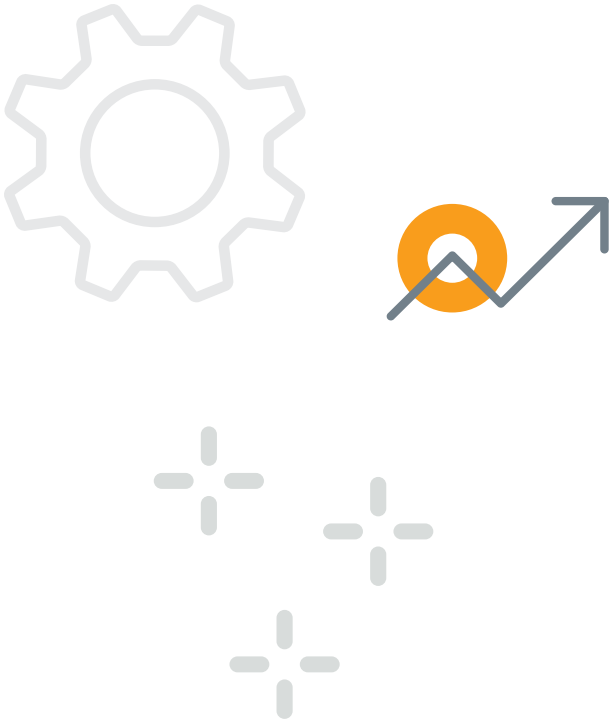
Der moderne Leitfaden zu **Container-Monitoring und Orchestrierung**

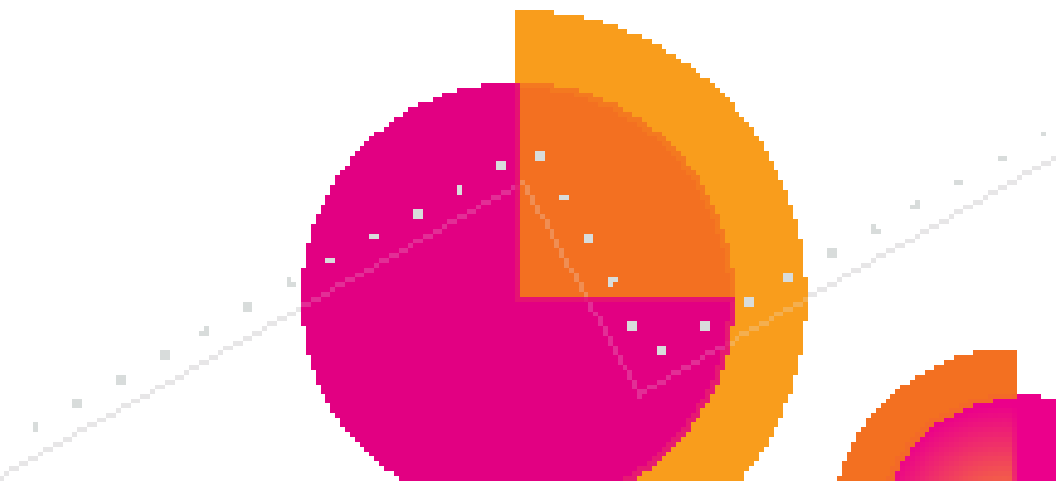
splunk>
turn data into doing



Seit der Einführung des Konzepts im Jahr 2013 sind Container zu einem der großen Themen der IT-Welt geworden. Es ist leicht zu verstehen, warum: Die Anwendungscontainer-Technologie revolutioniert die App-Entwicklung und bringt bisher ungeahnte Flexibilität und Effizienz in den Entwicklungsprozess.

Unternehmen wenden sich Containern in Scharen zu. Nach Zahlen von **Gartner** werden im Jahr 2025 mehr als 85 % der Unternehmen weltweit containerisierte Anwendungen in der Produktion einsetzen – im Jahr 2019 waren es noch weniger als 35 %. Eine so massenhafte Einführung macht deutlich, dass Unternehmen heute einen Container-basierten Entwicklungsansatz verfolgen müssen, um wettbewerbsfähig zu bleiben.

