Automatizzazione di distribuzioni pratiche e sicure

Clare Liguori

Copyright © 2020, Amazon Web Services, Inc. o società affiliate. Tutti i diritti riservati

Durante il mio colloquio di lavoro ad Amazon, mi sono assicurata di chiedere a un esaminatore: "Con quale frequenza effettuate la distribuzione in produzione?" All'epoca lavoravo a un prodotto che prevedeva la distribuzione di una versione principale una o due volte l'anno, ma talvolta dovevo rilasciare piccoli aggiornamenti tra una versione e l'altra. Per ogni aggiornamento rilasciato, dedicavo ore del mio tempo a una distribuzione accurata. Quindi, controllavo in modo frenetico log e parametri per verificare di non aver danneggiato nulla al termine della distribuzione e se era necessario eseguire il rollback.

Ho letto che Amazon si occupava di distribuzione continua, per cui in fase di colloquio volevo sapere quanto tempo avrei dovuto dedicare alla gestione e visualizzazione di distribuzioni in qualità di sviluppatore. L'esaminatore mi ha detto che le pipeline di distribuzione continua si occupavano della distribuzione automatica delle modifiche in produzione più volte al giorno. Quando ho chiesto informazioni sul tempo che dedicava ogni giorno al controllo di queste distribuzioni e alla visualizzazione di log e parametri relativi agli impatti generati, come facevo io, mi ha risposto che non dedica tempo a queste attività, poiché le pipeline se ne occupavano per conto del suo team. Per questo, nessuno dedicava tempo a visualizzare attivamente la maggior parte delle distribuzioni. "Wow!", ho detto. Dopo essere entrata a far parte di Amazon, non vedevo l'ora di scoprire il preciso funzionamento di queste "pratiche" distribuzioni automatiche.

In che modo una distribuzione continua sicura fornisce più tempo allo sviluppatore

Da allora, ho visto di persona il modo in cui Amazon configura pipeline di distribuzione continua per distribuire in maniera rapida e sicura. Ho imparato ad apprezzare il modo in cui le pratiche sicure di distribuzione continua permettessero allo sviluppatore di lavorare meno sulle distribuzioni. Quando eseguo il push di un codice di produzione in un ramo principale del repository del codice sorgente del servizio, me ne dimentico e passo all'attività seguente, mentre la pipeline del team se ne occupa apportando quella modifica in produzione. La pipeline automatizza interamente la distribuzione del codice modificato al servizio di produzione, il che significa che l'ultima volta che io o un altro sviluppatore modifica o controlla una parte del codice è al momento in cui viene unito nel repository del codice sorgente.

Il mio team si occupa della configurazione della pipeline con fasi automatizzate che distribuiscono in modo sicuro le modifiche in produzione. Così facendo, non è necessario visualizzare ogni singola distribuzione. La pipeline esegue le modifiche più recenti tramite un set di test e controlli di sicurezza sulla distribuzione. Queste fasi automatizzate evitano che i problemi abbiano un impatto sul cliente, dal manifestarsi in produzione al limitarne l'impatto sul cliente, nel caso la produzione presenti problemi. In qualità di sviluppatore, mi fido del fatto che la pipeline distribuisca in modo sicuro e accurato le modifiche apportate in produzione per mio conto, senza dover controllare attivamente.

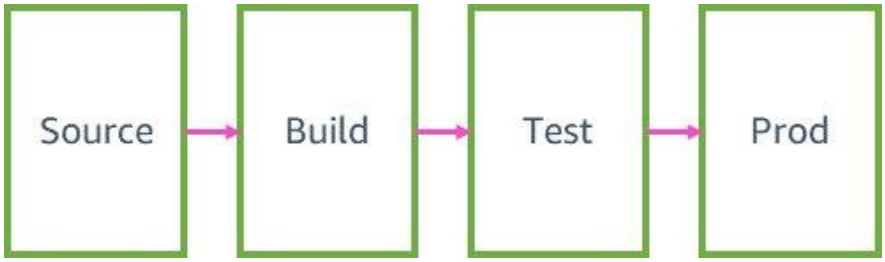
Il percorso verso una distribuzione continua

Amazon non sempre ha fatto uso della distribuzione continua e gli sviluppatori hanno dedicato giorni e ore del loro tempo alla gestione delle distribuzioni di codici in produzione. Abbiamo adottato le distribuzioni continue all'interno dell'azienda per automatizzare e standardizzare la distribuzione del software e per ridurre il tempo necessario ad apportare le modifiche in produzione. Le migliorie al nostro processo di distribuzione hanno inciso progressivamente sul tempo dedicato a questa attività.

Abbiamo identificato i rischi di distribuzione e i modi per ridurli, grazie a una nuova automazione sicura nelle pipeline. Continuiamo a iterare il processo di distribuzione, identificando nuovi rischi e modi per renderlo più sicuro. Per ulteriori informazioni sul percorso di distribuzione continua e sulle costanti migliorie apportate, leggi l'articolo della Builders' Library Più velocità con una consegna continua.

