



Certified Tech Developer

The Ultimate Degree

Infraestructura II

Actividad obligatoria y grupal

Dificultad: Alta

Deploy de infraestructura y aplicación

En las prácticas anteriores desplegamos infraestructura mediante un pipeline en GitLab. En esta ocasión vamos a ir un poco más allá y agregaremos a nuestra infraestructura un servidor instalando le un sitio web.

Etapas

Nuestro trabajo será modularizado siguiendo este esquema:

1. Consideración de prerequisites.
2. Obtención del código fuente.
3. Configuración del proyecto.
- 4. Configuración del ambiente local.**
- 5. Integración del código al proyecto.**
- 6. Ejecución del pipeline.**

Tené en cuenta que cada una de estas etapas es descrita en la infografía “Mapeamos el proceso de trabajo”, disponible en Playground. ¡No la pierdas de vista! ¡Va a ser una buena brújula para esta práctica!

Acerca de esta ejercitación

Dada la complejidad de esta práctica y el de cada uso que podamos hacer al combinar GitLab y Terraform, no estaremos contemplando las buenas prácticas que en un ambiente real serían necesarias; un ejemplo de esto es el uso de una rama “Master” en GitLab, en vez de “pushear” previamente a una rama de “Development”, o similar.

No obstante, recomendamos e invitamos a investigar “buenas prácticas” en el uso de:

- Git Branch Strategies
- Gitlab Security
- Gitlab CI Architectures
- Terraform Remote Backend
- Terraform Modules

¡Y conversar con tu profesor cualquier inquietud que pudiera aparecer!

Nota: *nuestro proceso ha sido ejecutado desde una computadora con MAC OS. Sin embargo, los mismos principios –quizás no así los comandos– aplicarán para una computadora con Linux o Windows.*

¡Vamos paso a paso!

4. Configuración del ambiente local

¡Vamos ahora a preparar nuestro puesto de trabajo! El primer paso será retomar la configuración pendiente de las llaves SSH que dejamos previamente.

Para hacerlo, abrimos una terminal de línea de comando para generar el par de llaves SSH privada-pública. Vamos a asociarlas a nuestro proyecto y así autenticarnos sin el uso de credenciales. Usaremos el comando:

```
ssh-keygen -t rsa -b 4096 -C "DH"
```

Que creará un par de llaves criptográficas, generalmente dentro del directorio oculto “.ssh”. En nuestro caso, “/Users/[TU NOMBRE DE USUARIO]/.ssh”

Abriremos el archivo con extensión .PUB (que se generará con la instrucción anterior), en nuestro caso "dh.pub", y copiaremos todo su contenido para luego pegarlo en la ventana de llaves:

Lanzamos el proceso ssh-agent que contendrá nuestra llave privada (archivo que no tiene extensión .PUB) para poder ser utilizada durante la validación de identidad contra GitLab:

```
ssh-add .ssh/dh
```

Nota: verifiquen estar en el directorio .ssh y/o apuntarlo apropiadamente.

La salida debería ser similar a la siguiente:

```
Identity added: .ssh/dh (DH key)
```

Nota: podremos validar su correcta adición mediante la orden:

```
ssh-add -l
```

Luego, para verificar que tenemos conexión y acceso autenticado a gitlab.com con la llave SSH que creamos, invocamos la siguiente orden:

```
ssh -T git@gitlab.com
```

El output será similar al siguiente:

```
The authenticity of host 'gitlab.com (172.65.251.78)' can't be established.
```

```
ECDSA key fingerprint is
```

```
SHA256:HbW3g8zUjNSksFbqTiUWPWg2Bq1x8xdGUrliXFzSnUw.
```

```
Are you sure you want to continue connecting (yes/no)?
```

Este error es normal y se debe a que el sistema operativo nos está diciendo que la llave creada “aún” no es conocida y por lo tanto, debemos agregarla. **¡Vamos a hacerlo escribiendo “yes”!** El output del comando será similar al siguiente:

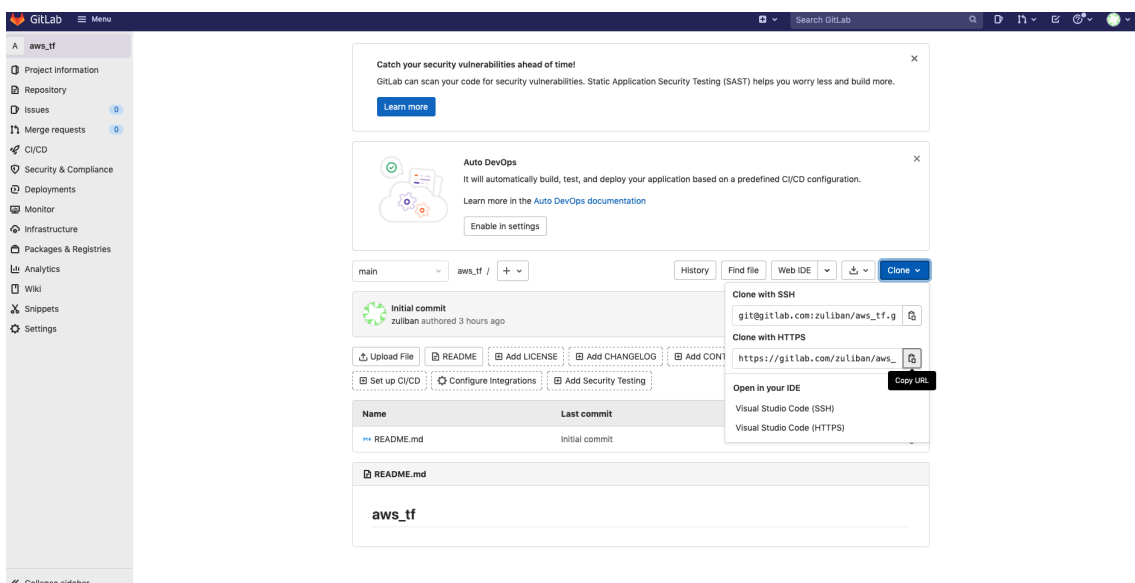
```
Welcome to GitLab, @zuliban!
```

Nota: En caso de que nuestro proceso `ssh-agent` no esté en memoria, nos arrojará un error el cual podremos by-pasear mediante la orden:

```
eval "$(ssh-agent)"
```

Y luego volviendo a agregar la llave mediante `ssh-add ./ssh/dh`

Ahora deberemos obtener la dirección de nuestro repo. La tomamos de la página inicial del proyecto:



Como estamos utilizando la autenticación vía SSH Keys, copiaremos la correspondiente a “Clone with SSH”. Volvemos a nuestra terminal y ejecutamos la siguiente orden:



```
git remote add origin git@gitlab.com:zuliban/aws_tf.git
```

Mediante la instrucción `git config -l` deberemos ser capaces de ver la entrada a nuestro repo y asegurarnos que ya fue agregado:

```
remote.origin.url=git@gitlab.com:zuliban/aws_tf.git
```

Con estos pasos, le estamos indicando a nuestra computadora en qué proyecto vamos a estar colaborando.

5. Integración del código al proyecto

Vamos ahora a subir nuestro código Terraform al proyecto para que podamos comenzar a colaborar, trackear código y, finalmente, correr nuestro pipeline. Vía consola, nos posicionamos en el directorio donde hemos dispuesto nuestro código y ejecutamos:

```
git status
```

```
git add .
```

```
git commit -m "este es mi primer commit de código terraform para generar  
infraestructura mediante la ejecución de un pipeline. Tambien, soy bien explícito  
en mis mensajes al comitear!"
```

El output se verá similar a:

```
[master (root-commit) 0b3e929] este es mi primer commit de código terraform  
para generar infraestructura mediante la ejecución de un pipeline. Tambien, soy  
bien explícito en mis mensajes al comitear
```

```
3 files changed, 177 insertions(+)
```

```
create mode 100644 main.tf
```

```
create mode 100644 providers.tf
```

```
create mode 100644 variables.tf
```

¡Y pusheamos!



```
git push --set-upstream origin master
```

Nuestro output se verá como el que sigue:

```
Enumerating objects: 5, done.
```

```
Counting objects: 100% (5/5), done.
```

```
Delta compression using up to 8 threads
```

```
Compressing objects: 100% (5/5), done.
```

```
Writing objects: 100% (5/5), 1.93 KiB | 394.00 KiB/s, done.
```

```
Total 5 (delta 0), reused 0 (delta 0)
```

```
remote:
```

```
remote: To create a merge request for master, visit:
```

```
remote:
```

```
https://gitlab.com/zuliban/aws\_tf/-/merge\_requests/new?merge\_request%5Bsource\_branch%5D=master
```