Sehen wir uns einmal an, was es mit der Containerisierung auf sich hat und wie Ihr Unternehmen sich damit einen Vorsprung verschaffen kann.

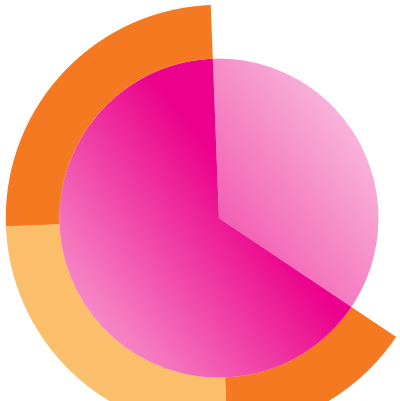


Was ist ein Container?

Die einfachste Möglichkeit, sich dem Container-Konzept zu nähern, besteht darin, dass man mit dem Namen beginnt. Ein physischer Container ist ein Behälter, der zum Aufbewahren und Transportieren von Waren von einem Ort zum anderen dient.

Ein Software-Container erfüllt eine ganz ähnliche Funktion. Er ermöglicht es Ihnen, den Code einer Anwendung, Konfigurations-dateien, Bibliotheken, Systemtools und alles andere, was zum Ausführen der Anwendung benötigt wird, in einer eigenständigen Einheit zu verpacken, sodass Sie sie überallhin verschieben und dort ausführen können.

Container sind wichtige Bestandteile von Microservices-Architekturen. Bei dieser Methode werden Anwendungen in Module mit jeweils einer einzelnen Funktion zerlegt, auf die nur bei Bedarf zugegriffen wird. Wenn Änderungen erforderlich sind, müssen die Entwickler anstelle der gesamten Anwendung nur einen einzigen dieser Services ändern oder erneut bereitstellen.



Warum sind Container so eine große Sache?

Container bieten eine Lösung für ein im Betrieb allzu häufiges Problem: Software zuverlässig und gleichförmig laufen zu lassen, egal wo sie bereitgestellt wird. Wenn eine App von einer Computerumgebung in eine andere verschoben wird – z. B. vom Staging in die Produktion – kann es zu Problemen kommen, wenn das Betriebssystem, die Netzwerktopologie, die Sicherheitsrichtlinien oder andere Aspekte der Umgebung unterschiedlich sind. Container isolieren die App von ihrer Umgebung und abstrahieren diese Umgebungsunterschiede. Darüber hinaus verhindern Container Probleme mit Abhängigkeiten, die z. B. entstehen, wenn einzelne Komponenten unterschiedliche Versionen derselben gemeinsam genutzten Bibliothek benötigen. Denn alle Abhängigkeiten sind im Container enthalten.

Vor Containern waren virtuelle Maschinen (VMs) die hauptsächlich eingesetzte Methode, um viele isolierte Anwendungen auf einem einzigen Server auszuführen. Wie Container abstrahieren VMs die zugrunde liegende Infrastruktur, sodass die App-Performance von Änderungen an Hard- und Software nicht beeinträchtigt wird. Es gibt aber einige wichtige Unterschiede in der Art und Weise, wie diese beiden Lösungen ihr Ziel erreichen.

Eine VM abstrahiert Hardware, um einen physischen Server in mehrere virtuelle Server zu verwandeln. Dies erreicht sie durch Ausführung auf einem Hypervisor, der seinerseits auf einem physischen Rechner ausgeführt wird, der als Host-Computer bezeichnet wird. Der Hypervisor ist im Wesentlichen ein Koordinationssystem, das den Zugriff auf die Ressourcen des Host-Computers (CPU, RAM usw.) regelt und sie der VM oder dem „Gastcomputer“ zur Verfügung stellt.