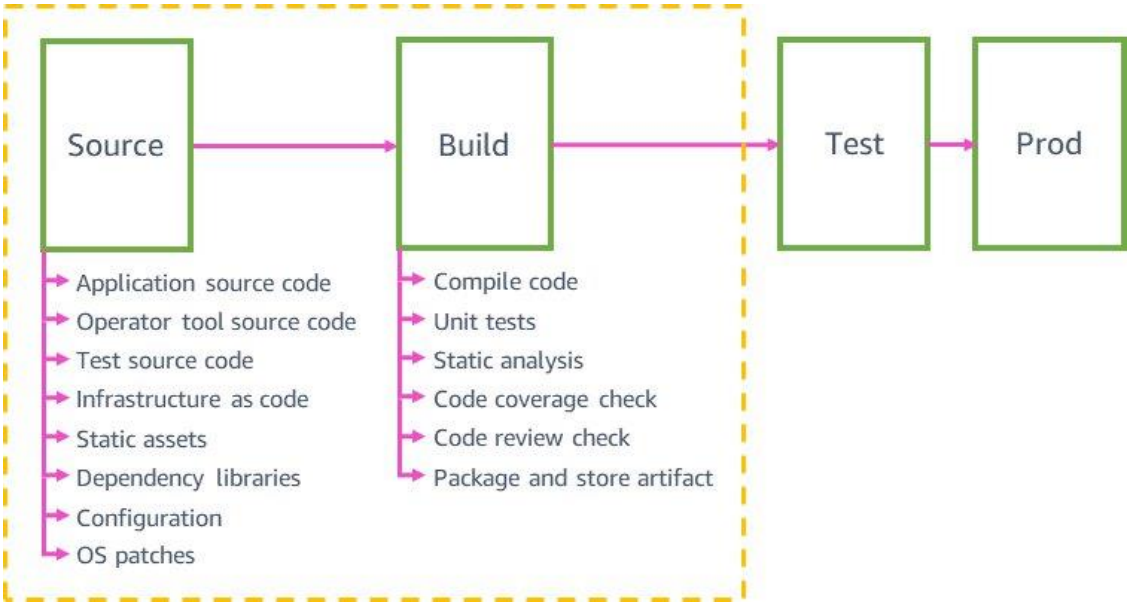
Le quattro fasi della pipeline

In questo articolo, illustriamo le fasi di modifica al codice in una pipeline di Amazon nel percorso verso la produzione. Generalmente, una pipeline di distribuzione continua presenta quattro fasi: origine, build, test e produzione (prod). Per un comune servizio AWS, vedremo nel dettaglio cosa succede in ogni fase della pipeline e forniremo un esempio di come il team che si occupa di quel servizio potrebbe configurare una pipeline.



Origine e build

Il diagramma seguente fornisce una panoramica delle fasi di origine e build tipiche delle pipeline di un team di un servizio AWS.



Origini della pipeline

In Amazon le pipeline convalidano automaticamente e distribuiscono tutte le modifiche apportate all'origine in produzione in modo sicuro. Pertanto, non si occupa esclusivamente delle modifiche al codice dell'applicazione. Possono convalidare e distribuire le modifiche alle origini, come test, strumenti, infrastrutture, configurazione, risorse statiche di siti Web e sistemi operativi (SO) sottostanti dell'applicazione. Tutte queste modifiche sono versioni controllate presenti in repository del singolo codice sorgente. Le dipendenze del codice sorgente, come librerie, linguaggi di programmazione e parametri (tra cui ID AMI), vengono aggiornati automaticamente alla versione più recente almeno una volta alla settimana.

Queste origini sono distribuite in pipeline individuali con gli stessi meccanismi di sicurezza (tra cui rollback automatico) utilizzati per la distribuzione del codice dell'applicazione. Ad esempio, i valori di configurazione di un servizio che possono subire modifiche in fase di runtime (come gli aumenti dei limiti del tasso di API e i flag delle caratteristiche) vengono distribuiti automaticamente in una pipeline di configurazione dedicata. Nel caso in cui le modifiche all'origine sono causa di problemi in produzione per il servizio (ad esempio, errori nell'elaborazione di un file di configurazione), eseguiamo automaticamente il rollback di queste modifiche.

Un microservizio tipico potrebbe disporre delle pipeline seguenti: code pipeline dell'applicazione, pipeline dell'infrastruttura, di patch del sistema operativo, di flag delle caratteristiche/di configurazione e degli strumenti operativi. Disporre di più pipeline per lo stesso microservizio permette di distribuire le modifiche in produzione con maggiore rapidità. Le modifiche al codice dell'applicazione che non superano i test sull'integrazione e bloccano la pipeline dell'applicazione non influenzano le altre pipeline. Ad esempio, non bloccano le modifiche al codice dell'infrastruttura dal raggiungere la produzione nella pipeline dell'infrastruttura. Tutte le pipeline per lo stesso microservizio tendono ad assomigliarsi. Ad esempio, una pipeline di flag delle caratteristiche usa le stesse tecniche di distribuzione sicura della pipeline del codice dell'applicazione, perché una modifica scadente alla configurazione della prima, così come una modifica inadeguata al codice della seconda, possono influenzare allo stesso modo la produzione.

Revisione del codice

Tutte le modifiche in produzione iniziano con la revisione del codice e devono essere approvate da un membro del team prima di essere accorpate al *ramo principale* (la nostra versione di "principale" o "tronco"), che avvia automaticamente la pipeline. La pipeline vincola il requisito secondo cui tutti i commit del ramo principale devono essere codici rivisti e approvati da un membro del team di servizio di quella pipeline. La pipeline bloccherà la distribuzione di tutti i commit non rivisti.

Con pipeline completamente automatizzate, la revisione del codice è l'ultima revisione manuale e approva la revisione di una modifica al codice da parte di un ingegnere prima di distribuirla in produzione. Si tratta di una fase cruciale. I revisori di codici ne valutano la correttezza e determinano se la modifica apportata può essere distribuita in produzione in sicurezza. Valutano inoltre se il codice dispone di test sufficienti (test di unità, di integrazione e di canary), se presenta la strumentazione adeguata al monitoraggio della distribuzione e se è possibile eseguire il rollback in modo sicuro. Alcuni team usano una checklist personalizzata, come nell'esempio seguente, aggiunta automaticamente

a ogni revisione del codice da parte del team per un controllo esplicito relativo ai problemi di sicurezza della distribuzione.

