Garantire la sicurezza del rollback durante le distribuzioni

Sandeep Pokkunuri

Uno dei principi alla base del modo in cui Amazon crea le soluzioni è: Evitare di intraprendere strade senza ritorno. Questo significa che evitiamo di compiere scelte che sono difficilmente reversibili o che non possono essere applicate a un livello più ampio. Adottiamo questo principio in tutte le fasi dello sviluppo del software, dalla progettazione dei prodotti, delle funzionalità, delle API e dei sistemi backend alle distribuzioni. In questo articolo verrà descritto il modo in cui applichiamo questo principio alle distribuzioni del software.

Una distribuzione consente di portare un ambiente software da uno stato (versione) a un altro. I software può funzionare perfettamente in ognuno di questi stati, tuttavia, potrebbe non funzionare in modo corretto durante o dopo la transizione in avanti (upgrade o rollforward) o la transizione all'indietro (downgrade o rollback). Quando il software non funziona bene, determina l'interruzione del servizio rendendolo inaffidabile per i clienti. In questo articolo si parte dal presupposto che entrambe le versioni del software funzionino come previsto. Il punto centrale è come assicurare che rollforward o il rollback non determini errori durante la distribuzione.

Prima di rilasciare una nuova versione del software, questo viene testato in un ambiente beta o gamma insieme a più dimensioni, quali funzionalità, concorrenza, prestazioni, dimensionamento e gestione degli errori a valle. I test consentono di rilevare eventuali problemi della nuova versione e di correggerli. Tuttavia, non sono sempre sufficienti ad assicurare una distribuzione senza errori. Negli ambienti di produzione si potrebbero incontrare circostanze impreviste o il software potrebbe non funzionare in modo ottimale. Noi di Amazon, desideriamo evitare di trovarci nella situazione in cui il rollback della distribuzione può causare errori ai nostri clienti. Pertanto, per far sì che ciò non accada, ci prepariamo completamente per il rollback prima di ogni distribuzione. Una versione del software che può essere sottoposta a rollback senza errori o interruzioni delle funzionalità disponibili nella versione passate è chiamata compatibile con le versioni precedenti. Noi pianifichiamo e verifichiamo che il nostro software sia compatibile con le versioni precedenti a ogni revisione.

Prima di entrare nel dettaglio sul modo in cui Amazon affronta gli aggiornamenti del software, parliamo di alcune differenze tra le distribuzioni del software standalone e quelle distribuite.

Distribuzioni del software standalone e distribuite

Per il software standalone eseguito come un processo su un singolo dispositivo, le distribuzioni sono atomiche, ossia non vengono mai eseguite contemporaneamente due versioni del software. Se il software standalone mantiene lo stato, la nuova versione deve leggere, ossia deserializzare i dati scritti, ovvero serializzati, dalla versione precedente e viceversa. La soddisfazione di questa condizione rende la distribuzione sicura per il rollforward e il rollback.

In un sistema distribuito le distribuzioni diventano più complesse. Le distribuzioni vengono effettuate mediante gli aggiornamenti, affinché la disponibilità rimanga inalterata. La nuova versione viene distribuita immediatamente in un sottoinsieme di host affinché gli altri host possano continuare a soddisfare le richieste. In genere questi host comunicano gli uni con gli altri mediante una chiamata di procedura remota (Remote Procedure Call, RPC) o uno stato persistente condiviso (ad esempio metadati o checkpoint). Tale comunicazione o stato condiviso può presentare ulteriori sfide. Lo scrittore e il lettore possono eseguire versioni del software differenti. Di conseguenza, possono interpretare in modo diverso i dati. Il lettore potrebbe persino non riuscire a leggere i dati tutti insieme causando un'interruzione dell'attività.

Problemi dovuti alle modifiche del protocollo

Abbiamo rilevato che il motivo più comune per cui il rollback non riesce è la modifica di un protocollo. Pensa, ad esempio, a un codice che inizia a comprimere i dati e che li rende al contempo persistenti nel disco. Quando la nuova versione scrive dati compressi, il rollback non è possibile. La versione precedente non sa che deve decomprimere i dati dopo averli letti dal disco. Se i dati sono memorizzati in un blob o in un archivio documenti, gli altri server non saranno in grado di leggerli neanche quando la distribuzione è in corso. Se questi dati vengono trasferiti tra due processi o server, il sistema ricevente non riuscirà a leggerli.

