Universidad de León

Trabajo de Fin de Máster

Integración continua DevOps con Google Cloud, Kubernetes, Jenkins y Github

Autor:
Daniel Álvarez Cristóbal

Tutor:
Isaias García Rodríguez

Máster en Ingeniería Informática

Diciembre de 2021

UNIVERSIDAD DE LEÓN
Escuela de Ingenierías I.I.
MÁSTER EN INGENIERÍA INFORMÁTICA
Trabajo de Fin de Máster

ALUMNO: Daniel Álvarez Cristóbal

TUTOR: Isaias García Rodríguez

TÍTULO: Integración continua DevOps con Google Cloud, Kubernetes, Jenkins y Github

CONVOCATORIA: Diciembre, 2021

RESUMEN:

En este proyecto se estudia la aplicación de metodologías DevOps mediante las herramientas Google Cloud Platform, Kubernetes, Jenkins y Github

Palabras clave: DevOps, Computación en nube, Kubernetes, Jenkins.

Firma de la alumna:	**VºBº Tutor:**

Resumen

Este proyecto se ha investigado la metodología ágil de DevOps y cómo se puede implantar mediante computación en la nube y contenedores de Docker inspirado en el modelo presentado en el libro "Pro DevOps with Google Cloud" [1].

En el documento se explicará qué es DevOps y cómo aplicarlo. Además se realizará un ejemplo paso a paso de la implementación de DevOps en un proyecto sencillo. En el ejemplo se usarán las siguientes tecnologías. Google Cloud para servidor en la nube. Kubernetes (y consecuentemente Docker) para la contenerización. Jenkins para la automatización del servidor. Github como repositorio del proyecto.

Palabras clave: DevOps, Computación en nube, Kubernetes, Jenkins.

Glosario

Metodologías ágiles : Conjunto de prácticas orientadas a desarrollar los proyectos de manera dinámica y flexible.

Repositorio de código : Un servidor que almacena todas las versiones del código subido y los cambios entre versiones. Permite revertir el código a un estado anterior.

Automatizador de servidor : Software que permite administrar y automatizar un servidor mediante configuraciones.

IaC (Infraestructura como Código) : Es una estrategia de DevOps que consiste en establecer y administrar infraestructura mediante archivos definitorios que el sistema puede leer en vez de configurarlo directamente mediante hardware o herramientas interactivas [2].

Virtualización : Ejecutar un software huésped sobre un sistema informático simulado por un segundo sistema informático anfitrión. Los dos sistemas no tienen por qué estar conectado.

Índice general

Índice de figuras **6**

Índice de cuadros **12**

1. Introducción **13**
1.1. Objetivos 13
1.2. Organización del documento 14

2. Situación del tema **15**
2.1. DevOps 15
2.1.1. Primeros pasos para implementar DevOps 16
2.1.2. Motivos para adoptar DevOps 17
2.1.3. Requisitos para implementar DevOps 19
2.2. Computación en la nube 21
2.2.1. Tipos de computaciones en la nube 22
2.2.2. GCP (Google Cloud Platform) 25
2.3. Integración y despliegue contínuos 27
2.3.1. Integración continua (CI) 27
2.3.2. Despliegue Continuo (CD) 29
2.4. Implementación de procesos CI/CD 30
2.4.1. Testeo e inspección continuos 32
2.4.2. Diseñando los canales de un sistema de integración y entrega continuos 33
2.4.3. Integración continua de bases de datos 34
2.4.4. Preparando la compilación para el despliegue 35
2.4.5. Desplegar el código 36
2.5. Introduccióna a contenedores (Docker y kubernetes) 38
2.5.1. ¿Cómo Funciona Docker? 39
2.6. Introduccióna a Continuous Delivery with GCP and Jenkins 40

2.6.1. Integración y despliegue contínuo con Jenkins 41

3. Metodología y herramientas 43
3.1. Modelo de desarrollo 43
3.2. Planificación 43
3.3. Herramientas 45
3.3.1. Software 45
3.3.2. Hardware 46

4. Sistema desarrollado 48
4.1. Google Cloud Platform (GCP) 48
4.1.1. Crear una cuenta 48
4.1.2. Crear una organización 50
4.1.3. Crear un proyecto 53
4.1.4. Planes de pago 55
4.1.5. Recursos de GCP 56
4.1.6. SDK de GCP 57
4.2. Instancias de GCP 60
4.2.1. Crear y configurar un proyecto 61
4.2.2. Habilitar la API 64
4.2.3. Crear una instancia de máquina virtual 66
4.2.4. Comandos básicos de instancias 69
4.2.5. Conectar con una instancia 69
4.3. Docker 71
4.3.1. ¿Qué es Docker? 71
4.3.2. Trabajando con Docker 72
4.3.3. Docker en GCP 73
4.3.4. Dockerfile 77
4.4. Kubernetes 79
4.4.1. ¿Qué es Kubernetes? 80
4.4.2. Trabajando con kubernetes 80
4.5. Ejemplo práctico de Kubernetes en GCP 83
4.5.1. Preparando un proyecto para usar Kubernetes 84
4.5.2. Crear un cluster 87
4.6. Aplicación web con Node.js 90
4.6.1. La aplicación 90
4.6.2. Ejecutar el servidor en local 93
4.7. Ejecutar en Kubernetes desde GCP 95

4.7.1. Archivos de configuración 95
4.7.2. Compilar y subir la imagen 98
4.7.3. Publicar la imagen con Kubernetes 100
4.8. Jenkins 102
4.8.1. Instalando Jenkins en local 102
4.8.2. Configuración de Jenkins 103
4.8.3. Actualizar plugins 114
4.8.4. Crear usuarios 114
4.8.5. Crear un canal a GitHub 118
4.8.6. Componentes de un canal 122
4.8.7. Estructura del canal 124
4.8.8. Trabajo desde el punto de vista del usuario 125
4.9. Servidor de Integración Continua 128
4.9.1. Desplegar Jenkins en Kubernetes con GCP 128
4.9.2. Configuración básica de Jenkins 139
4.9.3. CI con Jenkins 144
4.9.4. CD con Jenkins 150

5. Conclusiones y trabajos futuros **151**
5.1. Conclusiones 151
5.2. Trabajos futuros 151
5.3. Agradecimientos 152

Bibliografía **153**

A. Anexo **155**
A.1. AddWebpage 155
A.1.1. AddWebpage.HTML 155
A.1.2. AddScript.js 156
A.1.3. AddScriptTest.js 156
A.1.4. server.js 158
A.2. deploy 158
A.2.1. service.YAML 158
A.2.2. deployment.YAML 159
A.2.3. Ingress.YAML 159
A.2.4. Jenkinsfile 160

Índice de figuras

2.1. Cadena de DevOps 19
2.2. Fases de la integración continua 28
2.3. Funcionamiento de un sistema de CI de la publicación "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices" [3] 28
2.4. Fases del despliegue contínuo 29
2.5. Ejemplo de jenkinsfile básico. Aquí se establece el canal. 34
2.6. Esquema de servidores azules y verdes del libro [1], figura 3-6 37
2.7. Esquema de servidores canario del libro [1], figura 3-7 37
2.8. Esquema de servidores incrementales del libro [1], figura 3-8 38
2.9. Diferencias entre máquinas virtuales y contenedores [4] 39

3.1. Diagrama de Gannt de la planificación original 44
3.2. Diagrama de Gannt de la ejecución final 45

4.2. Pantalla de inicio 49
4.1. Inscripción y ventajas 49
4.3. Lista de comprobacion de primeros pasos 50
4.4. Creación de proyecto 51
4.5. Menú principal 51
4.6. Creación de proyecto 52
4.7. Menú de creación de organización 52
4.8. Menú de creación de proyecto 53
4.9. Seleccionar proyecto 54
4.10. Menú de inicio con un proyecto seleccionado 54
4.11. Menú de facturación 55
4.12. Conexión con el SDK 58
4.13. Permisos del SDK 59
4.14. Selección de proyecto 59

