GITLAB VS AWS CODEPIPELINE: THE ULTIMATE BATTLE ROYAL!

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

beSharp | 2 October 2020

GitLab has become a widely used DevOps tool as it packs a lot of features into a single service. It can be deployed on-premise, or you can use the SaaS version.

The free tier is enough for common tasks like maintaining a codebase, running pipelines and managing project information. Amazon Web Services offers a suite of services (**AWS CodeCommit, AWS CodeBuild, AWS CodePipeline**) to implement **CI/CD** best practices with a pay-as-you-go pricing model.

It's never easy to choose the perfect set of tools to use when planning a new business or to migrate legacy applications to the cloud. Fear not, dear developers: this article is going to clear things out through an ultimate Royal Battle! Since we're talking about cloud environments, we need to find a common playfield to start the fight: for an equal challenge, we choose the **AWS Well-Architected Framework**, the framework designed by AWS to help you design maintainable, secure, resilient, efficient, and cost-effective applications and architectures

Based on 5 pillars – Operational Excellence, Security, Reliability, Performance Efficiency, and Cost Optimization – it does not involve any AWS service in particular, so it can be used as a design reference to build any service or infrastructure.

Let's welcome today's fighters: on one side GitLab. On the other side, AWS CodePipeline is warming up!

Rules of the game

Let's begin an hypothetical match between GitLab and AWS Codepipeline.

Each pillar will be used as a round; scores will be based on the design principles of the pillar.

Round 1: Operational Excellence

For this game, points will be defined based on the following principles:

- Perform operations as code
- Make frequent, small, reversible changes
- Refine operations procedures frequently
- Anticipate failure
- Learn from all operational failures

GitLab and CodePipeline can define pipelines for various tasks, build projects and deploy them using a yaml template. It is possible to define different stages and execution steps for each stage.

We'll use a sample

GitLab at serve:

```
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
```

And now CodePipeline turn

```
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 allows you to define the image used for building, while CodePipeline allows you to define the image in pipeline definition, not in build steps. The syntax is quite similar and very clear in both cases and the deriving flexibility is considerable.

It's 1 – 1.

Anyway, CodePipeline allows you to deploy entire infrastructure stacks using CloudFormation, too. This means: Continuous Delivery for entire infrastructures!

CodePipeline takes the lead. **It's 2 – 1!**

Round 2: Security

For this game, points will be defined based on the following principles:

- Enable traceability
- Apply security at all layers
- Automate security best practices
- Protect data in transit and at rest
- Keep not authorized people away from data
- Prepare for security events

GitLab offers authentication mechanisms that can leverage other IdP and federations like Active Directory and SAML and you can take advantage of centralized user management if you plan your configuration carefully. Anyway, if you also need SAML SSO for groups, you have to opt for the paid tier.

AWS CodePipeline uses the same authentication and authorization layer of AWS: it features Active Directory, SAML, and… SAML SSO for groups. Given that not every authentication feature is available on GitLab's free tier, AWS takes the lead.

Let's speak about traceability and keeping people away from data: the audit log is available only on GitLab paid plans while CloudTrail is included and configured for every AWS service in your account.

AWS scores: **it's 3 – 1.**

The hardest part to deal with for GitLab comes with open issues and discussions on GitLab about runner security and about data encryption at rest (cache and artifacts).

If you need to interact with AWS services to deploy your application, you'll need to give the runner-manager a role with the right permissions. If you need to deploy multiple applications that run on different services, you'll need to provide additional permissions to your runner or to spawn multiple runners, giving them the minimum access required (note that, by doing so, the number of the resources will increase, along with costs)

This round is won by AWS services, as every service provides encryption at rest and in transit for data. Also, IAM roles allow you to avoid using tokens, access keys, and other vulnerable authentication mechanisms for services.

CodePipeline scores again and it's 4 – 1.

Round 3: Reliability

For this game, points will be defined based on the following principles:

- Automatically recover from failure
- Test recovery procedures
- Scale horizontally to increase aggregate workload availability
- Stop guessing capacity
- Automatic management of changes

Before diving into this round, it is necessary to keep in mind that in this case, a lot depends on your expectations when implementing your build environment in both services.

Both GitLab and AWS CodePipeline are characterized by horizontal scalability, they both automate changes, and don't require any intervention for advanced capacity planning (except for using a single big on-premise runner for GitLab).

They both score: it's a 5 – 2.

In terms of flexibility in scaling, GitLab implements a plugin system, and you can also use custom executors. Definitely an extra gear: **nice shot, GitLab: 5 – 3.**

AWS CodePipeline allows you to use a multi-az deployment. This ensures, for example, your build can run on the other 2 availability zones in case of failure in eu-west-1-a.

Note that if you are using GitLab's EC2 Autoscaling plugin, multi-az deployment is not implemented. Here's the sample config taken from this sample.