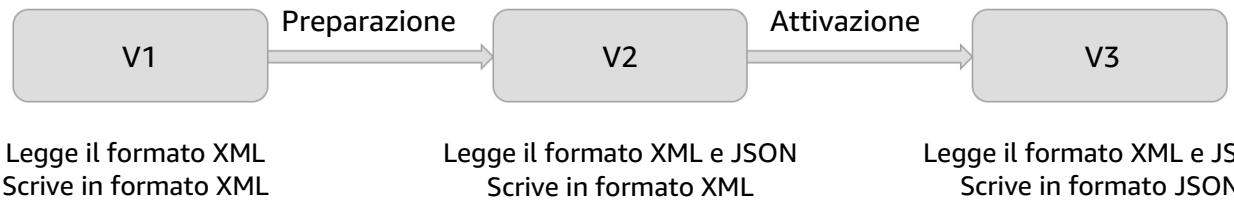
Talvolta, le modifiche del protocollo possono essere minime. Considera, ad esempio, due server che comunicano in modo asincrono su una connessione. Per mantenersi reciprocamente consapevoli della propria attività, i server si accordano per inviare un heartbeat l'uno all'altro ogni cinque secondi. Se un server non riceve un heartbeat entro il periodo di tempo concordato, presuppone che l'altro server sia inattivo e chiude la connessione.

Considera ora una distribuzione che aumenta il periodo dell'heartbeat a 10 secondi. Il commit del codice sembra di lieve entità, in quanto esiste solo la modifica di un numero. Tuttavia, ora né il rollforward né il rollback sono sicuri. Durante la distribuzione, il server che esegue la nuova versione invia un heartbeat ogni 10 secondi. Di conseguenza, il server che esegue la versione precedente non rileva un heartbeat per oltre cinque secondi e termina la connessione con il server che esegue la nuova versione. In una flotta di grandi dimensioni questa situazione può verificarsi su diverse connessioni, determinando pertanto una riduzione della disponibilità.

Tali piccole modifiche sono difficili da analizzare mediante la lettura del codice o dei documenti di progettazione. Pertanto, verifichiamo in modo esplicito che ciascuna distribuzione sia sicura per eseguire il rollforward e il rollback.

Tecnica di distribuzione in due fasi

Un modo in cui ci assicuriamo di poter eseguire il rollback in sicurezza consiste nell'utilizzare una tecnica comunemente denominata distribuzione in due fasi. Considera il seguente scenario ipotetico con un servizio che gestisce i dati (scrittura, lettura) su Amazon Simple Storage Service (Amazon S3). Questo servizio viene eseguito su una flotta di server in più zone di disponibilità per motivi di dimensionamento e disponibilità.
Attualmente, per la persistenza dei dati, il servizio utilizza il formato XML. Come mostrato nel seguente diagramma nella versione V1, tutti i server scrivono e leggono in formato XML. Per motivi aziendali, desideriamo eseguire la persistenza de dati in formato JSON. Se apportiamo questa modifica in una distribuzione, i server che implementano la modifica scriveranno in formato JSON. Tuttavia, gli altri server non saranno in grado di leggere il formato JSON, pertanto si verificheranno errori. Dividiamo quindi tale modifica in due parti ed eseguiamo una distribuzione in due fasi.



Come mostrato nel diagramma precedente, chiamiamo la prima fase Preparazione. In questa fase prepariamo tutti i server affinché leggano il linguaggio JSON (oltre all'XML), ma continuino a scrivere in XML mediante la distribuzione della versione V2. Questa modifica non determina alcun cambiamento da un punto di vista operativo. Tutti i server possono ancora leggere il formato XML e tutti i dati vengono ancora scritti in XML. Se decidiamo di effettuare il rollback di questa modifica, i server tornano a una condizione in cui non possono leggere il formato JSON. Tuttavia, questo non rappresenta un problema perché nessuno dei dati è stato ancora scritto in formato JSON.

Come mostrato nella diagramma precedente, chiamiamo questa seconda fase Attivazione. In questa fase attiviamo i server affinché utilizzino il formato JSON per la scrittura mediante la distribuzione della versione V3. Man mano che il server implementa questa modifica inizia a scrivere in formato JSON. I server che devono ancora implementare questa modifica possono comunque leggere il formato JSON perché sono stati preparati nella prima fase. Se decidiamo di eseguire il rollback di questa modifica, tutti i dati scritti dai server che si trovavano temporaneamente nella fase Attivazione saranno in formato JSON. I dati scritti dai server che non si trovavano nella fase Attivazione saranno in formato XML. Questa situazione va bene perché, come mostrato in V2, dopo il rollback i server sono ancora in grado di leggere sia il formato XML che il formato JSON.
Sebbene la figura precedente mostri la modifica del formato di serializzazione da XML in JSON, la tecnica generale è applicabile a tutte le situazioni descritte nella sezione relativa alle modifiche del protocollo. Pensa, ad esempio, allo scenario precedente in cui il periodo dell'heartbeat tra i server deve essere aumentato da 5 a 10 secondi. Nella fase Preparazione possiamo far sì che tutti i server adottino il periodo di heartbeat previsto di 10 secondi sebbene tutti i server continuino a inviare un heartbeat una volta ogni 5 secondi. Nella fase Attivazione possiamo modificare la frequenza a un heartbeat ogni 10 secondi.