4.15. Resultado del comando **gcloud compute -h** 60
4.16. Configuración de servidor 61
4.17. Selección de proyecto 61
4.18. Con proyecto 62
4.19. Sin proyecto 62
4.20. Elegir región del proyecto 63
4.21. Uso de **gcloud config set compute/zone <Nombre del servidor>** 63
4.22. Uso de **gcloud config set project <ID del proyecto>** 63
4.23. Mensaje de error por no tener la API habilitada 64
4.24. dirección de la URL 65
4.25. Advertencia de facturación de la API 66
4.26. **gcloud gcloud compute images list** 67
4.27. **gcloud compute images list –filter cos-cloud** 67
4.28. **gcloud compute instances create instanciacos –image-project cos-cloud –image-family cos-stable** 68
4.29. Primera ejecución de **gcloud compute ssh instanciacos** 70
4.30. Consola de comandos de la instancia de COS 70
4.31. **gcloud compute ssh** 73
4.32. **docker –version** 74
4.33. **docker search ubuntu** 74
4.34. **docker pull ubuntu** 75
4.35. **docker run ubuntu** 75
4.36. Borrando un contenedor mediante **docker container rm <ID>** . . 75
4.37. No se pueden usar contenedores en segundo plano en GCP 76
4.38. Exportación de contenedor. 76
4.39. Ejemplo de Dockerfile 79
4.40. Ejemplo de archivo YAML definiendo un servicio. 81
4.41. Establecer cuotas de un servicio en un archivo YAML. 82
4.42. Estructura básica de un archivo de despliegue en un archivo YAML. . 83
4.43. Configuración del nuevo proyecto 84
4.44. **gcloud components install kubectl** 87
4.45. **kubectl version** 87
4.46. **gcloud container clusters create pruebacluster** 88
4.47. **gcloud container clusters get-credentials pruebacluster** . . . 88
4.48. **kubectl create deployment hello-server –image=gcr.io/google-samples/hello-app:1.0** 89
4.49. **kubectl delete deployment hello-server** 89

4.50. **kubectl expose deployment hello-server –type=LoadBalancer –port=8080** . . . 89
4.51. **kubectl get service hello-server** . . . 89
4.52. Muestra del servidor funcionando desde otro equipo. . . . 90
4.53. Esquema de la estructura de aplicaciones de la aplicación web "Add-Webpage" . . . 91
4.54. Código de Addwebpage.html . . . 91
4.55. Aspecto de Addwebpage.html . . . 92
4.56. Código de server.js . . . 93
4.57. **npm test** . . . 94
4.58. **npm test** . . . 94
4.59. Localhost 8081 . . . 95
4.60. service.YAML . . . 96
4.61. deployment.YAML . . . 97
4.62. Ingress.YAML . . . 98
4.63. Crear servidor en kubernetes . . . 99
4.64. Acceso a localhost 8081 . . . 100
4.65. Puerto 8081 . . . 100
4.66. **docker push eu.gcr.io/$PROJECT_ID/addwebpage:latest** . 100
4.67. Despliegue de un servicio de Kubernetes . . . 101
4.68. Escalado del servicio de Kubernetes . . . 101
4.69. Servidor desplegado desde otro navegador . . . 102
4.70. Menú de administración de Jenkins . . . 104
4.71. Menú de plugins de Jenkins . . . 105
4.72. Instalación de plugins en Jenkins, parte I . . . 105
4.73. Instalación de plugins en Jenkins, parte II . . . 106
4.74. Reinicio de Jenkins . . . 106
4.75. Blue Ocean en plugins instalados . . . 106
4.76. Blue Ocean en el menú principal de Jenkins . . . 107
4.77. Plugin de Node.js . . . 107
4.78. Configuración de Node.js . . . 108
4.79. Plugin de Mail extension . . . 108
4.80. Creación de servidor STMP . . . 109
4.81. Menú de opciones de seguridad de la cuenta de Google . . . 109
4.82. Verificación en dos pasos . . . 110
4.83. Creación de contraseña de app . . . 110
4.84. Establecer e-mail de administrador. . . . 111

4.85. Configuración del correo, parte I . . . 112
4.86. Configuración del correo, parte II . . . 112
4.87. Configuración del correo, parte III . . . 113
4.88. Configuración del correo en el jenkinsfile . . . 113
4.89. Actualizar plugins, parte I . . . 114
4.90. Actualizar plugins, parte II . . . 114
4.91. Crear usuarios, parte I . . . 115
4.92. Crear usuarios, parte I . . . 115
4.93. Crear usuarios, parte II . . . 116
4.94. Crear usuarios, parte III . . . 116
4.95. Vinculación de correo a git . . . 116
4.96. Menú de configuración de GitHub . . . 117
4.97. Invitación de participación de GitHub aceptada . . . 118
4.98. Creación de Jenkinsfile por BlueOcean, Parte I . . . 119
4.99. Creación de Jenkinsfile por BlueOcean, Parte II . . . 119
4.100Creación de Jenkinsfile por BlueOcean, Parte III . . . 120
4.101Creación de Jenkinsfile por BlueOcean, Parte IV . . . 120
4.102Creación de Jenkinsfile por BlueOcean, Parte V . . . 121
4.103Creación de Jenkinsfile por BlueOcean, Parte VI . . . 121
4.104Creación de Jenkinsfile por BlueOcean, Parte VII . . . 121
4.105Creación de Jenkinsfile por BlueOcean, Parte VIII . . . 122
4.106Creación de Jenkinsfile por BlueOcean, Parte IX . . . 122
4.107Ejemplo de Jenkinsfile básico . . . 124
4.108Agente de Jenkins en el canal . . . 125
4.109Lista de ramas del repositorio . . . 126
4.110Lista de compilaciones . . . 126
4.111Compilación exitosa . . . 127
4.112Compilación errónea . . . 128
4.113Esquema de servidor de CI/CD en Jenkins . . . 130
4.114Archivo de configuración de ejemplo, Parte I . . . 131
4.115Archivo de configuración de ejemplo, Parte II . . . 131
4.116Archivo de configuración de ejemplo, Parte III . . . 132
4.117Archivo de configuración de ejemplo, Parte IV . . . 132
4.118Jenkins UI Service . . . 133
4.119Jenkins Discovery Service . . . 134

4.120.**gcloud container clusters create jenkins-devops –zone europe-west1-b –num-nodes 2 –scopes "https://www.googleapis.com/auth/source.read_ platform"** 135
4.121. **kubectl cluster-info** 135
4.122.**wget https://get.helm.sh/helm-v3.2.1-linux-amd64.tar.gz || tar zxfv helm-v3.2.1-linux-amd64.tar.gz || cp linux-amd64/helm .** 136
4.123. **kubectl create clusterrolebinding cluster-admin-binding –clusterrole=cluster-admin –user=$(gcloud config get-value account)** 136
4.124.**./helm repo add jenkinsci https://charts.jenkins.io || ./helm repo update** 136
4.125.**./helm install cd-jenkins -f jenkins/services_jenkins.yaml jenkinsci/jenkins –wait || kubectl get pods** 137
4.126. **export POD_NAME=$(kubectl get pods –namespace default -l "app.kubernetes.io/component=jenkins-master" -l "app.kubernetes.io/in -o jsonpath="{.items[0].metadata.name}") || kubectl port-forward $POD_NAME 8080:8080 » /dev/null kubectl get svc** 137
4.127.**printf $(kubectl get secret jenkins -o jsonpath="{.data.jenkins-admin-password}" | base64 –decode);echo** 138
4.128Conexión a Jenkins mediante el puerto 8080 de GCP 138
4.129LogIn de Jenkins 139
4.130Menú de Jenkins 139
4.131Diferencia en ejecutores de local a GCP 140
4.132Jenkinsfile, Parte I 141
4.133Jenkinsfile, Parte II 142
4.134Jenkinsfile, Parte III 142
4.135Jenkinsfile, Parte IV 143
4.136Jenkinsfile, Parte V 143
4.137Panel de control de Jenkins 144
4.138Configuración del panel de control de Jenkins 145
4.139Configuración del panel de control de Jenkins 145
4.140Resultado de compilación 146
4.141Detalles de la compilación 146
4.142Correo de compilación exitosa 147
4.143Fallo inducido en el código 147

4.144Cola de trabajos de Jenkins 148
4.145Compilación errónea 148
4.146Log de la compilación errónea 149
4.147Correo de compilación errónea 149

Índice de cuadros

2.1. Ventajas y desventajas de la contratación de un servicio de nube externo 25
2.2. Etapas de un sistema CI/CD 42

4.1. Caption 111

Capítulo 1

Introducción

En este proyecto se explican los principios de implementación de la metodología DevOps mediante un ejemplo práctico. Primero se expone qué es DevOps, cómo funciona y qué es necesario para implementarse en una empresa o proyecto. Después se repasan las tecnologías que se van a usar. Por último se muestra un

El objetivo de este proyecto es presentar la metodología DevOps y las bases de cómo implementarla. La primera parte es una explicación de qué es DevOps, cómo funciona y qué es necesario para implementarse en un equipo de desarrollo. La segunda parte es un ejemplo práctico que permitirá alterar el código de una aplicación web en tiempo real (sin recargar la página).

La principal fuente de información para entender DevOps ha sido el libro "Pro DevOps with Google Cloud" [1]. Para la parte más práctica este proyecto ha habido influencia del mismo libro pero se ha apoyado principalmente en la documentación oficial de Google Cloud [5], Kubernetes [6], Docker [7] y Jenkins [8].

