**Sprint 3: Creación de pipelines**

**Grupo 4**

**Integrantes:**

**David Carvalho**

**Andrea Giraldo**

**Stivel Pinilla**

**Sebastián Fuentes**

**Oscar Arbelaez**

**Andres Agudelo**

**Misión TIC**

**Desarrollo Web**

**Universidad de Antioquia**

**26 de septiembre de 2021**

**Historia de usuario: HU013**

Se configuran los Jobs para pruebas unitarias.

Con el uso de las librerías *chai*, *mongoose* y *mocha*, se desarrollaron las pruebas unitarias para validar las operaciones CRUD sobre la base de datos. En el Git hub que se encuentra en el siguiente link <https://github.com/ancagi/Proyecto_Ciclo_IV_Grupo_1_Equipo_4/> en la carpeta *test*.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

*Figura 1. Organización de los archivos github*

* En *src/models* en el archivo *student\_schema.js* se encuentra el esquema para la colección estudiante de acuerdo con la estructura de base de datos desarrollado en el sprint anterior (figura 2).

Texto

Descripción generada automáticamente

*Figura 2. Schema de la colección estudiante*

* En la carpeta test archivo test.js se encuentran las pruebas unitarias desarrolladas para las operaciones sobre la colección estudiante:

Texto

Descripción generada automáticamente

*Figura 3. Test para validar el ingreso de un campo requerido*

Texto

Descripción generada automáticamente

*Figura 4. Test para validar el tipo de dato ingresado*

Texto

Descripción generada automáticamente

*Figura 5. Test para validar la inserción de un estudiante en la colección*

Texto

Descripción generada automáticamente

*Figura 6. Test para validar el update de un estudiante en la colección*

Texto

Descripción generada automáticamente

*Figura 7. Test para validar la eliminación de un estudiante en la colección*

Al usar npm test se encuentra que todas las pruebas de validación se cumplen.

Texto

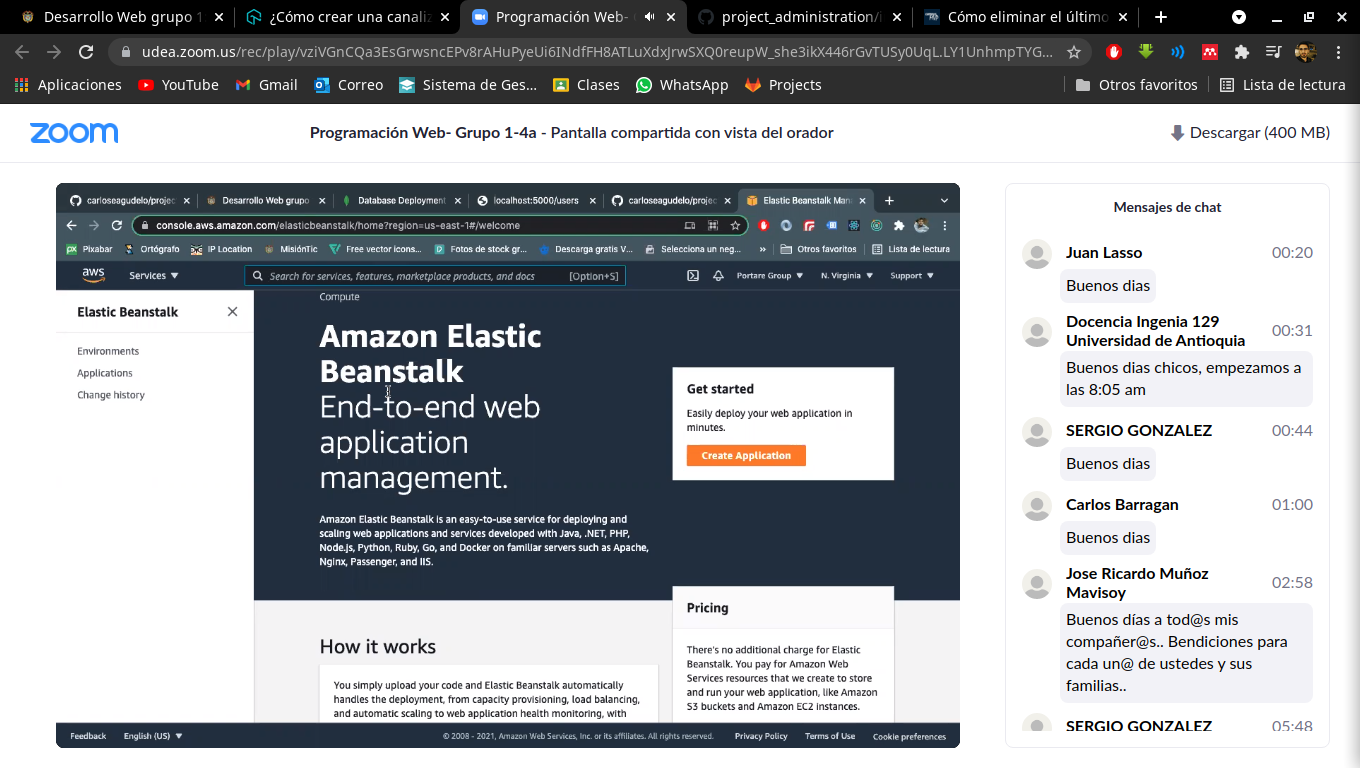
Descripción generada automáticamente

*Figura 8. Ejecución de las pruebas unitarias*

Similar a lo ilustrado anteriormente se pueden probar todas las colecciones y campos requeridos para la base de datos diseñada en el anterior sprint.

