AWS-Whitepaper

Implementieren von Microservices in AWS

Implementieren von Microservices in AWS: AWS-Whitepaper

Table of Contents

Implementieren von Microservices in AWS

Veröffentlichungsdatum: 9. November 2021 (Dokumentverlauf und Mitwirkende)

Überblick

Microservices sind ein architektonischer und organisatorischer Ansatz der Softwareentwicklung. Sie dienen dazu, Bereitstellungszyklen zu beschleunigen, Innovationen und eigenverantwortliche Beiträge zu fördern, die Verwaltbarkeit und Skalierbarkeit von Softwareanwendungen zu verbessern und den Umfang der von Unternehmen gelieferten Software und Services zu erhöhen. Die Basis dafür bildet ein agiler Ansatz, der Teams ein unabhängiges Arbeiten ermöglicht. Bei einem Microservices-Ansatz besteht die Software aus kleinen Services (Microservices), die über genau definierte Programmierschnittstellen (APIs) kommunizieren, die unabhängig bereitgestellt werden können. Diese Services sind das Eigentum kleiner, autonomer Teams. Der agile Ansatz ist für die erfolgreiche Skalierung Ihres Unternehmens ausschlaggebend.

Beim Entwickeln von Microservices durch unsere AWS-Kunden lassen sich drei gängige Muster erkennen: API-gesteuert, ereignisgesteuert und Daten-Streaming. In diesem Whitepaper erhalten Sie einen Überblick über alle drei Ansätze und die gemeinsamen Merkmale von Microservices. Es werden die wesentlichen Herausforderungen beim Entwickeln von Microservices erörtert und es wird beschrieben, wie Produktteams diese mithilfe von Amazon Web Services (AWS) bewältigen können.

Aufgrund der Komplexität der verschiedenen Themen, die in diesem Whitepaper behandelt werden, einschließlich Datenspeicher, asynchrone Kommunikation und Service Discovery, wird empfohlen, dass Sie zusätzlich zu den bereitgestellten Anleitungen spezifische Anforderungen und Anwendungsfälle ihrer Anwendungen berücksichtigen, bevor Sie architektonische Entscheidungen treffen.

Einführung

Microservice-Architekturen sind kein völlig neuer Ansatz der Softwareentwicklung, sondern eine Kombination aus verschiedenen erfolgreichen Konzepten, die sich bewährt haben, wie beispielsweise:

- Agile Softwareentwicklung
- Serviceorientierte Architekturen

- API-First-Design
- Continuous Integration (Kontinuierliche Integration)/Continuous Delivery (Kontinuierliche Bereitstellung) (CI/CD)

In vielen Fällen werden Designmuster der Twelve-Factor App für Microservices genutzt.

In diesem Whitepaper werden zunächst verschiedene Aspekte einer hochskalierbaren, fehlertoleranten Microservice-Architektur (Benutzeroberfläche, Microservice-Implementierung und Datenspeicher) erörtert und beschrieben, wie diese auf AWS mithilfe von Containertechnologien entwickelt werden kann. Anschließend erhalten Sie Empfehlungen für AWS-Services, mit denen sich eine typische Serverless Microservice-Architektur am besten implementieren lässt, um Betriebsabläufe zu vereinfachen.

Serverless bezeichnet ein Betriebsmodell, das auf folgenden Grundsätzen basiert:

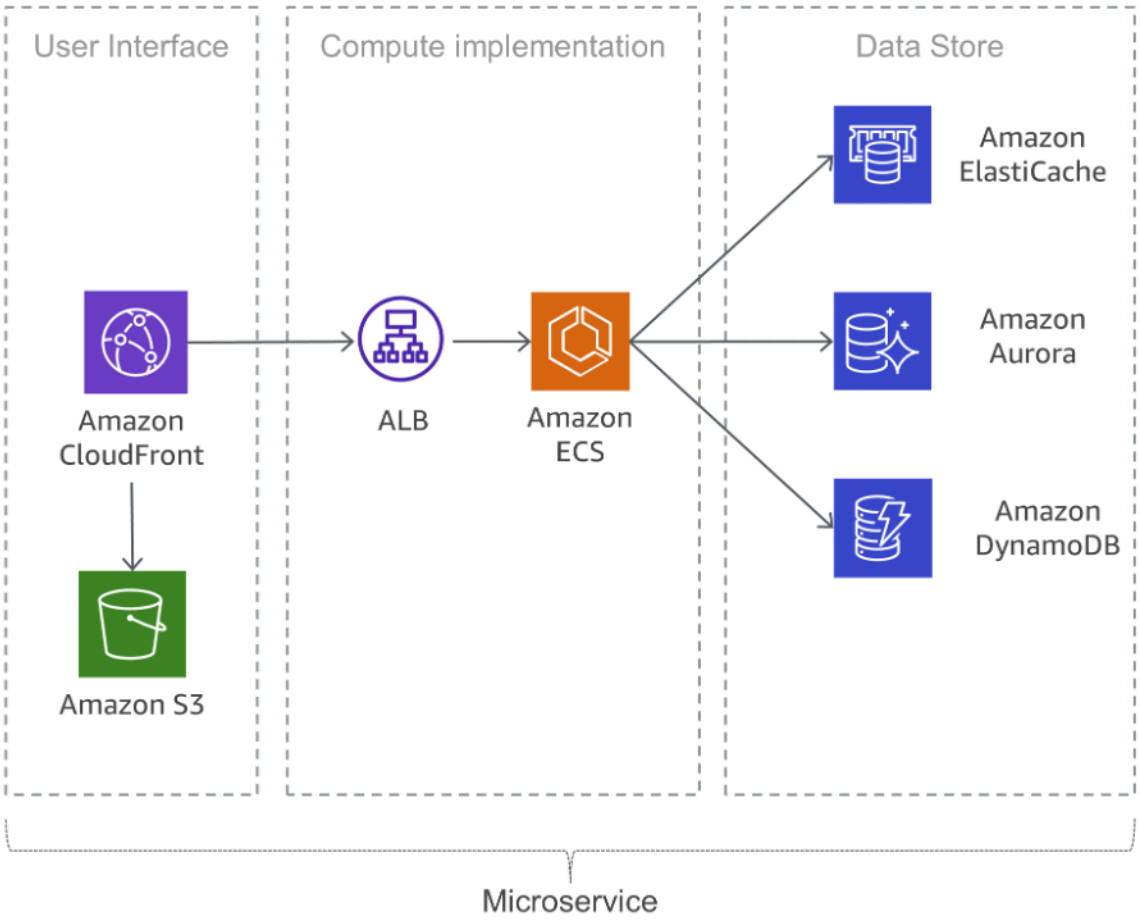
- Keine Infrastruktur bereitzustellen oder zu verwalten
- Automatische nutzungsbasierte Skalierung
- Pay-for-Value-Abrechnungsmodell
- Integrierte Verfügbarkeit und Fehlertoleranz

Schließlich behandelt dieses Whitepaper das Gesamtsystem und beschreibt die serviceübergreifenden Aspekte einer Microservice-Architektur, wie die verteilte Überwachung und Prüfung, Datenkonsistenz sowie die asynchrone Kommunikation.

Dieses Whitepaper konzentriert sich nur auf Workloads, die in der AWS Cloud ausgeführt werden. Hybridszenarien oder Migrationsstrategien werden nicht behandelt. Weitere Informationen zur Migration finden Sie im Whitepaper zur Methodik der Container-Migration.

Microservice-Architektur in AWS

Monolithische Anwendungen sind in der Regel aus verschiedenen Schichten aufgebaut, beispielsweise einer Benutzeroberfläche (UI), einer Business-Schicht und einer Persistenzschicht. Eine zentrale Idee einer Microservice-Architektur ist die Aufteilung von Funktionalitäten in kohärente Vertikalen – nicht nach technologischen Schichten, sondern durch die Implementierung einer bestimmten Domäne. Die folgende Abbildung zeigt eine Referenzarchitektur für eine typische Microservice-Anwendung in AWS.



Typische Microservice-Anwendung in AWS

Themen

- Benutzeroberfläche
- Microservices
- Datastore

Benutzeroberfläche

Moderne Webanwendungen verwenden häufig JavaScript-Frameworks, um eine aus nur einer Seite bestehende Anwendung zu implementieren, die mit einer RESTful-API (Representational State Transfer) kommuniziert. Statische Webinhalte können mit dem Amazon Simple Storage Service (S3) und Amazon CloudFront bereitgestellt werden.

Da die Clients eines Microservices vom nächstgelegenen Edge-Standort bedient werden und Antworten entweder aus einem Cache oder einem Proxy-Server mit optimierten Verbindungen zum Ursprung erhalten, können Latenzen deutlich reduziert werden. Microservices, die nahe beieinander ausgeführt werden, profitieren jedoch nicht von einem Content Delivery Network. Dieser Ansatz kann in der Tat mitunter die Latenz erhöhen. Es hat sich bewährt, andere Caching-Mechanismen zu implementieren, um Chats zu reduzieren und Latenzen zu minimieren. Weitere Informationen erhalten Sie im Thema the section called “Kommunikationsaufwand”.

Microservices

APIs sind das Tor zu Microservices, was bedeutet, dass APIs als Einstiegspunkt für die Anwendungslogik hinter einer Reihe programmatischer Schnittstellen dienen. Dabei handelt es sich in der Regel um eine RESTful-API für Webservices. Diese API akzeptiert und verarbeitet Aufrufe von Clients und implementiert möglicherweise Funktionen wie Traffic-Management, Request-Filterung, Routing, Caching, Authentifizierung und Autorisierung.

Implementierung von Microservices

AWS hat Bausteine integriert, um die Entwicklung von Microservices zu erleichtern. Zwei gängige Ansätze sind die Verwendung von AWS Lambda und Docker-Containern mit AWS Fargate.

Mit AWS Lambda laden Sie einfach Ihren Code hoch und überlassen Lambda all das, was für die Ausführung und die Skalierung der Ausführung erforderlich ist, um Ihre tatsächliche Nachfragekurve mit hoher Verfügbarkeit zu erfüllen. Es ist keine Verwaltung der Infrastruktur erforderlich. Lambda unterstützt mehrere Programmiersprachen und kann von anderen AWS-Services ausgelöst oder direkt aus jeder Webanwendung oder mobilen Anwendung aufgerufen werden. Zu den wesentlichen Vorteilen von AWS Lambda zählt die effiziente Vorgehensweise. Da AWS die Verwaltung der Sicherheit und Skalierung übernimmt, können Sie sich auf Ihre Geschäftslogik fokussieren. Der Opinionated-Ansatz von Lambda unterstützt die Skalierbarkeit der Plattform.

Eine typische Maßnahme zur Reduzierung des operativen Aufwands für den Einsatz ist die containerbasierte Bereitstellung. Containertechnologien wie Docker haben in den vergangenen

Jahren aufgrund ihrer Portierbarkeit, Produktivität und Effizienz an Popularität gewonnen. Die Lernkurve kann in Verbindung mit Containern steil sein. Außerdem müssen Sie sich Gedanken über Sicherheitsvorkehrungen für Ihre Docker-Images und die Überwachung machen. Mit Amazon Elastic Container Service (Amazon ECS) und Amazon Elastic Kubernetes Service (Amazon EKS) erübrigt sich das Installieren, Betreiben und Skalieren Ihrer eigenen Infrastruktur für die Cluster-Verwaltung. Mit API-Aufrufen können Sie Docker-fähige Anwendungen starten und stoppen, den kompletten Zustand Ihres Clusters abfragen und auf viele bekannte Funktionen wie Sicherheitsgruppen, Lastenausgleich, Amazon Elastic Block Store (Amazon EBS)-Volumes und AWS Identity and Access Management (IAM)-Rollen zugreifen.

AWS Fargate ist eine Serverless-Computing-Engine für Container, die sowohl mit Amazon ECS als auch mit Amazon EKS funktioniert. Mit Fargate brauchen Sie sich keine Gedanken mehr über die Bereitstellung der für Ihre Containeranwendungen erforderlichen Rechenressourcen zu machen. Mit Fargate lassen sich Zehntausende Container starten und auf einfache Weise für die Ausführung Ihrer geschäftskritischen Anwendungen skalieren.

Amazon ECS unterstützt Containerplatzierungsstrategien und -beschränkungen. Sie können somit festlegen, wie Aufgaben mit Amazon ECS platziert und beendet werden. Eine Platzierungsbeschränkung ist eine Regel, die bei der Aufgabenplatzierung berücksichtigt wird. Sie können Ihren Container-Instances Attribute, die im Prinzip Schlüssel-Wert-Paare sind, zuweisen und dann mit diesen Attributen in Verbindung mit einer Beschränkung Aufgaben platzieren. Mit Beschränkungen lassen sich beispielsweise bestimmte Microservices basierend auf dem Instance-Typ oder der Instance-Funktion platzieren, etwa auf GPU-basierten Instances.

Amazon EKS führt aktuelle Versionen der Open-Source-Software Kubernetes aus. Sie können somit alle bestehenden Plug-Ins und Tools der Kubernetes-Community verwenden. In Amazon EKS ausgeführte Anwendungen sind vollständig mit Anwendungen kompatibel, die in Kubernetes-Standardumgebungen ausgeführt werden – sowohl in lokalen Rechenzentren als auch in öffentlichen Clouds. Amazon EKS integriert IAM in Kubernetes. Sie können IAM-Entitäten dadurch über das native Authentifizierungssystem von Kubernetes registrieren. Das manuelle Einrichten von Anmeldeinformationen für die Authentifizierung bei der Kubernetes-Steuerebene erübrigt sich somit. Dank der Integration von IAM können Sie sich damit direkt bei der Steuerebene selbst authentifizieren und erhalten detaillierte Zugriffsrechte für den öffentlichen Endpunkt Ihrer Kubernetes-Steuerebene.

In Amazon ECS verwendete Docker-Images können in der Amazon Elastic Container Registry (Amazon ECR) gespeichert werden. Mit Amazon ECR erübrigen sich der Betrieb und die Skalierung der für Ihre Container-Registry erforderlichen Infrastruktur.

Continuous Integration und Continuous Delivery (CI/CD) sind bewährte Methoden und wichtiger Bestandteil einer DevOps-Initiative. Sie können damit schnell Softwareänderungen vornehmen, während die Stabilität und Sicherheit des Systems aufrechterhalten bleiben. Diese Methoden werden jedoch in diesem Whitepaper nicht behandelt. Weitere Informationen finden Sie im Whitepaper Continuous Integration und Continuous Delivery in AWS.

Private Links

AWS PrivateLink ist eine hochverfügbare, skalierbare Technologie, die Ihnen ermöglicht, Ihre Virtual Private Cloud (VPC) privat mit unterstützten AWS-Services, von anderen AWS-Konten gehostete Services (VPC-Endpunktservices) und unterstützten AWS Marketplace-Partnerservices zu verbinden. Sie benötigen für die Kommunikation mit dem Service weder ein Internet-Gateway noch ein NAT-Gerät, eine öffentliche IP-Adresse oder eine AWS Direct Connect- oder VPN-Verbindung. Der Datenverkehr zwischen Ihrer VPC und dem Service verlässt das Amazon-Netzwerk nicht.