Die Apps und alles, was zu ihrer Ausführung benötigt wird, einschließlich Bibliotheken und Systembinärdateien, liegen auf dem Gastcomputer. Jeder Gastcomputer hat auch ein vollständiges eigenes Betriebssystem. So weist z. B. ein Server, auf dem vier VMs ausgeführt werden, vier Betriebssysteme auf – über den Hypervisor hinaus, der sie alle koordiniert. Das ist eine erhebliche Anforderung an die Ressourcen des einzelnen Rechners, und die Dinge können unter Last schnell ins Stocken geraten, was letztlich die Anzahl der VMs, die ein einzelner Server betreiben kann, begrenzt.

Container hingegen abstrahieren auf Betriebssystemebene. Auf dem Host wird ein einzelnes Host-Betriebssystem ausgeführt (dies kann ein physischer Server, eine VM oder ein Cloud-Host sein). Die Container teilen sich – unter Verwendung eines Containerisierungsmoduls wie der Docker Engine – den Betriebssystemkernel mit anderen Containern, die jeweils einen eigenen isolierten Benutzerbereich haben. Der Mehraufwand ist hier also weit geringer als bei einer virtuellen Maschine. Im Ergebnis sind Container erheblich schlanker und ressourceneffizienter als VMs.

5 Container-Vorzüge

Eine Container-basierte Infrastruktur bietet eine ganze Reihe von Vorteilen. Hier sind die fünf größten:

1. **Schneller verfügbar –** Anwendungen auf einer virtuellen Maschine benötigen normalerweise mehrere Minuten für den Start. Container brauchen nicht auf den Betriebssystemstart zu warten, daher können sie in Sekundenbruchteilen starten. Sie laufen außerdem schneller, da sie weniger Ressourcen des Host-Systems verbrauchen, und zum Erstellen, Klonen und Entfernen sind nur einige Sekunden erforderlich. All dies hat großen Einfluss auf den Entwicklungsprozess, denn Sie können damit schneller Software auf den Markt bringen, schneller Fehler beheben und schneller neue Funktionen hinzuzufügen.

2. **DevOps-geeignet –** Microservices und Container eignen sich aufgrund ihrer Geschwindigkeit, ihres schmalen Footprints und ihrer Ressourceneffizienz perfekt für eine DevOps-Umgebung. Bei einer Microservices-Infrastruktur können Entwickler durchgängig für bestimmte Teile der Anwendung verantwortlich sein, sodass sie die Funktionsweise voll und ganz verstehen. Performance-Optimierung und Problembehebung kann daher auch viel effizienter geschehen als bei monolithischen Anwendungen.

3. **Problemlos portabel –** In Containern wird die App mit all ihren Abhängigkeiten „zusammengepackt". Dadurch wird es einfach, Container auf Windows-, Linux- oder Mac-Hardware zu verschieben und dort zuverlässig auszuführen. Container können auf Bare Metal oder auf virtuellen Servern oder in Public/Private Clouds ausgeführt werden. Dies hilft auch, eine übermäßige Anbieterbindung (Vendor Lock-in) zu vermeiden, wenn Sie Ihre Apps von einer Cloud in eine andere verschieben wollen.

4. **Besser skalierbar –** Container sind in der Regel klein, weil sie anders als VMs kein separates Betriebssystem benötigen. Ein Container hat normalerweise eine zweistellige Megabyte-Größe, während eine einzelne VM leicht auf das Tausendfache kommt. Das bedeutet: Sie können viel mehr Container unter einem einzelnen Host-Betriebssystem speichern. Das wiederum erhöht die Skalierbarkeit signifikant.