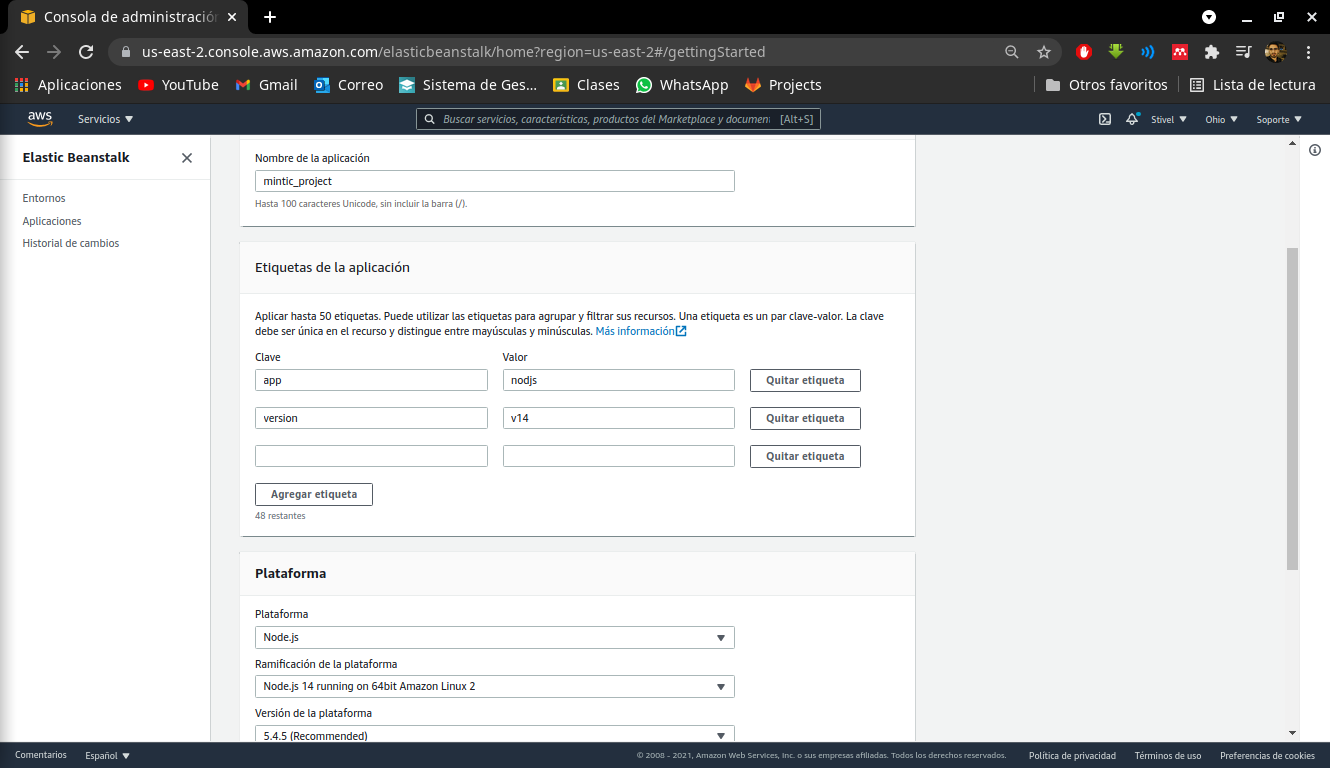
**Historia de usuario: HU\_ 014**

Usando los servicios que AWS ofrece realizamos los pipelines necesarios para CI/CD. Para ello lo primero que hacemos es crear una cuenta gratuita en la plataforma. Hacemos uso de *Amazon Elastic Beanstalk* (figura 9) que corresponde a un servicio para el despliegue y la escalabilidad de aplicaciones web.



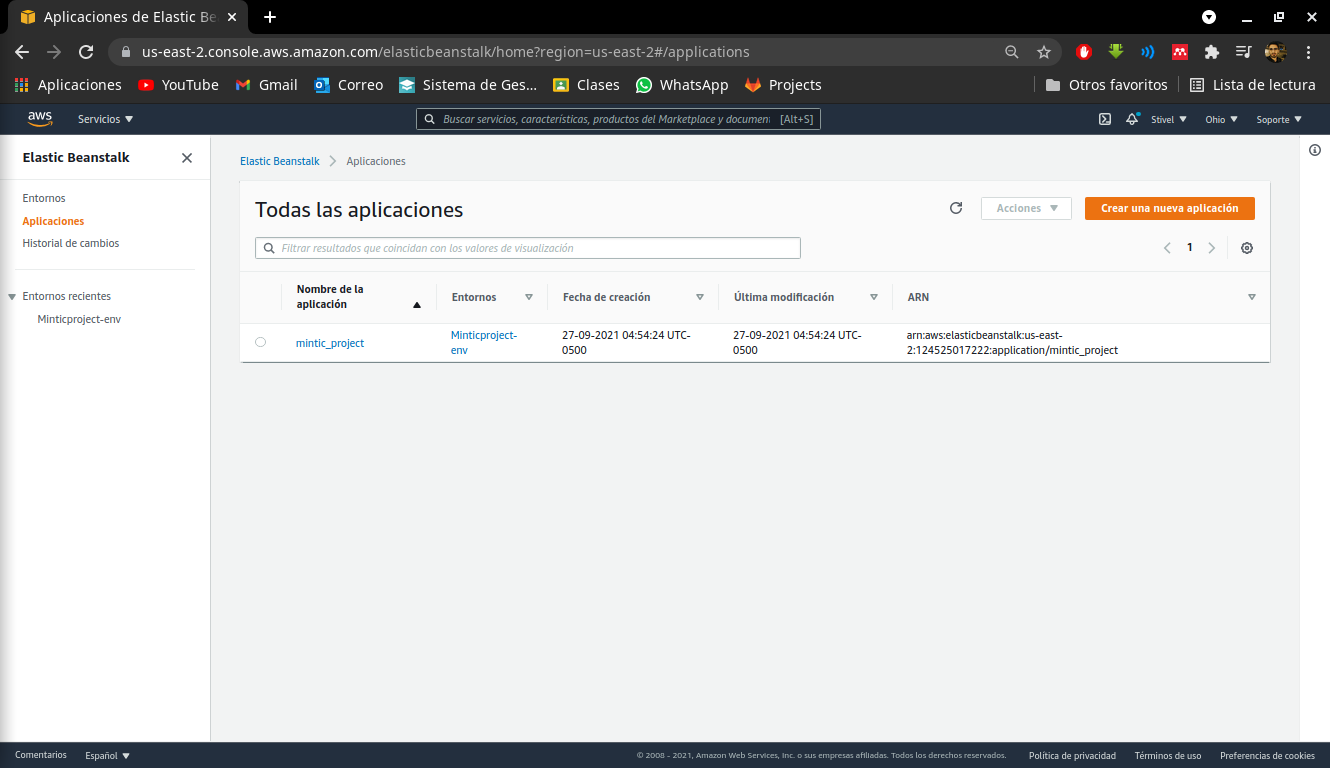
*Figura 9. Servicio de Amazon Elastic Beanstalk.*