Private Links sind eine hervorragende Möglichkeit, die Isolation und Sicherheit der Microservices-Architektur zu erhöhen. Ein Microservice könnte beispielsweise in einer völlig separaten VPC bereitgestellt werden, mit einem Load Balancer verbunden und über einen AWS PrivateLink-Endpunkt anderen Microservices zugänglich gemacht werden. Mit diesem Setup, das AWS PrivateLink verwendet, durchläuft der Netzwerkverkehr zum und vom Microservice niemals das öffentliche Internet. Ein Anwendungsfall für eine solche Isolierung ist die Einhaltung gesetzlicher Vorschriften für Services, die sensible Daten wie PCI, HIPPA und EU/US Privacy Shield verarbeiten. Mit AWS PrivateLink können Sie Ihre Microservices über mehrere Konten und Amazon VPCs verbinden – ganz ohne Firewallregeln, Pfaddefinitionen oder Routing-Tabellen. Somit wird das Netzwerkmanagement vereinfacht. Mithilfe von PrivateLink können Software as a Service (SaaS)-Anbieter und ISVs ihre Microservices-basierten Lösungen auch mit vollständiger betrieblicher Isolation und sicherem Zugriff anbieten.

Datastore

Der Datenspeicher wird verwendet, um die von den Microservices benötigten Daten zu erhalten. Beliebte Speicher für Sitzungsdaten sind In-Memory-Caches wie Memcached oder Redis. AWS bietet beide Technologien im Rahmen des verwalteten Amazon ElastiCache-Services an.

Das Einrichten eines Caches zwischen Anwendungsservern und einer Datenbank ist ein gängiger Mechanismus, um die Leselast in der Datenbank zu verringern. Dadurch können mitunter mehr Ressourcen für Schreibvorgänge genutzt werden. Caches können auch die Latenzzeiten verbessern.

Relationale Datenbanken sind zum Speichern strukturierter Daten und Business-Objekte nach wie vor sehr beliebt. AWS bietet sechs Datenbank-Engines (Microsoft SQL Server, Oracle, MySQL, MariaDB, PostgreSQL und Amazon Aurora) als verwaltete Services über den Amazon Relational Database Service (Amazon RDS) an.

Relationale Datenbanken sind jedoch nicht für die endlose Skalierung konzipiert. Daher kann die Anwendung von Techniken zur Unterstützung einer hohen Anzahl von Abfragen schwierig und zeitintensiv sein.

NoSQL-Datenbanken wurden entwickelt, um Skalierbarkeit, Leistung und Verfügbarkeit gegenüber der Konsistenz relationaler Datenbanken zu verbessern. Ein wichtiger Faktor von NoSQL-Datenbanken ist, dass sie in der Regel kein strenges Schema erzwingen. Die Daten werden über horizontal skalierbare Partitionen verteilt und über Partitionsschlüssel abgerufen.

Da Microservices jeweils für eine bestimmte Aufgabe konzipiert sind, verfügen sie typischerweise über ein vereinfachtes Datenmodell, das für die NoSQL-Persistenz von Vorteil sein kann. Es ist wichtig zu verstehen, dass NoSQL-Datenbanken andere Zugriffsmuster als relationale Datenbanken haben. Beispielsweise ist es nicht möglich, Tabellen zu verbinden. Wenn dies erforderlich ist, muss die Logik in der Anwendung implementiert werden. Sie können mit Amazon DynamoDB eine Datenbanktabelle zum Speichern und Abrufen beliebiger Datenmengen erstellen und unterschiedlichste Grade von Anforderungsdatenverkehr erfüllen. Die Leistung von DynamoDB liegt im einstelligen Millisekundenbereich. Es gibt jedoch Anwendungsfälle, die Reaktionszeiten in Mikrosekunden erfordern. Amazon DynamoDB Accelerator (DAX) bietet Caching-Funktionen für den Datenzugriff.

Darüber hinaus kann die Durchsatzkapazität des tatsächlichen Datenverkehrs mithilfe einer automatischen Skalierfunktion von DynamoDB dynamisch angepasst werden. Mitunter ist die Kapazitätsplanung jedoch aufgrund von hohen, kurzen Aktivitätsspitzen der Anwendung schwierig oder gar unmöglich. In diesen Fällen bietet DynamoDB eine On-Demand-Option mit einfachen nutzungsabhängigen Preisen. Mit der On-Demand-Funktion von DynamoDB lassen sich Tausende Anfragen pro Sekunde sofort und ohne Kapazitätsplanung bedienen.

Vereinfachung der operativen Komplexität

Die zuvor in diesem Whitepaper beschriebene Architektur verwendet bereits verwaltete Services. Dennoch müssen Amazon Elastic Compute Cloud (Amazon EC2)-Instances verwaltet werden. Der operative Aufwand für den Betrieb, die Wartung und die Überwachung von Microservices kann mithilfe einer vollständigen Serverless Architektur weiter reduziert werden.

Themen

- API-Implementierung
- Serverless Microservices
- Notfallwiederherstellung
- Hohe Verfügbarkeit
- Bereitstellung von Lambda-basierten Anwendungen

API-Implementierung

Die Entwicklung, Bereitstellung, Überwachung, kontinuierliche Verbesserung und Wartung einer API kann zeitaufwendig sein. Mitunter müssen verschiedene API-Versionen ausgeführt werden, um die Abwärtskompatibilität für alle Clients sicherzustellen. Die verschiedenen Phasen des Bereitstellungszyklus (wie Entwicklung, Test und Produktion) vervielfachen den operativen Aufwand weiter.

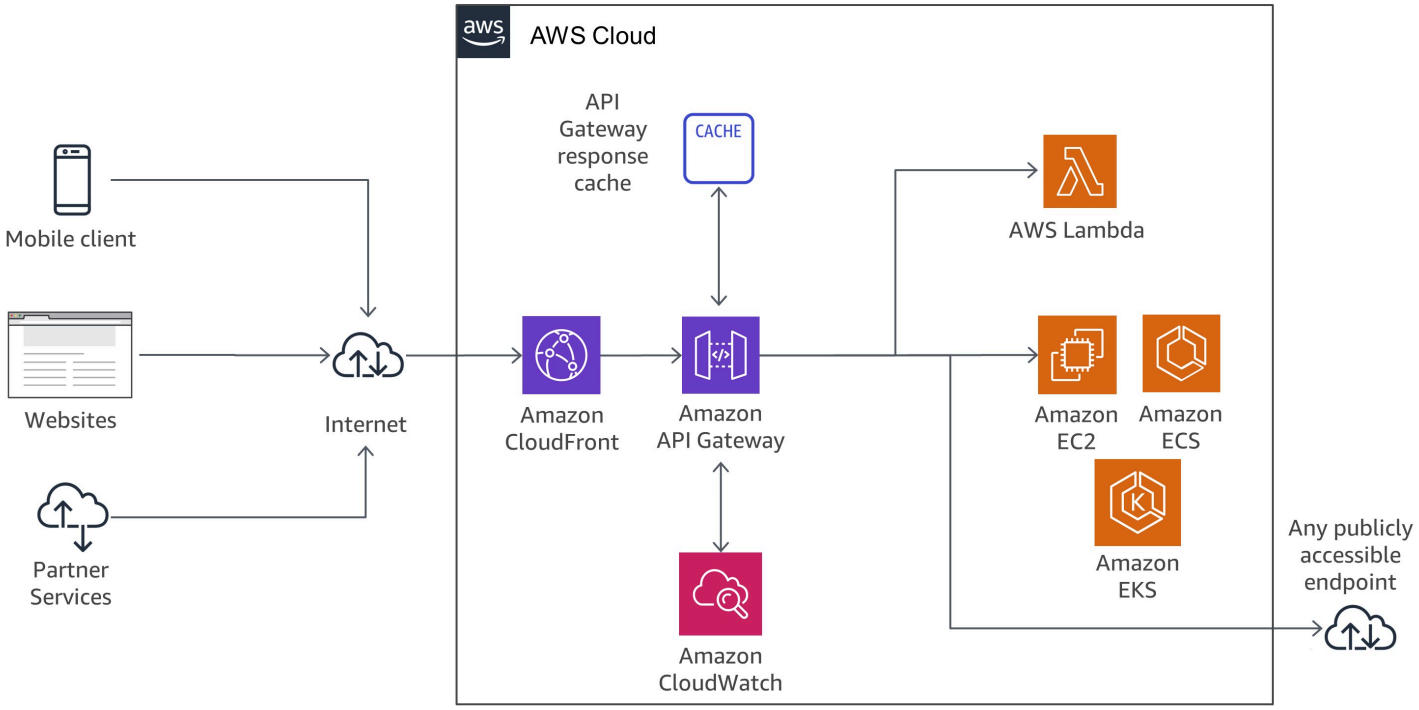
Die Autorisierung ist ein kritisches Merkmal für alle APIs. Die Entwicklung ist jedoch in der Regel komplex und beinhaltet wiederkehrende Aufgaben. Wenn eine API veröffentlicht wird und Erfolg erlangt, besteht die nächste Herausforderung in der Verwaltung, Überwachung und Monetarisierung des Ökosystems von Drittanbietern, die die API nutzen.

Weitere wichtige Funktionen und Herausforderungen sind das Drosseln von Anfragen zum Schutz der Backend-Services, das Zwischenspeichern von API-Antworten, die Transformation von Anfragen und Antworten sowie das Erstellen von API-Definitionen und -Dokumentationen mit Tools wie Swagger.

Amazon API Gateway löst diese Herausforderungen und reduziert die betriebliche Komplexität bei der Erstellung und Wartung von RESTful-APIs. API Gateway ermöglicht es Ihnen, Ihre APIs programmgesteuert zu erstellen, indem Sie Swagger-Definitionen über die AWS-API oder über die

AWS-Managementkonsole importieren. API Gateway dient als Zugang zu jeder Webanwendung, die in Amazon EC2, Amazon ECS, AWS Lambda oder einer lokalen Umgebung ausgeführt wird. Im Grunde ermöglicht Ihnen API Gateway, APIs auszuführen, ohne Server verwalten zu müssen.

Die folgende Abbildung veranschaulicht, wie API Gateway API-Aufrufe behandelt und mit anderen Komponenten interagiert. Anfragen von mobilen Geräten, Websites oder anderen Backend-Services werden an den nächstgelegenen CloudFront Point of Presence (PoP) weitergeleitet, um die Latenz zu minimieren und ein optimales Benutzererlebnis zu gewährleisten.



Verlauf eines API Gateway-Aufrufs

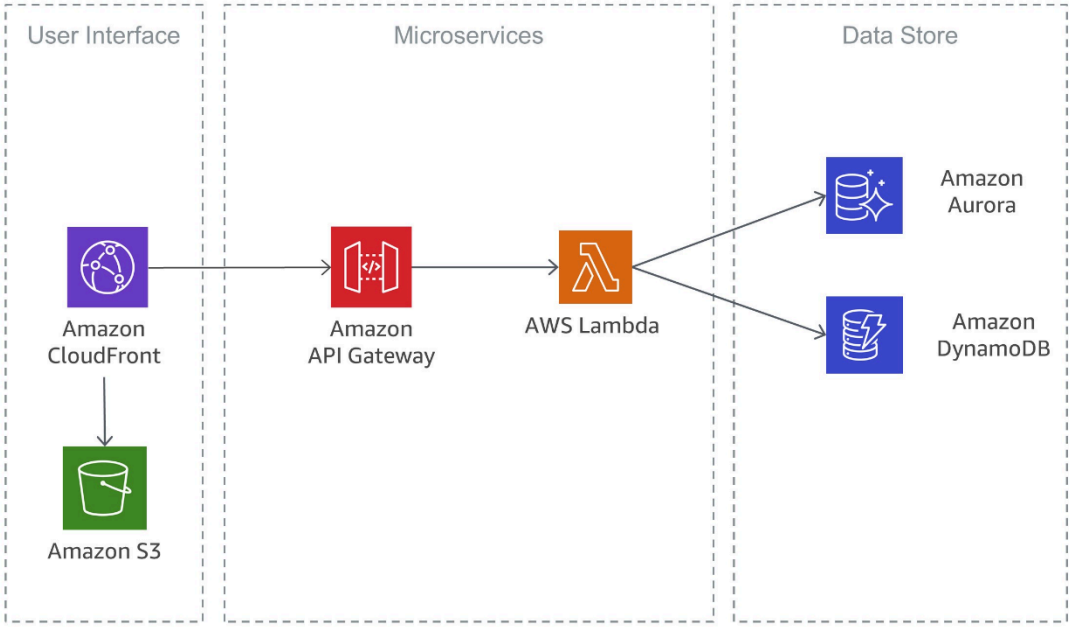
Serverless Microservices

„Kein Server ist einfacher zu verwalten als kein Server.“

Durch die Eliminierung von Servern lässt sich die operative Komplexität erheblich reduzieren.

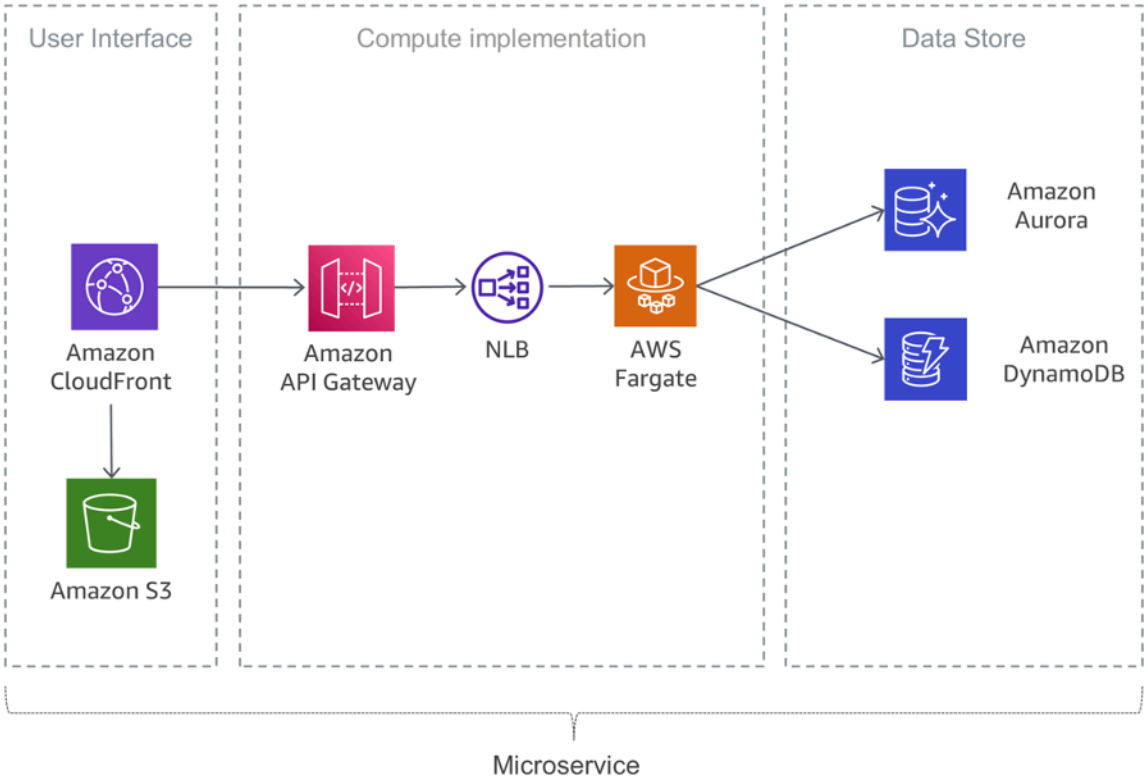
Lambda ist eng in API Gateway integriert. Die Fähigkeit, synchrone Aufrufe von API Gateway an Lambda zu tätigen, ermöglicht die Erstellung von vollständig Serverless Anwendungen und ist im Amazon API Gateway Entwicklerleitfaden ausführlich beschrieben.