Esempio di una checklist per la revisione del codice

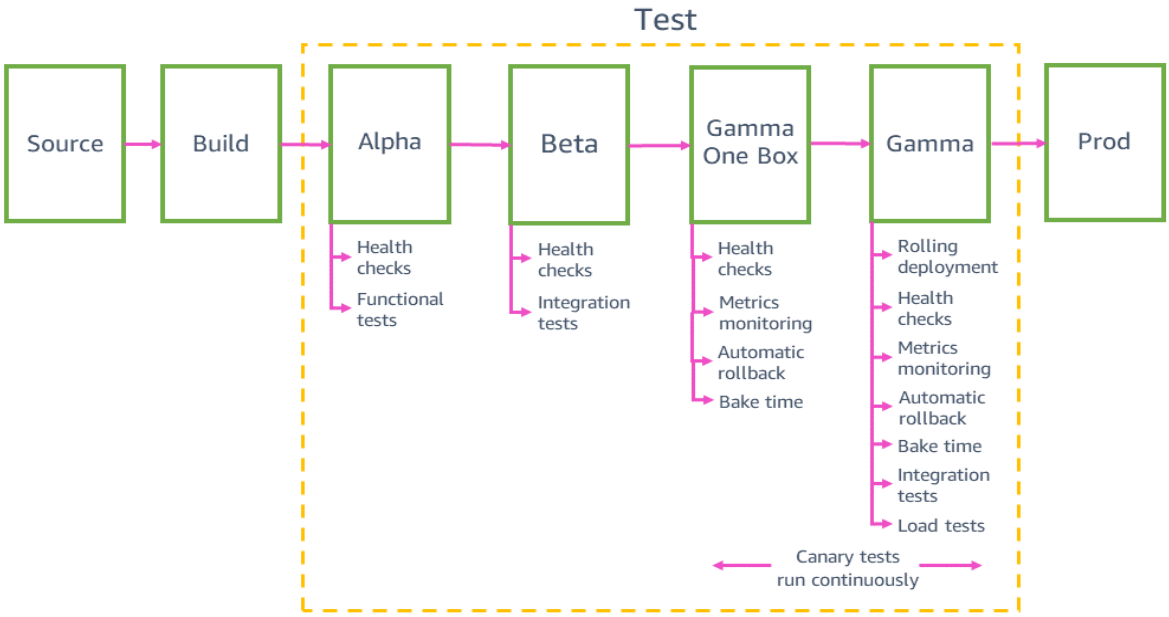
```
## Testing
[ ] Did you write new unit tests for this change?
[ ] Did you write new integration tests for this change?

Include the test commands you ran locally to test this change:
```
mvn test && mvn verify
```

## Monitoring
[ ] Will this change be covered by our existing monitoring?
 (no new canaries/metrics/dashboards/alarms are required)
[ ] Will this change have no (or positive) effect on resources and/or
limits?
 (including CPU, memory, AWS resources, calls to other services)
[ ] Can this change be deployed to Prod without triggering any alarms?

## Rollout
[ ] Can this change be merged immediately into the pipeline upon approval?
[ ] Are all dependent changes already deployed to Prod?
[ ] Can this change be rolled back without any issues after deployment to
Prod?
```

Build e unit test

Nella fase di build, il codice è compilato e dotato di unit test. Gli strumenti e la logica della build possono variare in base alla lingua e al team. Ad esempio, i team possono scegliere non solo gli strumenti di analisi statistica, i linter e i framework per gli unit test più adatti a loro, ma anche la relativa configurazione, come il code coverage minimo accettabile nel framework per l'unit test. Questi strumenti e tipi di test eseguiti cambieranno a seconda del tipo di codice distribuito dalla pipeline. Ad esempio, si usano unit test per il codice dell'applicazione e linter per l'infrastruttura come modelli di codici. Tutte le build sono eseguite senza effettuare l'accesso alla rete per isolarle e promuoverne la riproducibilità. In genere, gli unit test simulano tutte le chiamate API alle dipendenze, come altri servizi AWS. Le interazioni con dipendenze non simulate "in diretta" vengono testate nella pipeline in un secondo momento tramite test di integrazione. Rispetto a questi ultimi, gli unit test con dipendenze simulate possono esercitare casi limite, come errori imprevisti restituiti da chiamate API e assicurare una gestione degli errori ottimizzata nel codice. Una volta completata la build, il codice compilato viene convertito in pacchetto e firmato.

Distribuzioni di test in ambienti di pre-produzione

Prima della distribuzione in produzione, la pipeline distribuisce e convalida le modifiche in più ambienti di pre-produzione, ad esempio alfa, beta e gamma. Alfa e beta verificano che il codice più recente funzioni come previsto, eseguendo test dell'API funzionali e test di integrazione completi. Gamma verifica, invece, che il codice sia funzionale e che possa essere distribuito in produzione in modo sicuro. Gamma simula inoltre la produzione nel modo più fedele possibile, usando la stessa configurazione di distribuzione, gli stessi allarmi e monitoraggi, e gli stessi test di canary continui della produzione. Gamma è distribuito anche su più regioni AWS per cogliere potenziali impatti generati dalle differenze regionali.



Test di integrazione

I test di integrazione ci aiutano a usare automaticamente un servizio, proprio come i clienti, nell'ambito della pipeline. Questi test esercitano l'intero stack completo, effettuando chiamate API reali in esecuzione sull'infrastruttura reale in ogni fase di pre-produzione per tutti gli scenari cliente significativi. I test di integrazione si pongono l'obiettivo di cogliere comportamenti errati o imprevisti del servizio, prima di distribuirlo in produzione.

Mentre gli unit test sono eseguiti su dipendenze simulate, i test di integrazione vengono svolti in un sistema di pre-produzione che chiama dipendenze reali e convalida le previsioni delle simulazioni sul comportamento di queste dipendenze. I test di integrazione convalidano il comportamento di singole API con input diversi e tutti i flussi di lavoro che uniscono più API, come la creazione di una nuova risorsa, la descrizione di quest'ultima fino al relativo completamento, e l'uso della risorsa.