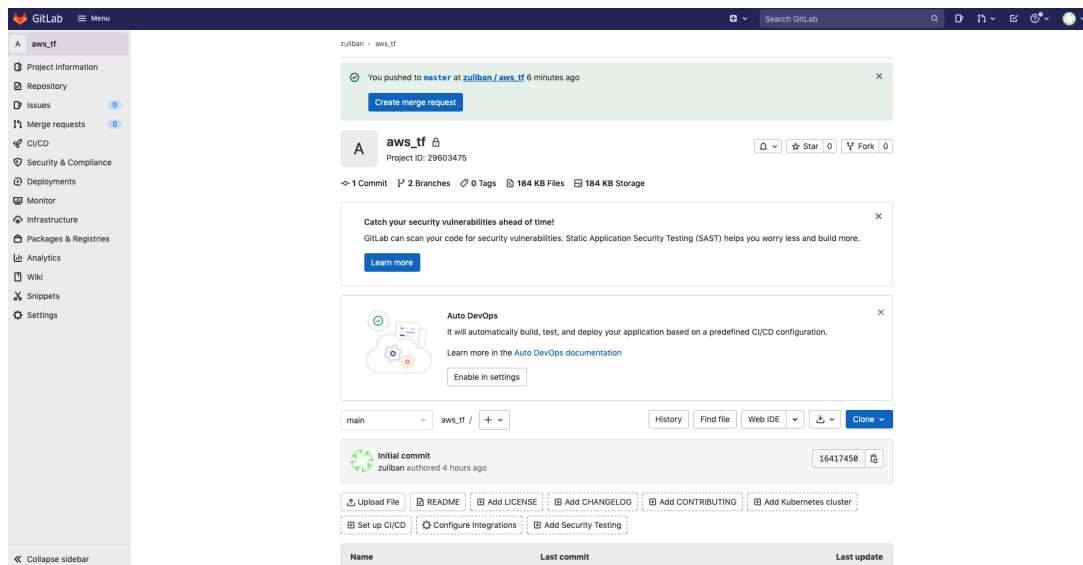
```
remote:
```

```
To gitlab.com:zuliban/aws_tf.git
```