Die folgende Abbildung zeigt die Architektur eines Serverless Microservice in Verbindung mit AWS Lambda. Der vollständige Service besteht aus verwalteten Services, wodurch Sie sich keine Gedanken hinsichtlich der Skalierung und Hochverfügbarkeit zu machen brauchen. Außerdem erübrigt sich der operative Aufwand für die Ausführung und Überwachung der Infrastruktur, die dem Microservice zugrunde liegt.



Serverless Microservice in Verbindung mit AWS Lambda

Eine ähnliche Implementierung, die ebenfalls auf Serverless Services basiert, ist in der folgenden Abbildung dargestellt. In dieser Architektur werden Docker-Container zusammen mit Fargate verwendet. Dadurch erübrigt sich auch hier die Verwaltung der zugrunde liegenden Infrastruktur. Ergänzend zu DynamoDB wird Amazon Aurora Serverless verwendet. Bei dieser Auto-Scaling- und On-Demand-Konfiguration für Amazon Aurora (MySQL-kompatible Edition) wird die Datenbank automatisch hoch- und heruntergefahren und die Kapazität entsprechend den Anforderungen der Anwendung skaliert.



Serverless Microservice in Verbindung mit Fargate

Notfallwiederherstellung

Wie bereits in der Einleitung dieses Whitepapers erwähnt, werden typische Microservices-Anwendungen mithilfe der Zwölf-Faktor-Anwendungsmuster implementiert. Im Abschnitt „Prozesse“ wird erläutert, dass Zwölf-Faktor-Prozesse zustandslos und ohne gemeinsame Komponenten sind. Alle Daten, die bestehen bleiben müssen, müssen in einem zustandsbehafteten Sicherungsdienst gespeichert werden, normalerweise in einer Datenbank.

Für eine typische Microservices-Architektur bedeutet dies, dass das Hauptaugenmerk bei der Notfallwiederherstellung auf den nachgeschalteten Diensten liegen sollte, die den Status der Anwendung aufrechterhalten. Dies können beispielsweise Dateisysteme, Datenbanken oder Warteschlangen sein. Bei der Erstellung einer Strategie zur Notfallwiederherstellung planen Unternehmen überwiegend Recovery Time Objective (RTO) und Recovery Point Objective (RPO) ein.

Recovery Time Objective ist die maximal akzeptable Verzögerung zwischen der Unterbrechung und der Wiederherstellung des Service. Dieses Ziel wird von der Organisation festgelegt und gibt an, was als akzeptables Zeitfenster gilt, wenn der Service nicht verfügbar ist.

Recovery Point Objective ist die maximal zulässige Zeitspanne seit dem letzten Wiederherstellungspunkt. Dieses Ziel wird von der Organisation festgelegt und bestimmt, was als akzeptabler Datenverlust zwischen dem letzten Wiederherstellungspunkt und der Service-Unterbrechung gilt.

Weitere Informationen finden Sie im Whitepaper Notfallwiederherstellung von Workloads auf AWS: Wiederherstellung in der Cloud.

Hohe Verfügbarkeit

In diesem Abschnitt wird die Hochverfügbarkeit für verschiedene Rechenoptionen genauer untersucht.

Um eine hohe Verfügbarkeit zu gewährleisten, führt Amazon EKS Kubernetes-Steuer- und Datenebenen-Instances in mehreren Availability Zones aus. Amazon EKS erkennt und ersetzt automatisch degenerierte Steuerebenen-Instances und stellt automatisch Versions-Upgrades und Patches dafür bereit. Diese Steuerebene besteht aus mindestens zwei API-Serverknoten und drei etcd-Knoten, die sich über drei Availability Zones innerhalb einer Region erstrecken. Amazon EKS verwendet die Architektur von AWS-Regionen, um die Hochverfügbarkeit aufrechtzuerhalten.

Amazon ECR hostet Images in einer hochverfügbaren und hochleistungsfähigen Architektur, sodass Sie Images für Containeranwendungen in Availability Zones zuverlässig bereitstellen können. Amazon ECR arbeitet mit Amazon EKS, Amazon ECS und AWS Lambda und erleichtert so den Arbeitsablauf von Entwicklung bis zu Produktion.

Amazon ECS ist ein regionaler Service zur einfacheren, hochverfügbaren Ausführung von Containern über mehrere Availability Zones innerhalb einer AWS-Region hinweg. Amazon ECS enthält mehrere Planungsstrategien, mit denen Container basierend auf Ihren Ressourcenanforderungen (z. B. CPU oder RAM) und Verfügbarkeitsanforderungen über Ihre Cluster verteilt werden.

AWS Lambda führt Ihre Funktion in mehreren Availability Zones aus, um sicherzustellen, dass sie für die Verarbeitung von Ereignissen verfügbar ist, wenn der Service in einer Zone unterbrochen wird. Wenn Sie Ihre Funktion für die Herstellung von Verbindungen mit einer Virtual Private Cloud (VPC) in Ihrem Konto konfigurieren, geben Sie Subnetze in mehreren Availability Zones an, um eine hohe Verfügbarkeit sicherzustellen.

Bereitstellung von Lambda-basierten Anwendungen

Sie können AWS CloudFormation verwenden, um Serverless Anwendungen anzugeben, bereitzustellen und zu konfigurieren.

Das AWS Serverless Application Model (AWS SAM) ist eine bequeme Methode, um Serverless Anwendungen zu definieren. AWS SAM wird von CloudFormation nativ unterstützt und definiert eine vereinfachte Syntax zum Ausdrücken von Serverless Ressourcen. Um Ihre Anwendung bereitzustellen, geben Sie die Ressourcen, die Sie als Teil Ihrer Anwendung benötigen, zusammen mit den zugehörigen Berechtigungsrichtlinien in einer CloudFormation-Vorlage an, verpacken Sie Ihre Bereitstellungsartefakte und stellen Sie die Vorlage bereit. SAM Local ist ein AWS Command Line Interface (AWS CLI)-Tool, das auf AWS SAM basiert und stellt eine Umgebung bereitstellt, in der Sie Serverless Anwendungen lokal entwickeln, testen und analysieren können, bevor diese in die Lambda-Laufzeitumgebung hochgeladen werden. AWS SAM Local ermöglicht Ihnen, eine lokale Testumgebung zu erstellen, die die AWS-Laufzeitumgebung simuliert.

Verteilte Systemkomponenten

Nachdem wir uns angesehen haben, wie sich mit AWS Herausforderungen im Zusammenhang mit einzelnen Microservices lösen lassen, rücken serviceübergreifende Herausforderungen in den Fokus. Hierzu zählen unter anderem Serviceerkennung, Datenkonsistenz, asynchrone Kommunikation sowie verteilte Überwachung und Prüfung.

Themen

Serviceerkennung

Eine der größten Herausforderungen bei Microservices-Architekturen besteht darin, dass Services sich gegenseitig entdecken und miteinander interagieren können. Die verteilten Eigenschaften von Microservice-Architekturen erschweren nicht nur die Kommunikation zwischen den Services, sondern bergen auch andere Herausforderungen, wie das Überprüfen des Zustands dieser Systeme sowie das Ankündigen der Verfügbarkeit neuer Anwendungen. Darüber hinaus müssen Sie entscheiden, wie und wo Meta-Store-Informationen wie etwa von Anwendungen verwendbare Konfigurationsdaten gespeichert werden sollen. In diesem Abschnitt erörtern wir verschiedene Techniken zur Serviceerkennung in AWS für Microservice-basierte Architekturen.

DNS-basierte Serviceerkennung

Amazon ECS bietet ab sofort eine integrierte Service-Discovery-Funktion. Dies erleichtert die gegenseitige Erkennung und die Verbindung containerisierter Services untereinander.

Bisher mussten Sie Ihr eigenes Service-Discovery-System basierend auf Amazon Route 53, AWS Lambda und ECS Event Stream konfigurieren und ausführen oder jeden Service mit einem Load Balancer verbinden, um sicherzustellen, dass sich die Services gegenseitig erkennen und miteinander verbinden können.

Amazon ECS erstellt und verwaltet mithilfe der Route 53 Auto Naming API eine Registry mit Servicenamen. Die Namen werden automatisch einer Reihe von DNS-Datensätzen zugewiesen. Sie können dadurch in Ihrem Code auf den Namen eines Service verweisen und DNS-Abfragen schreiben, sodass der Name während der Laufzeit in den Endpunkt des Services aufgelöst wird. Sie haben die Möglichkeit, in der Aufgabendefinition eines Services Bedingungen für eine Zustandsprüfung anzugeben. Amazon ECS stellt daraufhin sicher, dass bei einer Servicesuche nur intakte Serviceendpunkte zurückgegeben werden.

Des Weiteren können Sie für von Kubernetes verwaltete Services die einheitliche Serviceerkennung nutzen. Damit diese Integration möglich ist, hat AWS einen Beitrag zum Incubator-Projekt „External DNS“ von Kubernetes geleistet.

Eine weitere Option besteht darin, die Funktionen von AWS Cloud Map zu nutzen. AWS Cloud Map erweitert die Funktionen der Auto Naming APIs durch die Bereitstellung einer Service Registry für Ressourcen wie Internet Protocols (IPs), Uniform Resource Locators (URLs) und Amazon-Ressourcennamen (ARNs). Ein API-basierter Serviceerkennungsmechanismus ermöglicht zudem eine beschleunigte Änderungspropagation. Darüber hinaus können Sie die Ressourcenerkennung mithilfe von Attributen eingrenzen. Bestehende Ressourcen für Route 53 Auto Naming werden automatisch auf AWS Cloud Map aktualisiert.

Software von Drittanbietern

Ein anderer Ansatz bei der Implementierung der Serviceerkennung ist die Verwendung von Software von Drittanbietern wie HashiCorp Consul, etcd oder Netflix Eureka. Alle drei Beispiele sind verteilte, zuverlässige Schlüssel-Werte-Datenbank. Für HashiCorp Consul gibt es einen AWS Quick Start, der eine flexible, skalierbare AWS Cloud-Umgebung einrichtet und HashiCorp Consul automatisch in einer Konfiguration Ihrer Wahl startet.

Service-Meshes

In einer erweiterten Microservice-Architektur kann die tatsächliche Anwendung aus Hunderten oder sogar Tausenden Services bestehen. Häufig besteht der komplexeste Teil der Anwendung nicht in den Services an sich, sondern in der Kommunikation zwischen ihnen. Service-Meshes sind eine zusätzliche Verwaltungsschicht für die Kommunikation zwischen den Services. Diese ist für die Überwachung und Steuerung des Datenverkehrs in Microservices-Architekturen zuständig. Aufgaben wie die Serviceerkennung können dadurch vollständig von dieser Schicht verarbeitet werden.

Ein Service-Mesh ist in der Regel in eine Datenebene und eine Steuerungsebene unterteilt. Die Datenebene besteht aus einer Reihe intelligenter Proxys, die mit dem Anwendungscode als

spezieller Sidecar-Proxy bereitgestellt werden, der die gesamte Netzwerkkommunikation zwischen Microservices erfasst. Die Steuerungsebene ist für die Kommunikation mit den Proxys zuständig.

Service-Meshes sind transparent. Dies bedeutet, dass sich Anwendungsentwickler dieser zusätzlichen Schicht nicht bewusst sein müssen und den vorhandenen Anwendungscode beibehalten können. AWS App Mesh ist ein Service-Mesh, das Networking auf Anwendungsebene bietet, um Services die Kommunikation untereinander über unterschiedliche Arten von Computing-Infrastruktur zu ermöglichen. App Mesh standardisiert die Kommunikation Ihrer Services. Sie erhalten dadurch vollständige Transparenz und stellen Hochverfügbarkeit für Ihre Anwendungen sicher.

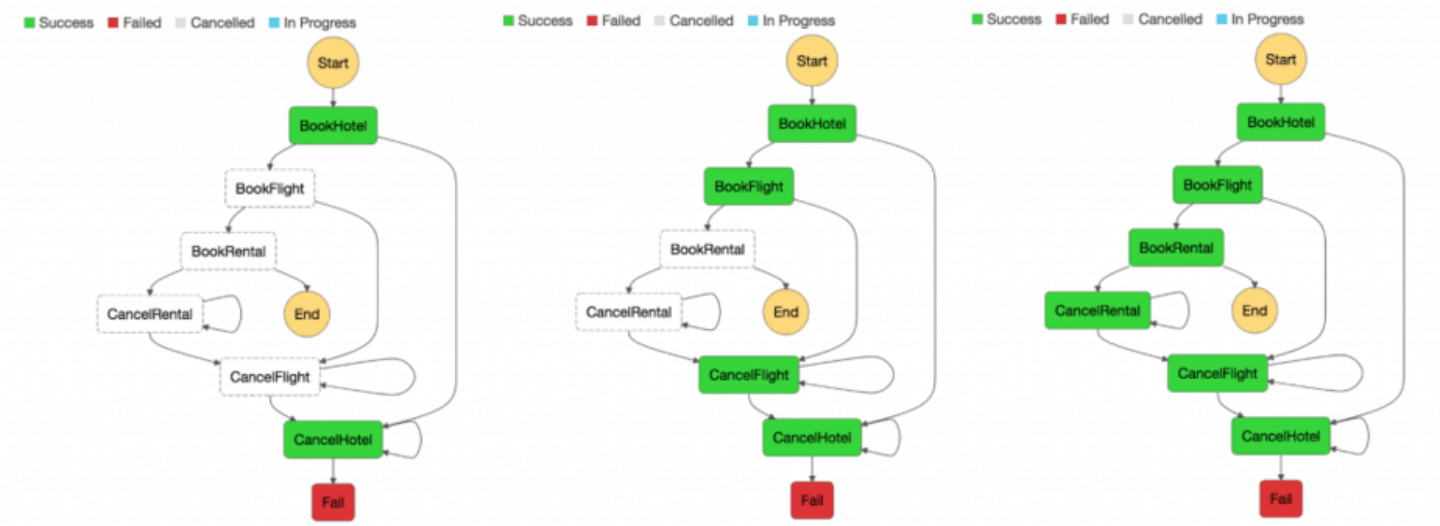
Sie können AWS App Mesh mit vorhandenen oder neuen Microservices verwenden, die in AWS Fargate, Amazon ECS, Amazon EKS und selbstverwalteten Kubernetes in AWS ausgeführt werden. App Mesh kann die Kommunikation für Microservices überwachen und steuern, die in Clustern, Orchestrierungssystemen oder VPCs als eine Anwendung ausgeführt werden, ohne den Code zu ändern.