1.1. Objetivos

Para este proyecto se marcaron los siguientes objetivos:

- Investigar qué es DevOps y qué características tiene.
- Crear un ejemplo como el del libro y ponerlo en práctica con una aplicación web sencilla.
- Explicar paso a paso para que sirva como introducción a esta metodología.

1.2. Organización del documento

Esta memoria se ha dividido como se detalla a continuación:

En el capítulo 2 se explica qué es DevOps y cómo se diferencia de las metodologías ágiles convencionales. Qué es la computación en la nube y por qué es beneficiosa. Que es la contenerización y cómo se diferencia de la virtualización convencional.

En el capítulo 3 se trata la metodología utilizada para el desarrollo del software y las tecnologías que se han empleado.

El capítulo 4 se muestra una explicación detallada de cómo usar cada tecnología investigada en este proyecto. Cada apartado asume que se ha leído los anteriores.

Por último, en el capítulo 5 se recogen las conclusiones y las ideas que se han tomado para trabajos futuros.

Capítulo 2

Situación del tema

En esta sección se revisará y explicarán las tecnologías y técnicas básicas para este proyecto.

2.1. DevOps

El término "Dev Ops" apareció en 2008 en una conferencia de metodologías Ágiles en Toronto, Canadá y se volvió popular en 2014 con el libro "Phoenix Project", una novela donde los autores explican el procedimiento DevOps. Es una filosofía de trabajo para acercar el desarrollo de aplicaciones y sistemas a su uso, reduciendo el tiempo de los proyectos y evitando fallos. Debido a su éxito ha surgido la figura de los ingenieros DevOps que funcionan como puente entre el desarrollo y operaciones.

Un ingeniero DevOps debe conocer los campos de desarrollo y dirección. Algunas de sus ocupaciones son la continua integración del producto y la supervisión de las infraestructuras, normalmente de código (IoC). Las ocupaciones de un ingeniero DevOps difiere en cada empresa, en general su labor es asegurar que el paso de desarrollo a implementación se integre de manera correcta y eficiente.

DevOps sólo funciona en metodologías ágiles. Una de las estructuras más básicas consiste en tres equipos, un jefe de proyecto y un representante del cliente. Los equipos son el de desarrollo, que se encarga de escribir el código; control de calidad, que se encarga de testar que el código pasa todos los requisitos de calidad y buscan errores; y despliegue, que se encarga de integrar el proyecto y desplegarlo. El jefe de proyecto coordina al grupo. El representante del cliente explica las necesidades del proyecto al equipo.

Uno de los procesos de desarrollo más habituales es una iteración continua. El proceso de desarrollo comienza con la planificación del proyecto por parte del cliente y el jefe de proyecto. El arquitecto software planifica la estructura del producto. Si no es la primera iteración, el arquitecto planifica cómo implementar los cambios sin alterar la estructura existente o adaptándola. El equipo de desarrollo escribe el código probando que es funcional. Cada periodo de tiempo se integra el código y este es testado por el equipo de control de calidad para analizar en detalle que funciona apropiadamente y cumple los requisitos de calidad. Con el código listo, este se pasa al equipo de despliegue para que lance la aplicación. Las actualizaciones sólo se hacen de manera periódica, unificando las actualizaciones en grandes parches. El código desplegado se monitoriza para analizar su funcionamiento. Salvo que el proyecto haya acabado, se vuelve al primer paso para buscar

2.1.1. Primeros pasos para implementar DevOps

Los directivos deben promover el cambio. Se necesita un fuerte apoyo por parte de la empresa y también es importante predicar con el ejemplo. Si no se apoya suficientemente o se implementa mal, entonces los trabajadores perderán la confianza en la metodología y poco a poco volverán a hacer las cosas como las hacían antes.

Las prácticas ágiles se deben promover en toda la compañía. DevOps es una práctica de las tecnologías ágiles. Por tanto funciona más eficientemente si todo el sistema usa el mismo tipo de planificación, evitando cuellos de botellas y problemas de comunicaciones. Si todos usan tecnologías ágiles, eso también ayudará a que los trabajadores puedan ver el trabajo de otros departamentos. La transparencia entre departamentos es positiva porque permite ver la efectividad del sistema y preparar mejor las iteraciones.

Se deben eliminar los obstáculos en la comunicación entre departamentos de TI. Generalmente los equipos de desarrollo y operaciones usan distintos software. Esto puede ocasionar incompatibilidades y dificultades para implementar DevOps. Se deben unificar las herramientas de comunicación para hacer más rápidas las comunicaciones. Así el equipo de desarrollo conoce antes la existencia de bugs y el equipo de operaciones sabe cuánto queda para el próximo lanzamiento.

El desarrollador debe ser responsable del software. Normalmente los desarrolladores son responsables del software hasta que sale en vivo. Después sólo se ocupan de actualizaciones mayores y algunos bugs, pero del grueso se ocupa el equipo de operaciones. Es más ventajoso que se ocupen los propios desarrolladores porque es más

fácil para ellos encontrar la raíz de los problemas y aplicar una solución permanente (en vez de parchear los fallos)

Hay que tratar al equipo de operaciones con respeto. Cuando se realizan cambios en el código, se tiene que incluir al equipo de operaciones, porque son los que más información tienen del uso en vivo del programa. Esto puede prevenir errores de despliegue.

Se debe crear una política de integración e implementación continua. Estas políticas hacen que el tiempo desde que el código es desplegado hasta que es implementado sea mínimo Esto ayuda a que el equipo de operaciones encuentre antes el problema y que el equipo de desarrollo lo solucione antes. Esto es especialmente crítico en vulnerabilidades de seguridad y similares. Para que esto funcione se deben dedicar muchos recursos en tiempo y personal en hacer suficientes test unitarios y automáticos. En caso contrario es probable que se acabe implementando código dañino, inseguro o peligroso.

El proceso de lanzamiento debe ser automático. Según los datos de [1] la principal causa de fallos es el error humano. Para reducir este tipo de fallos se debe automatizar el proceso de lanzamiento. Así se reduce la cantidad de interacción de personas y se obtiene un proceso uniforme y repetible. Además se puede obtener un modelo de IaC (Infrastructure as Code) para definir la estructura del software.

2.1.2. Motivos para adoptar DevOps

Es cierto que adoptar DevOps, como otras metodologías ágiles, significa un cambio general del funcionamiento de la empresa. Este es un cambio difícil, pero aporta muchos beneficios si se implementa apropiadamente.

El principal motivo es para mejorar la calidad del software y optimizar los procesos de lanzamiento. También mejora la comunicación entre departamentos. Al adoptar DevOps se mejora la calidad, al disponer de un sistema más procedural y planificado que con otras metodologías. Permite detectar antes fallos y hacer una integración continua y dinámica. Y también se consiguen procesos de lanzamiento más óptimos porque son más rápidos y automatizados. Esto reduce el tiempo de lanzamiento al mercado de los productos a la vez que permite detectar y solventar errores en tiempo real sin dejar de dar servicio.

Para adaptar DevOps es importante saber a qué y quiénes les va a afectar y cómo. Un mito sobre DevOps es que sólo afecta a ingenieros de software, ingenieros de sistemas o administradores de sistemas. Esto es erróneo, ya que cambia la forma

en la que los equipos interactúan entre sí. El primer paso de DevOps es mejorar las comunicaciones entre equipos, por tanto afecta a todos los equipos, no sólo desarrollo y operaciones. Para que funcione DevOps se necesita que todos los equipos trabajen juntos.

Una de las metas al adoptar DevOps es reducir la coordinación necesaria. Liberando tiempo y esfuerzo a los responsables del proyecto. Esto es importante cuando se pasa una aplicación del servidor de pruebas, porque no necesita pedir permiso una vez está listo. Cada persona que tenga que dar el visto bueno es un retraso en el lanzamiento.

En el desarrollo clásico son los propios integrantes de los equipos quienes se comunican una vez acaban. DevOps también asume la responsabilidad de esas comunicaciones, reduciendo otra vez las interacciones humanas prescindibles que puedan provocar retrasos en un proyecto (Por ejemplo, no ver un e-mail)

Principalmente hay tres estilos de coordinación: Directo, Indirecto y persistente. Cada estilo tiene sus ventajas y desventajas, por ello hay que elegir según las necesidades de la empresa.

Coordinación directa : Hay unos responsables que coordinan a los trabajadores. Se conocen entre ellos y a los coordinados. Con este método los responsables pueden recibir feedback y tomar decisiones durante cada reunión. El problema es que este método requiere más esfuerzo y recursos, aunque a veces se puede mitigar mediante metodologías como scrum.

Coordinación indirecta : Hay unos responsables que coordinan a unos equipos y estos se coordinan sólos. Esta coordinación conlleva menos detalle y da una imagen más general, por tanto necesitando menos recursos que una coordinación directa. Necesita pocos recursos de coordinación y se puede dejar en manos del product owner. El problema es que requiere una mejor coordinación porque las instrucciones son dadas a nivel general sin detalles.