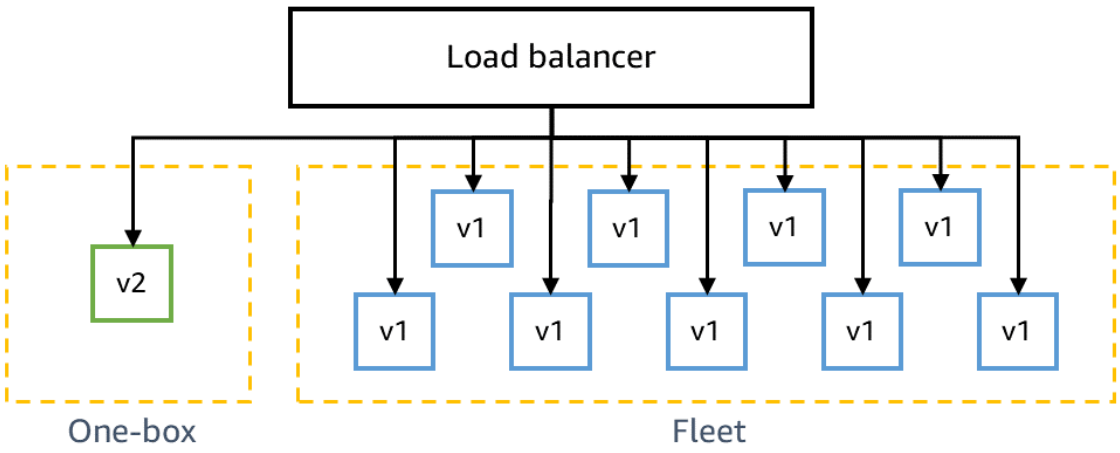
I test di integrazione eseguono casi di test positivi e negativi, come la fornitura di un input non valido a un'API e il controllo circa la restituzione di un errore "input non valido", come previsto. Alcune pipeline eseguono un test con dati casuali per generare un'ampia gamma di possibili input API

e convalidare che non causino errori interni al servizio. Alcune pipeline, inoltre, eseguono un breve test di carico nella fase di pre-produzione per accertarsi che le modifiche più recenti non causino latenza o regressioni throughput ai livelli di carico reali.

Compatibilità con le versioni precedenti e test one-box

Prima di distribuire in produzione, bisogna assicurarsi che il codice più recente sia compatibile con le versioni precedenti e possa essere distribuito in sicurezza nel codice corrente. Ad esempio, occorre rilevare se il codice più recente scrive dati in un formato che il codice corrente non elabora. La fase *one-box* in gamma distribuisce il codice più recente all'unità di distribuzione più piccola, come un'unica macchina virtuale o un singolo container, oppure a una percentuale ridotta delle chiamate alla funzione AWS Lambda. La distribuzione one-box abbandona il resto dell'ambiente gamma distribuito con il codice corrente per un determinato arco di tempo, ad es. 30 minuti o un'ora. Il traffico non deve essere indirizzato specialmente a quella distribuzione one-box. Può essere aggiunto allo stesso sistema di bilanciamento del carico o al sondaggio della stessa coda dell'ambiente gamma. Ad esempio, in un ambiente gamma di dieci container con un sistema di bilanciamento del carico, la distribuzione one-box riceve il dieci percento del traffico gamma generato dai test canary continui. La distribuzione one-box monitora i tassi di test canary con esito positivo e i parametri di servizio per rilevare impatti causati dalle distribuzioni o dalla presenza di una flotta "mista" distribuita in parallelo.

Il diagramma seguente mostra lo stato di un ambiente gamma in seguito alla distribuzione del nuovo codice in fase one-box, ma prima della distribuzione al resto della flotta gamma:



Dobbiamo anche assicurarci che il codice più recente sia compatibile con le versioni precedenti delle nostre dipendenze, ad es. se bisogna apportare una modifica tra microservizi in un ordine preciso. In genere, i microservizi in ambienti di pre-produzione chiamano l'endpoint di produzione di tutti i servizi di proprietà del team, ad es. Amazon Simple Storage Service (S3) o Amazon DynamoDB. Tuttavia, chiamano l'endpoint di pre-produzione di altri microservizi del team nella stessa fase. Ad esempio, il microservizio A di un dato team in gamma chiama il microservizio B dello stesso team in gamma, ma chiama l'endpoint di produzione di Amazon S3.

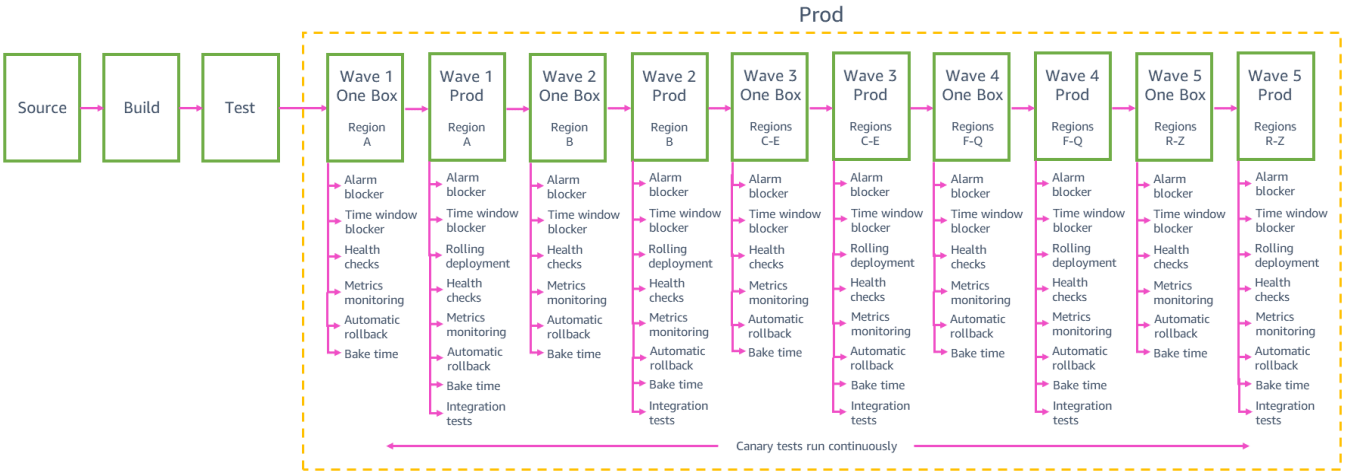
Inoltre, alcune pipeline eseguono test di integrazione in fasi di compatibilità con versioni precedenti distinte, denominate *zeta*, ossia un ambiente separato in cui ogni microservizio chiama solo gli

endpoint di produzione e testa che le modifiche in produzione siano compatibili con il codice attualmente distribuito in produzione tra più microservizi. Ad esempio, il microservizio A in zeta chiama l'endpoint di produzione del microservizio B e di Amazon S3.