Verteilte Datenverwaltung

Monolithische Anwendungen werden typischerweise durch eine große relationale Datenbank unterstützt, die ein einziges Datenmodell definiert, das allen Anwendungskomponenten gemein ist. Bei einem Microservice-Ansatz würde eine solche zentrale Datenbank das Ziel, dezentrale und unabhängige Komponenten aufzubauen, verhindern. Jede Microservice-Komponente sollte eine eigene Datenpersistenzschicht haben.

Dezentrales Datenmanagement stellt jedoch neue Herausforderungen dar. Infolge des CAP-Theorem machen verteilte Microservice-Architekturen zugunsten der Leistung Abstriche bei der Konsistenz und müssen einen „Eventually Consistent“-Zustand (letztendliche Datenkonsistenz) erreichen.

In einem verteilten System können Geschäftstransaktionen mehrere Microservices umfassen. Da dabei keine einzelne ACID-Transaktion genutzt werden kann, erfolgt die Transaktion möglicherweise nur zum Teil. In diesem Fall müssten die bereits verarbeiteten Transaktionen mithilfe einer Steuerungslogik wiederhergestellt werden. Zu diesem Zweck wird in der Regel das verteilte Saga-Muster verwendet. Wenn eine Geschäftstransaktion fehlschlägt, orchestriert Saga eine Reihe von kompensierenden Transaktionen, um die während der vorangegangenen Transaktionen durchgeführten Änderungen rückgängig zu machen. AWS Step Functions erleichtert die Implementierung eines Koordinators zur Ausführung von Saga, wie in der nächsten Abbildung dargestellt.



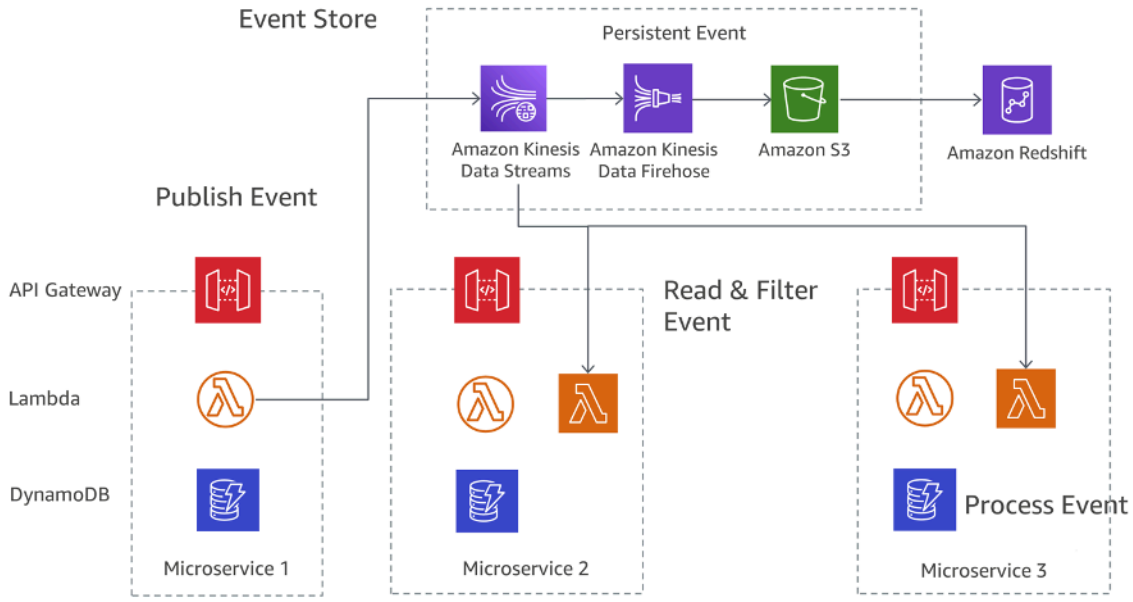
Ausführungskoordinator für Saga

Die Entwicklung eines zentralen Speichers kritischer Referenzdaten, der von Stammdatenmanagement-Tools und -Verfahren kuratiert wird, bietet Microservices die Möglichkeit, ihre kritischen Daten und möglicherweise Wiederherstellungspunkte zu synchronisieren. Mit Lambda und geplanten Amazon CloudWatch Events können Sie einen einfachen Bereinigungs- und Deduplizierungsmechanismus entwickeln.

Es kommt häufig vor, dass Zustandsänderungen mehr als einen einzelnen Microservice betreffen. In diesen Fällen hat sich das Event-Sourcing als nützliches Muster erwiesen. Die Grundidee des Event-Sourcing besteht darin, jede Anwendungsänderung als Ereignisprotokoll darzustellen und beizubehalten. Anstatt den Anwendungszustand zu persistieren, werden die Daten als Ereignisstrom gespeichert. Datenbank-Transaktionsprotokollierung und Versionskontrollsysteme sind zwei bekannte Beispiele für Event-Sourcing. Event-Sourcing hat einige Vorteile: Der Zustand kann zu jedem Zeitpunkt ermittelt und rekonstruiert werden. Es erzeugt auf natürliche Weise einen persistenten Audit-Trail und erleichtert auch das Debugging.

Im Rahmen von Microservice-Architekturen ermöglicht Event-Sourcing die Entkopplung verschiedener Teile einer Anwendung unter Verwendung eines Publish/Subscribe-Musters und speist die gleichen Ereignisdaten in verschiedene Datenmodelle für einzelne Microservices ein. Event-Sourcing wird häufig in Verbindung mit dem CQRS-Muster (Command, Query, Responsibility, Segregation) verwendet, um das Lesen von Schreib-Workloads zu entkoppeln und sowohl Leistung, Skalierbarkeit als auch Sicherheit zu optimieren. In traditionellen Datenmanagementsystemen werden Befehle und Abfragen im selben Datenspeicher ausgeführt.

Die folgende Abbildung zeigt, wie das Event-Sourcing-Muster in AWS implementiert werden kann. Amazon Kinesis Data Streams dient als Hauptkomponente des zentralen Ereignisspeichers, der Anwendungsänderungen als Ereignisse erfasst und dauerhaft in Amazon S3 speichert. Die Abbildung zeigt drei verschiedene Microservices bestehend aus Amazon API Gateway, AWS Lambda und Amazon DynamoDB. Die Pfeile geben den Ablauf der Ereignisse an: Wenn sich der Ereignisstatus des ersten Microservice ändert, veröffentlicht dieser ein Ereignis, indem er eine Nachricht in Kinesis Data Streams schreibt. Alle Microservices führen ihre eigene Kinesis Data Streams-Anwendung in AWS Lambda aus. AWS Lambda liest eine Kopie der Nachricht, filtert sie entsprechend ihrer Relevanz für den Microservice und leitet die Nachricht gegebenenfalls zur weiteren Verarbeitung weiter. Wenn Ihre Funktion einen Fehler zurückgibt, wiederholt Lambda die Batch-Verarbeitung, bis diese erfolgreich ist oder die Daten ablaufen. Um blockierte Shards zu vermeiden, können Sie die Ereignisquellen-Zuordnung so konfigurieren, dass der Vorgang bei einer kleineren Batch-Größe neu versucht wird, die Anzahl der Neuversuche begrenzt wird oder Datensätze, die zu alt sind, verworfen werden. Um verworfene Ereignisse beizubehalten, können Sie die Ereignisquellen-Zuordnung so konfigurieren, dass Details über fehlgeschlagene Batches an eine Amazon Simple Queue Service (Amazon SQS)-Warteschlange oder an ein Amazon Simple Notification Service (Amazon SNS)-Thema gesendet werden.



Event-Sourcing-Muster in AWS

Amazon S3 speichert alle Ereignisse sämtlicher Microservices dauerhaft und ist die einzige Informationsquelle, wenn es um Debugging, die Wiederherstellung des Anwendungszustands

oder die Überprüfung von Anwendungsänderungen geht. Es gibt zwei primäre Gründe, warum Datensätze mehr als einmal an Ihre Kinesis-Data-Streams-Anwendung übermittelt werden: Wiederholungsversuche des Produzenten und Wiederholungsversuche des Konsumenten. Ihre Anwendung muss in der Lage sein, einzelne Datensätze mehrere Male angemessen zu verarbeiten.

Konfigurationsmanagement

In einer typischen Microservices-Architektur mit Dutzenden verschiedener Services benötigt jeder Service Zugriff auf mehrere nachgeschaltete Services und Infrastrukturkomponenten, die Daten für den Service bereitstellen. Dies können beispielsweise Nachrichtenwarteschlangen, Datenbanken und andere Microservices sein. Eine der größten Herausforderungen besteht darin, jeden Service konsistent zu konfigurieren, um Informationen über die Verbindung zu nachgeschalteten Services und Infrastrukturen bereitzustellen. Darüber hinaus sollte die Konfiguration auch Informationen über die Umgebung enthalten, in der der Service ausgeführt wird. Ein Neustart der Anwendung zur Verwendung neuer Konfigurationsdaten sollte nicht erforderlich sein.

Das dritte Prinzip der Zwölf-Faktor-App-Muster behandelt dieses Thema: „Die Zwölf-Faktor-App speichert die Konfiguration in Umgebungsvariablen (oft abgekürzt mit env vars oder env).“ Für Amazon ECS können Umgebungsvariablen mithilfe des Parameters für die Containerdefinition der Umgebung an den Container übergeben werden, der der `--env`-Option zum Docker-Lauf zugeordnet ist. Sie können Umgebungsvariablen in großen Mengen an Ihre Container übergeben, indem Sie den Containerdefinitionsparameter `environmentFiles` verwenden, um eine oder mehrere Dateien aufzulisten, die die Umgebungsvariablen enthalten. Die Datei muss in Amazon S3 gehostet werden. In AWS Lambda stellt die Laufzeit Umgebungsvariablen für den Code zur Verfügung und legt zusätzliche Umgebungsvariablen mit Informationen über die Funktion und die Aufrufanforderung fest. Für Amazon EKS können Sie Umgebungsvariablen im env-Feld des Konfigurationsmanifests des entsprechenden Pods definieren. Eine andere Möglichkeit, env-Variablen zu nutzen, ist die Verwendung einer ConfigMap.

Asynchrone Kommunikation und einfaches Messaging

In traditionellen, monolithischen Anwendungen ist die Kommunikation eher einfach:
Teile der Anwendung kommunizieren über Methodenaufrufe oder einen internen Ereignisverteilungsmechanismus mit anderen Teilen. Wird die gleiche Anwendung über entkoppelte Microservices implementiert, muss die Kommunikation zwischen verschiedenen Teilen der Anwendung über Netzwerkkommunikation erfolgen.

REST-basierte Kommunikation

Das HTTP/S-Protokoll ist der beliebteste Weg, um synchrone Kommunikation zwischen Microservices zu implementieren. In den meisten Fällen verwenden RESTful-APIs HTTP als Transportschicht. Der REST-Architekturstil basiert auf zustandsloser Kommunikation, einheitlichen Schnittstellen und Standardmethoden.

Mit API Gateway können Sie ein API erstellen, das als Haupteingang für Anwendungen dient, um auf Daten, Geschäftslogik oder Funktionen von Ihren Backend-Services zuzugreifen. API-Entwickler können APIs erstellen, die auf AWS oder andere Webservices sowie auf Daten zugreifen, die in der AWS Cloud gespeichert sind. Ein mit dem API Gateway-Service definiertes API-Objekt besteht aus einer Gruppe von Ressourcen und Methoden.

Eine Ressource ist ein typisiertes Objekt innerhalb der Domäne einer API und kann ein Datenmodell oder Beziehungen zu anderen Ressourcen zugeordnet haben. Jede Ressource kann konfiguriert werden, um auf eine oder mehrere Methoden zu reagieren, d. h. auf Standard HTTP-Verben wie GET, POST oder PUT. REST-APIs können in verschiedenen Phasen bereitgestellt und versioniert sowie auf neue Versionen geklont werden.

API Gateway übernimmt alle Aufgaben, die mit der Annahme und Verarbeitung von mitunter Hunderttausenden gleichzeitigen API-Aufrufen verbunden sind. Hierzu zählen die Verwaltung des Datenverkehrs, die Autorisierung, die Zugriffskontrolle, die Überwachung sowie die API-Versionsverwaltung.

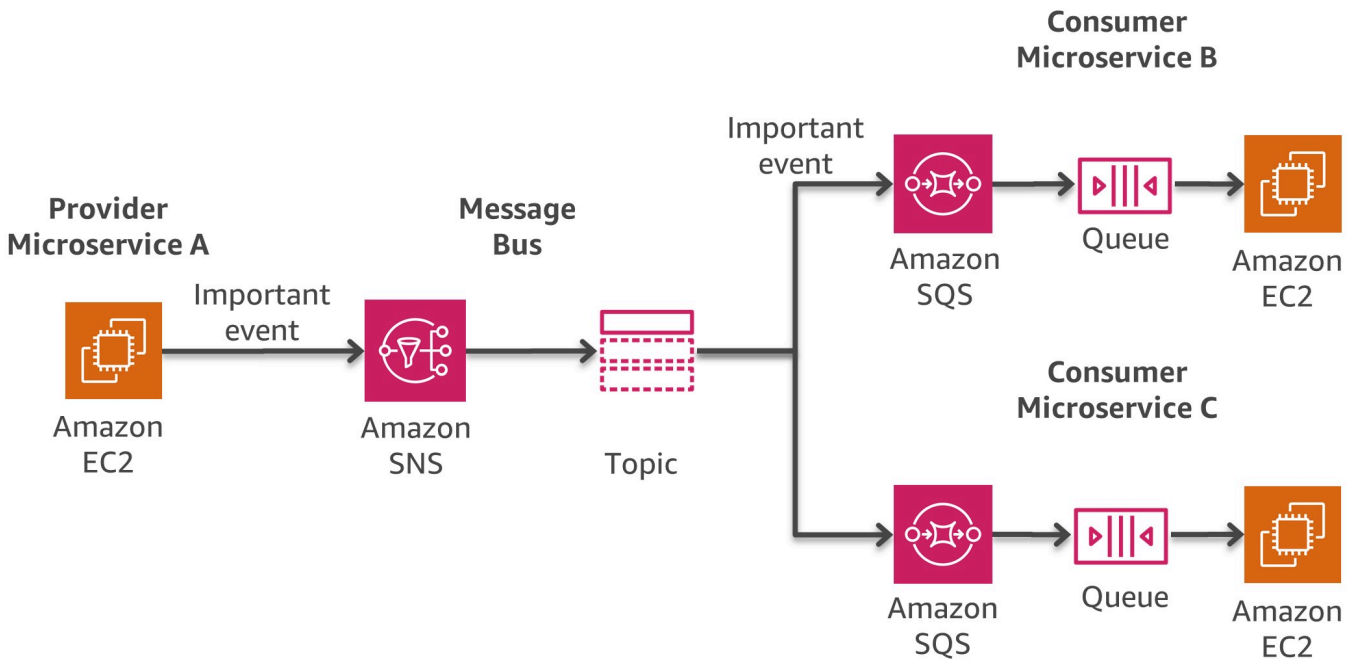
Asynchrones Messaging und Ereignisübermittlung

Ein weiteres Muster zur Implementierung der Kommunikation zwischen Microservices ist die Nachrichtenübermittlung. Services kommunizieren durch den Austausch von Nachrichten über eine Warteschlange. Ein großer Vorteil dieses Kommunikationsstils ist, dass keine Serviceerkennung erforderlich ist und Services lose gekoppelt sind.