Coordinación persistente : Este tipo de coordinación se basa en acceder a toda la comunicación sobre las decisiones tomadas, para que los equipos puedan entender el contexto y evitar fallos de comunicación. Este estilo evita fallos de comunicación. El problema es que en proyectos grandes o con restricciones de información podría no ser posible.

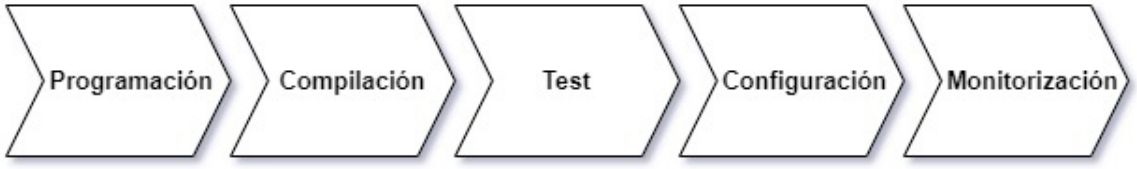


Figura 2.1: Cadena de DevOps

2.1.3. Requisitos para implementar DevOps

Las estrategias DevOps establecen procesos de producción robustos. El eje principal es la llamada "Cadena de DevOps" que establece el proceso de producción. La cadena se compone de las siguientes fases:

Programación : Los programadores crean un código que suben a un repositorio común como Git.

Compilación : Cuando se hace un commit del código, en el servidor de compilar se compila automáticamente y se pasan test en los cambios.

Test :Si se han pasado los anteriores tests, se realizarán más tests automáticos pero de toda la aplicación en general.

Configuración : Esta fase se salta cuando hay una metodología DevOps bien establecida, aunque también se usa en lanzamientos sensibles. En vez de lanzar la aplicación directamente, se lanza antes en unos "servidores canarios" que son servidores usados por usuarios reales pero una versión por delante del resto, de tal forma de que si hay un fallo sólo afecte a una pequeña parte de los clientes y no a todos.

Lanzamiento : En esta fase los servidores se configuran para usar el nuevo código y define la IaC (Infrastructure as Code).

Monitorización : Se mantiene un continuo feedback en busca del software y arquitectura para conseguir estadísticas que permitan mejorar el producto final y para buscar posibles bugs o fallos que hayan escapado a los tests.

Para mantener esta cadena en movimiento es importante una buena comunicación y coordinación. Por este motivo las metodologías DevOps ponen tanto énfasis en optimizar la interacción entre los trabajadores y se consideran imprescindibles las herramientas y metodologías que reducen o facilitan la interacción necesaria.

También es necesario establecer un canal de desarrollo efectivo. El canal de desarrollo es la implementación práctica de la cadena de DevOps. Es de gran ayuda

disponer de un software de integración continua como Jenkins. Es importante definir el canal de desarrollo porque va a ser la base de los cambios realizados por DevOps. Las partes a definir en el canal son:

Tests unitarios : Cada vez que se hace un commit del código a la plataforma, este pasa por varios tests unitarios. En caso de fallo, se le enviará un correo al programador. Es recomendable usar desarrollo dirigido por tests directamente donde primero se diseñan los tests y luego el propio código.

Política de branch :Siempre se tiene definido en git una única rama principal con el software listo para funcionar (aunque esté incompleto). El resto de ramas deben pasar por los tests unitarios y realizar una revisión del código para unirla a la rama principal.

Sistema de integración continua :Se debe mantener un buen sistema de comunicación, concretamente el apartado de mensajería electrónica. Así los desarrolladores podrán dar el visto bueno a los cambios más rápidamente y lanzar antes el producto.

Es crucial centralizar el servidor de compilación, incluidos los de desarrollo y despliegue, para minimizar tiempo y errores por traspaso de servidor. Usar servidores distintos para compilar código puede incurrir en errores difíciles de detectar. También hay que tener en cuenta que para mantener varias versiones de un código en un mismo servidor, es obligatorio seguir una estricta política de nombres para evitar perder información. El uso de Docker permite tener en un mismo servidor varias versiones del mismo servidor, lo que ayuda a las tareas de QA.

El lanzamiento automatizado de un software ocurre después de la producción tras realizar los tests de servidor y de QA (Control de calidad). La monitorización comienza desde que sale de la etapa de QA. Es importante para mantener el sistema estable y encontrar bugs y fallos antes de que sean un problema. Es recomendable usar algún sistema de monitorización, específicamente uno que genere un log que permita ver las razones de los fallos.

La monitorización se debe combinar con otras prácticas para ser efectivo. Es necesario tratar la información en tiempo real, no sólo recolectarla. Una de las prácticas más importantes para la corrección de errores es el análisis de un registro. El software más habitual para esta tarea es ELK (Elasticsearch, Logstash, and Kibana).

Un buen software de análisis del registro puede mejorar la calidad del código, en especial si indica dónde están los errores y no sólo el número. Las representaciones

gráficas ayudan a entender mejor y más rápido los problemas. DevOps se apoya completamente en la monitorización. Se necesita un buen sistema de monitorización para alcanzar un desarrollo adecuado del proyecto. La monitorización se puede dividir en caja negra y caja blanca. En la monitorización de caja negra se prueban fragmentos internos del código, como uso de la CPU o el funcionamiento de un fragmento del código. En la monitorización de caja blanca se prueba la interacción externa del código, como los mensajes HTTP que mande o querys de SQL que pida.

El equipo de operaciones tiene una gran responsabilidad en el resultado final del proyecto. Es el primero que interactúa con el cliente cuando hay un error. Suelen ser los primeros en enterarse de los errores, pero su comunicación con el equipo de desarrollo suele limitarse a los logs. Por ello es importante establecer un canal de comunicación entre operaciones y desarrollo. Tres formas de conseguir esto son:

1. Tener un empleado trabajando en ambos departamentos a la vez.
2. Tener a un ingeniero software en el equipo de operaciones para encontrar e identificar errores más rápido.
3. Que un ingeniero de operaciones redacte una documentación para solucionar los problemas más comunes. Esta documentación es más efectiva si es revisada por otro miembro de operaciones sin conocimiento del funcionamiento interno del sistema.

2.2. Computación en la nube

Las tecnologías basadas en la nube son muy populares ya que permiten ofrecer un servicio de manera telemática y sin necesidad de una instalación física, además de ofrecer servicios modulares y ajustados a las necesidades.

Los servicios de computación en la nube son aquellos en los que se alquila un servidor a una empresa externa, accediendo de manera telemática. Estos servidores pueden servir tanto para para pequeñas operaciones puntuales como para sustituir los servidores locales.

Para hacerlo rentable los proveedores de servicios de computación ofrecen un servidor virtual dentro de un centro de computación de gran potencia. No todos los usuarios usan el servicio a la vez, lo que significa que pueden abastecer a más usuarios con menos potencia. Incluso pueden apagar ciertas partes del servidor en horas de menor uso para ahorrar.

Los tres servicios de nube más populares son Amazon Web Services (AWS), Microsoft Azure y Google Cloud Platform(GCP) [9]. En este apartado se realiza una breve presentación a los servicios de nube y de GCP concretamente.

2.2.1. Tipos de computaciones en la nube

La computación en nube es un paradigma IT de la última década [10]. La computación en nube es un recurso compartido mediante una red informática con las siguientes características:

Auto-servicio y a demanda : Cualquier usuario puede acceder por sí mismo a lo que necesite y pagar sólo cuando se use.

Acceso desde red : La computación en nube tiene una distribución en una red (privada o pública) y los usuarios pueden acceder a los diversos recursos. Normalmente también se puede acceder desde distintos tipos de dispositivos.

Agrupación de recursos : Los usuarios pueden usar un espacio virtual común compartiendo los recursos de manera simultánea. También pueden guardar sus configuraciones individualmente.

Respuesta a cambios más rápida : Al usar la nube los lanzamientos de versiones son prácticamente instantáneos y sin depender de que los usuarios actualicen sus versiones.

Servicio medible : Todo sistema en la nube debe permitir al usuario administrar y revisar cualquier servicio contratado. Generalmente el usuario paga sólo por lo que usa.

Tipo de servicio y despliegue : Cada sistema en la nube tiene un modelo de negocio y uso dependiendo de su público objetivo.