Precauzioni con le distribuzioni in due fasi

Descriverò ora le precauzioni che adotteremo durante l'esecuzione della tecnica di distribuzione in due fasi. Sebbene mi stia riferendo allo scenario di esempio descritto nella sezione precedente, queste precauzioni si applicano alla maggior parte delle distribuzioni in due fasi.

Molti strumenti di distribuzione consentono agli utenti di considerare una distribuzione eseguita correttamente se un numero minimo di host adotta la modifica e segnala se stessi come integri. Ad esempio, AWS CodeDeploy ha una configurazione della distribuzione denominata minimumHealthyHosts.

Un presupposto fondamentale della distribuzione in due fasi di esempio è che alla fine della prima fase tutti i server sono stati sottoposti a upgrade affinché potessero leggere il linguaggio XML e JSON. Se uno o più server non riescono a eseguire l'upgrade durante la prima fase non riusciranno a leggere i dati durante e dopo la seconda fase. Pertanto, verifichiamo esplicitamente che tutti i server abbiano implementato la modifica nella fase Preparazione.

Quando stavo lavorando su Amazon DynamoDB, abbiamo deciso di modificare il protocollo di comunicazione tra un elevato numero di server che distribuivano più microservizi. Ho coordinato le distribuzioni tra tutti i microservizi affinché tutti i server raggiungessero prima la fase Preparazione, quindi procedessero alla fase Attivazione. Come precauzione ho esplicitamente verificato che, alla fine di ciascuna fase, la distribuzione fosse corretta su ogni singolo server.

Mentre una delle due fasi è sicura per il rollback, non possiamo eseguire il rollback di entrambe le modifiiche. Nell'esempio precedente, alla fine della fase Attivazione, i server scrivono i dati in formato JSON. La versione del software in uso prima delle modifiche delle fasi Preparazione e Attivazione non è in grado di leggere il formato JSON. Pertanto, come precauzione, abbiamo fatto passare un considerevole periodo di tempo tra le fasi Preparazione e Attivazione. Chiamiamo questo tempo periodo di messa a punto e la sua durata è in genere pochi giorni. Attendiamo per essere certi di non dover eseguire il rollback a una versione precedente.

Dopo la fase Attivazione possiamo rimuovere senza rischi la funzione del software per leggere il formato XML. Rimuovere tutti i dati scritti prima della fase Preparazione in formato XML non è sicuro. Possiamo eliminare la capacità di leggere il formato XML solo dopo esserci assicurati che ogni singolo oggetto sia stato riscritto in JSON. Questo processo è denominato backfilling. Potrebbero essere necessari ulteriori strumenti che possono essere eseguiti contemporaneamente mentre il servizio scrive e legge i dati.

Best practice per la serializzazione

La maggior parte dei software richiede la serializzazione dei dati per la persistenza o il trasferimento degli stessi su una rete. Man mano che la logica della serializzazione dei dati evolve, è normale che cambi. Le modifiche possono andare dall'aggiunta di un nuovo campo alla modifica completa del formato. Nel corso degli anni abbiamo sviluppato alcune best practice che adottiamo per la serializzazione:

- In genere evitiamo di sviluppare formati di serializzazione personalizzati.

  Sebbene la logica iniziale per la serializzazione personalizzata può sembrare richiedere poco sforzo assicurando anche prestazioni migliori, le iterazioni successive del formato pongono delle sfide, che sono state tuttavia già superate grazie a framework ben collaudati quali JSON, Protocol Buffers, Cap’n Proto e FlatBuffers. Se utilizzati in modo appropriato, questi framework forniscono funzionalità di sicurezza quali utilizzo dei caratteri di escape, compatibilità con le versioni precedenti e monitoraggio dell'esistenza degli attributi, per vedere, ad esempio, se un campo è stato impostato in modo esplicito oppure gli è stato assegnato un valore predefinito.

- Con ogni modifica assegniamo esplicitamente una versione distinta ai serializzatori.

  Eseguiamo questa operazione in modo indipendente dal codice sorgente o da versioni multiple della build. Inoltre, memorizziamo la versione del serializzatore con i dati serializzati o nei metadati. Le versioni precedenti del serializzatore continuano a funzionare nel nuovo software. In genere troviamo utile utilizzare una metrica per la versione dei dati scritti o letti, che fornisca agli operatori informazioni sulla visibilità e sulla risoluzione dei problemi, in caso di errori. Tutto questo viene applicato anche alle versioni delle RPC e delle API.
- Evitiamo la serializzazione delle strutture dei dati che non possiamo controllare.