5. **Konsistent programmierbar –** Da Container bereits alle Abhängigkeiten und die gesamte Konfiguration in sich enthalten, stellen sie sicher, dass Entwickler in einer konsistenten Umgebung arbeiten, unabhängig davon, wo die Container bereitgestellt sind. Das bedeutet, dass Entwickler keine Zeit damit verschwenden müssen, Umgebungsunterschiede zu beheben, sondern sich auf die Funktionalität der neuen App konzentrieren können.
Das bedeutet auch, dass Sie ein- und denselben Container auf den ganzen Weg von der Entwicklung bis zur Produktion bringen können, wenn es Zeit für den Regelbetrieb ist. Da Container nach ihrer Erstellung unveränderlich sind, müssen sich Entwickler außerdem keine Gedanken über Konfigurationsunterschiede in der Bereitstellung oder andere Problemquellen bei der Fehlerbehebung machen.

Einführung in die Orchestrierung: Kubernetes

Für den Einstieg in die Container-Orchestrierung benötigen Sie spezielle Software zur Verteilung, Verwaltung und Skalierung containerisierter Anwendungen. Eine der etabliertesten und beliebtesten Optionen ist Kubernetes, eine Open-Source-Automatisierungsplattform, die von Google entwickelt wurde und heute von der Cloud Native Computing Foundation betreut wird.

Kubernetes kann den Entwicklungsprozess erheblich verbessern, indem es das Containermanagement vereinfacht, Updates und Skalierung automatisiert und Ausfallzeiten minimiert, sodass sich Entwickler auf das Verbessern von Anwendungen und das Hinzufügen neuer Funktionen konzentrieren können. Um deutlich zu machen, wie das funktioniert, sehen wir uns die Grundkomponenten von Kubernetes und ihr Zusammenwirken an.

Kubernetes verwendet mehrere Abstraktionsebenen, die in seiner eigenen Sprache definiert sind. Kubernetes hat viele Bestandteile. Diese folgende Liste ist keineswegs vollständig, bietet aber einen vereinfachten Überblick darüber, wie Hard- und Software im System dargestellt werden.

Knoten: Im Kubernetes-Sprachgebrauch ist jeder einzelne Arbeits-computer ein „Knoten“ (Node). Dabei kann es sich um einen physischen Server oder um eine virtuelle Maschine bei einem Cloud-Anbieter wie AWS oder Microsoft Azure handeln. Knoten wurden ursprünglich als „Diener“ (Minions) bezeichnet, woraus ihr Zweck ganz gut ersichtlich ist: Sie bekommen und erfüllen Aufgaben, die ihnen der Master-Knoten zuweist, und enthalten alle Services, die für die Verwaltung und Zuweisung von Ressourcen an Container erforderlich sind.

Master-Knoten: Dies ist der Computer, der alle Arbeitsknoten orchestriert und ihren Punkten der Interaktion mit Kubernetes bildet. Alle zugewiesenen Aufgaben stammen von hier.

Cluster: Ein Cluster besteht aus einem Master-Knoten und mehreren Arbeitsknoten. Cluster bündeln alle diese Rechner zu einer einzigen, leistungsstarken Einheit. Containerisierte Anwendungen werden in einem Cluster bereitgestellt, und der Cluster verteilt die Arbeitslast dann auf verschiedene Knoten und verschiebt die Workload, wenn Knoten hinzugefügt oder entfernt werden.

Pods: Ein Pod stellt eine Sammlung von Containern dar, die zusammen verpackt werden und auf einem Cluster laufen. Alle Container eines Pods teilen sich das lokale Netzwerk und andere Ressourcen. Sie können Daten untereinander austauschen, so als ob sie sich auf dem gleichen Computer befänden, bleiben aber voneinander isoliert. Zugleich isolieren Pods Netzwerk und Massenspeicher vom zugrunde liegenden Container.

Ein einzelner Arbeitsknoten kann mehrere Pods enthalten. Wenn ein Knoten ausfällt, kann Kubernetes einen Pod als Ersatz auf einem funktionierenden Knoten bereitstellen.