El modelo de servicio de la nube es lo que más diferencia un servicio de otro. Los tres principales son:

SaaS (Software como Servicio) : Se ofrece el uso de un software en línea, pero el cliente no tiene acceso a la infraestructura como las redes o servidores. Generalmente el cliente puede acceder desde cualquier terminal con acceso a internet. (Ejemplos: Google Docs, Office 365 o ZenDesk)

PaaS (Plataforma como Servicio) : Se ofrece un espacio para desarrollar el software al cliente, sin la carga de lidiar con la infraestructura. El cliente puede

importar sus librerías, software, bases de datos o cualquier recurso que necesite. (Ejemplo: Heroku)

IaaS (Infraestructura como servicio) : Es similar a alquilar un ordenador online. El cliente puede usar un servicio estableciendo su infraestructura como el sistema operativo, aplicaciones y todo lo necesario, sin conocer el hardware utilizado. Esta opción es la más flexible y permite al cliente establecer incluso firewalls propios. (Ejemplos: GCP, AWS y Azure)

El modelo de despliegue es cómo se va a desplegar la nube, pero no el tipo de nube. Todos los modelos de servicio pueden usar todos los modelos de despliegue.

Privado : El uso de la nube es sólo para uso interno. La nube está en propiedad y es administrada por una unidad de la empresa.

Comunitario : Cualquiera que forme parte de una comunidad puede acceder a la nube.

Público : Está abierta a cualquiera con acceso a internet. Es habitual que se cobre un alquiler por los servicios.

Mixto : Combina varios tipos. Normalmente es una combinación con el modelo público para ofrecer a la población general algún dato o servicio. También es común que sea pública, pero una parte esté cortada por motivos de privacidad.

Contratar servicios de nube tiene varios beneficios, pero también desventajas respecto a usar un servidor privado. En la siguiente tabla se analizan las ventajas y desventajas de contratar un servicio en la nube frente a establecer un servidor propio.

Ventajas	**Desventajas**
Es la opción más accesible para empresas nuevas, porque se pueden alquilar los servicios de unos servidores sin la elevada inversión inicial de los servidores, su infraestructura necesaria y su mantenimiento.	Si una empresa se puede permitir la inversión inicial y va a tener proyectos lo suficientemente longevos como para amortizar los servidores, es más barato a la larga un servidor local.

Es una forma eficiente para que una empresa pueda ofrecer servicios al público abierto y ajustarse a la demanda, contratando más o prescindiendo de los servicios según se necesite.	Se depende aún más de una conexión en red con el exterior. Con este modelo de servicio sería imposible trabajar únicamente con una conexión local desde dentro de la empresa. Muchas empresas ya dependen de la conexión de la red externa para desplegar sus servicios, pero esto agrava las consecuencias del riesgo de que se corte la conexión a internet (o a la empresa proveedora).
Las averías, costes y mantenimiento van de parte de la empresa contratada, por lo que es más sencillo de administrar. A la hora de calcular gastos no se tienen que calcular los imprevistos relacionados al servidor. Lo más importante es que mitiga uno de los riesgos más preocupantes en una empresa que dependa de su servidor: Que el hardware del servidor caiga y haya que comprar uno nuevo. De esta forma si el servidor se avería, la empresa no tiene que pagar uno nuevo ni queda paralizada esperando a instalar un nuevo hardware. Esto no evita otros problemas derivados de la caída del servidor como pérdida de datos y la interrupción temporal del servicio.	Requiere estar pendiente de una tecnología más para mantener el servidor. Esto significa que al menos un empleado debe estar pendiente de los cambios y actualizaciones en la licencia, funcionamiento y precios del servicio. Esto puede suponer más carga a los administradores del servidor.
El Coste Total de Operación (TCO) es más bajo que adquirir y mantener la infraestructura en proyectos de baja o mediana duración. Esto se debe a que es un alquiler por periodo de tiempo y si se necesita reducir el coste, se pueden recortar los servicios fácilmente en el siguiente periodo de contratación.	Se depende de otra compañía para mantener el servidor de la empresa. Este problema añade un pequeño riesgo pero con grandes repercusiones. Si hay problemas en la empresa que ofrece el servicio, puede afectar al servicio de nube y por tanto al servidor.

Al poder incrementar rápida y fácilmente la infraestructura de la empresa, el Retorno de la Inversión (ROI) es mayor. Esto se debe a que no lleva una inversión inicial adicional al coste de mantenimiento cada vez que se amplía el servicio contratado.	

Tabla 2.1: Ventajas y desventajas de la contratación de un servicio de nube externo

En general, para empresas pequeñas o que están comenzando, los beneficios superan los inconvenientes de esta tecnología. Para empresas medianas que puedan permitirse su propio servidor, los beneficios e inconvenientes están más ajustados y sería recomendable según el caso específico. Para las grandes empresas no es rentable contratar este tipo de servicios porque los beneficios de las grandes empresas suelen venir de optimizar al máximo sus procesos y alquilar no es óptimo a largo plazo, además de que es más sensible depender de otras empresas.

Por otro lado, a las grandes empresas sí les interesa desarrollar estas tecnologías para uso propio. Esto cambia radicalmente las ventajas y desventajas, haciéndolo una opción más interesante para quien pueda costear el precio y mantenimiento. Usar una arquitectura de nube permite centralizar y optimizar los recursos informáticos de la empresa, unificando los costes y distribuyendo el servicio a las necesidades de manera flexible y en tiempo real. También abre la posibilidad de hacer un servidor más grande y alquilar servicios de computación en la nube. A cambio, se mantiene el problema de la dependencia a la red y también se acentúan las consecuencias de que el servidor caiga al completo, afectando a toda la organización. Estas nuevas ventajas y desventajas sólo le benefician a grandes empresas que tienen el capital para la inversión y la distribución para necesitarlo.

2.2.2. GCP (Google Cloud Platform)

Google Cloud Platform es el servicio público de nube ofrecido por Google. Se puede contratar como un conglomerado de servicios vinculados a la infraestructura de Google (como Gmail, Google Calendar, Google Play o Youtube).

Entre los tres servicios de nube más populares vamos a centrarnos en GCP, pero todo lo que se va a mostrar en este proyecto se puede realizar en cualquiera de

los tres servicios. Se ha elegido GCP porque es el más compatible con el ejemplo práctico que se va a realizar en este documento.

Como breve comparación, AWS destaca en potencia y calidad, pero es más caro y menos seguro (aunque sigue siendo seguro). Microsoft Azure es bastante similar en capacidades y precio a GCP y actualmente tiene el mejor sistema SaaS de los tres. GCP destaca por su seguridad y por su reducido precio, pero más específicamente en su capacidad para crear aplicaciones en la nube con la mejor combinación de de servicios de PaaS (Plataforma como Servicio) y SaaS (Software como Servicio). GCP ha ganado popularidad en los últimos años. [11]

GCP ofrece una amplia gama de servicios que se pueden dividir en: computación y hosting, almacenamiento, redes, big data y machine learning. Para cada área se ofrece un set completo con todas las funcionalidades necesarias para realizar una aplicación web. Estas son algunas de las más populares:

Google Compute Engine : Permite crear máquinas virtuales dentro de la nube. Se puede instalar en cada máquina el sistema operativo deseado.

Google App Engine : Es una PaaS (Plataforma como Servicio) para desarrollar aplicaciones web. Soporta Go, PHP, Java, .NET, Ruby, Python y Node.js.

Google Kubernetes Engine : Es un orquestador de contenedores. Permite desplegar, escalar y lanzar contenedores.

Google Cloud Bigtable : Una herramienta para almacenar datos comprimidos de alto rendimiento

Google BigQuery : Una aplicación basada en REST para realizar datasets de tamaño masivo.

Google Cloud Function : Una plataforma sin servidor que reacciona a eventos. Funciona con el paradigma de infraestructura como código.

Google Cloud Datastore : Una base de datos altamente escalable y NoSQL. Tiene de base Bigtable y Megastore.

Google Storage : Un servicio REST para almacenar datos en Google Cloud Platform (GCP). Es similar a Amazon S3 service.

2.3. Integración y despliegue contínuos

La Integración Continua (CI) y el despliegue contínuo (CD) son prácticas cada vez más populares. Permiten reducir el tiempo desde producción hasta lanzamiento y mejorar la calidad de código. Están estrechamente vinculadas, implementar ambas ahorra gastos e incrementa el beneficio que aportan. En este apartado se explicará la importancia de ambos para DevOps.

2.3.1. Integración continua (CI)

La Integración Continua (CI) es una práctica de desarrollo software en la que los desarrolladores integran el código en un repositorio centralizado de manera automatizada a medida que lo van desarrollando. Cada vez que un desarrollador realiza un commit el código pasa por varios tests antes de ser integrado, devolviendo un feedback dependiendo del resultado e integrando el código si todo está correcto.

Habitualmente los desarrolladores van juntando el código y cada cierto tiempo se integra para testarlo y lanzarlo como un nuevo parche. Esta forma es ineficiente porque se debe esperar al final del ciclo para poder comenzar a revisar el código. Por ejemplo, si un desarrollador acaba antes que el resto, tiene que esperar para poder continuar. No obstante el mayor problema es cuando la compilación tiene un error. El proceso para detectar un error es más lento y difícil mientras más extenso sea el código revisado, y en este modelo precisamente se espera a juntar código.