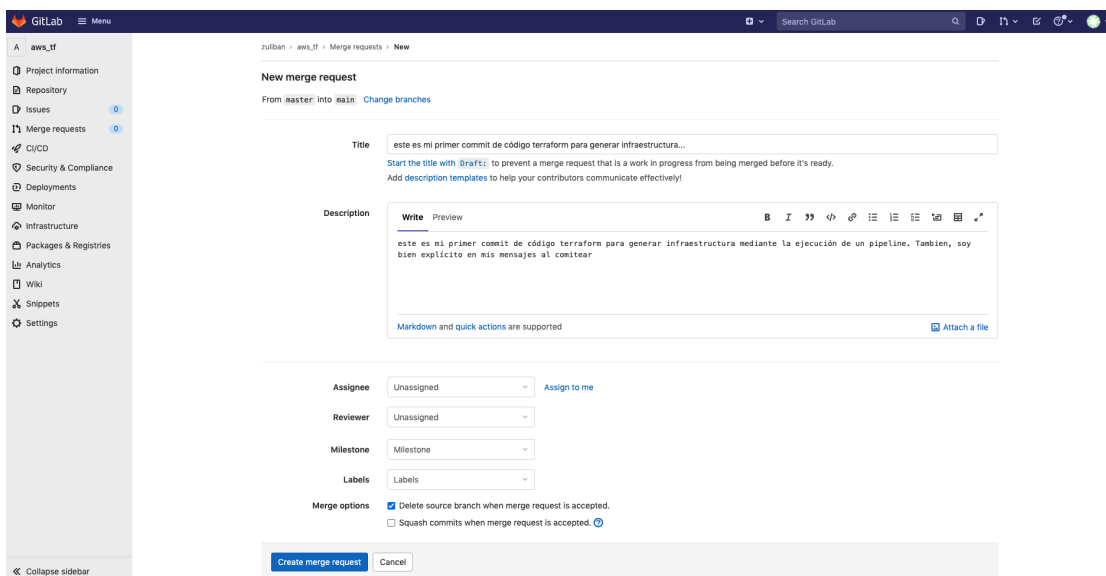
```
* [new branch]    master -> master
```

```
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

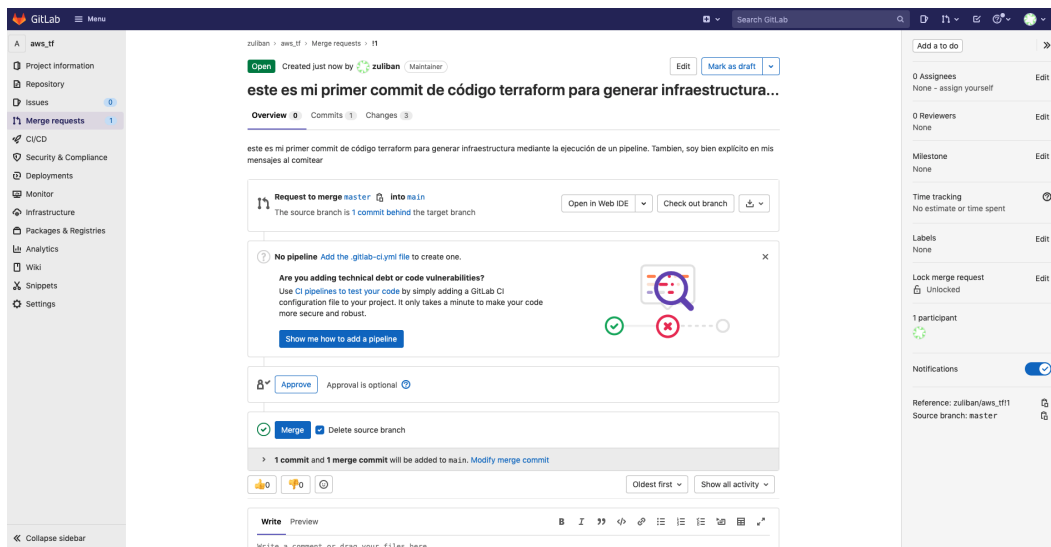
Esto creará una solicitud de “Meryeo”:



GitLab nos pide generar esta solicitud porque la filosofía que emplea implica tener control sobre todo lo queremos ubicar en nuestro repositorio. En otras palabras, es una forma de garantizar que todos nuestros “commits” son validados por alguien (otro colaborador) antes de cambiar el área de trabajo “Master” o “Main”.



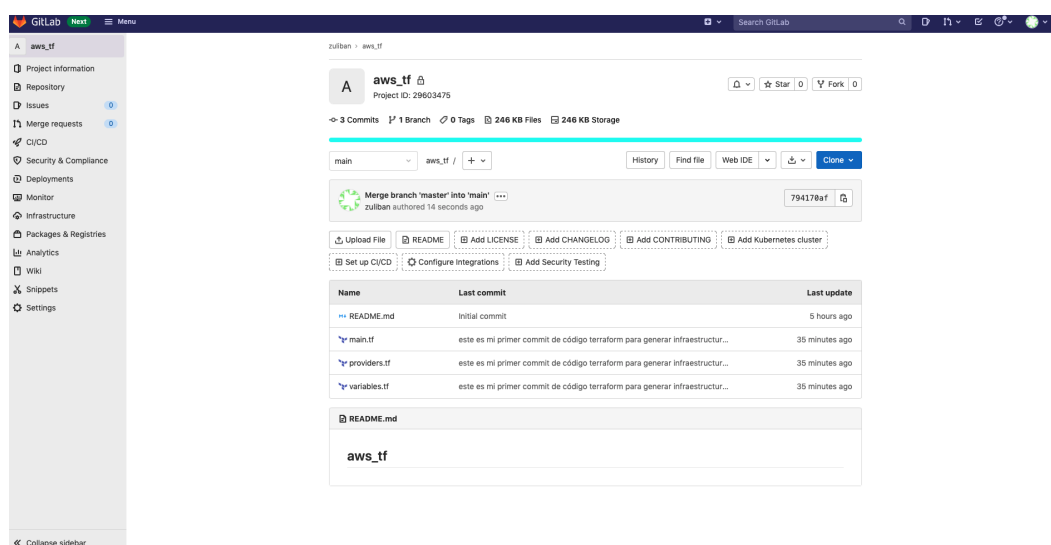
Clic en el botón “Create merge request” para saltar a esta página:



Clic en el botón “Merge” para hacer efectivo el cambio sobre nuestra rama “Master” o “Main”.

Nota: En este punto volvemos a hacer hincapié en la recomendación de aplicar buenas prácticas. Invitamos a recorrer –con suficiente tiempo– cada una de las opciones disponibles. Estas opciones son muy importantes no solo a nivel seguridad, sino también para que podamos comprender con mayor detalle las características de este sistema.

Al volver a nuestro proyecto, veremos nuestro primer commit realizado satisfactoriamente:



¡Ya tenemos nuestro código centralizado e integrado con nuestro sistema GIT local!

6. Ejecución del pipeline

Llegó el momento más esperado: **¡Construir nuestro pipeline!** Para esto, GitLab se sirve de un archivo muy particular: ".gitlab-ci.yml". Nuestro primer paso será copiar el siguiente contenido a un archivo nuevo y guardarlo con el nombre ".gitlab-ci.yml":

```
include:

  - template: Terraform/Base.gitlab-ci.yml #
    https://gitlab.com/gitlab-org/gitlab/blob/master/lib/gitlab/ci/templates/Terraform/Base.latest.gitlab-ci.yml

stages:

  - init

  - validate

  - build

  - deploy

init:

  extends: .init

validate:

  extends: .validate

build:

  extends: .build

deploy:
```

```
extends: .deploy

dependencies:

  - build
```

Nota: Este archivo debe estar ubicado en el mismo directorio que nuestro proyecto.

Ahora vamos a “pushear” nuevamente nuestro build:

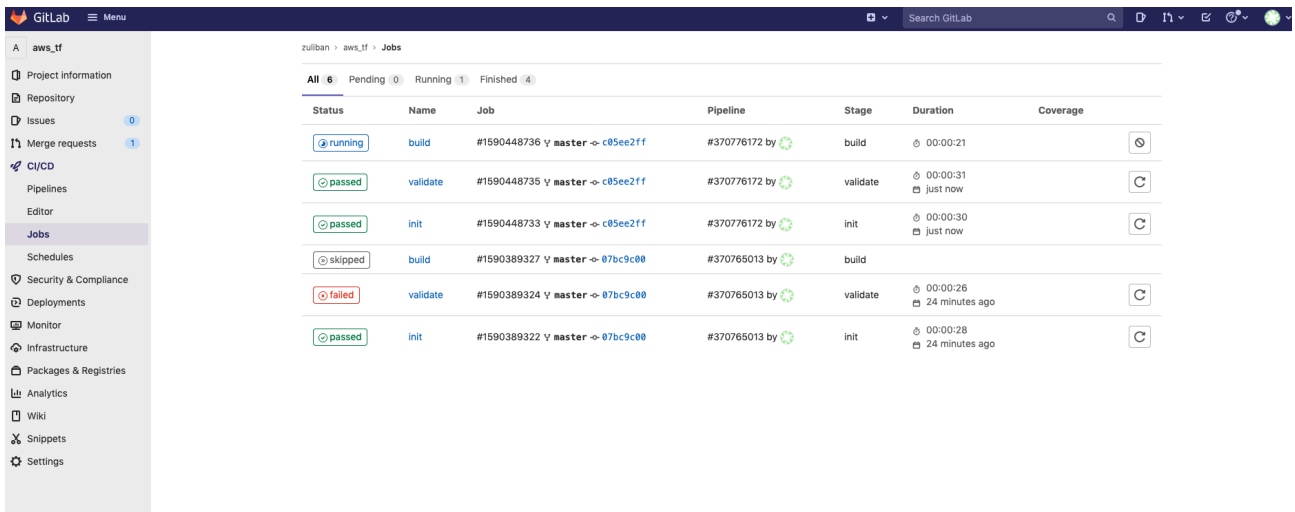
```
git add .
```

```
git commit -m "agregamos el .gitlab-ci.yml para que se ejecute nuestro pipeline"
```