Zwar kann ein Pod viele Container enthalten, es empfiehlt sich aber, nur so viele wie erforderlich zu verpacken: einen Hauptprozess und seine Hilfscontainer, die als „Sidecars“ bezeichnet werden. Pods skalieren als Einheit, unabhängig von ihren individuellen Anforderungen; überfüllte Pods können eine Belastung der Ressourcen darstellen.

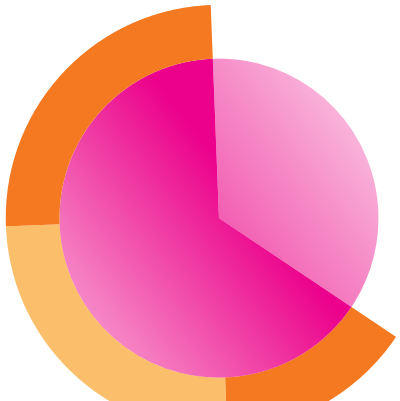
Verteilungen: Anstatt Pods direkt auf einem Cluster bereitzustellen, verwendet Kubernetes eine zusätzliche Abstraktionsschicht, die als „Verteilung“ bezeichnet wird. Eine Verteilung gibt Ihnen die Möglichkeit, festzulegen, wie viele Replikate eines Pods gleichzeitig ausgeführt werden sollen. Sobald die betreffende Anzahl Pods auf einem Cluster bereitgestellt wurde, werden sie fortlaufend überwacht und erneut bereitgestellt, falls ein Pod ausfällt.

Zugang: Kubernetes isoliert Pods von der Außenwelt, sodass Sie einen Kommunikationskanal für jeden Dienst öffnen müssen, den Sie verfügbar machen möchten. Dies ist eine weitere Abstraktionsschicht, die als Zugang („Ingress“) bezeichnet wird. Es gibt eine Reihe von Möglichkeiten, einem Cluster einen Zugang hinzuzufügen, etwa durch einen LoadBalancer, einen NodePort oder einen Ingress-Controller. Das ist in etwa vergleichbar mit dem Internet-seitigen Webserver, den Sie vielleicht bei einer herkömmlichen Architektur eingesetzt haben.

Herausforderungen von Kubernetes und Containerisierung für das Monitoring

Bei allen Vorteilen, die Container und Frameworks für die Orchestrierung in Unternehmen mit sich bringen, können sie die Verwaltung von Cloud-Anwendungen auch komplexer machen. Es ergeben sich unter anderem die folgenden Herausforderungen:

- **Tote Winkel** – Container sind sozusagen Einwegartikel. Darum führen sie mehrere Abstraktionsebenen zwischen der Anwendung und der zugrunde liegenden Hardware ein, um Portabilität und Skalierbarkeit zu gewährleisten. Daraus ergibt sich dann allerdings ein beträchtlicher toter Winkel aus Sicht des konventionellen Monitoring. Klassische Monitoring-Tools können diese Abstraktionen nicht verstehen.



- **Größeres Datenvolumen** – Die bequeme Portabilität hat einen Preis: Es braucht jetzt mehr Telemetriedaten, wenn man die Observability von Leistung und Zuverlässigkeit der Anwendungs-, Container- und Orchestrierungsplattform sicherstellen will. Viele Microservice-Architekturen sind so gebaut, dass sie ihre Microservices bei Bedarf skalieren oder entfernen können. Angesichts dieser Flüchtigkeit wird es erforderlich, die Daten in ein Observability-System zu streamen. Darüber hinaus wird die Anzahl der Elemente, die überwacht und geprüft werden müssen, wenn etwas schief läuft, durch zusätzliche Systemkomponenten noch einmal erhöht.

