen Benutzer Ihrer Firma an Service Y geht, während der Traffic aus der restlichen Welt immer noch zu Service X geleitet wird.

Experimente sind für eine Vielzahl von Szenarien nützlich, unter anderem in der Entwicklung, wo man beim Programmieren eine begrenzte Menge an echtem Traffic (meist 1 % oder weniger) an ein experimentelles Backend schickt. Oder Sie führen ein A/B-Experiment durch, bei dem 50 % der Anwender die eine Umgebung erhalten, die anderen 50 % aber eine andere, sodass Sie statistische Modelle aufbauen können, die zeigen, welcher Ansatz der effektivere ist. Experimente sind eine unglaublich gute Möglichkeit, für mehr Zuverlässigkeit, Agilität und Einblicke in Ihre Anwendung zu sorgen, aber oft lassen sie sich in der Praxis nur schwierig umsetzen, weshalb sie nicht so oft zum Einsatz kommen, wie sie es sollten.

Service Meshes ändern das, indem sie das Experimentieren in das Mesh selbst einbauen. Statt Code zum Implementieren Ihres Experiments zu schreiben oder eine ganz neue Kopie Ihrer Anwendung auf neuer Infrastruktur zu deployen, definieren Sie deklarativ die Parameter des Experiments (10 % des Traffics an Version Y, 90 % an Version X) und das Service Mesh implementiert das dann für Sie. Sie sind zwar beim Entwickeln in das Definieren des Experiments involviert, aber die Implementierung ist transparent und geschieht automatisch, sodass viel mehr Experimente laufen können und damit Zuverlässigkeit, Agilität und Einblicke entsprechend steigen.

15.3 Introspection

Wenn Sie so wie die meisten Menschen beim Entwickeln sind, werden Sie ein Programm nach dem ersten Schreiben wiederholt debuggen, da immer neue Fehler auftauchen. Mit der Fehlersuche im Code verbringt man in der Entwicklung einen Großteil des Tages. Das Debuggen ist noch schwieriger, wenn Anwendungen auf mehrere Microservices verteilt sind. Es ist schwierig, einen einzelnen Request zusammenzubringen, wenn er über mehrere Pods verteilt ist. Die Informationen, die zum Debuggen erforderlich sind, müssen aus mehreren Quellen zusammengeführt werden, wenn die relevanten Informationen überhaupt vorhanden sind.

Eine automatische Introspection ist eine weitere wichtige Fähigkeit von Service Meshes. Weil diese in der gesamten Kommunikation zwischen Pods involviert sind, weiß das Service Mesh, wohin Requests geroutet wurden, und kann die erforder-

lichen Informationen bereitstellen, um wieder einen vollständigen Request-Trace zusammenzusetzen. Statt einen Berg von Requests an eine Reihe verschiedener Microservices vor sich zu haben, kann man beim Entwickeln einen einzelnen, aggregierten Request erhalten, der die User Experience ihrer vollständigen Anwendung enthält. Zudem ist das Service Mesh für ein ganzes Cluster implementiert. So funktioniert das Request Tracing unabhängig davon, welches Team den Service entwickelt hat. Die Monitoring-Daten werden für alle Services konsistent in einem Cluster-weiten Service Mesh zusammengetragen.

15.4 Brauchen Sie wirklich ein Service Mesh?

Die hier beschriebenen Vorteile verleiten Sie vielleicht dazu, auf jeden Fall ein Service Mesh auf Ihrem Cluster installieren zu wollen. Aber vorher sollten Sie sich überlegen, ob das für Ihre Anwendung wirklich notwendig ist. Ein Service Mesh ist ein verteiltes System, das Ihr Anwendungsdesign komplexer macht. Es ist stark mit der zentralen Kommunikation Ihrer Microservices verknüpft. Hat ein Service Mesh ein Problem, kommt Ihre gesamte Anwendung zum Stillstand. Setzen Sie ein Service Mesh ein, müssen Sie sicher sein, dass Sie entsprechende Probleme auch beheben können. Zudem müssen Sie ein Auge auf die Software-Releases des Service Meshs haben, um sicherzustellen, dass Sie immer die letzten Sicherheitspatches und Bug Fixes implementiert haben. Und wenn neue Fixes zur Verfügung stehen, müssen Sie zudem die neue Version ausrollen können, ohne dass dies Einfluss auf Ihre Anwendung hat. Dieser zusätzliche operationale Overhead führt dazu, dass ein Service Mesh für viele kleine Anwendungen für unnötige Komplexität sorgt.

Nutzen Sie Kubernetes als Managed Service, der auch ein Service Mesh enthält, ist es viel einfacher, dieses Mesh zu verwenden, weil der Cloud-Provider den Support, das Debugging und das problemlose Einspielen neuer Releases übernimmt. Aber auch wenn ein Service Mesh von einem Cloud-Provider bereitgestellt wird, muss sich Ihre Entwicklung mit der zusätzlichen Komplexität herumschlagen. Letztendlich muss jedes Anwendungs- oder Plattform-Team auf Ebene des Clusters abwägen, ob die Kosten durch die Vorteile aufgewogen werden. Um die Vorzüge eines Service Meshs zu maximieren, hilft es, wenn im Cluster alle Microservices gleichzeitig darauf umsteigen.

