GITLAB VS AWS CODEPIPELINE: UN MATCH ALL'ULTIMO COMMIT!

AWS CloudFormation | AWS CodeBuild | AWS CodeCommit | AWS CodeDeploy | AWS CodePipeline | CI/CD

beSharp | 2 Ottobre 2020

GitLab è diventato uno tra gli strumenti più utilizzati dai Devops. Include tante funzionalità in un solo servizio ed è possibile installarlo on-premise o utilizzarlo in versione SaaS.

Il piano gratuito è sufficiente per alcuni dei più comuni casi d'uso come ad esempio mantenere una codebase, eseguire pipeline e gestire le informazioni sui progetti. Anche Amazon Web Services permette di implementare sistemi di **CI/CD** con un modello di pagamento a consumo attraverso l'uso dei servizi **AWS CodeCommit, AWS CodeBuild, AWS CodePipeline**.

Selezionare il miglior strumento da utilizzare durante la progettazione di un nuovo business o per migrare applicazioni legacy in Cloud non è mai banale. Ma non abbiate paura, cari sviluppatori: questo articolo vi aiuterà a fare chiarezza! I due servizi stanno per dare vita ad un match alla pari su un terreno di gioco comune che permetta un test ad armi pari: stiamo parlando dell'**AWS Well-Architected Framework**, il framework di AWS pensato per la progettazione di applicazioni ed architetture che siano manutenibili, sicure, resilienti, efficienti e ottimizzate dal punto di visto dei costi.

È basato su 5 pilastri fondamentali – Eccellenza Operativa, Sicurezza, Affidabilità, Efficienza delle Prestazioni, Ottimizzazione dei Costi – e non riguarda nessun servizio AWS in particolare, così da poter essere usato come riferimento per implementare qualsiasi servizio o infrastruttura.

Accogliamo quindi i contendenti di oggi: da un lato Gitlab, dall'altro si sta scaldando AWS CodePipeline!

Le regole del gioco

Iniziamo un ipotetico match all'ultimo commit fra GitLab e AWS CodePipeline.

Ogni pilastro dell'AWS Well Architected Framework sarà usato come round; i punteggi saranno basati sui principi di progettazione di ogni pilastro.

Round 1: Eccellenza operativa

Per questo round, i punti saranno definiti su questi principi:

- Eseguire le operazioni come codice;
- Effettuare cambiamenti frequenti, piccoli e reversibili;
- Raffinare le procedure operative frequentemente;
- Anticipare il fallimento;
- Imparare dai fallimenti.

GitLab e CodePipeline permettono entrambi di utilizzare template yaml per definire delle pipeline per eseguire compiti come compilare progetti e distribuirli. E' possibile definire fasi di esecuzione e differenti passi per ogni fase.

Useremo un progetto di esempio.

Partiamo con GitLab:

```
image: python:latest

variables:
  PIP_CACHE_DIR: "$CI_PROJECT_DIR/.cache/pip"

test:
  script:
        - python setup.py test
        - pip install tox flake8  # you can also use tox
        - tox -e py36,flake8

run:
  script:
    - python setup.py bdist_wheel
    - pip install dist/*
  artifacts:
    paths:
      - dist/*.whl
```

Passiamo ora a AWS CodePipeline

```
version: 0.2
env:
        variables: PIP_CACHE_DIR: "$CI_PROJECT_DIR/.cache/pip"
phases:
        test:
          - python setup.py test
          - pip install tox flake8  # you can also use tox
          - tox -e py36,flake8
```

```
        run:
          - python setup.py bdist_wheel
          - pip install dist/*

  artifacts:
        files:
          - dist/*.whl
```

Gitlab permette di definire l'immagine da usare per la build, mentre per CodePipeline l'immagine è definita nella configurazione della pipeline. La sintassi è molto chiara per entrambi i servizi e la flessibilità che deriva dal loro utilizzo è notevole.

Il punteggio è di 1 – 1.

Andando più nel dettaglio, CodePipeline consente anche di distribuire stack sfruttando CloudFormation e di fare quindi **Continuous Delivery anche su intere infrastrutture**.

Con questo colpo basso, CodePipeline si aggiudica un punto ulteriore e passa al comando: **il punteggio è di 2 – 1 per CodePipeline.**

Round 2: Sicurezza

Per questo round, i punti saranno definiti su questi principi:

- Permettere la tracciabilità;
- Applicare la sicurezza a tutti i livelli;
- Automatizzare le migliori pratiche di sicurezza;
- Proteggere i dati (memorizzati ed in transito);
- Limitare l'accesso ai dati solo alle persone autorizzate;
- Prepararsi agli eventi di sicurezza.