En integración continua no existen esos problemas porque el código se prueba en cuanto está disponible, reduciendo los retrasos. Cuando hay un fallo se corrige antes de que se haga un código dependiente de ese código y antes de finalizar la versión final. Además es más fácil encontrar errores en fragmentos de código pequeños. Como la integración de código es gradual es más fácil detectar qué código provoca el error de compilación.

La CI se puede resumir en tres fases: Desarrollo, Commit y Test. En la fase de desarrollo, los desarrolladores escriben el código del software. En la fase de Commit, se compila y se sube el código al repositorio. La fase de tests comienza cuando se realiza el commit, pasando tests automatizados al código y actuando según los resultados. En cuanto el código supere la fase de tests, pasa a ser responsabilidad del equipo de control de calidad.

En un escenario de Integración Continua, cada vez que se realiza un commit ocurre lo siguiente:

Figura 2.2: Fases de la integración continua

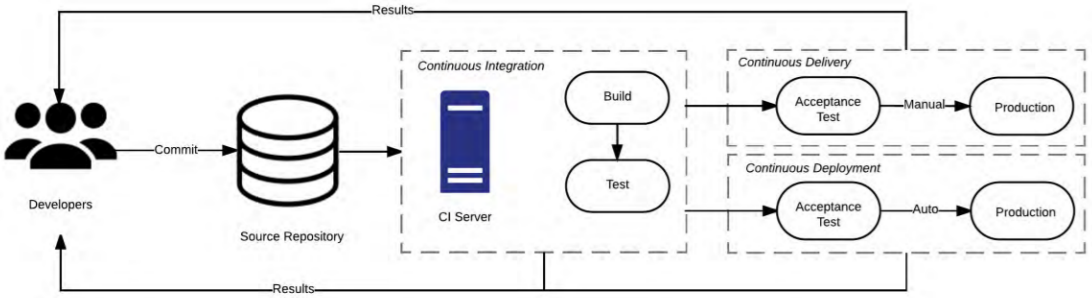


Figura 2.3: Funcionamiento de un sistema de CI de la publicación "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices" [3]

1. El desarrollador hace un commit del código en el repositorio centralizado.
2. El servidor CI detecta el cambio en el repositorio, descargando la última versión y empieza a probarlo con los tests establecidos. Si funciona correctamente el servidor lo compila.
3. El servidor CI devuelve una notificación mediante el sistema de comunicación (e-mail, slack, ...) con feedback sobre la integración.

El éxito de un sistema CI depende de dos elementos del servidor: El repositorio de código y el software de automatización.

El repositorio de código es donde se almacena el código, es necesario para tener en todo momento una versión estable que se pueda actualizar de forma segura, no es posible realizar Integración Continua si no se puede integrar el código. El repositorio se puede montar en un servidor interno o en uno externo. Las ventajas y desventajas se pueden ver en el apartado sobre Google Cloud Platform [**?**].Algunos ejemplos de software para un repositorio de código son Git o SVN

El software de automatización es necesario para que todo el proceso se ejecute automáticamente. En teoría se podría hacer a mano pero entonces las principales ventajas de velocidad y seguridad se reducirían considerablemente. El software de

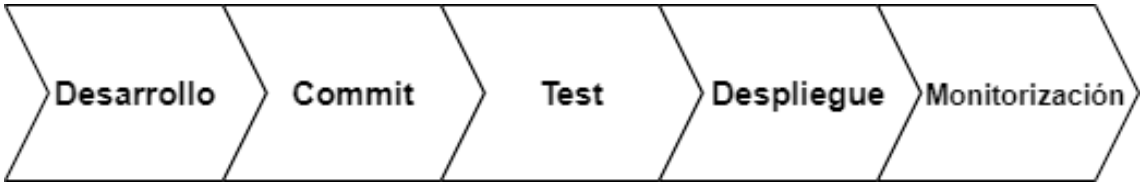


Figura 2.4: Fases del despliegue contínuo

automatización se encarga de iniciar el script de integración con cada commit en el repositorio. Algunos ejemplos de software para integración continua serían Jenkins, Travis CI o TeamCity. No es obligatorio el uso de un software dedicado, se puede sustituir por un conjunto de scripts de Bash, Ant, Maven o Makefile. El problema de usar scripts frente a un servidor dedicado es que añade una nueva capa de complejidad, es menos automatizado y más difícil de configurar. Otra ventaja de usar un servidor CI son las herramientas para monitorizar los commits realizados que ayudan a ver el estado de la aplicación fácilmente.

2.3.2. Despliegue Continuo (CD)

El Despliegue Continua (CD) es una práctica de ingeniería software y una extensión de la Integración Continua (CI). En el Despliegue Contínuo el código se despliega según el código pasa el proceso de CI y superando tests de integración, creando así ciclos cortos de despliegue. No es obligatorio desplegar todo el código desarrollado. La CD sólo comienza una vez el producto está en un estado lo suficientemente avanzado como para ser desplegado.

La CD añade dos fases nuevas a la CI (Despliegue y monitorización), además de expandir la fase de tests. En la fase de tests se añaden tests de integración, de aceptación y verificación. En la fase de despliegue se juntan la entrega y despliegue automáticos del código. Por último en la fase de monitorización se analiza el código desplegado para determinar si se está ejecutando apropiadamente y encontrar errores que no se hayan detectado en la fase de tests. Cuando el código pasa los tests y la revisión del equipo de control de calidad, el código pasa a ser responsabilidad del equipo de despliegue.

El objetivo de la CD es tener siempre una versión lista para usarse sin detener el servicio en ningún momento. La CD permite entre otras cosas realizar cambios en tiempo real y actualizar sin tener que esperar a un parche de versión.

Con un sistema de CI/CD, si hay un equipo de control de calidad, estos podrán empezar a trabajar inmediatamente en vez de esperar a la primera versión de la

compilación. Si el sistema de CI/CD automatiza también los test de aplicación, también le ahorra trabajo al equipo de control de calidad para poder centrarse más en casos no controlados y en la calidad general del código. Esto implica que el equipo de control de calidad puede empezar a trabajar antes y tiene menos carga de trabajo.

El problema de la CD es que hay que dedicar mucho esfuerzo para poder aplicarla. Primero hay que implementar un sistema de CI robusto y además preparar el sistema de CD. El propio equipo de despliegue también tiene que acostumbrarse porque la forma de trabajo es distinta a otros modelos, con una carga de trabajo menor pero constante en vez de grandes cargas de trabajo separadas en el tiempo. Además si no se implementa apropiadamente, el sistema de CD puede fallar fácilmente y el despliegue es una etapa crítica de cualquier proyecto.

Hay dos formas de mitigar el riesgo de que un fallo supere los tests y se lance una versión defectuosa. El primero es que el último paso de lanzar el código sea manual, aunque esto significa crear un tapón en el proceso, da más control sobre el software al equipo de despliegue. El segundo es el uso de servidores canarios, que son servidores que reciben las actualizaciones antes que el resto para que si hay un error, sólo afectará a parte de los clientes y no a todos a la vez.

2.4. Implementación de procesos CI/CD

Para implementar un sistema de integración y despliegue continuos, se necesitan hacer cambios en la infraestructura y arquitectura de la empresa.

Los cambios más importantes afectan a los desarrolladores. Estos cambios son la base de un buen sistema de integración y despliegue continuos, aunque hay que ajustarlos a las necesidades específicas de la empresa. Se debe usar un Desarrollo Dirigido por Tests (TDD, Test-Driven Development), en el que se crean los test unitarios antes de escribir el código. El código se debe integrar lo más frecuentemente posible (al menos una vez al día). Se deben utilizar herramientas para asistir la programación que ayuden a disminuir el esfuerzo para mantener el código. También se debe crear un script para la plataforma de integración continua para que realice las compilaciones de manera automática y por línea de comando, para integrarlo más fácilmente con otras herramientas. Para este script no importa el lenguaje (Maven, Ant, MSBuild, Bash, PowerShell, ...) pero debe dar siempre el mismo resultado ante un mismo input. El servidor de Integración y despliegue continuo deben cumplir los siguientes requisitos:

1. Realizar frecuentemente commits debidamente testados, por pequeña que sea la modificación.
2. No romper el código con los commits, cada cambio sustancial debe de tener su propia compilación.
3. Desarrollar test unitarios.
4. La compilación se debe hacer automáticamente por un script.
5. Diseñar vías para el despliegue del software.
6. Crear una estrategia de despliegue específica para cada etapa del proyecto.