Creamos una nueva aplicación a la cual llamamos *mintic\_project* y le agregamos algunos t*ags:*



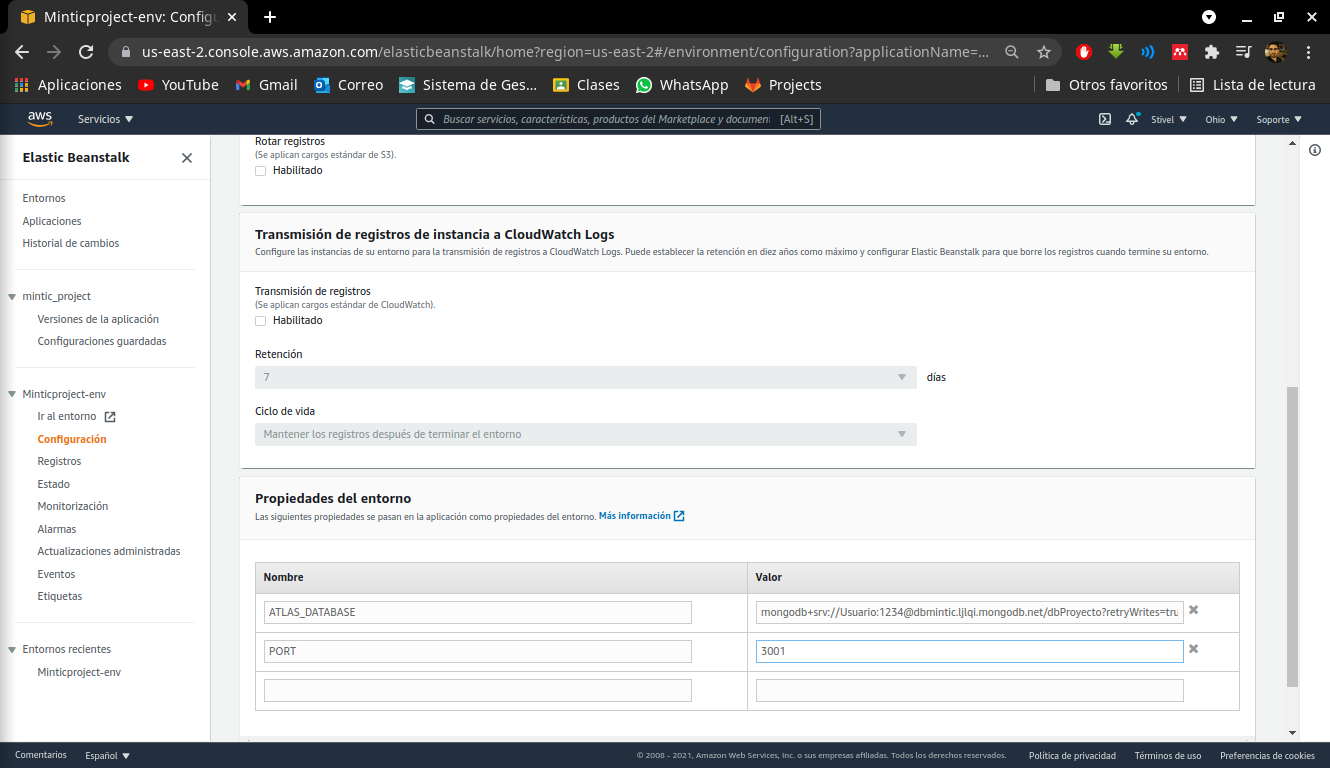
*Figura 10. Creación de un Elastic Beanstalk.*

Al crearlo y dejar por defecto una aplicación que base Nodejs tenemos desplegada una aplicación web:



*Figura 11 Aplicación Elastic Beanstalk.*

Cómo realizaremos el despliegue de nuestra aplicación web vamos a guardar dos variables de entorno en la plataforma de AWS, que corresponde a la dirección de la base de datos y el puerto a utilizar. Esto lo podemos realizar estando ubicados en *Elastic Beanstalk* y dirigiéndonos a *Configuración:*