Per una descrizione delle strategie di scrittura e distribuzione di modifiche compatibili con versioni precedenti, consulta l'articolo della Builders' Library [Garantire la sicurezza del rollback durante le distribuzioni](#).

Distribuzioni di produzione

In AWS, l'obiettivo #1 per le distribuzioni di produzione è evitare che causino conseguenze negative su più regioni in contemporanea e su più zone di disponibilità all'interno della stessa regione. Limitare l'ambito di ogni distribuzione limita il potenziale impatto sui clienti causato da distribuzioni di produzione con errori ed evita impatti tra più regioni o zone di disponibilità. Per limitare l'ambito di distribuzioni automatiche, suddividiamo la fase di produzione della pipeline in più fasi e distribuzioni su singole regioni. I team suddividono le distribuzioni regionali in distribuzioni con ambiti ancora più limitati, distribuendo in singole zone di disponibilità o singoli shard interni al servizio (noti come *celle*) nella pipeline in modo da limitare ulteriormente il potenziale impatto causato da una distribuzione di produzione che presenta errori.



Distribuzioni sfalsate

Ogni team deve bilanciare la sicurezza delle distribuzioni con ambiti limitati e la velocità con cui trasmettere le modifiche ai clienti in tutte le regioni. Apportare le modifiche di distribuzione in 24 regioni o 76 zone di disponibilità tramite una pipeline alla volta comporta un rischio inferiore di ampi impatti, ma potrebbe richiedere settimane prima che la pipeline sia in grado di trasmettere le modifiche ai clienti su scala globale. Secondo quanto identificato, se raggruppiamo le distribuzioni in "onde" crescenti, come visto nella pipeline di produzione di esempio precedente, otteniamo un buon equilibrio tra velocità e rischio di distribuzione. Ogni fase dell'onda nella pipeline gestisce le distribuzioni a un gruppo di regioni, promuovendo modifiche da un'onda all'altra. Le nuove modifiche possono entrare nella fase di produzione della pipeline in qualsiasi momento. Dopo che una serie di

modifiche viene promossa dalla prima fase alla seconda fase nell'onda 1, la serie di modifiche successive da gamma viene promossa nella prima fase dell'onda 1, in questo modo non si avranno grandi gruppi di modifiche in attesa della distribuzione di produzione.

Le prime due onde nella pipeline creano la massima fiducia nella modifica: la prima onda distribuisce a una regione con un basso numero di richieste per limitare il possibile impatto della prima distribuzione di produzione della nuova modifica. L'onda distribuisce a una sola zona (o cella) di disponibilità alla volta all'interno di quella regione per distribuire attentamente le modifiche nella regione. La seconda onda distribuisce quindi a una zona (o cella) di disponibilità alla volta in una regione con un elevato numero di richieste in cui è altamente probabile che i clienti eserciteranno tutti i nuovi percorsi di codice e dove otteniamo una buona verifica delle modifiche.

Successivamente abbiamo una maggiore fiducia nella sicurezza della modifica dalle distribuzioni delle onde della pipeline iniziali, possiamo distribuire in sempre più regioni in parallelo nella stessa onda. Ad esempio, la precedente pipeline di produzione di esempio distribuisce a tre regioni nell'onda 3, poi fino a 12 regioni nell'onda 4, quindi alle rimanenti regioni nell'onda 5. Il numero esatto e la scelta delle regioni in ciascuna di queste onde e il numero di onde in una pipeline del team del servizio dipende dai singoli modelli d'uso del servizio e dalla scalabilità. Le onde successive nella pipeline ci aiutano comunque a raggiungere il nostro obiettivo di evitare l'impatto negativo su più zone di disponibilità nella stessa regione. Quando un'onda distribuisce a più regioni in parallelo, segue lo stesso attento comportamento di implementazione per ciascuna regione usato nelle onde iniziali. Ciascuna fase nell'onda distribuisce solo a una singola zona o cella di disponibilità da ciascuna regione nell'onda.

Distribuzioni one-box e in sequenza

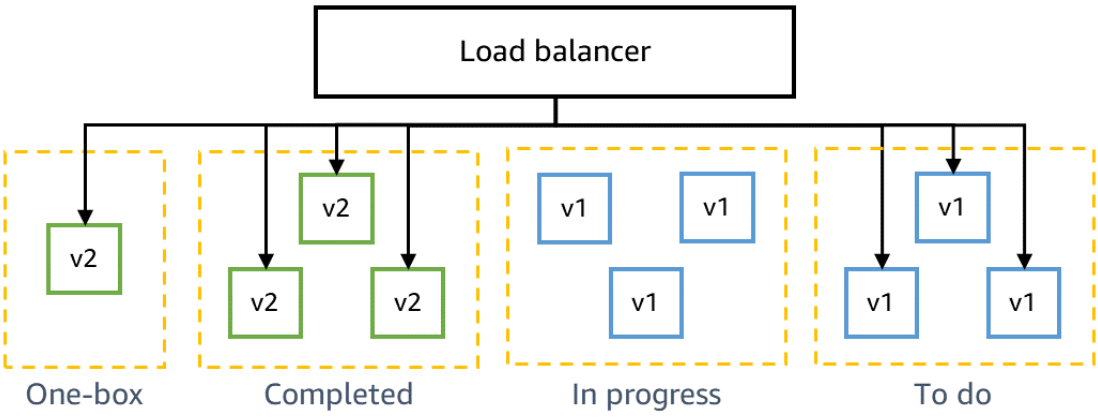
Le distribuzioni a ciascuna onda di produzione iniziano con una fase one-box. Come nella fase one-box gamma, ciascuna fase one-box di produzione distribuisce l'ultimo codice a una *scatola* (una singola macchina virtuale, un singolo container o una piccola percentuale di chiamate della funzione Lambda) in ciascuna delle regioni o zone di disponibilità dell'onda. La distribuzione one-box di produzione riduce al minimo il potenziale impatto delle modifiche sull'onda limitando inizialmente le richieste elaborate dal nuovo codice in quell'onda. Normalmente, una scatola gestisce al massimo il dieci percento di tutte le richieste per la regione o zona di disponibilità. Se la modifica causa un impatto negativo nell'unica scatola, la pipeline ripristina automaticamente la modifica e non la promuove alle restanti fasi di produzione.