Synchrone Systeme sind eng gekoppelt, wodurch sich ein Problem in einer synchronen nachgelagerten Abhängigkeit unmittelbar auf die vorgelagerten Aufrufer auswirkt. Wiederholversuche der vorgelagerten Aufrufer können sich schnell anhäufen und das Problem verstärken.

Abhängig von bestimmten Anforderungen wie Protokollen bietet AWS verschiedene Services zur Implementierung dieses Musters. Eine Möglichkeit der Implementierung besteht darin, Amazon Simple Queue Service (Amazon SQS) oder Amazon Simple Notification Service (Amazon SNS) zu kombinieren.

Beide Services arbeiten eng zusammen: Mit Amazon SNS können Anwendungen Nachrichten über einen Push-Mechanismus an mehrere Teilnehmer senden. Durch die Kombination von Amazon SNS und Amazon SQS kann eine Nachricht an mehrere Verbraucher verteilt werden. Die folgende Abbildung veranschaulicht die Integration von Amazon SNS und Amazon SQS.



Nachrichtenbusmuster in AWS

Wenn Sie eine Amazon-SQS-Warteschlange für ein SNS-Thema abonnieren, können Sie eine Nachricht zum Thema veröffentlichen. Amazon SNS sendet daraufhin eine Nachricht an die abonnierte Amazon-SQS-Warteschlange. Die Nachricht enthält den Betreff und den zum Thema veröffentlichten Nachrichtentext sowie Metadateninformationen im JSON-Format.

Eine weitere Option für die Entwicklung ereignisgesteuerter Architekturen mit Ereignisquellen, die interne Anwendungen, SaaS-Anwendungen von Drittanbietern und AWS-Services im großen Stil umfassen, ist Amazon EventBridge. EventBridge ist ein vollständig verwalteter Event-Bus-Service, der Ereignisse aus unterschiedlichen Quellen empfängt, ein Ziel basierend auf einer Routing-Regel identifiziert und nahezu in Echtzeit Daten an dieses Ziel liefert, einschließlich AWS Lambda, Amazon SNS und Amazon Kinesis Streams. Ein eingehendes Ereignis kann auch vor der Auslieferung per Input Transformer angepasst werden.

Um ereignisgesteuerte Anwendungen wesentlich schneller zu entwickeln, sammeln und organisieren EventBridge-Schema-Register Schemas, einschließlich Schemas für alle von AWS-Services generierten Ereignisse. Kunden können auch benutzerdefinierte Schemas definieren oder eine Schemaableitungsoption verwenden, um Schemas automatisch zu erkennen. Insgesamt gehen all diese Funktionen jedoch möglicherweise zu Lasten eines relativ höheren Latenzwerts für die EventBridge-Bereitstellung. Außerdem können der Standarddurchsatz und die Kontingente für EventBridge basierend auf dem jeweiligen Anwendungsfall eine Erhöhung durch eine Supportanforderung erfordern.

Eine andere Implementierungsstrategie basiert auf Amazon MQ. Dieser Service kann verwendet werden, wenn vorhandene Software für das Messaging offene Standard-APIs und Protokolle wie JMS, NMS, AMQP, STOMP, MQTT und WebSocket verwendet. Amazon SQS stellt eine benutzerdefinierte API bereit. Dies bedeutet, dass bei einer Migration einer vorhandenen Anwendung – beispielsweise von einer lokalen Umgebung zu AWS – der Code geändert werden muss. Mit Amazon MQ erübrigt sich dies in vielen Fällen.

Amazon MQ übernimmt die Verwaltung und Wartung von ActiveMQ, einem gängigen Open Source Message Broker. Die zugrunde liegende Infrastruktur wird für eine hohe Verfügbarkeit und Nachrichtenbeständigkeit automatisch bereitgestellt und erhöht die Zuverlässigkeit der Anwendungen.

Orchestrierung und Statusmanagement

Aufgrund der verteilten Natur von Microservices wird die Orchestrierung von Workflows mit mehreren beteiligten Microservices zur Herausforderung. Entwickler könnten versucht sein, Orchestrierungscode direkt in ihre Services einzufügen. Dies sollte vermieden werden, da es zu einer engeren Kopplung kommt und es schwieriger wird, einzelne Services schnell zu ersetzen.

Mithilfe von AWS Step Functions können Sie Anwendungen aus einzelnen Komponenten entwickeln, die jeweils eine diskrete Funktion ausführen. Step Functions stellt eine Statusmaschine zur Verfügung, die die Komplexität der Service-Orchestrierung wie Fehlerbehandlung, Serialisierung und Parallelisierung verbirgt. Auf diese Weise können Sie Anwendungen schnell skalieren und ändern und gleichzeitig zusätzlichen Koordinationscode innerhalb von Services vermeiden.

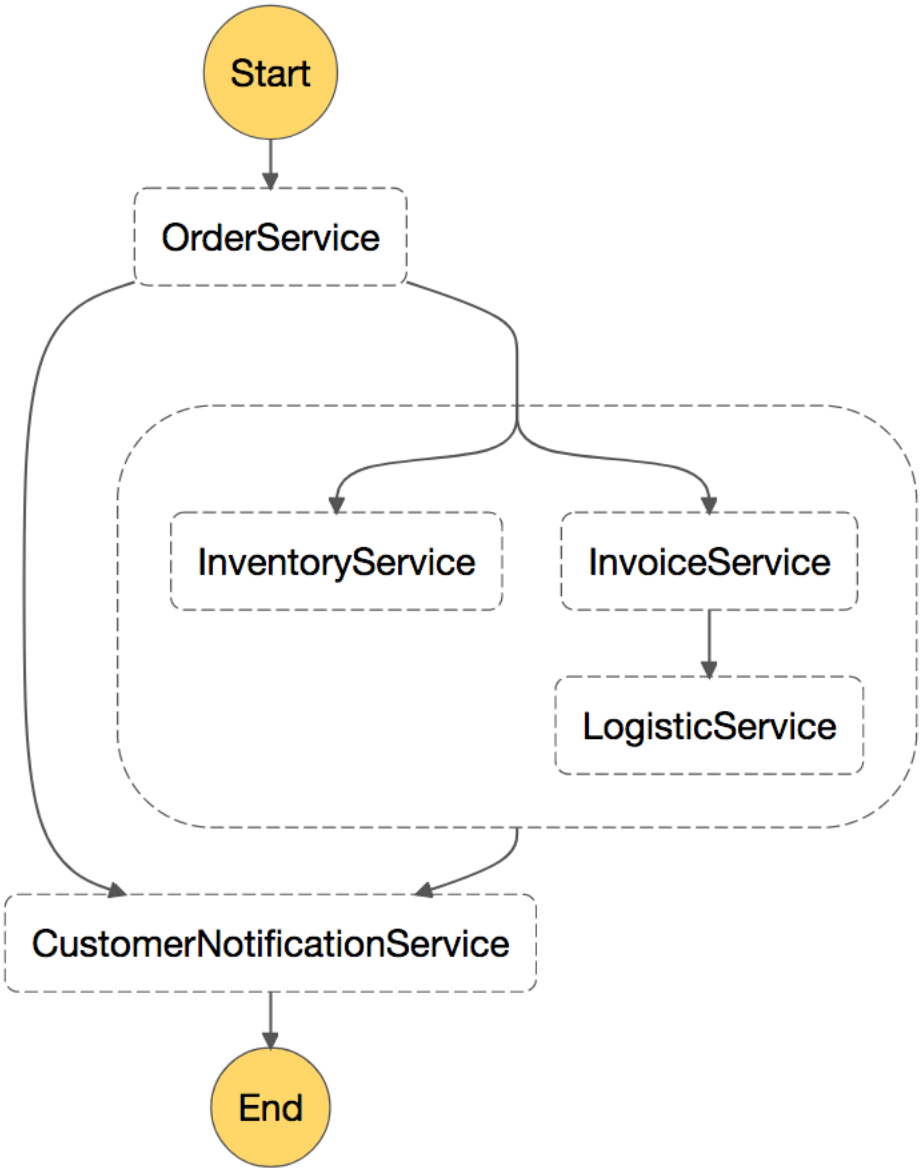
Step Functions ist eine zuverlässige Methode zur Koordination von Komponenten und Durchlaufen der Funktionen Ihrer Anwendungen. Step Functions bietet eine grafische Konsole zum Anordnen und Visualisieren der Komponenten Ihrer Anwendungen als eine Serie von Schritten. Dies macht es einfacher, verteilte Services zu entwickeln und auszuführen.

Step Functions startet jeden Schritt automatisch und verfolgt ihn und führt bei Fehlern Neuversuche aus, sodass Ihre Anwendung ordnungsgemäß und wie erwartet ausgeführt wird. Step Functions protokolliert den Status von jedem Schritt, wenn also etwas schiefgeht, können Sie Probleme schnell diagnostizieren und beheben. Sie können Schritte ändern und hinzufügen, ohne selbst Code zu schreiben, um Ihre Anwendung zu entwickeln und schneller zu innovieren.

Step Functions ist Teil der AWS Serverless Platform und unterstützt die Orchestrierung von Lambda-Funktionen und Anwendungen, die auf Computing-Ressourcen wie Amazon EC2, Amazon EKS, Amazon ECS sowie zusätzlichen Services wie Amazon SageMaker und AWS Glue basieren. Step Functions verwaltet für Sie den Betrieb und die zugrunde liegende Infrastruktur, um sicherzustellen, dass Ihre Anwendung in jeder Größenordnung verfügbar ist.

Zum Entwickeln von Workflows verwendet Step Functions die Amazon States Language. Workflows können sequenzielle oder parallele Schritte sowie Verzweigungsschritte enthalten.

Die folgende Abbildung zeigt einen exemplarischen Workflow für eine Microservice-Architektur, die sequenzielle und parallele Schritte kombiniert. Der Aufruf eines solchen Workflows kann entweder über die Step-Functions-API oder über API Gateway erfolgen.



Ein Beispiel für einen Microservice-Workflow, der durch Step Functions aufgerufen wird.

Verteilte Überwachung

Eine Microservices-Architektur besteht aus vielen verschiedenen verteilten Teilen, die überwacht werden müssen. Mit Amazon CloudWatch können Sie Metriken erfassen und verfolgen, Protokolldateien zentralisieren und überwachen, Alarme festlegen und automatisch auf Änderungen in Ihrer AWS-Umgebung reagieren. CloudWatch kann AWS-Ressourcen, wie Amazon-EC2-Instances, DynamoDB-Tabellen und Amazon-RDS-DB-Instances sowie von Ihren Anwendungen und Services generierte Metriken und Protokolldateien überwachen.

Überwachung

CloudWatch bietet Ihnen einen systemweiten Einblick in die Auslastung Ihrer Ressourcen, die Anwendungsleistung und die Integrität Ihrer Betriebsabläufe. CloudWatch bietet eine zuverlässige, skalierbare und flexible Überwachungslösung, die Sie innerhalb weniger Minuten einsetzen können. Sie müssen keine eigenen Überwachungssysteme und -infrastrukturen mehr einrichten, verwalten und skalieren. In einer Microservice-Architektur ist die Möglichkeit, benutzerdefinierte Metriken mit CloudWatch zu überwachen, ein zusätzlicher Vorteil, da Entwickler entscheiden können, welche Metriken für jeden Service gesammelt werden sollen. Darüber hinaus kann eine dynamische Skalierung basierend auf benutzerdefinierten Kennzahlen implementiert werden.

Zusätzlich zu Amazon CloudWatch können Sie auch CloudWatch Container Insights verwenden, um Metriken und Protokolle von Ihren containerisierten Anwendungen und Microservices zu sammeln, zu aggregieren und zusammenzufassen. CloudWatch Container Insights sammelt automatisch Metriken für viele Ressourcen wie CPU, Arbeitsspeicher, Festplatte und Netzwerk und aggregiert diese als CloudWatch-Metriken auf Cluster-, Knoten-, Pod-, Aufgaben- und Service-Ebene. Mit CloudWatch Container Insights können Sie auf die Dashboard-Metriken von CloudWatch Container Insights zugreifen. Der Service bietet außerdem Diagnoseinformationen, wie z. B.Fehler beim Container-Neustart, damit Sie Probleme schnell aufdecken und beheben können. Sie können für die von Container Insights gesammelten Metriken auch CloudWatch-Alarme einrichten.

Container Insights ist für Amazon ECS, Amazon EKS und Kubernetes-Plattformen in Amazon EC2 verfügbar. Der Amazon-ECS-Support umfasst Unterstützung für Fargate.

Eine weitere gängige Option, insbesondere für Amazon EKS, ist die Verwendung von Prometheus. Prometheus ist ein Open-Source-Überwachungs- und -Benachrichtigungs-Toolkit, das häufig zusammen mit Grafana verwendet wird, um die erfassten Metriken zu visualisieren. Zahlreiche Kubernetes-Komponenten speichern Metriken unter `/metrics`. Prometheus kann diese Metriken in einem regelmäßigen Intervall abrufen.

Amazon Managed Service for Prometheus (AMP) ist ein Prometheus-kompatibler Überwachungsservice, der die Überwachung von Anwendungen nach Maß ermöglicht. Mit AMP können Sie mithilfe der Open-Source Prometheus Query Language (PromQL) die Leistung von containerisierten Workloads überwachen, ohne die zugrunde liegende Infrastruktur verwalten zu müssen, die zur Erfassung, Speicherung und Abfrage von operativen Metriken erforderlich ist. Sie können Prometheus-Metriken aus Amazon-EKS- und Amazon-ECS-Umgebungen erfassen, indem Sie AWS Distro for OpenTelemetry oder Prometheus-Server als Erfassungsagenten verwenden.