```
git push
```

Esta acción creará una solicitud de “Meryeo”. Hacemos clic en “Create merge request” y en el botón “Merge” para hacer efectivo el cambio sobre nuestra rama “Master” o “Main”.

Notaremos una diferencia: el botón “Merge” ahora dice “Merge when pipeline succeeds”. Esto significa que nuestro cambio será agregado a nuestra rama “Master” o “Main” siempre y cuando la ejecución de nuestro pipeline haya sido satisfactoria.



The screenshot shows the GitLab interface for a project named 'aws_tf'. The left sidebar contains navigation links: Project information, Repository, Issues (0), Merge requests (1), CI/CD, Pipelines, Editor, Jobs (selected), Schedules, Security & Compliance, Deployments, Monitor, Infrastructure, Packages & Registries, Analytics, Wiki, Snippets, and Settings.

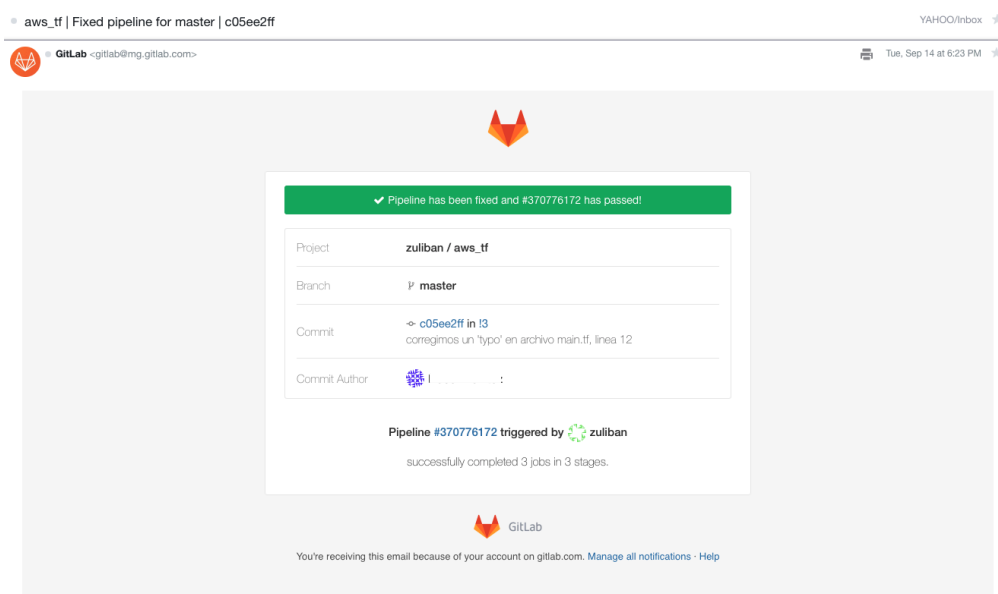
The main content area displays the 'Jobs' section for the 'aws_tf' project. It shows a table of pipeline jobs with columns: Status, Name, Job, Pipeline, Stage, Duration, and Coverage. The jobs are as follows:

Status	Name	Job	Pipeline	Stage	Duration	Coverage
running	build	#1590448736 y master -> c05ee2ff	#370776172 by	build	00:00:21	
passed	validate	#1590448735 y master -> c05ee2ff	#370776172 by	validate	00:00:31 just now	
passed	init	#1590448733 y master -> c05ee2ff	#370776172 by	init	00:00:30 just now	
skipped	build	#1590389327 y master -> 07bc9c00	#370765013 by	build		
failed	validate	#1590389324 y master -> 07bc9c00	#370765013 by	validate	00:00:26 24 minutes ago	
passed	init	#1590389322 y master -> 07bc9c00	#370765013 by	init	00:00:28 24 minutes ago	

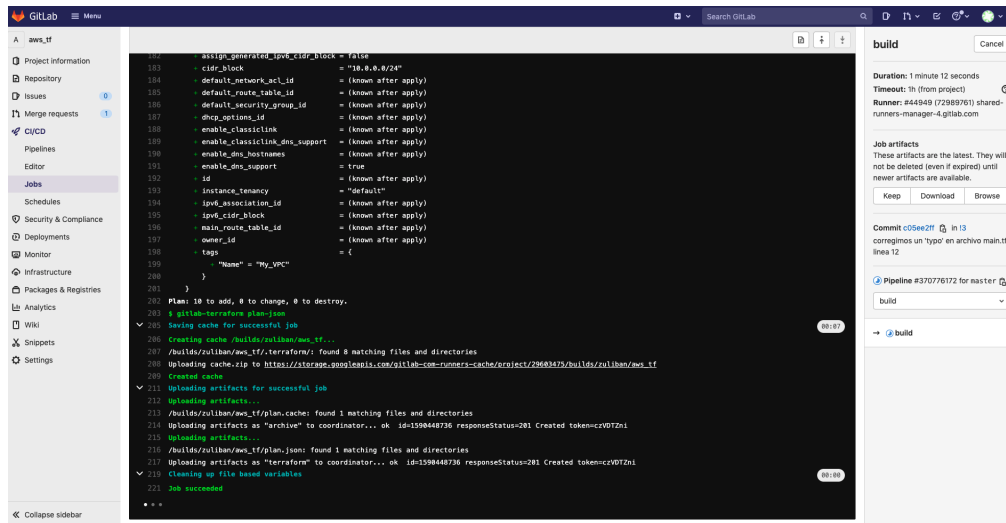
Nota: en esta captura, vemos como segundo paso un evento “failed”. Esto sucedió porque se encontró un ‘typo’ en el recurso “aws_vpc” del archivo “Main.tf” el cual ustedes también verán dado que el código fuente no fue corregido.

Luego de corregirlo, se efectuó un nuevo “commit” para poder proceder. ¡Ustedes deberán hacerlo de igual manera!

Dato: Como parte de la integración que GitLab nos ofrece, vamos a poder hacer el seguimiento de cada ejecución: por cada ejecución nos estará llegando un correo electrónico a la dirección que hemos configurado.



Continuando en la ventana de visualización de jobs, veremos en tiempo real la lectura y ejecución de cada etapa o “stage” del pipeline:



Si nos movemos a la opción “Jobs”, en el menú contextual de nuestra consola, veremos que todos los stages han finalizado correctamente y que ahora estamos listos para dar nuestro último paso de forma manual: el “Deploy”.

Esta etapa manual pone el énfasis en la interacción con “Producción” y en que tengamos todos los recaudos necesarios. La manualidad nos permite visualizar todo el plan ejecutado previamente evitando cualquier error que se pueda haber pasado por alto.

Status	Name	Job	Pipeline	Stage	Duration	Coverage
@ manual	deploy	#1590452244 ✓ main → 9f129ca5 allowed to fail manual	#370776829 by	deploy		
passed	build	#1590452242 ✓ main → 9f129ca5	#370776829 by	build	00:00:37 just now	
passed	validate	#1590452240 ✓ main → 9f129ca5	#370776829 by	validate	00:00:41 1 minute ago	
passed	init	#1590452238 ✓ main → 9f129ca5	#370776829 by	init	00:00:57 1 minute ago	
passed	build	#1590448736 ✓ master → c85ee2ff	#370776172 by	build	00:01:14 2 minutes ago	
passed	validate	#1590448735 ✓ master → c85ee2ff	#370776172 by	validate	00:00:31 3 minutes ago	
passed	init	#1590448733 ✓ master → c85ee2ff	#370776172 by	init	00:00:30 4 minutes ago	
skipped	build	#1590389327 ✓ master → 07bc9c00	#370765013 by	build		
failed	validate	#1590389324 ✓ master → 07bc9c00	#370765013 by	validate	00:00:26 27 minutes ago	
passed	init	#1590389322 ✓ master → 07bc9c00	#370765013 by	init	00:00:28 28 minutes ago	

Al hacer clic en el botón “Manual” pasaremos a esta otra vista:

Nuestro pipeline comenzará su etapa final: ¡Deploy!

Aquí continuamos el seguimiento del plan de ejecución a medida que avanza:

GRLab
Search GRLab

- A aws_tf
- Project Information
- Repository
- Issues
- Merge requests
- CIVCD**
- Pipelines
- Editor
- Jobs
- Schedules
- Security & Compliance
- Deployments
- Monitor
- Infrastructure
- Package & Registries
- Analytics
- Wiki
- Snippets
- Settings