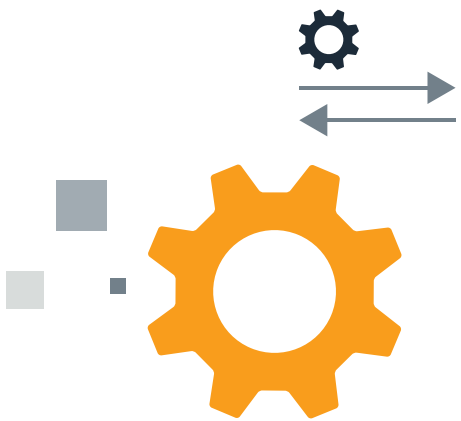
- **Anspruchsvolle Visualisierung** – Aufgrund des Umfangs und der Komplexität, die durch Microservices, Container und die Container-Orchestrierung ins Spiel kommen, müssen Sie Ihre Umgebung unbedingt visualisieren können, damit Sie direkten Einblick in den Zustand Ihrer Infrastruktur haben und einen Überblick über den Datenverkehrsfluss in Ihrer Umgebung gewinnen. Außerdem müssen Sie in der Lage sein, durch Ausschnittvergrößerungen Zustand und Leistung von einzelnen Containern, Knoten und Pods darzustellen. Eine geeignete Monitoring-Lösung sollte diesen Workflow ermöglichen.

- **Schleudergefahr für DevOps** – Container können blitzschnell skaliert und geändert werden. Durch diese beschleunigte Bereitstellung wird es für DevOps-Teams schwieriger, Leistungsveränderungen und deren Ursachen zu verfolgen oder zu erkennen, wann neue Serviceabhängigkeiten hinzugefügt werden.

So klappt‘s mit dem Container-Monitoring

Damit Sie die Übersicht über Ihre dynamische Container-Umgebung behalten, brauchen Sie eine fähige Monitoring-Lösung, die Containerdaten mit anderen Infrastrukturdaten zusammenführt und auf diese Weise eine bessere Kontextualisierung und Ursachenanalyse bietet. Am Beispiel von Docker, der beliebtesten Implementierung, wollen wir uns ansehen, wie sich ein Monitoring für die diversen Ebenen bewerkstelligen lässt.

Hosts: Die physischen und virtuellen Computer in Ihren Clustern können auf Verfügbarkeit und Leistung überwacht werden. Zu den wichtigsten Kennzahlen, die man im Blick behalten muss, gehören Arbeitsspeicher, CPU-Auslastung, belegter Auslagerungsspeicher und die Speichernutzung. Dies sollten Kernfunktionen jedes Tools zum Container-Monitoring sein.



Container: Die Observability Ihrer Container – ob aggregiert oder einzeln – ist entscheidend. Ein gutes Monitoring-Tool sollte Ihnen sagen, wie viele Container gerade laufen, welche am meisten Speicher brauchen und welcher Container zuletzt gestartet wurde. Es kann außerdem Einblick in die CPU- und Arbeitsspeichernutzung jedes Containers und den Status seiner Netzwerk-E/A geben.

Framework zur Orchestrierung: Kubernetes selbst muss ebenfalls überwacht werden. Wie viele verfügbare Knoten gibt es? Wie ist der Zustand des Master-Knotens? Gibt es Pods, die lange Zeit auf Neuzuweisung warten? Welches Datenvolumen durchläuft Ihren Zugang? Sie müssen in der Lage sein, all diese Fragen schnell zu beantworten, wenn Sie auf Dauer zuverlässige Services betreiben wollen. Darüber hinaus ist Kubernetes so beschaffen, dass Pods zur Optimierung der Ressourcennutzung geplant werden. Dadurch wird zwar einerseits die Effizienz gesteigert, andererseits wird aber auch schwerer vorhersagbar, wo Pods bereitgestellt und ausgeführt werden.

Anwendungsendpunkte: Wichtig ist auch, wann Ihr Service Benutzeranfragen verarbeiten kann und wie es um die Performance und Latenz dieser Anfragen bestellt ist. Darüber hinaus muss Ihre Monitoring-Lösung Health Checks bei der Anwendung selbst durchführen und Latenz- oder andere Performance-Metriken bestimmen können.