  Ad esempio, potremmo serializzare gli oggetti della raccolta di Java utilizzando il riflesso. Tuttavia, quando tentiamo di eseguire l'upgrade di JDK, l'implementazione sottostante di tali classi può cambiare, causando un errore della serializzazione. Il rischio esiste anche nelle classi delle librerie condivise tra i team.

- In genere, progettiamo serializzatori per consentire la presenza di attributi sconosciuti.

  Ove possibile, i nostri serializzatori mantengono attributi sconosciuti durante la riscrittura dei dati. In questo modo, anche se un server che esegue la nuova versione del software include nuovi attributi nei dati durante la serializzazione, i server che eseguono la versione precedente non cancelleranno gli attributi durante l'aggiornamento dei dati. In questo modo, la distribuzione in due fasi non è necessaria.

Come in molti casi, condividiamo le nostre best practice con la riserva che le nostre linee guida non sono idonee a tutte le applicazioni e gli scenari.

Verifica che una modifica è sicura per il rollback

In genere, verifichiamo in modo esplicito che una modifica del software sia sicura per il rollforward e il rollback mediante un processo che chiamiamo test dell'upgrade-downgrade. Per questo processo allestiamo un ambiente di test rappresentativo degli ambienti di produzione. Nel corso degli anni abbiamo identificato alcuni modelli che evitiamo di utilizzare per la creazione degli ambienti di test.

Ho visto situazioni in cui la distribuzione di una modifica in produzione aveva causato errori anche se aveva superato tutti i test a cui era stata sottoposta nell'ambiente di test. In un'occasione a ciascun servizio dell'ambiente di test era associato un singolo server. Pertanto, tutte le distribuzioni erano atomiche, condizione che precludeva la possibilità di eseguire contemporaneamente versioni del software differenti. Ora, anche se gli ambienti di test non ospitano così tanto traffico come gli ambienti di produzione, utilizziamo server multipli di zone di disponibilità differenti per ciascun servizio, proprio come in un ambiente di produzione. Amazon apprezza la parsimonia, ma non quando si tratta di assicurare la qualità.

In un'altra occasione l'ambiente di server includeva più server, tuttavia, la distribuzione veniva effettuata su tutti i server contemporaneamente per accelerare i test. Anche questo approccio aveva impedito l'esecuzione contemporanea di versioni nuove e precedenti. Il problema con il rollforward non era stato rilevato. Attualmente, utilizziamo la stessa configurazione della distribuzione in tutti gli ambienti di test e di produzione.

Per le modifiche che comportano il coordinamento tra i microservizi manteniamo lo stesso ordine di distribuzione tra gli stessi negli ambienti di test e di produzione. Tuttavia, l'ordine in cui vengono eseguiti il rollforward e il rollback può essere differente. Ad esempio, in genere seguiamo un ordine specifico nel contesto della serializzazione. Ossia, i lettori vengono prima degli scrittori durante il rollforward mentre gli scrittori vengono prima dei lettori durante il rollback. L'ordine appropriato viene comunemente seguito negli ambienti di test e di produzione.

Quando la configurazione di un ambiente di test è simile a quella degli ambienti di produzione, simuliamo il più strettamente possibile il traffico dell'ambiente di produzione. Ad esempio, creiamo e leggiamo più record o messaggi in rapida successione. Tutte le API vengono testate di continuo. Quindi, l'ambiente di test passa attraverso tre fasi, ciascuna delle quali ha una durata ragionevole per consentire di identificare potenziali errori. La durata è sufficientemente lunga per eseguire almeno una volta tutte le API, i flussi di lavoro backend e i processi in batch.
Prima, distribuiamo la modifica su circa metà della flotta per assicurare la coesistenza delle versioni del software. Quindi, completiamo la distribuzione. Infine, iniziamo la distribuzione del rollback

e seguiamo gli stessi passaggi fino a quando tutti i server eseguono il software precedente. Se durante queste fasi non vengono rilevati errori o comportamenti imprevisti, il test viene considerato superato.

Conclusione

Avere la certezza di poter eseguire il rollback di una distribuzione senza causare interruzioni dell'attività ai nostri clienti è fondamentale per rendere un servizio affidabile. Eseguire esplicitamente test per verificare la sicurezza del rollback elimina la necessità di effettuare l'analisi manuale, la quale può favorire errori. Quando scopriamo che una modifica non è sicura per il rollback, la suddividiamo in genere in due modifiche, ciascuna delle quali è sicura per il rollback e il rollforward.