**

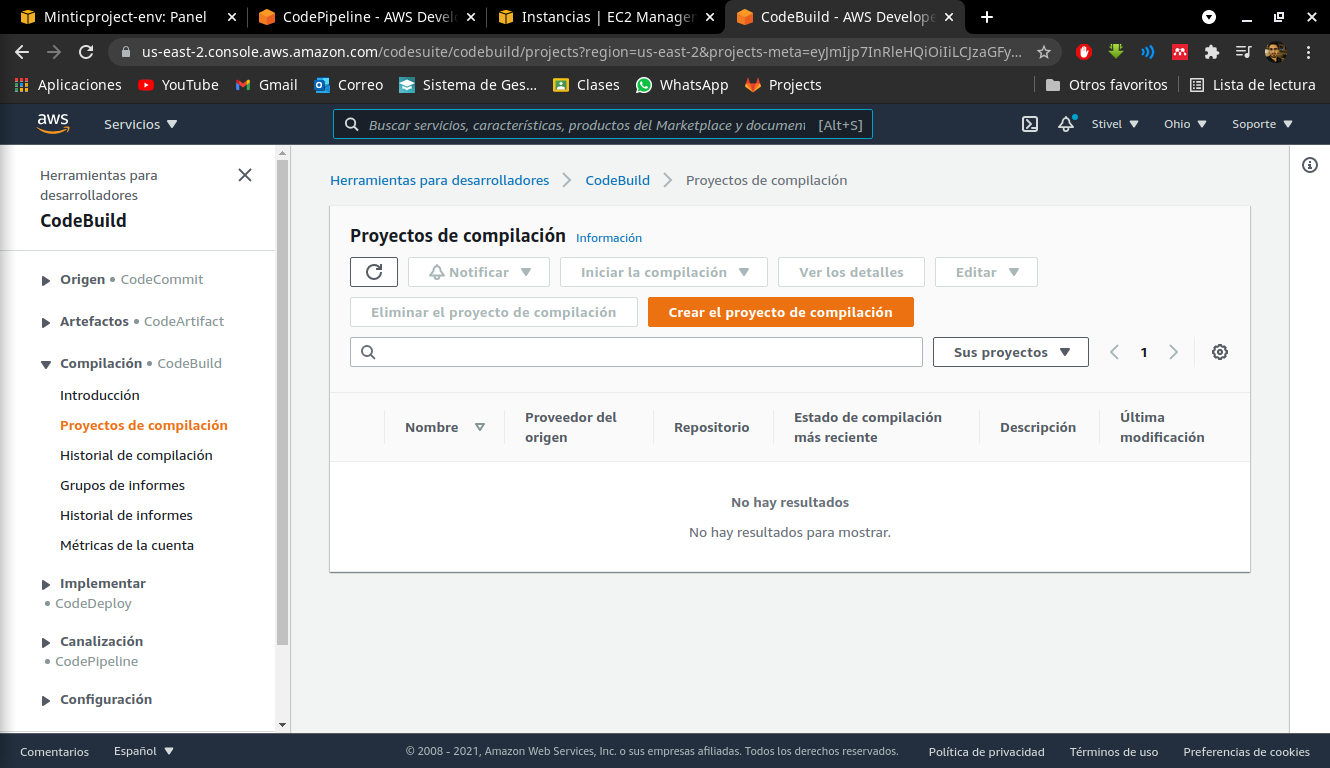
*Figura 12 Configuración de variables de entorno en Elastic Beanstalk.*

Además en configuraciones en el apartado de seguridad cambiamos el rol de servicio para poder acceder más tarde desde el pipeline.

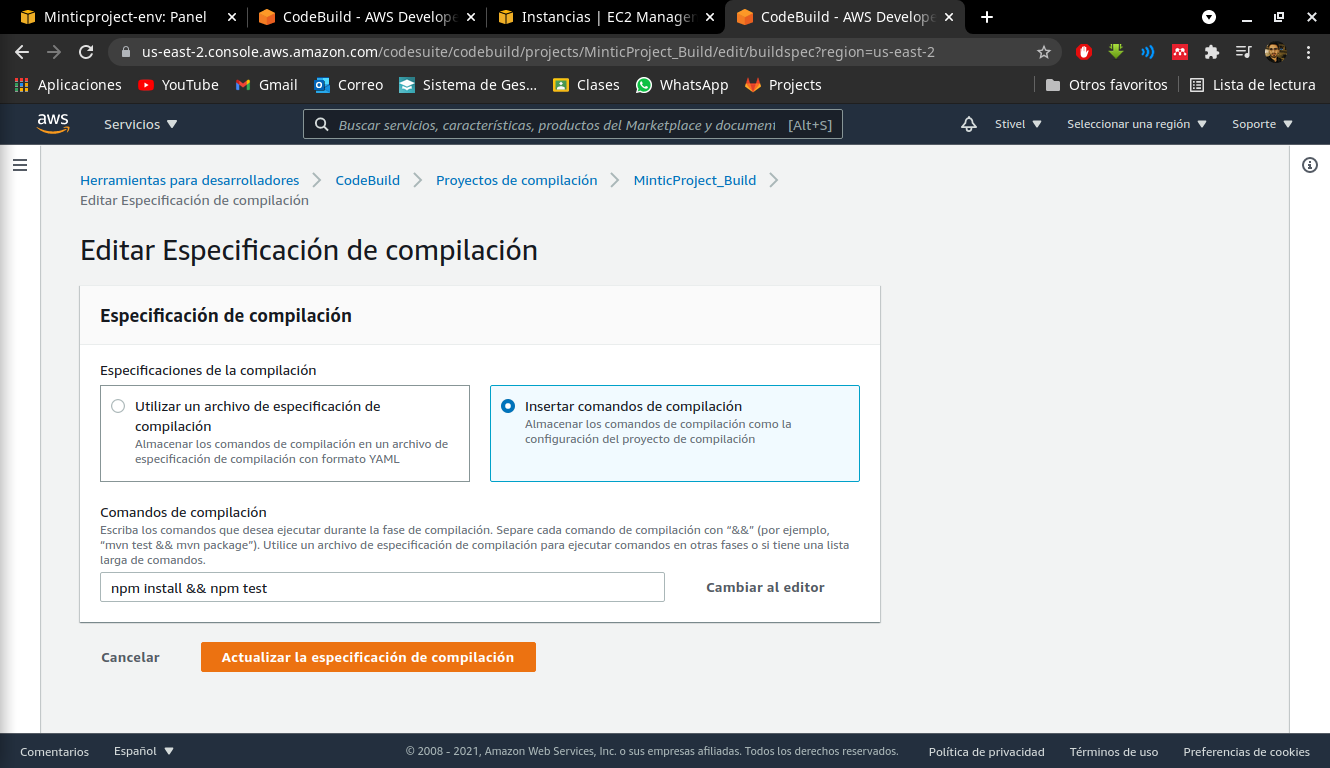


*Figura 13 Configuración del rol de servicio en Elastic Beanstalk.*

Ahora para realizar las pruebas unitarias utilizaremos *CodeBuild* de AWS (figura 14) y dentro de las configuraciones del proyecto agregamos los comandos *npm install* y *npm test* (figura 15).