Was muss Container-Monitoring können?

Wie gesagt: Anwendungen in Containern samt Kubernetes und anderen Frameworks zur Orchestrierung haben enorme Vorteile in Bezug auf moderne Entwicklungs- und Bereitstellungsabläufe. Das Monitoring ist aber nicht ganz trivial und deutlich komplizierter als man es von Legacy-Anwendungen gewohnt ist. Eine Monitoring-Lösung, die sich für Container und ihre Orchestrierungsabläufe eignet, sollte daher (mindestens) die folgenden Funktionen bieten.

Schlüsselmetriken erfassen

Pod-Metriken: Anzahl der gewünschten Pods, Anzahl der verfügbaren Pods, Pods nach Phase (fehlgeschlagen, ausstehend, aktuell ausgeführt), gewünschte Pods pro Bereitstellung.

Metriken der Ressourcennutzung: Von Docker Socket erfasste Metriken (Container-Metriken und Metriken zur Ressourcennutzung auf Knotenebene, wie etwa CPU- und Arbeitsspeicherauslastung).

Anwendungsmetriken: RED-Metriken (Rate, Error, Duration), Anwendungszustand, Datenbankverfügbarkeit und -Performance.

Konsolidierungs-, Korrelations- und Analysefunktionen

Einfache Bereitstellung: Die Bereitstellung von Collectors muss einfach und Cloud-nativ sein. Im Idealfall ist ein Helm-Chart verfügbar. So einfach kann dann die Bereitstellung sein:

```
helm repo add splunk-otel-collector-chart https://signalfx.github.io/splunk-otel-collector-chart
```

```
helm repo update
```

```
helm install --set splunkAccessToken='xxx' --set clusterName='sample' --set splunkRealm='us0' --set otelCollector.enabled='true' --generate-name splunk-otel-collector-chart/splunk-otel-collector
```

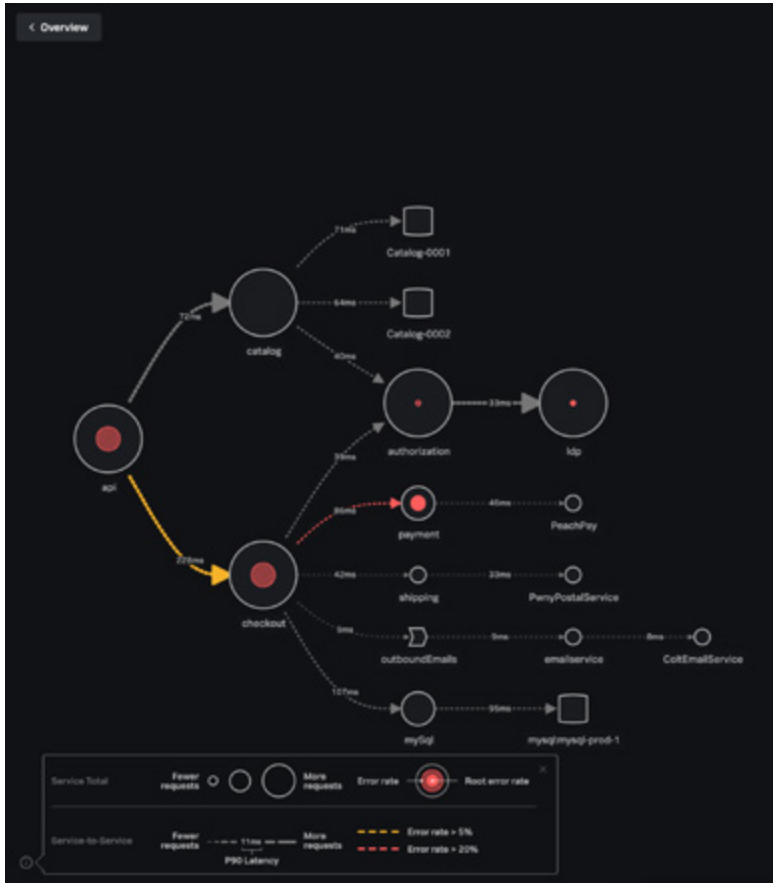
OpenTelemetry: OpenTelemetry ist die Monitoring-Zukunft, und die Instrumentierung Ihrer Umgebung muss einem möglichen Wechsel von Monitoring- oder Observability-Anbietern Rechnung tragen. Die eingeführte Lösung muss also OpenTelemetry unterstützen, damit Sie im Falle eines Wechsels nicht gezwungen sind, die ganze Instrumentierung neu anzugehen.