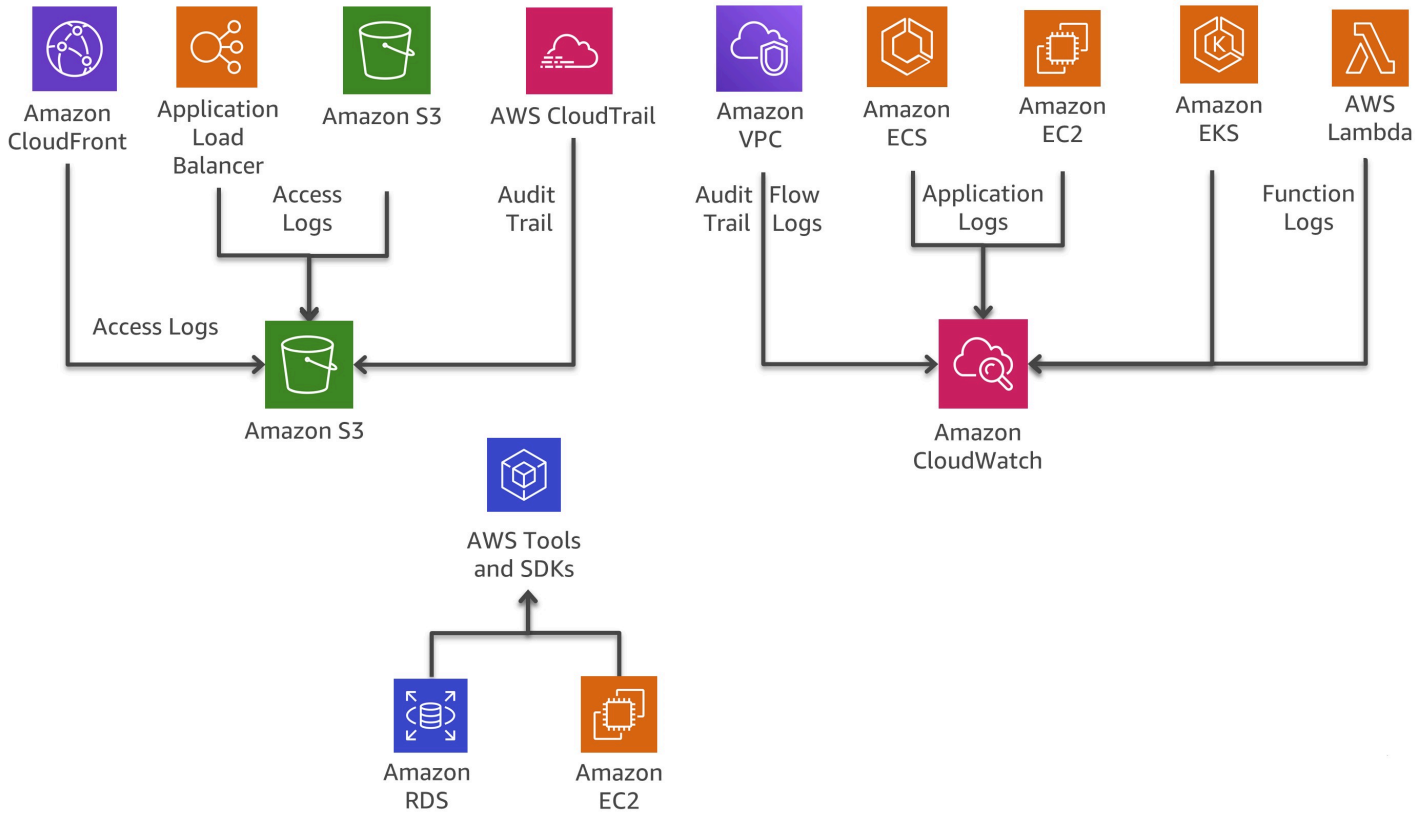
AMP wird häufig in Kombination mit Amazon Managed Service for Grafana (AMG) verwendet. Mit AMG können Sie Ihre Metriken ganz einfach abfragen, visualisieren, warnen und verstehen, unabhängig davon, wo sie gespeichert sind. Mit AMG können Sie Ihre Metriken, Protokolle und Traces analysieren, ohne Server bereitstellen, Software konfigurieren und aktualisieren oder den Arbeitsaufwand für die Sicherung und Skalierung von Grafana in der Produktion aufbringen zu müssen.

Zentralisierte Protokolle

Eine konsistente Protokollierung ist für die Fehlerbehebung und die Identifizierung von Problemen entscheidend. Microservices ermöglichen es Teams, viel mehr Releases als je zuvor zu liefern. Sie erleichtern es Entwicklern, in der Produktion mit neuen Funktionen zu experimentieren. Zu wissen, wie sich Funktionen einer Anwendung auf Kunden auswirken, ist wichtig, um diese stetig zu optimieren.

Bei den meisten AWS-Services werden Protokolldateien standardmäßig zentralisiert. In AWS werden Protokolldateien primär in Amazon S3 und Amazon CloudWatch Logs gespeichert. Für Anwendungen, die auf Amazon-EC2-Instances ausgeführt werden, steht ein Daemon zur Verfügung, um Protokolldateien an CloudWatch Logs zu senden. Lambda-Funktionen senden ihre Protokollausgabe nativ an CloudWatch Logs. Amazon ECS unterstützt den Protokolltreiber awslogs, der die Zentralisierung von Containerprotokollen in CloudWatch Logs ermöglicht. Für Amazon EKS kann entweder Fluent Bit oder Fluentd Protokolle von den einzelnen Instances im Cluster an eine zentrale Protokollierung von CloudWatch Logs weiterleiten, wo sie für übergeordnete Berichte mithilfe von Amazon OpenSearch Service und Kibana kombiniert werden. Aufgrund des geringeren Platzbedarfs und der Leistungsvorteile wird Fluent Bit anstelle von Fluentd empfohlen.

Die folgende Abbildung veranschaulicht die Protokollierungsfunktionen einiger Services. Teams können diese Protokolle dadurch mit Tools wie Amazon OpenSearch Service und Kibana suchen und analysieren. Mit Amazon Athena lassen sich einmalige Abfragen an zentralisierte Protokolldateien in Amazon S3 durchführen.



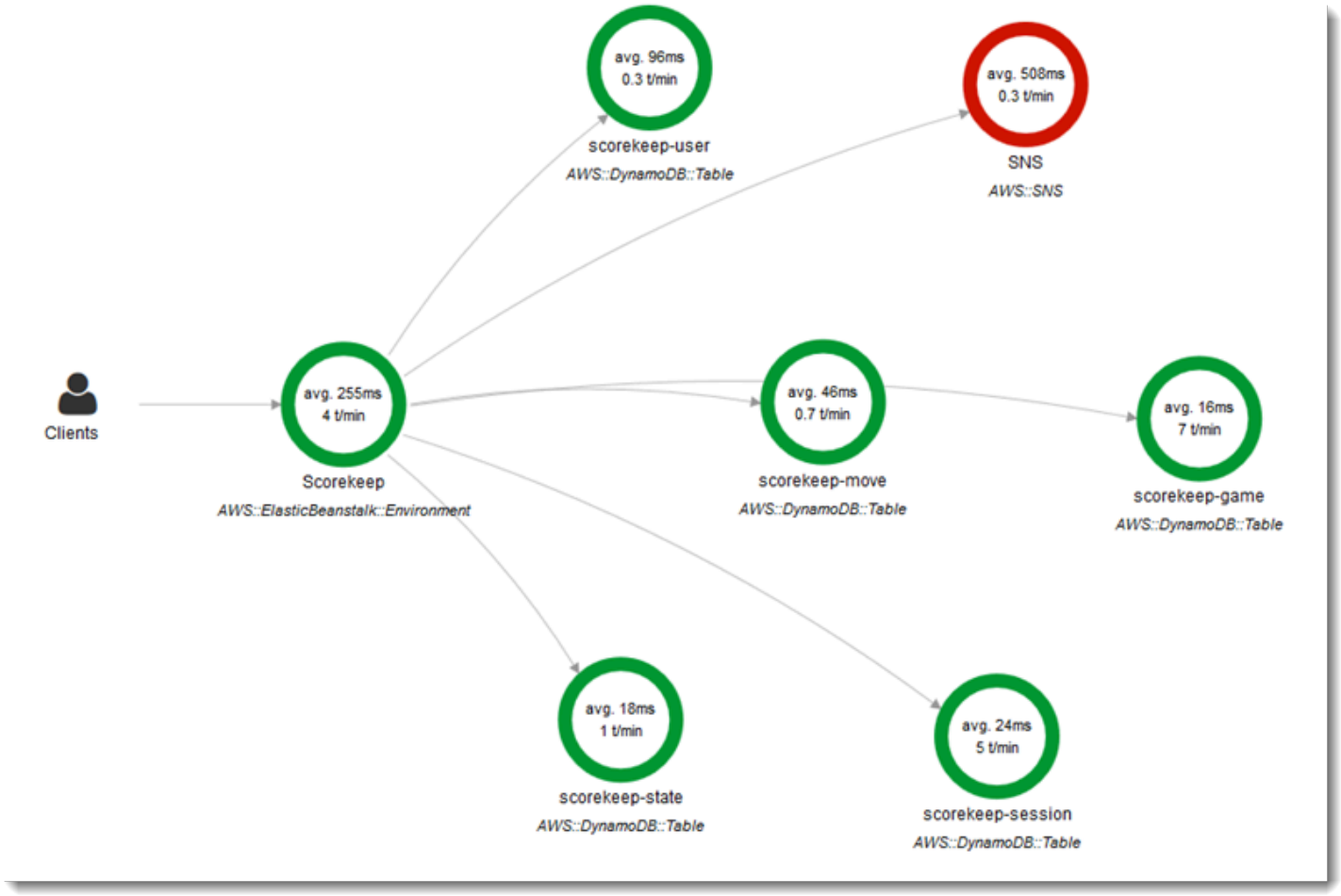
Protokollierungsfunktionen von AWS-Services

Verteilte Nachverfolgung

In vielen Fällen arbeitet eine Reihe von Microservices zusammen, um eine Anfrage zu bearbeiten. Doch was geschieht, wenn in einem komplexen System aus Dutzenden von Microservices in einem der Services in der Aufrufkette ein Fehler auftritt? Selbst wenn alle Microservices ordnungsgemäß protokolliert und die Protokolle in einem zentralen System konsolidiert werden, kann es schwierig sein, alle relevanten Protokollmeldungen zu finden.

Die zentrale Idee hinter AWS X-Ray ist die Verwendung von Korrelations-IDs. Die eindeutigen Identifikatoren werden an alle Anfragen und Nachrichten angehängt, die sich auf eine bestimmte Ereigniskette beziehen. Die Trace-ID wird HTTP-Anfragen in bestimmten Tracing-Headern mit dem Namen `X-Amzn-Trace-Id` hinzugefügt, sobald die Anfrage den ersten in X-Ray integrierten Service (z. B. einen Application Load Balancer oder API Gateway) erreicht und in die Antwort aufgenommen wird. Über das X-Ray-SDK kann jeder Microservice diesen Header lesen, hinzufügen und aktualisieren.

X-Ray funktioniert mit Amazon EC2, Amazon ECS, Lambda und AWS Elastic Beanstalk. Sie können X-Ray mit Anwendungen verwenden, die in Java, Node.js und .NET geschrieben wurden und auf diesen Services bereitgestellt werden.



AWS-X-Ray-Serviceübersicht

Epsagon ist eine vollständig verwaltete SaaS, die die Ablaufverfolgung für alle AWS-Services, APIs von Drittanbietern (über HTTP-Aufrufe) und andere gängige Services wie Redis, Kafka und Elastic umfasst. Der Epsagon-Service umfasst Überwachungsfunktionen, Warnungen für die gängigsten Services und Sichtbarkeit der Nutzlast bei jedem einzelnen Aufruf, den Ihr Code tätigt.

AWS Distro for OpenTelemetry ist eine sichere, produktionsreife, von AWS unterstützte Verteilung des OpenTelemetry-Projekts. Als Teil der Cloud Native Computing Foundation bietet AWS Distro for OpenTelemetry Open-Source-APIs, -Bibliotheken und -Agenten, um verteilte Traces und Metriken für die Überwachung von Anwendungen zu sammeln. Mit AWS Distro for OpenTelemetry können Sie Ihre Anwendungen nur einmal instrumentieren, um korrelierte Metriken und Traces an mehrere Überwachungslösungen von AWS und Partnern zu senden. Verwenden Sie Auto-Instrumentierungs-

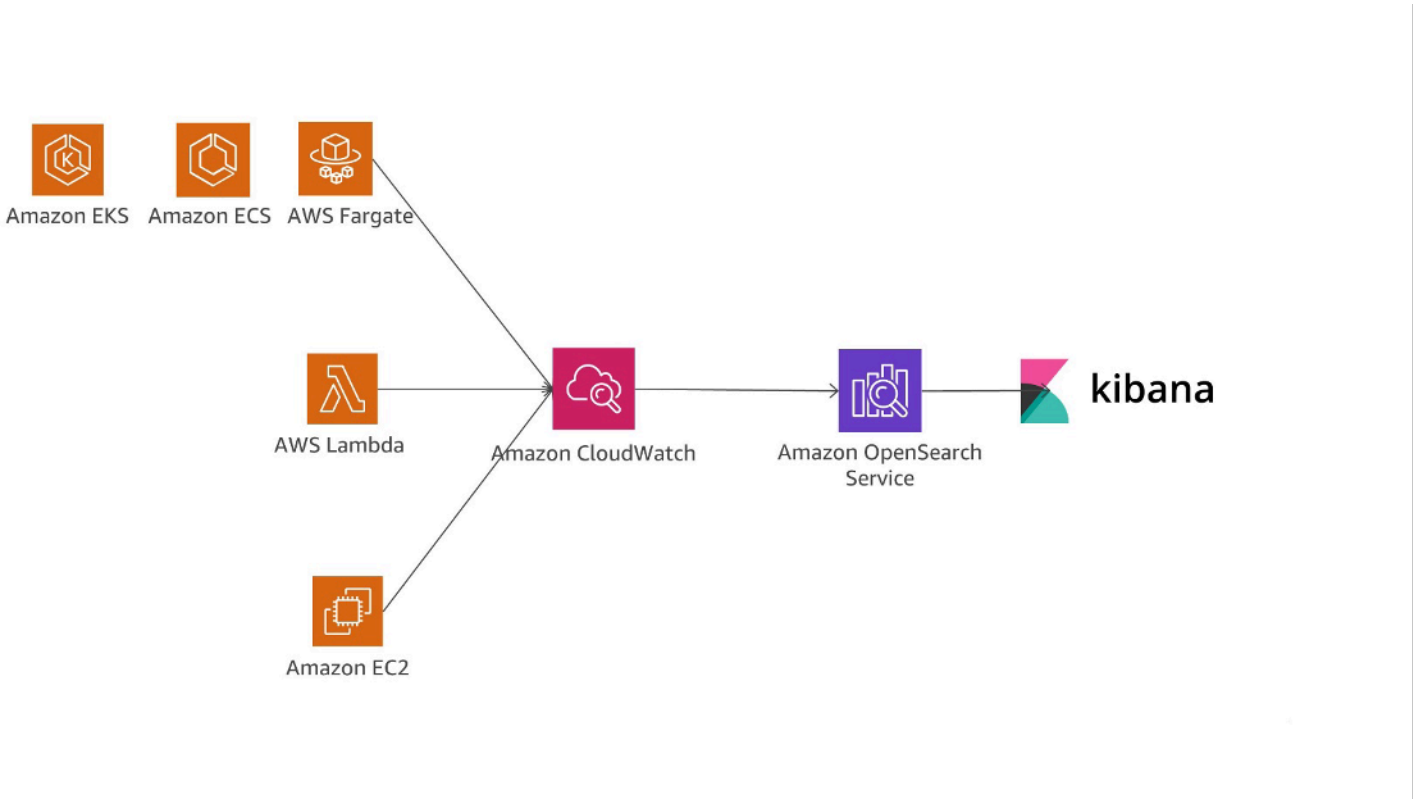
Agenten, um Traces zu erfassen, ohne Ihren Code zu ändern. AWS Distro for OpenTelemetry entnimmt auch Metadaten aus Ihren AWS-Ressourcen und verwalteten Services, so dass die Daten zur Anwendungsleistung mit den zugrunde liegenden Infrastrukturdaten korreliert werden können, was die durchschnittliche Zeit bis zur Problemlösung verkürzt. Verwenden Sie AWS Distro for OpenTelemetry, um Ihre Anwendungen zu instrumentieren, die in Amazon EC2, Amazon ECS, Amazon EKS auf Amazon EC2, Fargate und AWS Lambda sowie On-Premises ausgeführt werden.

