Gewährleistung von Rollback-Sicherheit während der Bereitstellung

Sandeep Pokkunuri

Einer der Leitsätze, wie wir bei Amazon Lösungen entwickeln, lautet: Vermeiden Sie es, durch Einbahntüren zu gehen. Dies bedeutet, dass wir uns von Entscheidungen fernhalten, die schwer rückgängig zu machen oder zu erweitern sind. Wir wenden diesen Grundsatz bei allen Schritten der Softwareentwicklung an – vom Entwerfen von Produkten, Funktionen, APIs und Back-End-Systemen bis hin zu Bereitstellungen. In diesem Artikel werde ich beschreiben, wie wir diesen Grundsatz auf Softwarebereitstellungen anwenden.

Eine Bereitstellung nimmt eine Softwareumgebung von einem Status (Version) in einen anderen auf. In beiden Fällen funktioniert die Software möglicherweise einwandfrei. Die Software funktioniert jedoch möglicherweise während oder nach dem Vorwärtsübergang (Upgrade oder Roll-Forward) oder dem Rückwärtsübergang (Downgrade oder Rollback) nicht richtig. Wenn die Software nicht einwandfrei funktioniert, führt dies zu einer Betriebsstörung, die sie für Kunden unzuverlässig macht. In diesem Artikel gehe ich davon aus, dass beide Versionen der Software wie erwartet funktionieren. Ich konzentriere mich darauf, wie ich sicherstellen kann, dass das Vorwärts- oder Rückwärtsrollen während der Bereitstellung nicht zu Fehlern führt.

Vor der Veröffentlichung einer neuen Softwareversion testen wir diese in einer Beta- oder Gamma-Testumgebung in mehreren Dimensionen, z. B. Funktionalität, Parallelität, Leistung, Skalierung und nachgelagerte Fehlerbehandlung. Diese Tests helfen uns, Probleme in der neuen Version aufzudecken und zu beheben. Es kann jedoch sein, dass dies nicht immer ausreicht, um eine erfolgreiche Bereitstellung sicherzustellen. In Produktionsumgebungen können unerwartete Umstände oder suboptimales Softwareverhalten auftreten. Wir bei Amazon möchten vermeiden, dass ein Rollback der Bereitstellung für unsere Kunden zu Fehlern führen kann. Um dies zu vermeiden, bereiten wir uns vor jeder Bereitstellung vollständig auf ein Rollback vor. Eine Softwareversion, die ohne Fehler oder Unterbrechung der in der vorherigen Version verfügbaren Funktionen zurückgesetzt werden kann, wird als abwärtskompatibel bezeichnet. Wir planen und überprüfen bei jeder Überarbeitung, ob unsere Software abwärtskompatibel ist.

Bevor ich näher auf die Vorgehensweise von Amazon bei Softwareaktualisierungen eingehe, möchten wir einige der Unterschiede zwischen eigenständigen und verteilten Softwarebereitstellungen erläutern.

Eigenständige oder verteilte Softwarebereitstellungen

Bei eigenständiger Software, die als ein Prozess auf einem Gerät ausgeführt wird, handelt es sich um atomare Bereitstellungen. Zwei Versionen der Software werden niemals gleichzeitig ausgeführt. Wenn die eigenständige Software den Status beibehält, muss die neue Version Daten lesen (d. h. deserialisieren), die von der alten Version geschrieben (d. h. serialisiert) wurden, und umgekehrt. Wenn diese Bedingung erfüllt ist, ist die Bereitstellung sicher für das Vor- und Zurückrollen.

In einem verteilten System werden Bereitstellungen komplexer. Die Bereitstellung erfolgt durch fortlaufende Updates, sodass die Verfügbarkeit nicht beeinträchtigt wird. Die neue Version wird sofort auf eine Untergruppe von Hosts übertragen, damit die anderen Hosts weiterhin Anforderungen bedienen können. In der Regel kommunizieren diese Hosts über einen Remoteprozeduraufruf (Remote Procedure Call, RPC) oder einen gemeinsam genutzten dauerhaften Status (z. B. Metadaten oder Prüfpunkte) miteinander. Eine solche Kommunikation oder ein gemeinsamer Zustand kann zusätzliche Herausforderungen darstellen. Der Schreiber und der Leser können unterschiedliche Versionen der Software ausführen. Infolgedessen könnten sie die Daten unterschiedlich interpretieren. Das Lesegerät kann die Daten möglicherweise sogar nicht vollständig lesen, was zu einem Ausfall führt.