Echtzeit-Streaming: In einem Umfeld, in dem Ausfallzeiten pro Stunde Unsummen kosten können, zählt jede Sekunde. Eine moderne Monitoring- und Observability-Plattform muss in der Lage sein, Benachrichtigungen und Datenanalysen in Echtzeit zu liefern, damit die MTTD (Mean Time to Detect) auf ein Minimum verkürzt wird und die Schlüsse, die Sie aus Ihren Daten ziehen, auf dem aktuellen Stand des Systems beruhen.

Vorausschauende Analysen: Auf der Grundlage von KI und ML können Sie mit Predictive Analytics Elemente ermitteln, die kurz vor einem Ausfall stehen, und einem solchen auf diese Weise vorbeugen. In der Komplexität moderner Umgebungen können Sie somit verhindern, dass Fehler überhaupt auftreten, und mögliche Performance- und Verfügbarkeitsprobleme bereits erkennen, bevor sie sich auf Ihre Kunden auswirken.

Vordefinierte Dashboards: Der Nutzen Ihres Monitoring-Systems sollte nicht davon abhängen, dass Sie im Vorfeld eine vollständige Dokumentation Ihrer Infrastruktur erstellt haben. Mit automatisierten, integrierten Dashboards können Sie sich im Handumdrehen ein Bild von den komplexen Wechselbeziehungen zwischen Knoten, Pods, Containern und Anwendungen sowie vom Status Ihrer Umgebung machen:

Automatische Service-Karten: Im Container-Universum ist es extrem schwierig, einen Überblick über den Datenverkehrsfluss durch den Zugang, die Container und die Anwendungen zu gewinnen. Neben der Erkennung von Fehlern und anderen Problemen in Ihrer Umgebung muss das Monitoring-Tool in der Lage sein, den Weg Ihrer Anforderungen nachzuvollziehen und Ihnen diese Anforderungen zu zeigen:



Nächste Schritte

Container sind ein mächtiges Werkzeug in Ihrem Entwicklungsarsenal. Aber es ist wichtig, dass Sie sehen können, ob und wie Ihre Container-Umgebungen funktionieren. Infrastruktur-Monitoring und Application Performance Monitoring werden mit der Bereitstellung von Containern noch wichtiger, und eine Container-fähige Monitoring-Lösung muss etlichen Anforderungen gerecht werden: Sie muss Container analysieren können, eine einfache Bereitstellung ermöglichen, Echtzeit-Streaming und Predictive Analytics beherrschen und eine out of the box einsatzfertige Oberfläche mitbringen, die Ihnen aussagekräftige Informationen liefert. Wenn Sie für ein Observability-System bereit sind, das all diese Anforderungen erfüllt und nativ mit Containern, Public und Private Clouds sowie selbst gehosteten Umgebungen betrieben werden kann, sollten Sie sich eine Demo der Splunk Observability Cloud ansehen oder gleich mit einer kostenlosen Testversion einsteigen. Wenn Sie mehr über Observability erfahren möchten, besuchen Sie unsere Website oder laden das E-Book Observability: Ein Leitfaden für Einsteiger herunter.

splunk>

21-14769-Splunk-Modern_GER