I meccanismi di autenticazione offerti da GitLab includo la possibilità di utilizzare altri IdP federati come Active Directory e SAML e, pianificando con cura la configurazione, potremo ottenere una gestione centralizzata degli utenti. Se volessimo, però, utilizzare SAML SSO per gruppi sarebbe necessario passare ad uno dei piani a pagamento.

AWS CodePipeline usa lo stesso strato di autenticazione ed autorizzazione di AWS, con integrazione con Active Directory, SAML… e anche di SAML SSO per i gruppi. AWS passa in leggero vantaggio visto anche il numero minore di meccanismi di autenticazione offerti gratuitamente da GitLab.

Anche per quanto riguarda la tracciabilità e limitazione dell'accesso ai dati AWS, GitLab accusa il colpo di un piano gratuito piuttosto limitato: il log di audit è incluso solamente nei piani a pagamento, mentre CloudTrail è incluso e configurato per ogni servizio AWS in ogni nostro account.

AWS incrementa il suo vantaggio: **3 – 1 per CodePipeline.**

Il colpo più duro a GitLab arriva però da alcune discussioni aperte e segnalazioni sulla sicurezza dei "runner" di GitLab come questa e sulla cifratura dei dati memorizzati come questa (cache e artifacts).

Il runner manager di GitLab ha bisogno di ruoli e permessi corretti per interagire con i servizi AWS e, ad esempio, distribuire le applicazioni. Se il numero di applicazioni cresce, diventa necessario assegnare permessi aggiuntivi al runner o aggiungerne altri, dando loro il minimo permesso possibile. In questo modo però il numero di risorse coinvolte cresce, facendo crescere anche complessità di gestione e costi.

Ogni servizio AWS ha la possibilità di cifrare i dati memorizzati ed in transito. In più, i ruoli IAM permettono di evitare l'utilizzo di token, access key e altri meccanismi di autenticazione vulnerabili.

CodePipeline prosegue la sua fuga: è un 4 – 1.

Round3: Affidabilità

Per questo round, i punti saranno definiti su questi principi:

- Ripristino automatico dopo un fallimento;
- Test delle procedure di ripristino;
- Aggiunta di capacità computazionale scalando orizzontalmente;
- Smettere di prevedere/indovinare le necessità sulle risorse future;
- Implementare i cambiamenti mediante automazione.

È giusto specificare che per questo specifico round, le singole necessità influenzano molto l'implementazione degli ambienti di build per tutti e due i servizi.

Sia GitLab che CodePipeline possono scalare orizzontalmente, automatizzare i cambiamenti e non richiedono nessuna pianificazione avanzata di uso delle risorse (a meno che non stiate usando un solo grosso runner on-premise per GitLab).

Entrambi guadagnano un prezioso punto: **siamo a 5 – 2.**

GitLab implementa un sistema a plugin ed è possibile usare executors personalizzati che aggiungono scalabilità e flessibilità: è sicuramente un vantaggio… e un punto per il servizio che accorcia le distanze : **5 – 3.**

AWS CodePipeline può però essere distribuito fra più availability zone, quindi in caso di fallimento in eu-west-1-a le build possono ancora funzionare nelle rimanenti 2 AZ.

Nel plugin che implementa l'autoscaling su EC2 di GitLab la ridondanza su più AZ non è invece possibile. Ecco un esempio di configurazione.

```
[runners.machine]
    IdleCount = 1
    IdleTime = 1800
    MaxBuilds = 10
    MachineDriver = "amazonec2"
    MachineName = "gitlab-docker-machine-%s"
    MachineOptions = [
      "amazonec2-access-key=XXXX",
      "amazonec2-secret-key=XXXX",
      "amazonec2-region=us-central-1",
      "amazonec2-vpc-id=vpc-xxxxx",
      "amazonec2-subnet-id=subnet-xxxxx",
      "amazonec2-zone=x",
      "amazonec2-use-private-address=true",
      "amazonec2-tags=runner-manager-name,gitlab-aws-autoscaler,gitlab,true,gitlab-run
ner-autoscale,true",
      "amazonec2-security-group=xxxxx",
      "amazonec2-instance-type=m4.2xlarge",
    ]
```

È possibile definire solamente una singola subnet ed una sola Availability Zone. Con questa configurazione la build fallirà se la zona scelta fallisce. **AWS conquista il punto, siamo a 6 – 3.**

Per GitLab è comunque possibile scrivere un executor personalizzato che permetta di aumentare l'affidabilità, ma svilupparlo richiede una considerevole quantità di tempo.

Continuiamo con gli ultimi due round.

Round 4: Efficienza Operativa.