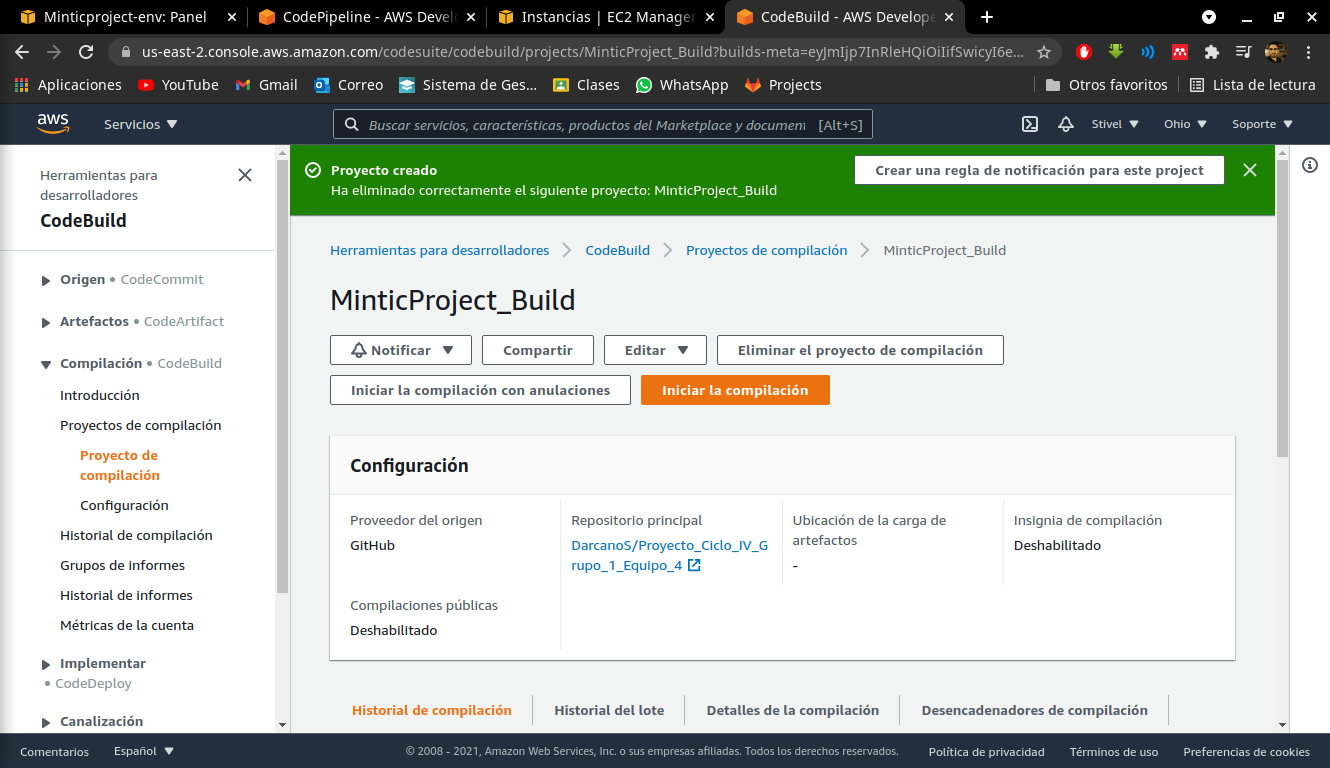
*Figura 14 CodeBuild*



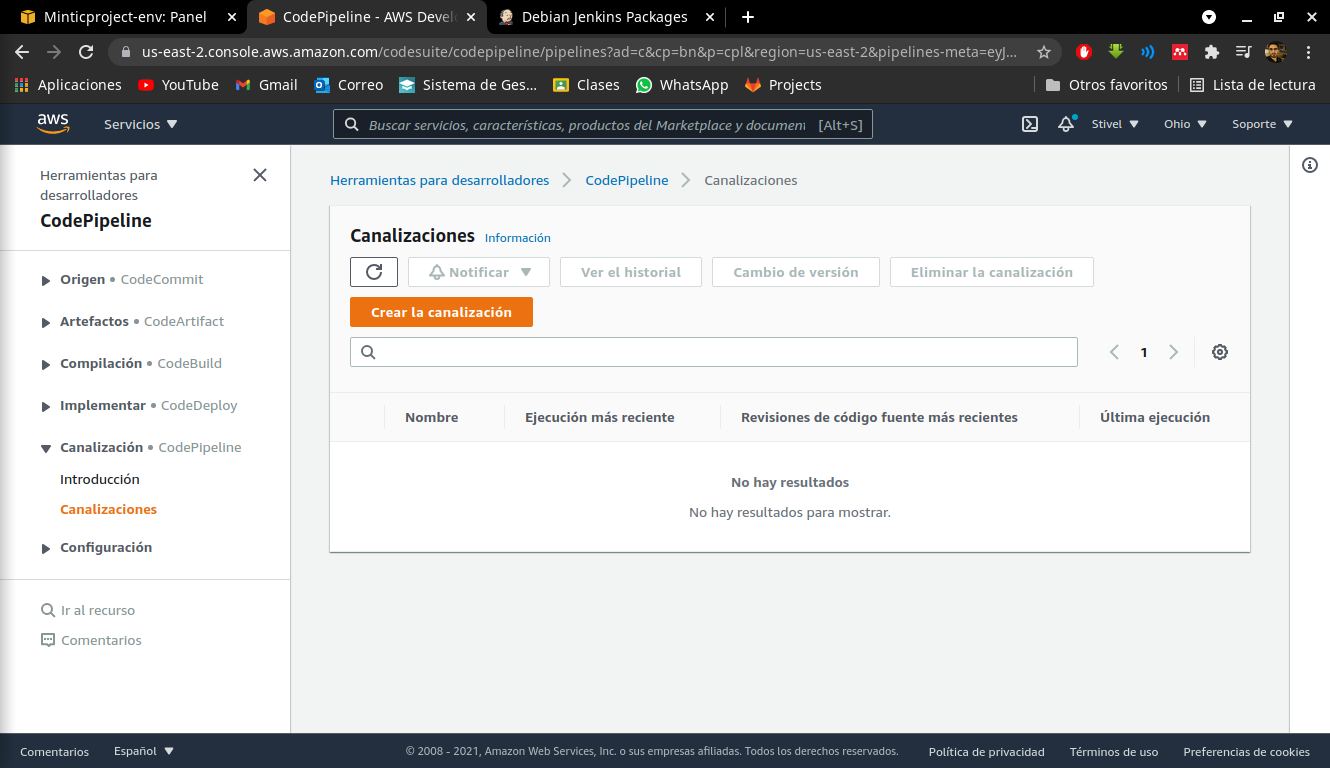
*Figura 15 Configuración de comandos en CodeBuild.*

Después de haber creado nuestro proyecto de compilación (figura 16) nos dirigimos a crear el pipeline (figura 17).