Probleme mit Protokolländerungen

Wir haben festgestellt, dass der häufigste Grund für das Nicht-Rollback eine Änderung des Protokolls ist. Betrachten Sie beispielsweise eine Codeänderung, bei der die Daten komprimiert werden, während sie auf dem Datenträger verbleiben. Nachdem die neue Version einige komprimierte Daten geschrieben hat, ist ein Rollback nicht mehr möglich. Die alte Version weiß nicht, dass Daten nach dem Lesen von der Festplatte dekomprimiert werden müssen. Wenn die Daten in einem Blob oder einem Dokumentenspeicher gespeichert sind, können andere Server sie nicht lesen, auch wenn die Bereitstellung ausgeführt wird. Wenn diese Daten zwischen zwei Prozessen oder Servern übertragen werden, kann der Empfänger sie nicht lesen.

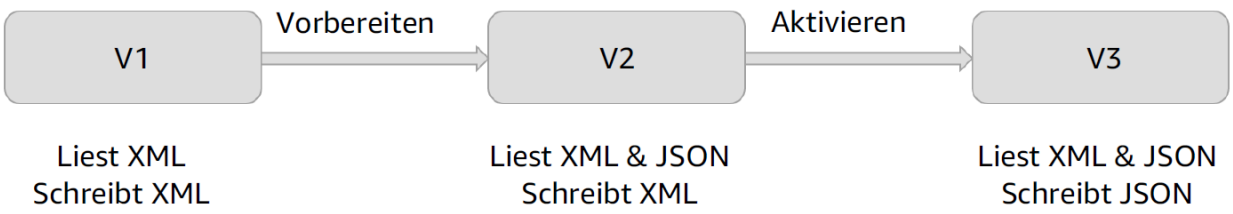
Manchmal können Protokolländerungen sehr subtil sein. Angenommen, zwei Server kommunizieren asynchron über eine Verbindung. Um sich gegenseitig zu informieren, dass sie am Leben sind, verpflichten sie sich, sich alle fünf Sekunden einen Heartbeat zu senden. Wenn ein Server innerhalb der festgelegten Zeit keinen Heartbeat sieht, wird davon ausgegangen, dass der andere Server heruntergefahren ist, und die Verbindung wird geschlossen.

Stellen Sie sich nun eine Bereitstellung vor, die den Heartbeat-Zeitraum auf 10 Sekunden erhöht. Das Festschreiben des Codes scheint geringfügig zu sein – es handelt sich lediglich um eine Änderung der Nummer. Es ist jetzt jedoch nicht sicher, sowohl vorwärts als auch rückwärts zu rollen. Während der Bereitstellung sendet der Server, auf dem die neue Version ausgeführt wird, alle 10 Sekunden einen Heartbeat. Infolgedessen wird dem Server, auf dem die alte Version ausgeführt wird, länger als fünf Sekunden kein Takt angezeigt, und die Verbindung mit dem Server, auf dem die neue Version ausgeführt wird, wird beendet. In einer großen Flotte kann dies bei mehreren Verbindungen vorkommen, was zu einem Verfügbarkeitsverlust führt.

Solche subtilen Änderungen lassen sich nur schwer durch Lesen von Code oder Designdokumenten analysieren. Aus diesem Grund stellen wir ausdrücklich sicher, dass für jede Bereitstellung ein Rollback vorwärts und rückwärts ausgeführt werden kann.

Zwei-Phasen-Bereitstellungstechnik

Eine Möglichkeit, um ein sicheres Rollback zu gewährleisten, ist die Verwendung einer Technik, die allgemein als zweiphasige Bereitstellung bezeichnet wird. Stellen Sie sich das folgende hypothetische Szenario mit einem Dienst vor, der Daten in Amazon Simple Storage Service (Amazon S3) verwaltet (schreibt, liest). Der Service wird auf einer Serverflotte in mehreren Availability Zones ausgeführt, um Skalierung und Verfügbarkeit zu gewährleisten.
Derzeit verwendet der Dienst das XML-Format, um Daten zu speichern. Wie im folgenden Diagramm in Version V1 gezeigt, schreiben und lesen alle Server XML. Aus geschäftlichen Gründen möchten wir Daten im JSON-Format beibehalten. Wenn wir diese Änderung in einer Bereitstellung vornehmen, schreiben die Server, die die Änderung übernommen haben, in JSON. Die anderen Server können JSON jedoch noch nicht lesen. Diese Situation verursacht Fehler. Aus diesem Grund teilen wir eine solche Änderung in zwei Teile und führen eine zweiphasige Bereitstellung durch.