```
[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-runner-autoscale,true",
      "amazonec2-security-group=xxxxx",
      "amazonec2-instance-type=m4.2xlarge",
    ]
```

As you can see, it is possible to set a single subnet only and a single Availability Zone. With this configuration, your build will fail in case of failure in the chosen zone. AWS gets the point. **It's 6 – 3.**

Note that is possible to implement a custom executor to increase reliability, but you'll need to develop it and test it on your own with a considerable waste of time.

Let's continue with the two last two rounds.

Round 4: Performance Efficiency

For this game, points will be defined based on the following principles:

- Democratizing advanced technologies
- Going global in minutes
- Using serverless architectures
- Experimenting more often
- Conoscere e tenere in considerazione l'infrastruttura sottostante (qualche specifica qui).

Even if GitLab and CodePipeline have a lot in common, there's a little more flexibility on the GitLab side, as it supports a lot of different deployments and configurations.

One point to GitLab: **6 – 4.**

On the other side, CodePipeline offers better support for serverless builds, in comparison with the limited GitLab support for some serverless approaches. For example, you can't use Docker in Docker leveraging on the same Docker socket on the fargate custom executor.

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

Round 5: Cost Optimization

For this game, points will be defined based on the following principles:

- Implement cloud financial management
- Adopt a consumption model
- Measure overall efficiency
- Stop spending money on undifferentiated heavy lifting
- Analyze and attribute expenditure

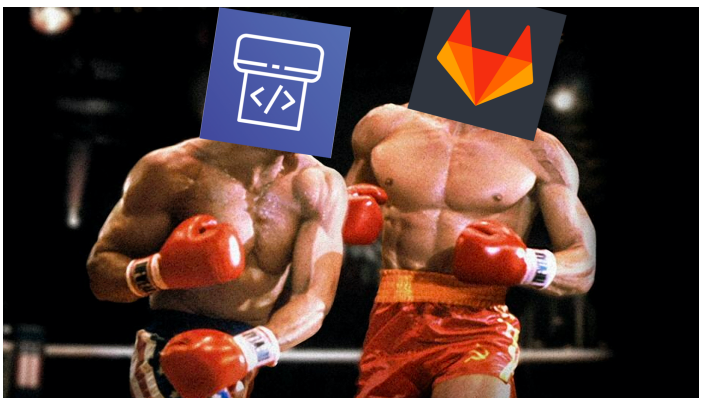
Let's analyze pricing models: GitLab is based on a per-user pricing model, while AWS users are free. CodeBuild minutes are priced based on the time you use resources for a build. If you manage to use a general1.small (2 cores and 3gb of RAM), you'll end up spending 5 dollars for 1.000 minutes.

With GitLab shared runners, instead, you can purchase 1.000 minutes of shared runners for 10$ (the first 2.000 minutes are free).

If you don't want to use GitLab's shared runners, you can use AWS Auto Scaling for builds, but keep in mind that you'll need an EC2 instance for the runner manager and other instances to allow your builds to scale. Other possibilities are purchasing reserved instances or using spot instances if your business allows you to do it.

For this round, no doubts for the winner: AWS CodePipeline gets the last point, as it offers a more pre-built and out-of-the-box and integrated solution.

Podium and conclusions



Our friendly fight between 2 of the most widely adopted services has ended. With a score of 8 – 4, we are welcoming **AWS CodePipeline on the top step of the podium!**

Will it keep its supremacy?

Don't get us wrong: GitLab remains an excellent service with great features for teams and top user experience for developers and under specific conditions, it is still the best choice, according to specific use cases.

In some other cases, choosing isn't a good idea at all... Depending on the needs, mixing both of the services can be a winning choice... but this is a plain and simple spoiler for the next series of articles... 🙂

Keep warm for the next fight. Till then, code secure and see you in 14 days on **#Proud2beCloud**!

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

Copyright © 2011-2021 by beSharp srl - P.IVA IT02415160189