￼

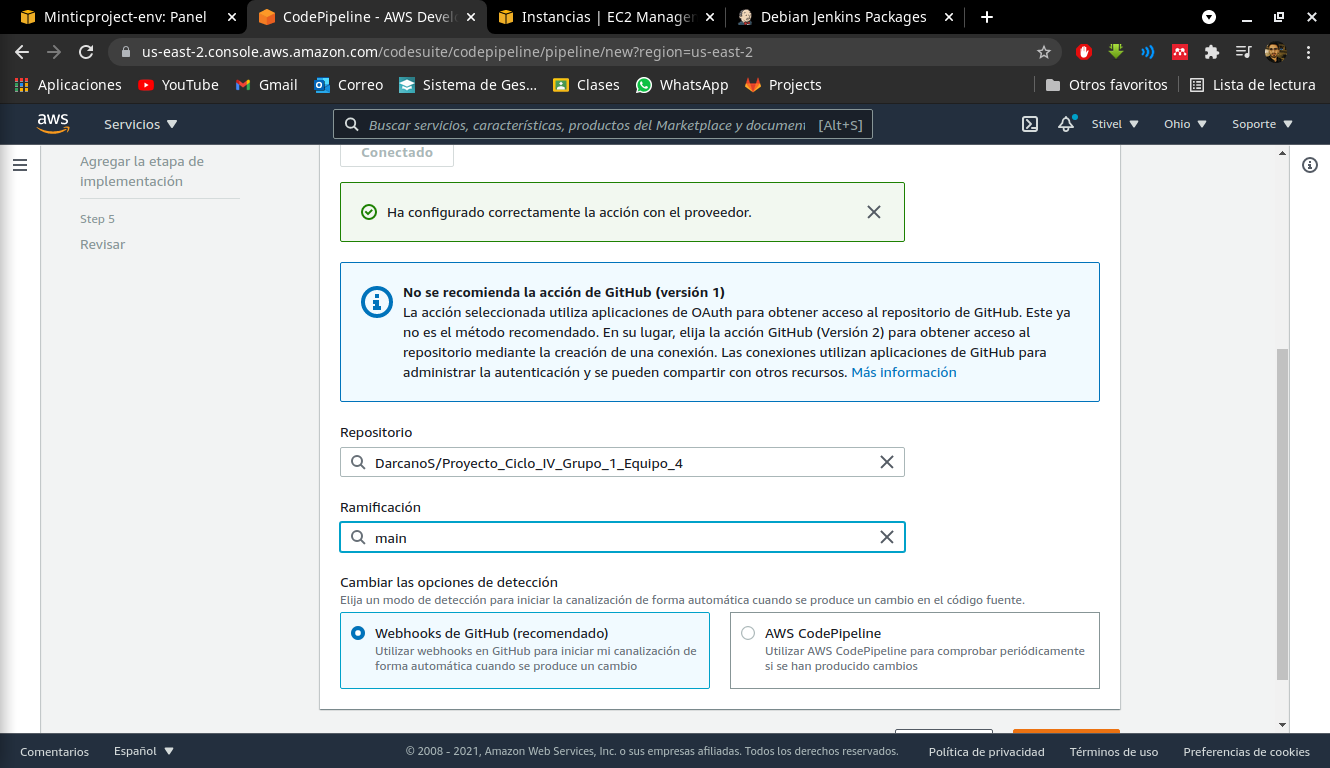


*Figura 16 Proyecto de compilación creado,*

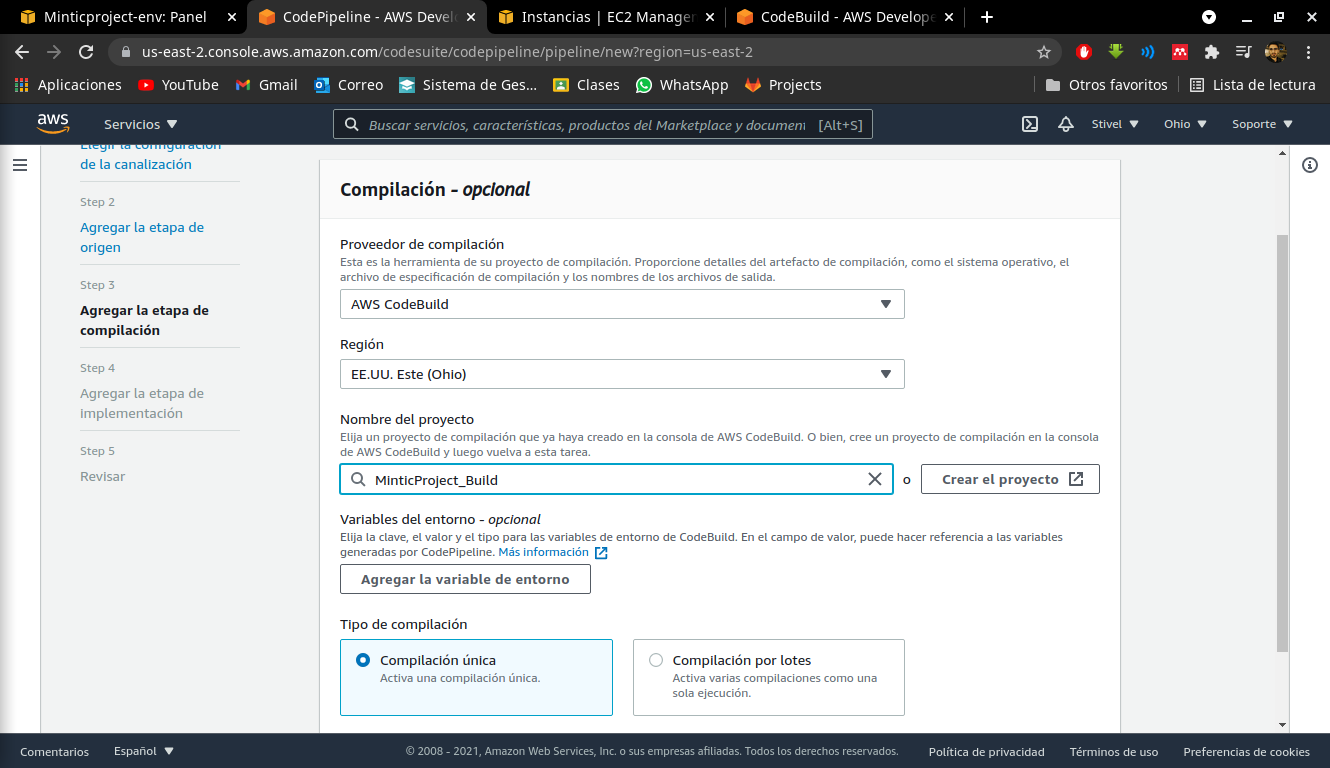


*Figura 17 CodePipeline*

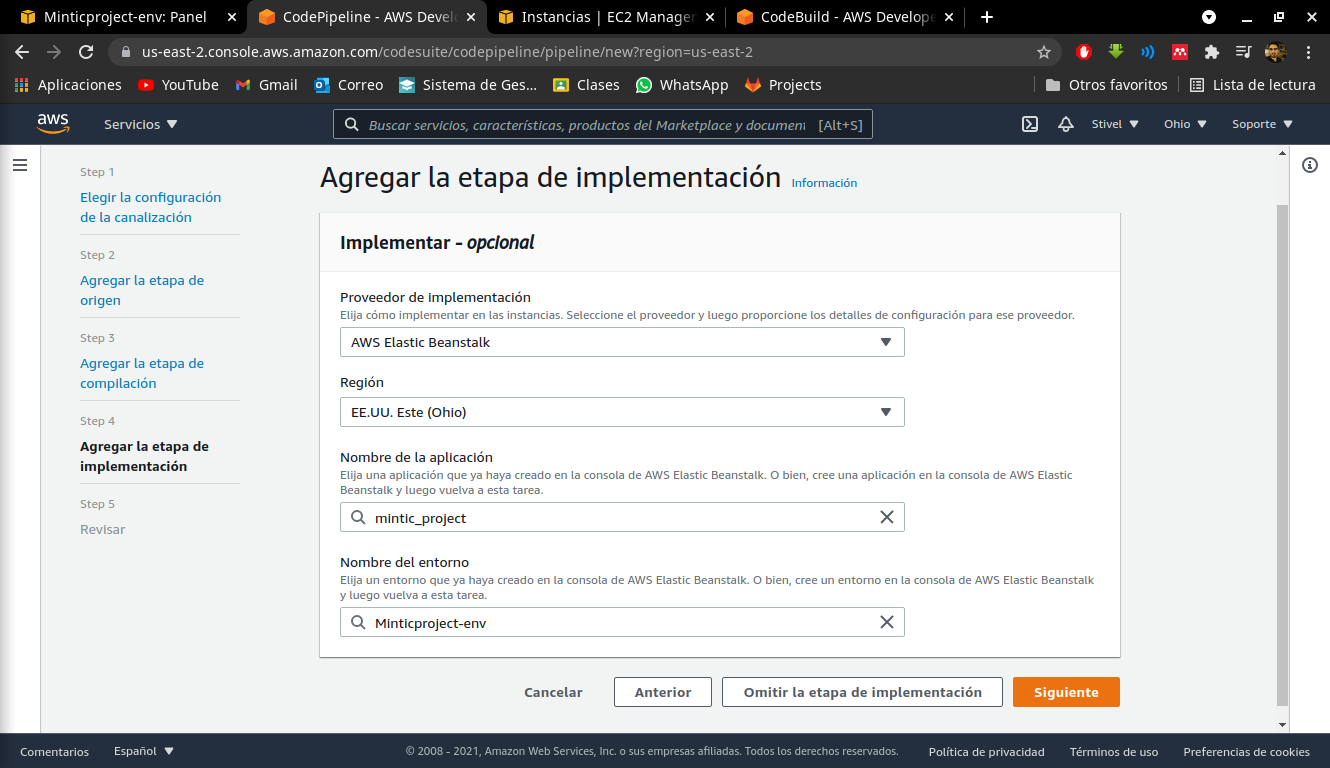
En nuestro pipeline configuramos nuestro repositorio en la rama *main* (figura 18), agregamos nuestro proyecto de compilación (figura 19) y en la etapa de implementación agregamos nuestro Elastic Beanstalk (figura 20).