Uno de los pasos más importantes a la hora de implementar un sistema de Despliegue Continuo efectivo es establecer una buena estrategia. Las partes que componen una buena estrategia de CD son:

Buenas estrategias de ramificación del repositorio : Se debe implementar una estrategia de repositorio que permita trabajar con ramas individuales de forma ordenada y segura. La estrategia más habitual es crear una rama para cada bug o característica desarrollada. De esta manera se puede experimentar más con el código sin riesgo de dañar la rama maestra y por tanto siempre se dispone de una versión potencialmente entregable.

Una fuerte política de tests unitarios : Para poder tener un Despliegue Continuo eficaz, se necesita un código libre de errores y que cumpla los requisitos del cliente. La forma más habitual de conseguirlo es mediante Desarrollo Dirigido por Tests (TDD, Test-Driven Development), con esta metodología se preparan los test unitarios antes del desarrollo basándose en los requisitos del proyecto. También es necesario tener una gran parte del código cubierta por los test. Alrededor del 85 % del código es lo ideal. Otra buena estrategia es la pirámide de tests de Mike Cohn. Según esta estrategia se harían pruebas pasando por tres fases, siendo las primeras (la base) más exhaustivas. Test de unidad, para probar que el software cumple con su cometido más básico; Test de servicio, para probar que el software se puede lanzar; Test de UI, para probar que la interfaz gráfica funciona apropiadamente.

Una fase de tests automatizada : Las pruebas de test unitario sirven para asegurar de que partes individuales del código funcionan correctamente, pero no demuestran que toda la aplicación funciona apropiadamente. Es necesario planificar una serie de test automatizados que prueben toda la aplicación al

completo. Las partes que aún no están desarrolladas se pueden suplir introduciendo los datos que deberían aparecer de manera forzada. El objetivo de estos test no es tanto probar que las partes individuales del código funcionan apropiadamente, sino que están conectadas correctamente.

Promoción de código automatizada : Debe haber un mecanismo que promueva automáticamente las versiones del código que pasan todos los tests. Normalmente los servidores de integración continua como Jenkins promueven el código por sí mismos. Esto se realiza mediante el uso de tags o por ramas. Al usar esta forma de promoción de código, cuando se lanza una versión del código lo que se hace en realidad es una copia a otro servidor de la última versión. Los servidores de Integración Continua suelen tener un archivo con la información de la estructura del software. A este archivo se le llama ("artifact immutability")

Inspección de código : La inspección de código es una práctica importante para la integración y despliegue continuos. A esta práctica se la conoce como "linting". El objetivo de la inspección de código es comprobar fácilmente la calidad estructural del código, comprobando por ejemplo si un método es demasiado largo. Se pueden realizar inspecciones de código sobre la marcha usando las herramientas de integración continua para realizar un control de calidad previo automatizado. Por ejemplo, se puede comprobar que se sigue un mismo convenio en el código u obligar a cumplir ciertas normas. Probablemente el factor más importante de revisar en una inspección de código es la "Complejidad Ciclomática". La complejidad ciclomática es la cantidad de caminos independientes que pasan por un método en el software. Sirve para medir la complejidad de lectura, comprensión y testeo de una aplicación.

2.4.1. Testeo e inspección continuos

Una parte importante de la implementación de CI/CD es asegurar la implementación automatizada de los tests e inspecciones del código para mantener o mejorar el nivel de calidad. El objetivo es producir código fiable con cada release de manera automatizada. La fase de tests se divide en tres partes, que en orden son:

Test unitarios : Prueba que las funcionalidades del código están bien implementadas y compile sin errores. El código se prueba de manera individual (por ejemplo una clase o función) y no el conjunto del código. No es realista ni eficiente usar tests para cubrir todo el código, pero se debería alcanzar un alto porcentaje. El porcentaje depende de cada software, pero debería estar

siempre entre un 80 % y un 90 %. Se puede comprobar la cobertura de código mediante el test unitario.

Test de integración : Sirve para asegurarse que el software funciona con información real en un entorno controlado. En vez de datos artificiales para cada método, se probaría el software como su conjunto introduciendo la información por los canales que usará cuando esté acabado.

Tests de aceptación y verificación : Comprueban que el software funciona correctamente desde el punto de vista del usuario. En especial comprueba que cumple los requisitos establecidos y que su interfaz es entendible.

Además del testeo, se deben hacer inspecciones de código. Las inspecciones del código se dividen en dos partes: Revisón de código y análisis de código estático.

La revisión del código se realiza antes de integrar el código. El código es revisado por otros desarrolladores para comprobar si se puede realizar un merge con el main branch (Rama principal). Si el código tiene algún problema, se señala con comentarios en el código. Cuando se resuelvan todos los problemas se podrá realizar el merge.

El análisis del código estático se realiza cuando se integra el código. Se divide en dos fases. En la primera fase el desarrollador ejecuta el commit local y se comprueban reglas de claridad del código (como el número de caracteres por línea, la complejidad de los métodos, ...), esto se puede automatizar con algunas herramientas durante el desarrollo. En la segunda fase se buscan bugs durante la ejecución.

Una vez se han realizado los tests y la inspección, se puede compilar y entregar el código.

2.4.2. Diseñando los canales de un sistema de integración y entrega continuos

Los canales de un sistema de integración y entrega continuas son esenciales. Permiten definir los pasos para compilar del software y entregarlo. Cuando se hace la compilación de un software normalmente existen tres fases, cada una con su propio canal. Las fases son en este orden: Fase de desarrollo, Fase de puesta en escena y Fase de producción

Al definir un canal, esencialmente se establece un sistema de promoción de software de un estado a otro. Este proceso debe ser automático, y se puede realizar con

```
pipeline{
    agent any

    stages{
        stage('Build'){
            steps{
                echo 'Starting building'
                //Build commands
            }
        }
        stage('Test'){
            steps{
                echo 'Starting testing'
                //test commands
            }
        }
        stage('deploy'){
            steps{
                echo 'Starting deploying'
                //Deploying command
            }
        }
    }
}
```

Figura 2.5: Ejemplo de jenkinsfile básico. Aquí se establece el canal.

programas de automatización de repositorios como: GoCD, Travis CI, GitBucket, Circle CI o Jenkins. Es habitual que la configuración se pueda guardar como un script para poder añadirlo a un repositorio de código.

En los canales se debe diseñar un proceso que procese el commit y lo teste, devolviendo un feedback de al menos las compilaciones fallidas con el log del error, los test fallados y el número de compilación.

2.4.3. Integración continua de bases de datos

El software suele tener asociadas una o más bases de datos para almacenar datos. Durante las actualizaciones del software, la estructura de las bases de datos puede variar y hay que adaptar la base de datos de tal manera que no se pierda información. A esto se le conoce como “migrar la base de datos”.

La integración continua de bases de datos trata de crear y llenar la base de datos en cada versión. Este proceso sirve para depurar la base de datos y obtener siempre un conjunto de datos "limpios" para trabajar. No obstante este método tiene un pequeño riesgo de corromper los datos por un error, por ello en los casos donde la información es extremadamente sensible (como en información financiera) se debería hacer de manera menos frecuente o buscar una alternativa.

Para manejar este proceso se utilizan los scripts de DML (Data Manipulation Language, Lenguaje de Manipulación de Datos) y el DDL (Data Definition Language, Lenguaje de Definición de Datos). El DML y DDL deben ir en repositorio software con el resto del código.

El proceso para recrear la base de datos es el siguiente:

1. Realizar un DROP en la base de datos para tener una versión completamente nueva.
2. Mantener el DML y DDL en el repositorio.
3. Mantener un espacio distinto para la nueva base de datos.
4. El equipo de Administración de bases de datos (DBA) debe hacer una revisión del código del DML y DDL, así como asegurarse de que la base de datos no tiene errores.
5. Si cambia la estructura de la base de datos, se debe comprobar que los datos se trasladan apropiadamente.

2.4.4. Preparando la compilación para el despliegue

Cuando ya se tiene la última versión actualizada del código, antes de iniciar la release final se debe subir a un repositorio y documentar. Hay que seguir tres reglas para subir el código: Identificar el código en el repositorio para poder distinguirlo del resto de releases, crear un reporte de compilación y subir la compilación a un repositorio compartido.

Hace falta identificar el código en el repositorio para distinguirlo del resto de versiones y buscarlo fácilmente. Es importante escoger un nombre descriptivo, indicar la versión y que todos los nombres sigan un convenio común. A parte del nombre, las principales formas de identificar el código son crear una etiqueta, un breve texto que permite identificar la versión de manera similar a un nombre; crear un sistema de tags, los tags son breves descripciones que se pueden añadir a varias versiones

distintas. Se puede identificar una versión concreta buscando una combinación de tags que coincida con la versión deseada; y usando otra branch, si se crea otra rama, puede servir como una etiqueta.