```

140 aws_elb_nat.EIP: Creating...
141 aws_vpc.Main: Creating...
142 aws_elb_nat.EIP: Creation complete after 1s [id=elb-palloc-e0bf9e7d5741423b]
143 aws_vpc.Main: Creation complete after 1s [id=vpc-f799336df7384941]
144 aws_internet_gateway.IDM: Creating...
145 aws_subnet.public_subnets: Creating...
146 aws_subnet.private_subnets: Creating...
147 aws_subnet.private_subnets: Creation complete after 1s [id=subnet-beafed5f4f8eb45e]
148 aws_subnet.public_subnets: Creation complete after 1s [id=subnet-e225ee6632859887]
149 aws_nat_gateway.NAT_GW: Creating...
150 aws_internet_gateway.IDM: Creation complete after 1s [id=idgw-8859669e8ea7c72d]
151 aws_route_table.Public_RT: Creating...
152 aws_route_table.Public_RT: Creation complete after 1s [id=rtn-8052771a549581538]
153 aws_route_table_association.Public_RT_Association: Creating...
154 aws_route_table_association.Public_RT_Association: Creation complete after 8s [id=rtassoc-81d738837d7738d37]
155 aws_nat_gateway.NAT_GW: Still creating... [18s elapsed]
156 aws_nat_gateway.NAT_GW: Still creating... [20s elapsed]
157 aws_nat_gateway.NAT_GW: Still creating... [20s elapsed]
158 aws_nat_gateway.NAT_GW: Still creating... [40s elapsed]
159 aws_nat_gateway.NAT_GW: Still creating... [50s elapsed]
160 aws_nat_gateway.NAT_GW: Still creating... [50s elapsed]
161 aws_nat_gateway.NAT_GW: Still creating... [1m0s elapsed]
162 aws_nat_gateway.NAT_GW: Still creating... [1m0s elapsed]
163 aws_nat_gateway.NAT_GW: Still creating... [1m20s elapsed]
164 aws_nat_gateway.NAT_GW: Still creating... [1m20s elapsed]
165 aws_nat_gateway.NAT_GW: Still creating... [1m30s elapsed]
166 aws_route_table.Private_RT: Creation complete after 1m36s [id=mat-8f1c2fc8b7fe6c56]
167 aws_route_table.Private_RT: Creating...
168 aws_route_table.Private_RT: Creation complete after 1s [id=rtn-8288cd1334518f7a]
169 aws_route_table_association.Private_RT_Association: Creating...
170 aws_route_table_association.Private_RT_Association: Creation complete after 8s [id=rtassoc-85396371d987461c]
171 Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
172 Saving state for successful job
173 Creating cache /builds/zulibaw/aws_tf...
174 /builds/zulibaw/aws_tf:terraform/. found 8 matching files and directories
175 Uploading cache.zip to https://storage.googleapis.com/gcp-lab-com-runners-cache/project/72603472/builds/zulibaw/aws_tf
176 Created cache
177 Cleaning up file based variables
178 Job Succeeded
        