Per questo round i punti saranno definiti su questi principi:

- Diffondere l'accesso alle tecnologie avanzate;
- Distribuire servizi globali nell'arco di minuti;
- Utilizzare architetture serverless;
- Fare esperimenti frequentemente;
- Conoscere e tenere in considerazione l'infrastruttura sottostante (qualche specifica qui).

Anche se GitLab e CodePipeline hanno molto in comune, GitLab è più flessibile: supporta molti tipi di installazione e di configurazione e questo vale sicuramente un punto extra.

Aggiorniamo il punteggio a 6 – 4.

Offre però un supporto più limitato in ambito serverless: non è possibile, ad esempio, utilizzare Docker in Docker sull'executor personalizzato GitLab Fargate, non essendo possibile utilizzare il socket Docker. Dal canto suo, CodePipeline offre supporto completo alle build serverless, anche per le build Docker in Docker.

Sembra che su questo argomento ci sia un pareggio! Entrambi i servizi guadagnano un punto: **7 – 4 per CodePipeline.**

Round 5: Ottimizzazione dei Costi

Per questo round i punti saranno definiti su questi principi:

- Implementare un processo di gestione delle finanze cloud;
- Utilizzare servizi con un modello di costi a consumo;
- Misurare l'efficienza totale;
- Smettere di investire in attività automatizzabili, ripetitive e di basso valore (https://nodramadevops.com/2019/02/identifying-undifferentiated-heavy-lifting/)
- Analizzare ed attribuire i costi.

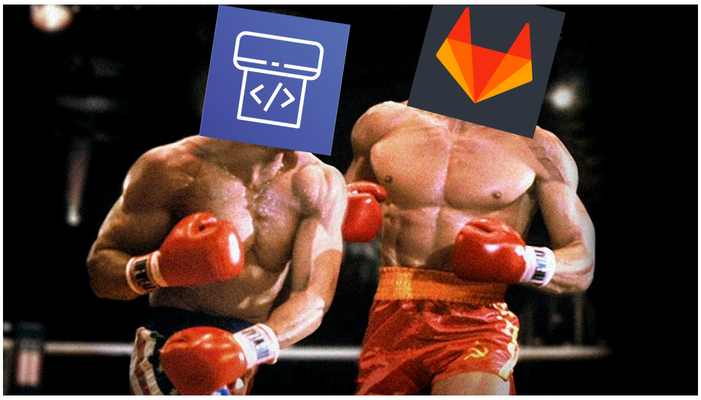
Analizzando i modelli di pricing dei due servizi, vediamo che mentre i piani a pagamento di GitLab prevedono un costo per singolo utente, le utenze AWS sono gratuite e il servizio CodeBuild prevede un pagamento in base al numero di minuti utilizzati. In concreto, utilizzando una istanza general1.small (2 core e 3gb di RAM) la spesa su AWS sarà di 5 dollari per 1.000 minuti di esecuzione.

Con i runner condivisi di GitLab, invece, sarà possibile acquistare 1.000 minuti per 10 dollari (i primi 2.000 minuti sono gratuiti).

È anche possibile utilizzare l'autoscaling AWS per le build se non si vogliono utilizzare i runner shared di GitLab, ma in questo caso saranno necessarie istanze EC2 per il runner manager ed altre per permettere alle build di scalare. Un'altra possibilità è l'acquisto di istanze reserved o spot, se il tipo di business lo permette.

Sull'ottimizzazione dei costi non abbiamo dubbi: AWS si accaparra l'ultimo match offrendo soluzioni pronte all'uso e "chiavi in mano".

Podio e conclusioni



Il nostro incontro amichevole fra i 2 servizi più utilizzati è terminato. Con un punteggio di **8 –** 4, accogliamo sul podio **il vincitore, AWS CodePipeline**!

Ma sarà in grado di mantenere il primato?

GitLab resta un ottimo servizio, dalle funzionalità eccellenti per i team ed dalla grande usabilità per gli sviluppatori e, per alcune soluzioni, potrebbe comunque essere la scelta ottimale.

In alcuni altri casi ancora, la strada migliore da intraprendere potrebbe addirittura essere utilizzare le funzionalità di entrambi i servizi, ma questo potrebbe essere uno spoiler uno dei nostri prossimi articoli..

Tenetevi pronti per il prossimo match, sviluppate consapevolmente e ci vediamo tra 14 giorni su **#prou2beCloud**!

beSharp

Dal 2011 beSharp guida le aziende italiane sul Cloud. Dalla piccola impresa alla grande multinazionale, dal manifatturiero al terziario avanzato, aiutiamo le realtà più all’avanguardia a realizzare progetti innovativi in campo IT.

Get in touch

beSharp.it
proud2becloud@besharp.it