Dopo la fase one-box, la maggior parte dei team usa le distribuzioni in sequenza per distribuire la flotta di produzione principale dell'onda. Una distribuzione in sequenza assicura che il servizio abbia abbastanza capacità per elaborare il carico di produzione durante l'intera distribuzione. Controlla la velocità a cui il nuovo codice viene *messo in funzione* (ovvero, quando inizia a elaborare il traffico di produzione) per limitare l'impatto delle modifiche. In una distribuzione in sequenza tipica a una regione, al massimo il 33 percento delle scatole del servizio in quella regione (container, chiamate lambda o software in funzione su macchine virtuali) è sostituito dal nuovo codice.

Durante una distribuzione, il sistema di distribuzione sceglie prima un batch iniziale di fino al 33 percento di scatole da sostituire con il nuovo codice. Durante la sostituzione, almeno il 66 percento della capacità complessiva è in buone condizioni ed elabora la richiesta. Tutti i servizi sono ridimensionati in modo da resistere alla perdita di una zona di disponibilità nella regione, quindi

sappiamo che il servizio può ancora elaborare il carico di produzione a questa capacità. Dopo che il sistema di distribuzione determina che una scatola nel batch iniziale di scatole supera i controlli dello stato, una scatola dalla flotta rimanente può essere sostituita con il nuovo codice e così via. Nel frattempo, manteniamo comunque almeno il 66 percento della capacità per elaborare le richieste in qualsiasi momento. Per limitare ulteriormente l'impatto delle modifiche, alcune pipeline dei team distribuiscono appena il cinque percento delle scatole alla volta. Tuttavia, successivamente eseguono *rollback rapidi*, in cui il sistema sostituisce il 33 percento delle scatole alla volta con il codice precedente per velocizzare il rollback.

Il diagramma seguente mostra lo stato di un ambiente di produzione a metà di una distribuzione in sequenza. Il nuovo codice è stato distribuito alla fase one-box e al primo batch della flotta di produzione principale. Un altro batch è stato rimosso dal sistema di bilanciamento del carico ed è stato disattivato per la sostituzione.



Monitoraggio dei parametri e rollback automatico

Tipicamente le distribuzioni automatiche nella pipeline non anno uno sviluppatore che guarda attivamente ciascuna distribuzione di produzione, controlla i parametri ed esegue manualmente il ripristino in caso di problemi. Queste distribuzioni sono completamente pratiche. Il sistema di distribuzione monitora attivamente un allarme per determinare se occorre ripristinare automaticamente una distribuzione. Un rollback riporterà l'ambiente all'immagine di container, al pacchetto di distribuzione della funzione AWS Lambda o al pacchetto di distribuzione interno che era stato precedentemente distribuito. I nostri pacchetti di distribuzione interni sono simili alle immagini di container, poiché i pacchetti sono immutabili e usano una checksum per verificare la loro integrità.

Ciascun microservizio in ciascuna regione normalmente ha allarmi di gravità elevata che si attivano al raggiungimento delle soglie dei parametri che influenzano i clienti del servizio (come i tassi di guasto e la latenza elevata) e dei parametri di integrità del sistema (come l'uso della CPU) come illustrato nell'esempio seguente. Questo allarme di gravità elevata viene usato per chiamare il tecnico reperibile e ripristinare automaticamente il servizio se è in corso una distribuzione. Spesso, il rollback è già in corso nel momento in cui il tecnico reperibile è stato chiamato e inizia a intervenire.

Esempio di allarme di microservizio di gravità elevata

```
ALARM("FrontEndApiService_High_Fault_Rate") OR
ALARM("FrontEndApiService_High_P50_Latency") OR
ALARM("FrontEndApiService_High_P90_Latency") OR
ALARM("FrontEndApiService_High_P99_Latency") OR
ALARM("FrontEndApiService_High_Cpu_Usage") OR
ALARM("FrontEndApiService_High_Memory_Usage") OR
ALARM("FrontEndApiService_High_Disk_Usage") OR
ALARM("FrontEndApiService_High_Errors_In_Logs") OR
ALARM("FrontEndApiService_High_Failing_Health_Checks")
```

Le modifiche introdotte da una distribuzione possono avere un impatto sui microservizi a monte e a valle, pertanto il sistema di distribuzione deve monitorare l'allarme di gravità elevata per il microservizio sottoposto a distribuzione *e* monitorare gli allarmi di gravità elevata per gli altri microservizi del team per determinare quando eseguire il ripristino. Le modifiche distribuite possono anche influenzare i parametri dei test di canary continui, quindi il sistema di distribuzione deve inoltre monitorare il mancato superamento dei test di canary. Per ripristinare automaticamente tutte queste possibili aree di impatto, i team creano allarmi aggregati di gravità elevata per i sistemi di distribuzione da monitorare. Gli allarmi aggregati di gravità elevata raggruppano lo stato di tutti gli allarmi di gravità elevata del singolo microservizio del team e lo stato degli allarmi canary in un unico stato aggregato, come nell'esempio seguente. Se uno qualsiasi degli allarmi di gravità elevata per i microservizi del team entrano nello stato di allarme, tutte le distribuzioni in corso del team su tutti i loro microservizi in quella regione vengono automaticamente ripristinate.