Wie im vorhergehenden Diagramm gezeigt, nennen wir die erste Phase Vorbereiten. In dieser Phase bereiten wir alle Server auf das Lesen von JSON (zusätzlich zu XML) vor, aber sie schreiben weiterhin XML, indem sie Version V2 bereitstellen. Diese Änderung ändert aus betrieblicher Sicht nichts. Alle Server können weiterhin XML lesen und alle Daten werden weiterhin in XML geschrieben. Wenn wir diese Änderung rückgängig machen, kehren die Server in einen Zustand zurück, in dem sie JSON nicht lesen können. Dies ist kein Problem, da noch keine Daten in JSON geschrieben wurden.

Wie im vorhergehenden Diagramm gezeigt, nennen wir die zweite Phase Aktivieren. In dieser Phase aktivieren wir die Server, um das JSON-Format zum Schreiben zu verwenden, indem wir Version V3 bereitstellen. Wenn jeder Server diese Änderung aufnimmt, beginnt er mit dem Schreiben in JSON. Die Server, die diese Änderung noch nicht übernommen haben, können weiterhin JSON lesen, da sie in der ersten Phase vorbereitet wurden. Wenn wir diese Änderung rückgängig machen, befinden sich alle Daten, die von den Servern geschrieben wurden, die sich vorübergehend in der Aktivierungsphase befanden, in JSON. Daten, die von Servern geschrieben wurden, die sich nicht in der Aktivierungsphase befanden, sind in XML. Diese Situation ist in Ordnung, da die Server, wie in V2 gezeigt, nach dem Rollback weiterhin sowohl XML als auch JSON lesen können.
Obwohl das frühere Diagramm die Änderung des Serialisierungsformats von XML zu JSON zeigt, ist die allgemeine Technik auf alle Situationen anwendbar, die im Abschnitt zu den früheren Protokolländerungen beschrieben wurden. Denken Sie beispielsweise an das vorherige Szenario zurück, in dem der Heartbeat-Zeitraum zwischen Servern von fünf auf 10 Sekunden erhöht werden musste. In der Vorbereitungsphase können alle Server die erwartete Heartbeat-Zeit auf 10 Sekunden verkürzen, obwohl alle Server weiterhin alle fünf Sekunden einen Heartbeat senden. In der Aktivierungsphase ändern wir die Frequenz auf einmal alle 10 Sekunden.

Vorsichtsmaßnahmen bei zweiphasigen Einsätzen

Nun werde ich die Vorsichtsmaßnahmen beschreiben, die wir treffen, wenn wir die Technik der zweiphasigen Bereitstellung befolgen. Obwohl ich mich auf das im vorherigen Abschnitt beschriebene Beispielszenario beziehe, gelten diese Vorsichtsmaßnahmen für die meisten zweiphasigen Bereitstellungen.

Viele Bereitstellungstools ermöglichen es Benutzern, eine Bereitstellung als erfolgreich zu betrachten, wenn eine Mindestanzahl von Hosts die Änderung aufnimmt und sich selbst als fehlerfrei meldet. Beispielsweise verfügt AWS CodeDeploy über eine Bereitstellungskonfiguration mit dem Namen minimumHealthyHosts.

Eine wichtige Annahme in der zweiphasigen Beispielbereitstellung ist, dass am Ende der ersten Phase alle Server auf XML und JSON aktualisiert wurden. Wenn ein oder mehrere Server in der ersten Phase nicht aktualisiert werden, können sie während und nach der zweiten Phase keine Daten lesen. Aus diesem Grund stellen wir ausdrücklich sicher, dass alle Server die Änderung in der Vorbereitungsphase übernommen haben.

Als ich an Amazon DynamoDB arbeitete, beschlossen wir, das Kommunikationsprotokoll zwischen einer großen Anzahl von Servern zu ändern, die sich über mehrere Mikrodienste erstrecken. Als ich an Amazon DynamoDB arbeitete, entschied ich mich für das Kommunikationsprotokoll zwischen einer großen Anzahl von Servern zu ändern, die sich über mehrere Mikrodienste erstrecken. Vorsorglich habe ich explizit überprüft, ob die Bereitstellung auf jedem einzelnen Server am Ende jeder Phase erfolgreich war.