*Figura 18 Agrega repositorio git al pipeline*

**

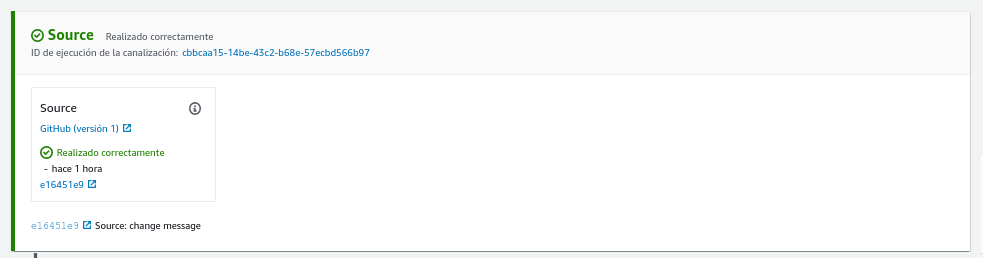
*Figura 19 Agrega el build al pipeline*

**

*Figura 20 Agrega el Elastic Beanstalk al pipeline*

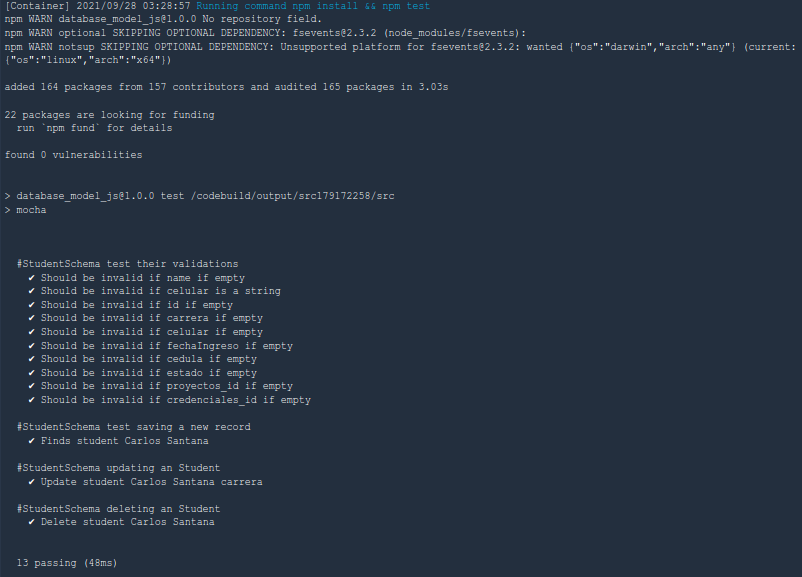
Después de esto se ejecutan todas las etapas Source (figura 21), build (figura 22), donde en el build de ejecutar las pruebas (figura 23), y por último el Deploy (figura 24) que nos da como resultado el despliegue de la página (figura 25) en la dirección:

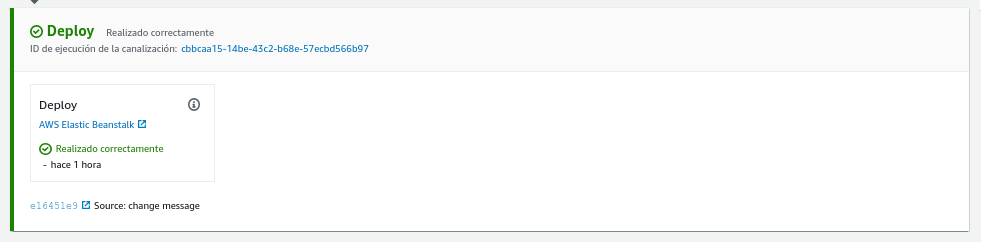
[**http://minticprojectv2-env.eba-mcphvvhq.us-east-2.elasticbeanstalk.com/**](http://minticprojectv2-env.eba-mcphvvhq.us-east-2.elasticbeanstalk.com/)

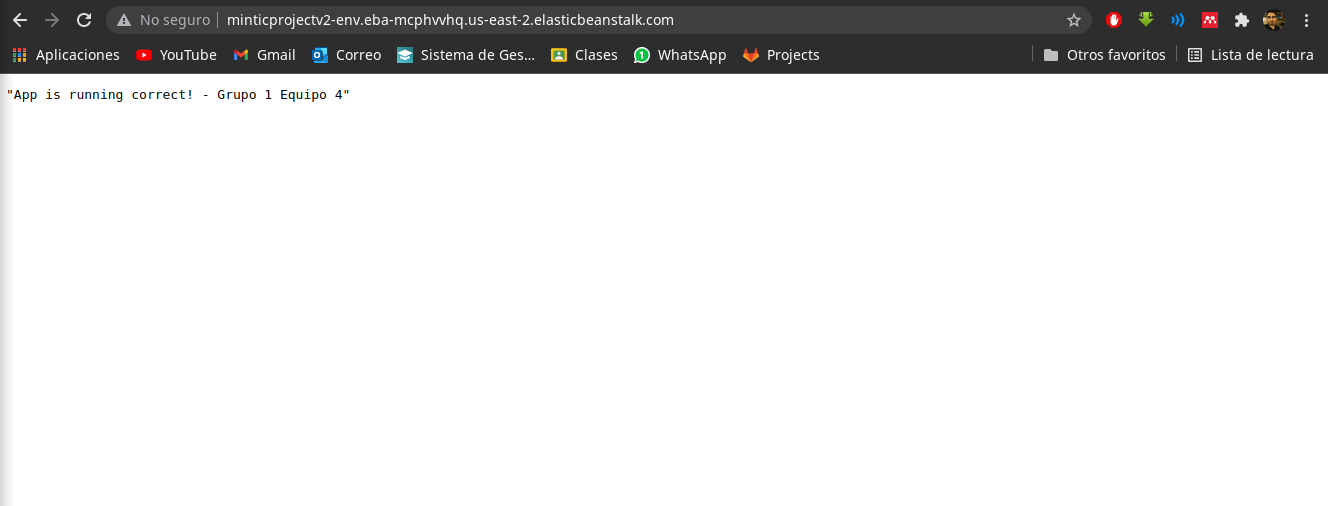


*Figura 21 Stage Source*

*Figura 22 Stage Build*

*Figura 23 Ejecución de pruebas unitarias*

*Figura 24 Stage Deploy*



*Figura 25 Aplicación web desplegada*

**Enlaces**

Trello: <https://trello.com/b/rxlH2t0x/mintic>

GitHub: <https://github.com/DarcanoS/Proyecto_Ciclo_IV_Grupo_1_Equipo_4>

Web Aplication: <http://minticprojectv2-env.eba-mcphvvhq.us-east-2.elasticbeanstalk.com/>