Esempio di allarme di rollback dell'aggregato di gravità elevata

```
ALARM("FrontEndApiService_High_Severity") OR
ALARM("BackendApiService_High_Severity") OR
ALARM("BackendWorkflows_High_Severity") OR
ALARM("Canaries_High_Severity")
```

Una fase one-box elabora una piccola percentuale del traffico totale, quindi i problemi introdotti da una distribuzione one-box possono non attivare l'allarme di rollback dell'aggregato di gravità elevata del servizio. Per trovare e ripristinare le modifiche che causano problemi nella fase one-box prima che raggiungano il resto delle fasi di produzione, le fasi one-box ripristinano inoltre i parametri riservati solo a una scatola. Ad esempio, ripristinano il tasso di guasto delle richieste elaborate specificamente dalla scatola, che costituisce una piccola percentuale del numero totale di richieste.

Esempio di allarme di rollback one-box

```
ALARM("High_Severity_Aggregate_Rollback_Alarm") OR
ALARM("FrontEndApiService_OneBox_High_Fault_Rate") OR
```

```
ALARM("FrontEndApiService_OneBox_High_P50_Latency") OR
ALARM("FrontEndApiService_OneBox_High_P90_Latency") OR
ALARM("FrontEndApiService_OneBox_High_P99_Latency") OR
ALARM("FrontEndApiService_OneBox_High_Cpu_Usage") OR
ALARM("FrontEndApiService_OneBox_High_Memory_Usage") OR
ALARM("FrontEndApiService_OneBox_High_Disk_Usage") OR
ALARM("FrontEndApiService_OneBox_High_Errors_In_Logs") OR
ALARM("FrontEndApiService_OneBox_Failing_Health_Checks")
```

Oltre a ripristinare gli allarmi definiti dal team del servizio, il nostro sistema di distribuzione può anche rilevare e ripristinare automaticamente anomalie nei parametri comuni emessi dal nostro framework di servizi Web interno. La maggior parte dei nostri microservizi emette parametri come il numero di richieste, la latenza delle richieste e il numero di guasti in un formato standard. Usando questi parametri standard, il sistema di distribuzione può eseguire automaticamente il ripristino in presenza di anomalie nei parametri durante una distribuzione. Alcuni esempi sono se il numero di richieste scende improvvisamente a zero, o se la latenza o il numero di guasti diventa più alto del normale.

Tempo di messa a punto

A volte un impatto negativo causato da una distribuzione non è subito evidente. Si *sviluppa lentamente*. Ovvero, non compare immediatamente durante la distribuzione, specialmente se il servizio è a basso carico al momento. Promuovere la modifica alla fase di pipeline successiva immediatamente dopo il completamento della distribuzione può avere un impatto in più regioni nel momento in cui l'impatto emerge nella prima regione. Prima di promuovere una modifica alla fase di produzione successiva, ciascuna fase di produzione nella pipeline ha *un tempo di messa a punto*, ovvero quando la pipeline continua a monitorare l'allarme dell'aggregato di gravità elevata per un qualsiasi impatto a lento sviluppo dopo il completamento di una distribuzione e prima di passare alla fase successiva.

Per calcolare la quantità di tempo impiegato nella messa a punto di una distribuzione, occorre bilanciare il rischio di causare un impatto maggiore se le modifiche vengono promosse a più regioni troppo velocemente rispetto alla velocità a cui le modifiche possono essere distribuite ai clienti nel mondo. Abbiamo scoperto che un buon modo per bilanciare questi rischi è, per le onde precedenti nella pipeline, avere un tempo di messa a punto più lungo mentre creiamo fiducia nella sicurezza della modifiche, e avere un tempo di messa a punto più breve per le onde successive. Il nostro obiettivo è quello di ridurre al minimo il rischio di un impatto che interessi più regioni. Poiché la maggior parte delle distribuzioni non è monitorata attivamente da un membro del team, i tempi di messa a punto standard tipici della pipeline sono contenuti e distribuiranno una modifica a tutte le regioni in circa quattro o cinque giorni lavorativi. I servizi che sono più ampi o altamente critici hanno tempi di messa a punto e tempi affinché le pipeline distribuiscano una modifica a livello globale ancora più contenuti.

Una pipeline tipica attente almeno un'ora dopo ciascuna fase one-box, almeno 12 ore dopo la prima onda regionale e almeno da due a quattro ore dopo ciascuna delle restanti onde regionali, con tempi di messa a punto aggiuntivi per le singole regioni, zone di disponibilità e celle all'interno di ciascuna onda. Il tempo di messa a punto include i requisiti di attente uno specifico numero di punti di dati nei parametri del team (ad esempio, "attendi almeno 100 richieste a Crea API") per garantire che siano

state effettuate richieste sufficienti a rendere probabile il pieno esercizio del nuovo codice. Durante l'intero tempo di messa a punto, la distribuzione viene ripristinata automaticamente se l'allarme dell'aggregato di gravità elevata entra in stato di allarme.

Sebbene sia estremamente raro, in alcuni casi può essere necessario erogare una modifica urgente (come una correzione di sicurezza o una mitigazione per un evento su larga scala che compromette la disponibilità del servizio) ai clienti più velocemente rispetto al tempo normalmente necessario alla pipeline per mettere a punto le modifiche e distribuirle. In questi casi è possibile ridurre il tempo di messa a punto della pipeline per accelerare la distribuzione, ma per farlo occorre un livello elevato di controllo della modifica. Per questi casi è necessario il controllo dei tecnici principali dell'organizzazione. Il team deve riesaminare le modifiche apportate al codice, nonché la loro urgenza e il rischio di impatto, con sviluppatori estremamente esperti in materia di sicurezza operativa. La modifica procede ancora attraverso le stesse fasi nella pipeline come sempre, ma viene promossa alla fase successiva più rapidamente. Gestiamo il rischio di una distribuzione più rapida limitando le modifiche in movimento nella pipeline durante questo tempo per consentire solo alle modifiche di codice più piccole necessarie per affrontare il problema corrente e monitorando attivamente le distribuzioni.