15.5 Introspection einer Service-Mesh-Implementierung

Es gibt viele verschiedene Projekte und Implementierungen eines Service Meshs im Cloud-nativen Ökosystem, aber die meisten greifen auf gleiche Design-Patterns und Technologie zurück. Weil das Service Mesh transparent in den Netzwerk-Traffic Ihres Anwendungs-Pods eingreift, diesen anpasst und im Cluster eventuell umleitet, muss ein Teil des Service Meshs auf jedem Ihrer Pods vorhanden sein. Es würde zu deutlichen Reibungsverlusten führen, wenn alle Entwickler gezwungen wären, jedes Container-Image zu ergänzen, zudem wäre es viel schwieriger, die Version des Service

Meshs zentral zu managen. Daher fügen die meisten Implementierungen von Service Meshes jedem Pod im Mesh einen Sidecar-Container hinzu. Weil das Sidecar im gleichen Netzwerk-Stack wie der Anwendungs-Pod sitzt, kann es Tools wie `iptable` oder neuerdings `eBPF` nutzen, um den Netzwerk-Verkehr aus Ihrem Anwendungs-Container im Service Mesh unter die Lupe zu nehmen und anzupassen.

Natürlich ist es nahezu genauso aufwendig wie ein Anpassen des Container-Image, jede Programmiererin und jeden Programmierer dazu zu nötigen, ein weiteres Container Image in die Pod-Definition aufzunehmen. Daher nutzen die meisten Service-Mesh-Implementierungen einen angepassten Admission Controller, der das Service-Mesh-Sidecar automatisch allen Pods hinzugibt, die in einem bestimmten Cluster erstellt werden. Jeder REST-API-Request zum Erstellen eines Pods wird zunächst an diesen Admission Controller umgeleitet. Dieser passt die Pod-Definition an und fügt das Sidecar hinzu. Weil dieser Admission Controller vom Cluster-Administrator installiert wird, implementiert er transparent und konsistent ein Service Mesh für ein ganzes Cluster.

Aber das Service Mesh ist nicht nur dazu da, in das Pod-Netzwerk einzugreifen. Sie müssen auch dazu in der Lage sein, das Verhalten des Service Meshs zu steuern – zum Beispiel durch das Definieren von Routing-Regeln für Experimente oder Zugriffsbeschränkungen für Services im Mesh. Wie alles andere in Kubernetes auch werden diese Ressourcen-Definitionen deklarativ per JSON oder YAML angegeben, die Sie mit `kubectl` oder anderen Tools, die mit dem API-Server von Kubernetes kommunizieren, erstellen. Service-Mesh-Implementierungen machen sich *Custom Resource Definitions* (CRDs) zunutze, um Ihr Kubernetes-Cluster um spezialisierte Ressourcen zu ergänzen, die nicht Teil der Standard-Installation sind. In den meisten Fällen sind diese Custom Resources eng mit dem Service Mesh selbst verbunden. In der CNCF gibt es aktuell Bestrebungen, mit einem Service Mesh Interface (SMI) einen herstellerneutralen Standard zu definieren, den viele verschiedene Service Meshes implementieren können.

15.6 Service-Mesh-Landschaft

Der kniffligste Aspekt der Service-Mesh-Landschaft kann darin bestehen, herauszufinden, welches Mesh man nehmen sollte. Bisher hat sich keines der Meshes als De-facto-Standard herauskristallisiert. Konkrete Statistiken lassen sich zwar nur schwer finden, aber das beliebteste Service Mesh ist vermutlich das Istio-Projekt. Neben Istio gibt es noch viele andere Open-Source-Meshes, unter anderem Linkerd, Consul Connect und Open Service Mesh. Zudem gibt es kommerzielle Meshes wie AWS App Mesh. Wir gehen davon aus, dass sich diese Schnittstellen in den kommenden Jahren in der Cloud-nativen Community vereinheitlichen werden.

Wie soll man sich nun als Entwickler oder Cluster-Administratorin entscheiden? Ehrlich gestanden ist das beste Service Mesh das, das Ihr Cloud-Provider für Sie managt. Es ist im Allgemeinen unnötig, noch Operations für ein Service Mesh aufbauen zu müssen, weil die Aufgaben Ihrer Cluster-Operatoren schon kompliziert genug sind. Viel besser ist es, einen Cloud-Provider das für Sie erledigen zu lassen.

Wenn das für Sie keine Option ist, müssen Sie sich selbst schlaumachen. Lassen Sie sich nicht durch schicke Präsentationen und Funktionalitätsversprechungen überreden. Ein Service Mesh lebt tief in Ihrer Infrastruktur und jeder Ausfall kann die Verfügbarkeit Ihrer Anwendung deutlich beeinträchtigen. Und weil Service-Mesh-APIs dazu tendieren, spezifisch für eine Implementierung zu sein, ist es schwierig, Ihre Wahl zu ändern, wenn Sie erst einmal Anwendungen damit gebaut haben. Letztendlich kann es auch sein, dass das richtige Mesh für Sie gar kein Mesh ist.

15.7 Zusammenfassung

Service Meshes bieten eine leistungsfähige Funktionalität, durch die Ihre Anwendungen sicherer und flexibler werden können. Gleichzeitig bringt ein Service Mesh zusätzliche Komplexität für Ihre Cluster-Operations mit und es ist eine potenzielle Quelle für Ausfälle. Wägen Sie Vor- und Nachteile eines Service Meshs für Ihre Infrastruktur sorgfältig ab. Wenn Sie die Wahl haben, nehmen Sie einen Managed Service, bei dem jemand anderes für die Operation-Details verantwortlich ist, Ihre Anwendungen aber von den Fähigkeiten des Meshs profitieren.