Optionen für die Protokollanalyse in AWS

Das Suchen, Analysieren und Visualisieren von Protokolldaten ist ein wichtiger Aspekt beim Verständnis verteilter Systeme. Amazon CloudWatch Logs Insights ermöglicht Ihnen die unmittelbare Erkundung, Analyse und Visualisierung Ihrer Protokolle. Sie können so operative Probleme beheben. Eine weitere Option zur Analyse von Protokolldateien ist die Verwendung von Amazon OpenSearch Service zusammen mit Kibana.

Amazon OpenSearch Service kann für die Volltextsuche, die strukturierte Suche und Analysen oder alle drei in Kombination verwendet werden. Kibana ist ein Open-Source-Plug-In zur Datenvisualisierung, das sich nahtlos in den Amazon OpenSearch Service integriert.

Die folgende Abbildung zeigt die Protokollanalyse mit Amazon OpenSearch Service und Kibana. CloudWatch Logs lässt sich so konfigurieren, dass Protokolleinträge in Amazon OpenSearch Service nahezu in Echtzeit über ein CloudWatch-Logs-Abonnement gestreamt werden. Kibana visualisiert die Daten und stellt eine komfortable Suchoberfläche für Datenspeicher in Amazon OpenSearch Service zur Verfügung. Diese Lösung kann in Kombination mit Software wie ElastAlert verwendet werden, um ein Alarmsystem zu implementieren und SNS-Benachrichtigungen, E-Mails, JIRA-Tickets usw. zu senden, wenn Anomalien, Spitzen oder andere Interessenmuster in den Daten erkannt werden.



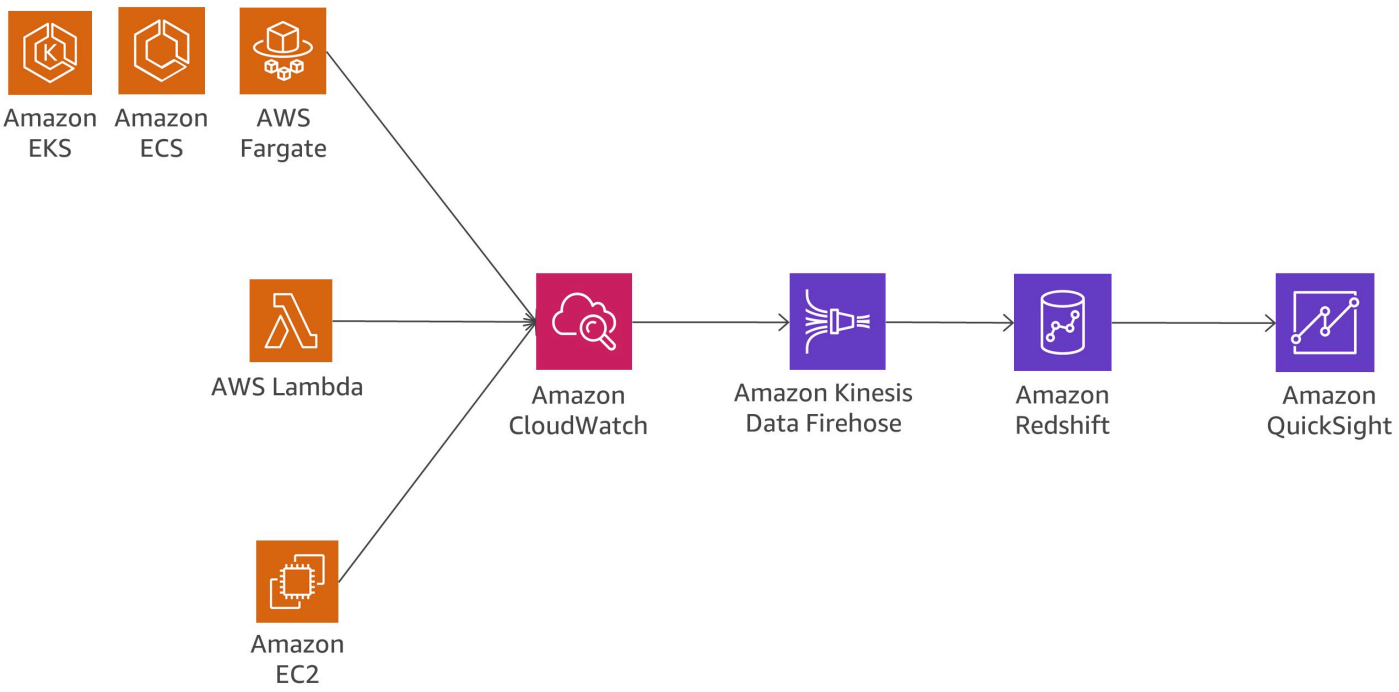
Protokollanalyse mit Amazon OpenSearch Service und Kibana

Eine weitere Möglichkeit zur Analyse von Protokolldateien besteht darin, Amazon Redshift mit Amazon QuickSight zu verwenden.

QuickSight kann einfach mit AWS-Datenservices wie Amazon Redshift, Amazon RDS, Amazon Aurora, Amazon EMR, Amazon DynamoDB, Amazon S3 und Amazon Kinesis verbunden werden.

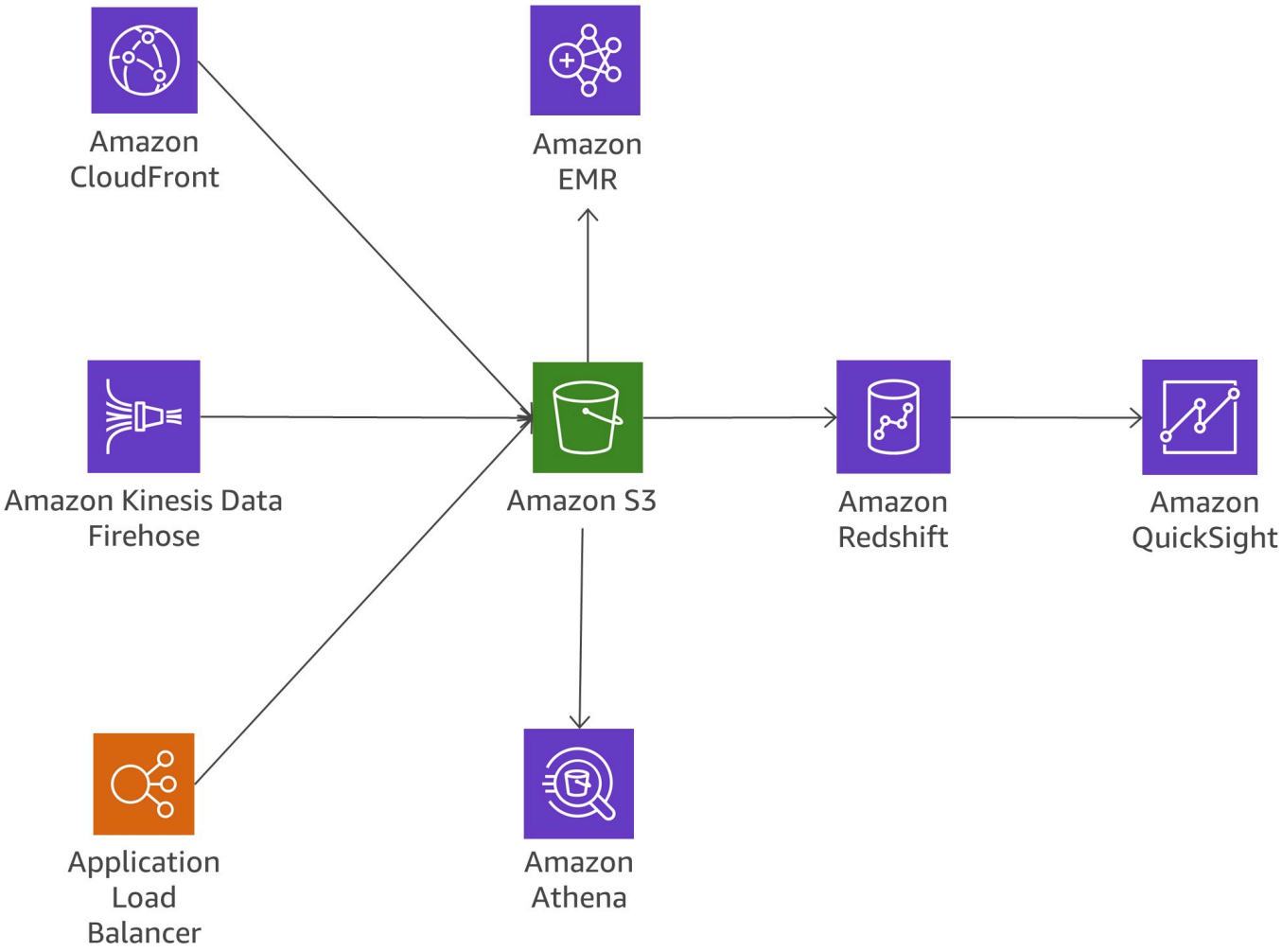
CloudWatch Logs kann als zentraler Speicher für Protokolldaten fungieren. Neben der Speicherung der Daten lassen sich Protokolleinträge auch in Amazon Kinesis Data Firehose streamen.

Die folgende Abbildung stellt ein Szenario dar, bei dem Protokolleinträge aus verschiedenen Quellen mit CloudWatch Logs und Kinesis Data Firehose an Amazon Redshift gestreamt werden. Amazon QuickSight verwendet die in Amazon Redshift gespeicherten Daten für Analysen, Berichterstellungen und Visualisierungen.



Protokollanalyse mit Amazon Redshift und Amazon QuickSight

Die folgende Abbildung zeigt ein Szenario der Protokollanalyse in Amazon S3. Wenn die Protokolle in Amazon-S3-Buckets gespeichert werden, können die Protokolldaten in verschiedene AWS-Datenservices wie Amazon Redshift oder Amazon EMR geladen werden, um die im Protokoll-Stream gespeicherten Daten zu analysieren und Anomalien zu ermitteln.



Protokollanalyse in Amazon S3

Kommunikationsaufwand

Durch die Aufteilung monolithischer Anwendungen in kleine Microservices steigt der Kommunikationsaufwand, da Microservices miteinander kommunizieren müssen. Aufgrund des einfachen Kommunikationsprotokolls wird häufig REST über HTTP verwendet. Doch hohe Nachrichtenvolumen können Probleme verursachen. In vielen Fällen empfiehlt sich die Konsolidierung von Services, die viele Nachrichten senden. Wenn Sie ein zunehmende Anzahl von Services konsolidieren, nur um den Kommunikationsaufwand zu reduzieren, sollten Sie die problematischen Domänen und Ihr Domänenmodell überprüfen.

Protokolle

In diesem Whitepaper werden im Abschnitt the section called “Asynchrone Kommunikation und einfaches Messaging” verschiedene mögliche Protokolle erörtert. Für Microservices ist es üblich, einfache Protokolle wie HTTP zu verwenden. Die von Services ausgetauschten Nachrichten können auf unterschiedliche Weise kodiert werden, z. B. in einem für Menschen lesbaren Format wie JSON oder YAML oder in effizienten Binärformaten wie Avro oder Protocol Buffers.

Caching

Caches sind eine großartige Möglichkeit, Latenz und Kommunikationsaufwand von Microservices-Architekturen zu reduzieren. Je nach Anwendungsfall und Engpässen sind mehrere Caching-Schichten möglich. Viele in AWS ausgeführten Microservice-Anwendungen verwenden Amazon ElastiCache, um die Anzahl der Aufrufe an andere Microservices zu reduzieren, indem Ergebnisse lokal zwischengespeichert werden. API Gateway bietet zur Entlastung der Backend-Server eine integrierte Caching-Schicht. Darüber hinaus ist das Caching nützlich, um die Last der Datenpersistenzschicht zu reduzieren. Die Herausforderung bei allen Caching-Mechanismen besteht darin, die richtige Balance zwischen einer guten Cache-Trefferrate und der Aktualität und Konsistenz der Daten zu finden.

Prüfung

Eine weitere Herausforderung bei Microservice-Architekturen mit potenziell Hunderten von verteilten Services besteht darin, die Transparenz der Benutzeraktionen in allen Services sicherzustellen und einen guten Gesamtüberblick auf organisatorischer Ebene zu erhalten. Um die Durchsetzung von Sicherheitsrichtlinien zu unterstützen, ist es wichtig, sowohl den Ressourcenzugriff als auch Aktivitäten zu überprüfen, die zu Systemänderungen führen.

Änderungen müssen sowohl für die einzelnen Services als auch systemweit verfolgt werden. Da in Microservice-Architekturen normalerweise häufig Änderungen vorkommen, ist deren Überprüfung besonders wichtig. In diesem Abschnitt betrachten wir die wichtigsten Services und Funktionen innerhalb von AWS, die Ihnen helfen können, Ihre Microservice-Architektur zu überprüfen.

Prüfprotokoll

AWS CloudTrail ist ein nützliches Werkzeug zur Nachverfolgung von Änderungen in Microservices, da alle API-Aufrufe in der AWS Cloud protokolliert und entweder nahezu in Echtzeit an CloudWatch Logs oder innerhalb weniger Minuten an Amazon S3 gesendet werden können.

Alle Aktionen von Benutzern und automatisierten Systeme werden durchsuchbar und können hinsichtlich unerwarteter Verhaltensweisen, Verstößen gegen Unternehmensrichtlinien oder Debugging analysiert werden. Zu den erfassten Informationen gehören Zeitstempel, Benutzer- und Kontoinformationen, der aufgerufene Service mit der angeforderten Aktion, die IP-Adresse des Aufrufers sowie Anfrageparameter und Antwortelemente.

CloudTrail ermöglicht die Definition mehrerer Prüfprotokolle für ein und dasselbe Konto, sodass verschiedene Stakeholders wie Sicherheitsadministratoren, Softwareentwickler oder IT-Auditoren ihr eigenes Prüfprotokoll erstellen und verwalten können. Wenn Microservice-Teams unterschiedliche AWS-Konten haben, ist es möglich, Prüfprotokolle in einem einzigen S3-Bucket zusammenzufassen.