Während für jede der beiden Phasen ein Rollback sicher ist, können wir nicht beide Änderungen zurücksetzen. Im vorherigen Beispiel schreiben die Server am Ende der Aktivierungsphase Daten in JSON. Die Softwareversion, die vor den Vorbereitungs- und Aktivierungsänderungen verwendet wird, kann JSON nicht lesen. Vorsorglich lassen wir daher zwischen den Phasen Vorbereiten und Aktivieren eine beträchtliche Zeitspanne verstreichen. Wir nennen diese Zeit die Backperiode und ihre Dauer beträgt normalerweise einige Tage. Wir warten, um sicherzustellen, dass wir nicht auf eine frühere Version zurücksetzen müssen.

Nach der Aktivierungsphase können wir die Fähigkeit der Software, XML zu lesen, nicht sicher entfernen. Das Entfernen ist nicht sicher, da alle Daten, die vor der Vorbereitungsphase geschrieben wurden, in XML vorliegen. Die Fähigkeit zum Lesen von XML kann erst entfernt werden, nachdem sichergestellt wurde, dass jedes einzelne Objekt in JSON neu geschrieben wurde. Wir nennen diesen Prozess Verfüllen. Möglicherweise sind zusätzliche Tools erforderlich, die gleichzeitig ausgeführt werden können, während der Dienst Daten schreibt und liest.

Best Practices für die Serialisierung

Die meiste Software umfasst die Serialisierung von Daten – sei es für die Persistenz oder die Übertragung über ein Netzwerk. Während der Entwicklung ändert sich häufig die Serialisierungslogik. Änderungen können vom Hinzufügen eines neuen Felds bis zur vollständigen Änderung des Formats reichen. Im Laufe der Jahre haben wir einige Best Practices für die Serialisierung entwickelt:

- Generell vermeiden wir es, benutzerdefinierte Serialisierungsformate zu entwickeln.

  Die anfängliche Logik für die benutzerdefinierte Serialisierung erscheint möglicherweise trivial und bietet sogar eine bessere Leistung. Nachfolgende Iterationen des Formats stellen jedoch Herausforderungen dar, die bereits von etablierten Frameworks wie JSON, Protocol Buffers, Cap'n Proto und FlatBuffers gelöst wurden. Bei entsprechender Verwendung bieten diese Frameworks Sicherheitsfunktionen wie Escape-, Abwärtskompatibilitäts- und Attributexistenznachverfolgung (d. h. ob ein Feld explizit festgelegt oder implizit mit einem Standardwert versehen wurde).

- Bei jeder Änderung weisen wir den Serialisierern explizit eine eigene Version zu.

  Wir tun dies unabhängig vom Quellcode oder der Build-Versionierung. Wir speichern auch die Serializer-Version mit den serialisierten Daten oder in den Metadaten. Ältere Serializer-Versionen funktionieren weiterhin in der neuen Software. Wir finden es normalerweise hilfreich, eine Metrik für die Version der geschriebenen oder gelesenen Daten auszugeben. Es bietet Bedienern Informationen zur Übersicht und zur Fehlerbehebung, wenn Fehler auftreten. All dies gilt auch für RPC- und API-Versionen.

- Wir vermeiden die Serialisierung von Datenstrukturen, die wir nicht kontrollieren können.

  Beispielsweise könnten wir Java-Sammlungsobjekte mithilfe von Reflektion serialisieren. Wenn wir jedoch versuchen, das JDK zu aktualisieren, kann sich die zugrunde liegende Implementierung solcher Klassen ändern, sodass die Deserialisierung fehlschlägt. Dieses Risiko gilt auch für Klassen aus Bibliotheken, die von Teams gemeinsam genutzt werden.
- Normalerweise entwerfen wir Serialisierer, um das Vorhandensein unbekannter Attribute zu ermöglichen.

  Wo immer möglich, behalten unsere Serialisierer unbekannte Attribute bei, während sie die Daten zurückschreiben. Selbst wenn ein Server, auf dem die neue Softwareversion ausgeführt wird, während der Serialisierung neue Attribute in die Daten einbezieht, werden die Attribute auf Servern, auf denen die alte Version ausgeführt wird, beim Aktualisieren derselben Daten nicht gelöscht. Eine zweiphasige Bereitstellung ist daher nicht erforderlich.