```

deploy

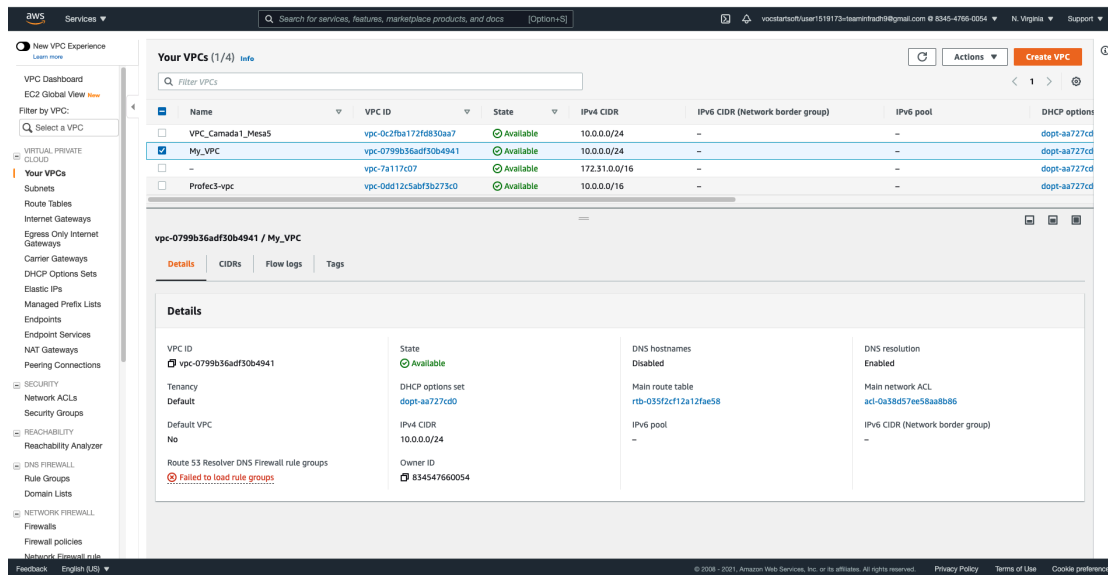
Duration: 2 minutes 16 seconds
 Timeout: 1h (from project)
 Runner: #380986 (0277ead0) shared
[runners-manager - gslab.com](#)

Commit 9f125ac
 Merge branch 'master' into 'main'

☐ Pipeline #370778829 for main
☒ deploy

→ @deploy

14



The screenshot shows the AWS Management Console interface. On the left, there's a navigation menu with categories like VPC, SECURITY, REACHABILITY, DNS, and NETWORK. The main content area is titled 'Your VPCs (1/4)' and displays a table of VPCs. Below the table, the details for the selected VPC 'vpc-0799b36adf30b4941 / My_VPC' are shown, including its state (Available), DHCP options set, IPv4 CIDR, and DNS settings.

Name	VPC ID	State	IPv4 CIDR	IPv6 CIDR (Network border group)	IPv6 pool	DHCP options
VPC_Camada1_Mesa5	vpc-0c2fba172f1d830aa7	Available	10.0.0.0/24	-	-	dopt-aa727cd
My_VPC	vpc-0799b36adf30b4941	Available	10.0.0.0/24	-	-	dopt-aa727cd
-	vpc-7a117cd07	Available	172.31.0.0/16	-	-	dopt-aa727cd
Protec3-vpc	vpc-0dd12c5abf3b273c0	Available	10.0.0.0/16	-	-	dopt-aa727cd

vpc-0799b36adf30b4941 / My_VPC

Details

VPC ID vpc-0799b36adf30b4941	State Available	DNS hostnames Disabled	DNS resolution Enabled
Tenancy Default	DHCP options set dopt-aa727cd0	Main route table rtb-035f2cf12a12fae58	Main network ACL acl-0a38d57ee58aa8b86
Default VPC No	IPv4 CIDR 10.0.0.0/24	IPv6 pool -	IPv6 CIDR (Network border group) -
Route 53 Resolver DNS Firewall rule groups Failed to load rule groups			
Owner ID 834547660054			

Conclusión

En esta ejercitación pudimos levantar una infra en AWS utilizando Terraform dentro de un pipeline en la nube... ¡Replicando de forma tangible un proceso de deployment tal y como se vería en un ambiente real!