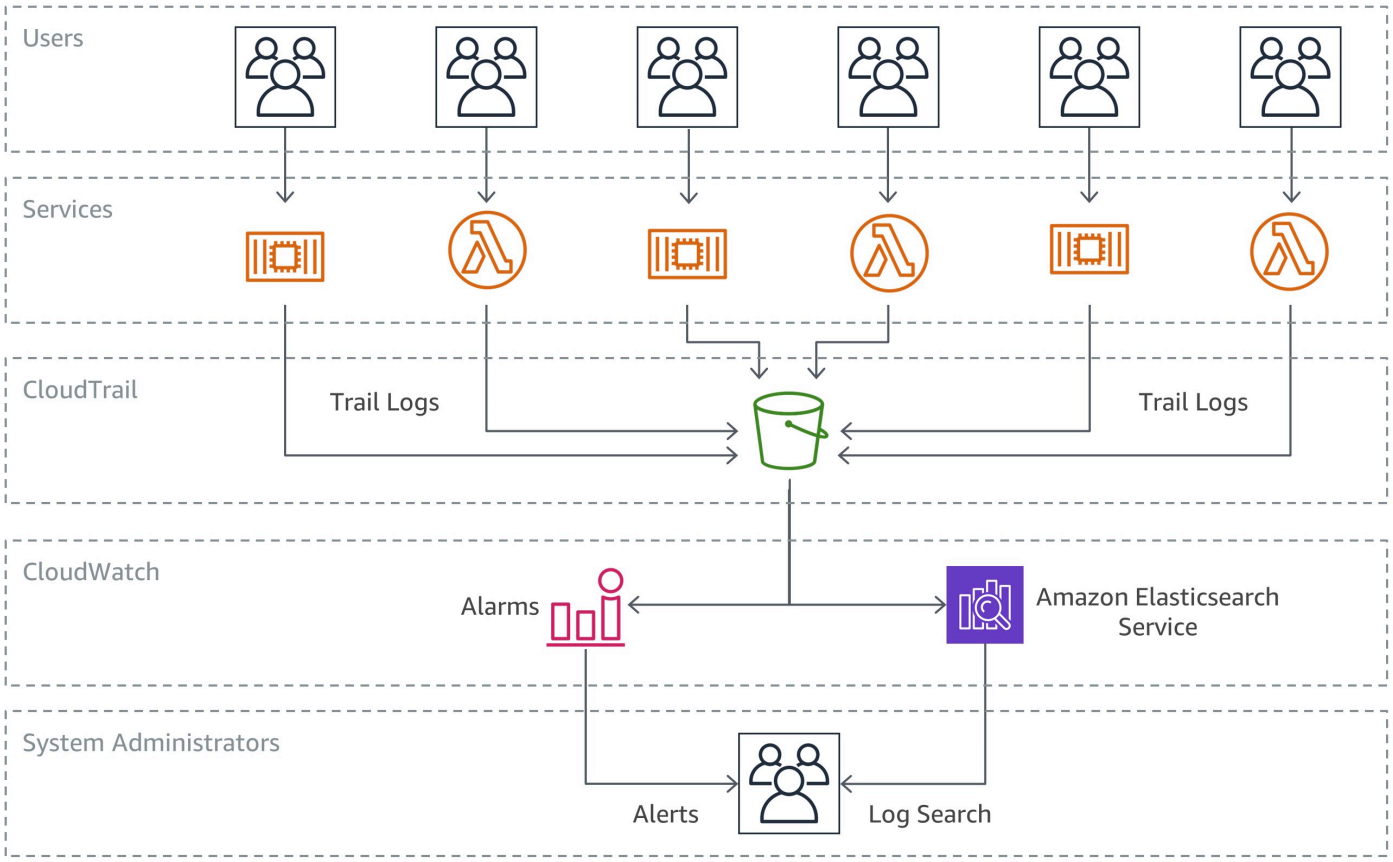
Das Speichern der Prüfprotokolle in CloudWatch hat den Vorteil, dass die Protokolldaten in Echtzeit generiert werden. Informationen lassen sich für die Suche und Visualisierung außerdem einfach an Amazon OpenSearch Service weiterleiten. Sie können CloudTrail für die Anmeldung in Amazon S3 und CloudWatch Logs konfigurieren.

Ereignisse und Echtzeitaktionen

Auf bestimmte Änderungen in Systemarchitekturen muss schnell reagiert werden. Dabei sind entweder Maßnahme zur Behebung erforderlich oder es müssen spezielle Governance-Verfahren zur Autorisierung der Änderung initiiert werden. Durch die Integration von Amazon CloudWatch Events in CloudTrail können für alle mutierenden API-Aufrufe über alle AWS-Services hinweg Ereignisse generiert werden. Außerdem lassen sich benutzerdefinierte Ereignisse definieren oder Ereignisse nach einem festen Zeitplan generieren.

Wenn ein Ereignis ausgelöst wird und einer definierten Regel entspricht, kann eine vorab definierte Personengruppe in Ihrer Organisation sofort benachrichtigt werden und entsprechende Maßnahmen ergreifen. Falls die erforderliche Aktion automatisierbar ist, kann die Regel automatisch einen integrierten Workflow auslösen oder eine Lambda-Funktion aufrufen, um das Problem zu beheben.

Die folgende Abbildung zeigt eine Umgebung, bei der CloudTrail und CloudWatch Events zusammenarbeiten, um Prüfungen und Abhilfemaßnahmen innerhalb einer Microservice-Architektur zu ermöglichen. Alle Microservices werden von CloudTrail verfolgt und das Prüfprotokoll wird in einem Amazon-S3-Bucket gespeichert. CloudWatch Events erkennt betriebsbezogene Veränderungen, sobald diese auftreten. Zudem reagiert CloudWatch Events auf diese betriebsbezogenen Änderungen und führt bei Bedarf Korrekturmaßnahmen durch, indem es an die Umgebung Nachrichten versendet, Funktionen aktiviert, Änderungen vornimmt und Statusinformationen erfasst. CloudWatch Events setzt auf CloudTrail auf und löst bei bestimmten Änderungen an der Architektur Warnmeldungen aus.



Überprüfung und Behebung von Problemen

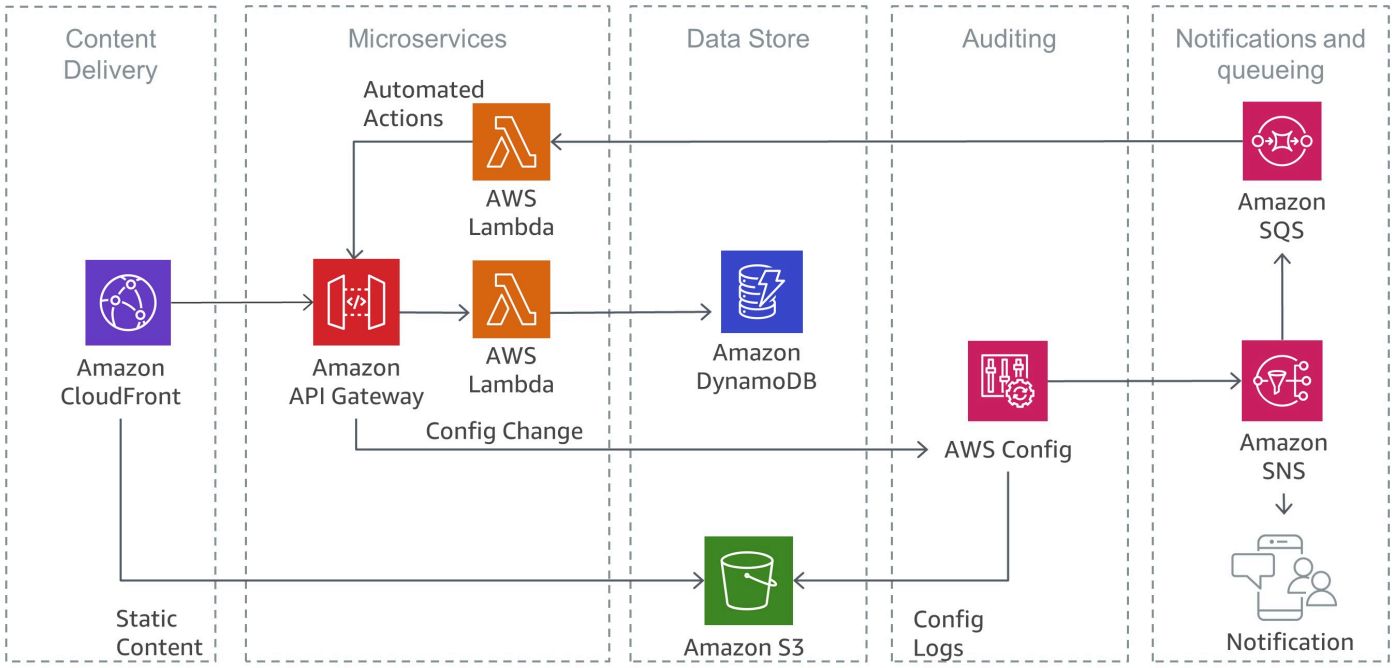
Ressourceninventarisierung und Änderungsmanagement

Um in einer agilen Entwicklungsumgebung die Kontrolle über schnell wechselnde Infrastrukturkonfigurationen zu behalten, ist zur Überprüfung und Steuerung Ihrer Architektur ein verstärkt automatisierter, verwalteter Ansatz wichtig.

Während CloudTrail und CloudWatch Events wichtige Bausteine sind, um Infrastrukturänderungen in verschiedenen Microservices zu verfolgen und darauf zu reagieren, ermöglichen die AWS-Config-Regeln einem Unternehmen, Sicherheitsrichtlinien anhand spezifischer Regeln zu definieren und Verstöße gegen diese Richtlinien automatisch zu erkennen, zu verfolgen und zu melden.

Das folgende Beispiel zeigt, wie Sie nicht konforme Konfigurationsänderungen innerhalb Ihrer Microservice-Architektur erkennen, verstehen und automatisch darauf reagieren können. Ein Mitglied des Entwicklungsteams hat in API Gateway eine Änderung an einem Microservice vorgenommen, damit der Endpunkt neben HTTPS-Anfragen auch eingehenden HTTP-Datenverkehr akzeptieren kann.

Da diese Situation zuvor vom Unternehmen als mögliches Sicherheitsproblem identifiziert wurde, wird diese Bedingung bereits durch eine AWS Config-Regel überwacht. Die Regel identifiziert die Änderung als Sicherheitsverletzung und führt zwei Aktionen aus: Es wird sowohl ein Protokoll zur Überprüfung einer erkannten Änderung in einem Amazon-S3-Bucket als auch eine SNS-Benachrichtigung erstellt. Amazon SNS wird in unserem Szenario für zwei Zwecke verwendet. Der Service benachrichtigt eine bestimmte Gruppe bezüglich der Sicherheitsverletzung und stellt eine Nachricht in eine SQS-Warteschlange. Die Nachricht wird abgerufen und der konforme Zustand wird durch eine Änderung der API-Gateway-Konfiguration wiederhergestellt.



Erkennung von Sicherheitsverletzungen mit AWS Config

Fazit

Die Microservice-Architektur ist ein verteilter Ansatz, der darauf abzielt, die Grenzen traditioneller monolithischer Architekturen zu überwinden. Microservices erleichtern das Skalieren von Anwendungen und Organisationen und verbessern gleichzeitig die Zykluszeiten. Sie stellen jedoch auch Herausforderungen dar, die zu zusätzlicher architektonischer Komplexität und Betriebslast führen können.

AWS bietet ein großes Portfolio an verwalteten Services, um Produktteams die Entwicklung von Microservice-Architekturen zu erleichtern und die architektonische und betriebliche Komplexität zu minimieren. In diesem Whitepaper werden die relevanten AWS-Services erläutert und wie Sie typische Muster wie die Serviceerkennung oder Event-Sourcing nativ mit AWS-Services implementieren können.

Ressourcen

- AWS-Architekturzentrum
- AWS-Whitepaper
- AWS Architecture Monthly
- AWS-Architekturblog
- „This Is My Architecture“-Videos
- AWS Answers
- AWS-Dokumentation

Dokumentverlauf und Mitwirkende

Dokumentverlauf

Abonnieren Sie den RSS-Feed, um über Aktualisierungen des Whitepapers benachrichtigt zu werden.

Update-Historie-Änderung	Update-Historie-Beschreibung	Update-Historie-Datum
Whitepaper aktualisiert	Integration von Amazon EventBridge, AWS OpenTelemetry, AMP, AMG, Container Insights, kleineren Textänderungen.	9. November 2021
Kleinere Aktualisierungen	Seitenlayout angepasst	30. April 2021
Kleinere Aktualisierungen	Kleinere Textänderungen.	1. August 2019
Whitepaper aktualisiert	Integration von Amazon EKS, AWS Fargate, Amazon MQ, AWS PrivateLink, AWS App Mesh, AWS Cloud Map	1. Juni 2019
Whitepaper aktualisiert	Integration von AWS Step Functions, AWS X-Ray und ECS-Ereignisströme.	1. September 2017
Erste Veröffentlichung	Implementieren von Microservices in AWS veröffentlicht.	1. Dezember 2016

> **Note**
>
> Um RSS-Aktualisierungen zu abonnieren, muss für den von Ihnen verwendeten Browser ein RSS-Plug-In aktiviert sein.

Mitwirkende

Dieses Dokument ist unter der Mitarbeit folgender Personen und Unternehmen entstanden:

- Sascha Möllering, Solutions Architecture, AWS
- Christian Müller, Solutions Architecture, AWS
- Matthias Jung, Solutions Architecture, AWS
- Peter Dalbhanjan, Solutions Architecture, AWS
- Peter Chapman, Solutions Architecture, AWS
- Christoph Kassen, Solutions Architecture, AWS
- Umair Ishaq, Solutions Architecture, AWS
- Rajiv Kumar, Solutions Architecture, AWS

Hinweise

Kunden sind eigenverantwortlich für die unabhängige Bewertung der Informationen in diesem Dokument zuständig. Dieses Dokument: (a) dient rein zu Informationszwecken, (b) spiegelt die aktuellen Produktangebote und Verfahren von AWS wider, die sich ohne vorherige Mitteilung ändern können, und (c) impliziert keinerlei Verpflichtungen oder Zusicherungen seitens AWS und dessen Tochtergesellschaften, Lieferanten oder Lizenzgebern. AWS-Produkte oder -Services werden im vorliegenden Zustand und ohne ausdrückliche oder stillschweigende Gewährleistungen, Zusicherungen oder Bedingungen bereitgestellt. Die Verantwortung und Haftung von AWS gegenüber seinen Kunden wird durch AWS-Vereinbarungen geregelt. Dieses Dokument ist weder ganz noch teilweise Teil der Vereinbarungen zwischen AWS und seinen Kunden und ändert diese Vereinbarungen auch nicht.