Blocchi delle finestre di allarme e tempo

La pipeline impedisce distribuzioni automatiche in produzione in presenza di un rischio maggiore di causare un impatto negativo. La pipeline usa una serie di "blocchi" per valutare il rischio della distribuzione. Ad esempio, distribuire automaticamente una nuova modifica in produzione quando è attualmente in corso un problema nell'ambiente può peggiorare o prolungare l'impatto. Prima di iniziare una nuova distribuzione in qualsiasi fase di produzione, la pipeline controlla l'allarme dell'aggregato di gravità elevata del team per stabilire se sono presenti eventuali problemi attivi. Se l'allarme è attualmente in stato di allarme, la pipeline impedisce l'avanzamento della modifica. Le pipeline possono anche controllare gli allarmi a livello di organizzazione, come un allarme di evento su larga scala che indica se è presente un impatto ampio in altri sistemi del team, e impedisce l'avvio di una nuova distribuzione che possa aumentare l'impatto generale. Questi blocchi della distribuzione possono essere disabilitati dagli sviluppatori quando occorre distribuire una modifica in produzione per eseguire il ripristino da un problema di gravità elevata.

La pipeline è inoltre configurata con una serie di finestre temporali che definiscono quando può iniziare una distribuzione. Quando si configurano le finestre temporali, occorre bilanciare due cause del rischio di distribuzione. Da un lato, finestre temporali molto piccole possono causare l'accumulo di modifiche nella pipeline mentre la finestra temporale è chiusa, aumentando la probabilità che una qualsiasi di queste modifiche nella distribuzione successiva abbia un impatto quando la finestra temporale si apre. Dall'altro, finestre temporali molto grandi che vanno oltre le normali ore lavorative aumentano il rischio di prolungare l'impatto derivante da una distribuzione fallita. Durante le ore non lavorative, occorre più tempo per contattare il tecnico reperibile durante la giornata, quando il tecnico reperibile e gli altri membri del team stanno lavorando. Durante le normali ore lavorative, il team può essere contattato più rapidamente a seguito di una distribuzione fallita se sono necessarie fasi di ripristino manuale.

La maggior parte delle distribuzioni non è monitorata attivamente da un membro del team, quindi ottimizziamo la temporizzazione delle distribuzioni per ridurre al minimo il tempo necessario per contattare un tecnico reperibile qualora fosse necessario il rollback dopo un ripristino automatico. Normalmente occorre più tempo per contattare i tecnici reperibili di notte, durante le ferie

e nei weekend, pertanto questi tempi sono esclusi dalle finestre temporali. A seconda dei modelli d'uso del servizio, alcuni problemi possono non presentarsi per ore dopo la distribuzione, pertanto molti team escludono anche le distribuzioni di venerdì e nel tardo pomeriggio dalle proprie finestre temporali per ridurre il rischio di dover contattare il tecnico reperibile di notte o durante il weekend dopo la distribuzione. Abbiamo scoperto che questa serie di finestre temporali consente un ripristino rapido anche quando è necessaria un'azione manuale, garantisce un coinvolgimento minore dei tecnici reperibili al di fuori del normale orario di lavoro e fa in modo che venga raggruppato un piccolo numero di modifiche mentre le finestre temporali sono chiuse.

Pipeline come codice

Il team del servizio AWS tipico possiede molte pipeline per distribuire più microservizi e tipi di origine del team (codici dell'applicazione, codici dell'infrastruttura, patch OS, ecc.). Ciascuna pipeline ha molte fasi di distribuzione per un numero sempre maggiore di regioni e zone di disponibilità. Questo si traduce in un'importante configurazione da gestire per il team nel sistema della pipeline, nel sistema di distribuzione e nel sistema di allarme e un grande impegno per rimanere aggiornati sulle best practice e sulle nuove regioni e zone di disponibilità. Negli ultimi anni, abbiamo abbracciato la pratica delle "pipeline come codice" come un modo per configurare in modo più semplice e coerente pipeline sicure e aggiornate modellando questa configurazione nel codice. Il nostro strumento di pipeline come codice interno esegue l'estrazione da una lista centralizzata di regioni e zone di disponibilità per aggiungere facilmente nuove regioni e zone di disponibilità alle pipeline in AWS. Lo strumento consente anche ai team di modellare le pipeline usando l'ereditarietà, definendo una configurazione che è comune per le pipeline del team in una classe genitore (come quali regioni vanno in ciascuna onda e quanto deve essere lungo il tempo di messa a punto per ciascuna onda) e definendo tutta la configurazione della pipeline del microservizio come una sottoclasse che eredita tutta la configurazione comune.

Conclusione

In Amazon, abbiamo creato le nostre pratiche di distribuzione automatizzate nel tempo in base a cosa ci aiuta a bilanciare la sicurezza della distribuzione rispetto alla velocità di distribuzione. Allo stesso tempo, vogliamo ridurre al minimo la quantità di tempo necessaria agli sviluppatori per preoccuparsi della distribuzione. Creare la sicurezza della distribuzione automatizzata nel processo di rilascio ricorrendo a numerosi test pre-produzione, rollback automatici e distribuzioni di produzione sfalsate ci consente di ridurre al minimo il potenziale impatto sulla produzione causato dalle distribuzioni. Questo significa che gli sviluppatori non devono monitorare attivamente le distribuzioni di produzione.

Grazie alle pipeline interamente automatizzate, gli sviluppatori usano valutazioni del codice per controllare il loro codice e anche per approvare il passaggio della modifica in produzione. Una volta unita la modifica nel repository del codice sorgente, lo sviluppatore può passare all'attività successiva e dimenticarsi della distribuzione, confidando che la pipeline arrivi in produzione in modo sicuro e accurato. La pipeline automatizzata si occupa della distribuzione continua di produzione più volte al giorno, bilanciando sicurezza e velocità. Modellando la nostra pratica di distribuzione continua nel codice, è ancora più semplice per i team dei servizi AWS configurare le pipeline per distribuire automaticamente e in modo sicuro le modifiche al codice.