Wie bei vielen unserer Best Practices teilen wir sie mit der Vorsicht, dass unsere Richtlinien nicht für alle Anwendungen und Szenarien gelten.

Überprüfen, ob eine Änderung für das Rollback sicher ist

Im Allgemeinen überprüfen wir ausdrücklich, ob eine Softwareänderung sicher ist, indem wir Up- und Downgrade-Tests durchführen. Für diesen Prozess richten wir eine Testumgebung ein, die für Produktionsumgebungen repräsentativ ist. Im Laufe der Jahre haben wir einige Muster identifiziert, die wir beim Einrichten von Testumgebungen vermeiden.

Ich habe Situationen erlebt, in denen die Bereitstellung einer Änderung in der Produktion Fehler verursachte, obwohl die Änderung alle Tests in der Testumgebung bestanden hatte. In einem Fall hatten die Services in der Testumgebung jeweils nur einen Server. Somit waren alle Bereitstellungen atomar, was die Möglichkeit ausschloss, verschiedene Versionen der Software gleichzeitig auszuführen. Selbst wenn in Testumgebungen nicht so viel Verkehr wie in Produktionsumgebungen auftritt, verwenden wir hinter jedem Service mehrere Server aus verschiedenen Availability Zones, so wie es in der Produktion der Fall wäre. Wir lieben Sparsamkeit bei Amazon, aber nicht, wenn es um die Sicherung von Qualität geht.

Bei einer anderen Gelegenheit hatte die Testumgebung mehrere Server. Die Bereitstellung wurde jedoch auf allen Servern gleichzeitig durchgeführt, um die Tests zu beschleunigen. Dieser Ansatz verhinderte auch, dass die alte und die neue Version der Software gleichzeitig ausgeführt wurden. Das Problem mit dem Vorrollen wurde nicht erkannt. Wir verwenden jetzt in allen Test- und Produktionsumgebungen dieselbe Bereitstellungskonfiguration.

Für Änderungen, die die Koordination zwischen Microservices erfordern, behalten wir die gleiche Bereitstellungsreihenfolge für Microservices in Test- und Produktionsumgebungen bei. Die Reihenfolge für das Vor- und Zurückrollen kann jedoch unterschiedlich sein. Beispielsweise befolgen wir im Rahmen der Serialisierung im Allgemeinen eine bestimmte Reihenfolge. Das heißt, Leser gehen vor Autoren, während sie vorwärts rollen, während Autoren vor Lesern gehen, während sie rückwärts rollen. Die entsprechende Reihenfolge wird üblicherweise in Test- und Produktionsumgebungen eingehalten.
Wenn eine Testumgebung den Produktionsumgebungen ähnlich ist, simulieren wir den

Produktionsverkehr so genau wie möglich. Zum Beispiel erstellen und lesen wir schnell hintereinander mehrere Datensätze (oder Nachrichten). Alle APIs werden kontinuierlich ausgeübt. Anschließend durchlaufen wir die Umgebung drei Phasen, von denen jede eine angemessene Dauer hat, um potenzielle Fehler zu identifizieren. Die Dauer ist lang genug, damit alle APIs, Back-End-Workflows und Batch-Aufgaben mindestens einmal ausgeführt werden können.
Erstens stellen wir die Änderung auf etwa die Hälfte der Flotte bereit, um die Koexistenz der Softwareversionen sicherzustellen. Zweitens schließen wir die Bereitstellung ab. Drittens initiieren wir die Rollback-Bereitstellung und führen dieselben Schritte aus, bis auf allen Servern die alte Software ausgeführt wird. Wenn in diesen Phasen keine Fehler oder unerwartetes Verhalten auftreten, wird der Test als erfolgreich angesehen.

Fazit

Um einen zuverlässigen Service zu gewährleisten, müssen wir sicherstellen, dass wir eine Bereitstellung für unsere Kunden störungsfrei rückgängig machen können. Das explizite Testen der Rollback-Sicherheit macht eine manuelle Analyse überflüssig, die fehleranfällig sein kann. Wenn wir feststellen, dass eine Änderung nicht rückgängig gemacht werden kann, können wir sie in der Regel in zwei Änderungen aufteilen, von denen jede vorwärts und rückwärts rückgängig gemacht werden kann.