Hay que crear reportes de compilación que cualquier persona pueda entender (incluso si no tiene conocimientos técnicos) para que otros compañeros puedan saber qué hay en la compilación sin tener que preguntar al desarrollador original. El reporte debe indicar el nombre, versión y forma de obtención de la release. También debería incluir una descripción de los cambios de la versión. El informe puede ser autogenerado si se conecta con un programa de administración de proyectos como Jira, que puede añadir automáticamente la descripción de la tarea.

Cuando se ha acabado con la compilación, se debe dejar accesible al resto de equipos. Para esto podemos usar un repositorio como git o crear un registro que permita descargar la versión más reciente. Es importante que una vez validado y subido, no se puede editar. Para realizar cambios se crea una nueva versión.

2.4.5. Desplegar el código

Esta es la última parte del ciclo, entregar el código para que empiece a funcionar en vivo.

Cuando se hace una release del código, este tiene que ser estable y no dejar sin servicio al cliente. Sólo se debería actualizar de forma periódica y planificada, o por una actualización de emergencia (como una brecha de seguridad). Si hay algún fallo y no se puede mantener la versión, se debe volver a la última versión estable hasta que se solucione el fallo.

Las releases no se deberían lanzar al servidor estable de forma directa, primero hay que probar que funciona en servidores en vivo. Estos servidores sirven como última prueba para detectar fallos que sólo ocurran en un entorno no simulado y obtener feedback de su uso real, pero sin exponer a todos los clientes. El concepto es similar al de las versiones beta, pero el software es un producto final. Hay tres formas de hacer releases de las versiones sin interrumpir al cliente: Despliegue Azul/Verde, Servidores canario y Despliegue incremental.

Los servidores azul y verde consisten en dos servidores que se pueden intercambiar sobre la marcha sin actualizar. En un servidor se tiene la última versión estable y el otro la nueva versión que se va a usar. Cuando se hace una release, se sube al servidor que no está en uso y una vez todo esté en orden se redirige a los usuarios al otro servidor, si hay algún fallo o problema de última hora sólo hay que cambiar

de nuevo al servidor estable. Este método, aunque eficaz, es algo complejo por las bases de datos que hay que modificar en ambos servidores y porque la información la debe guardar el cliente en su caché hasta que la guarde.

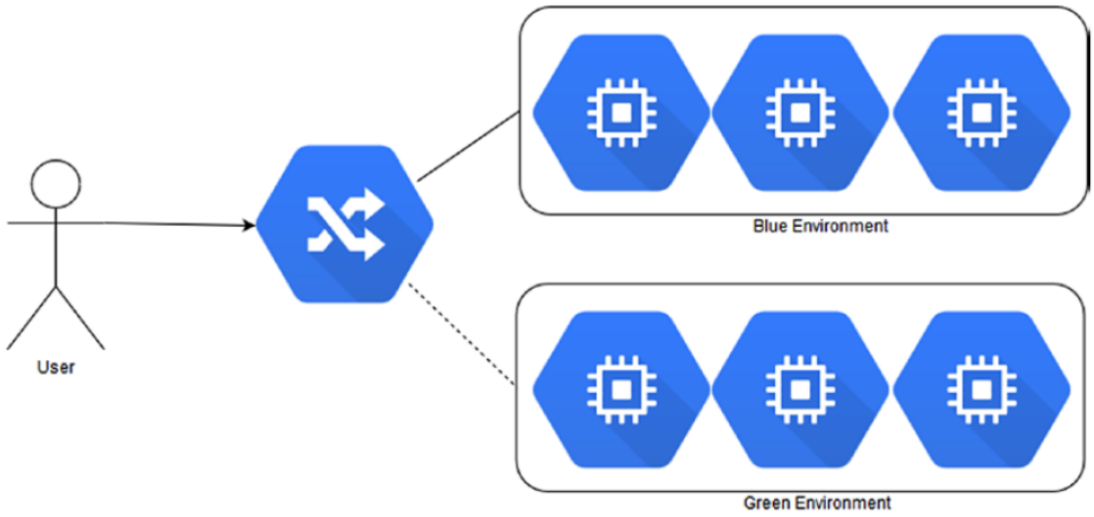


Figura 2.6: Esquema de servidores azules y verdes del libro [1], figura 3-6

Los servidores canarios consisten en que una pequeña parte de los usuarios (sobre un 10 %) usen la nueva versión. Si no funciona es más fácil realizar un rollback menos extenso o conseguir un feedback más personalizado con una cantidad sustancialmente menor de clientes. Si funciona se incrementa el número de clientes paulatinamente sin llegar al 100 %.

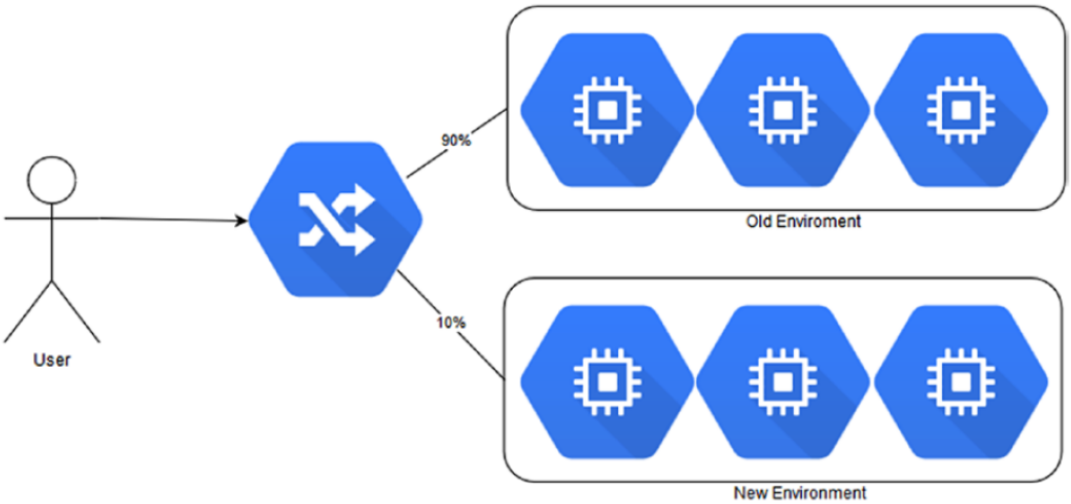


Figura 2.7: Esquema de servidores canario del libro [1], figura 3-7

El despliegue incremental consiste en aplicar las actualizaciones a un pequeño porcentaje de usuarios (sobre un 5 %) y una vez se está satisfecho con los resultados, ir aumentando poco a poco. Un beneficio respecto a los otros métodos es que sólo necesita un servidor.

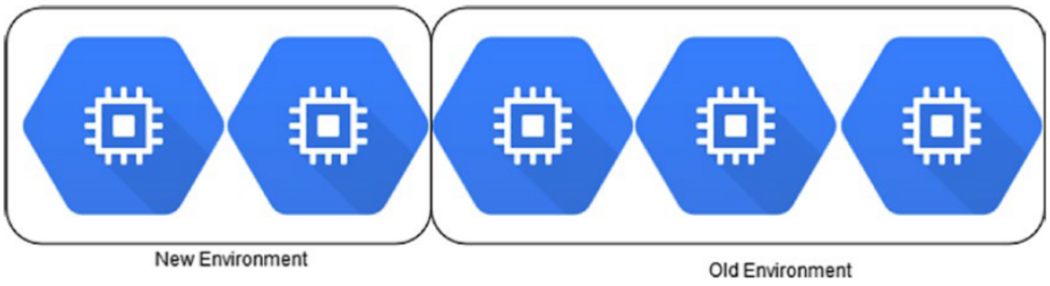


Figura 2.8: Esquema de servidores incrementales del libro [1], figura 3-8

2.5. Introduccióna a contenedores (Docker y kubernetes)

Los contenedores existen desde hace muchos años, pero fueron popularizados por Docker. Docker es una plataforma abierta para trabajar con contenedores. Existen más plataformas de contenerización como RunC, Podman o LXD, pero la más extendida es Docker y vamos a ver esta tecnología mediante el punto de vista de Docker.

La contenerización es una forma de empaquetar aplicaciones de tal manera que puedan ser ejecutadas independientemente de sus dependencias en entornos virtuales. En la práctica los contenedores funcionan de una manera similar a las máquinas virtuales, pero en vez de cargar una máquina completa, virtualiza sólo lo mínimo y necesario para la ejecución.

La ventaja de los contenedores frente a las máquinas virtuales son su ligereza, portabilidad y sencillez. Los contenedores son más ligeros tanto computacionalmente como en el disco duro en relación a una máquina virtual. No necesita de un Hypervisor (que es una plataforma que permite usar varios sistemas operativos simultáneamente) y usan directamente el kernel del host. Esto significa un descenso de consumo de memoria de alrededor de un 80 % respecto a máquinas virtuales. Además los contenedores tampoco hay que instalar un sistema operativo por máquina virtual.

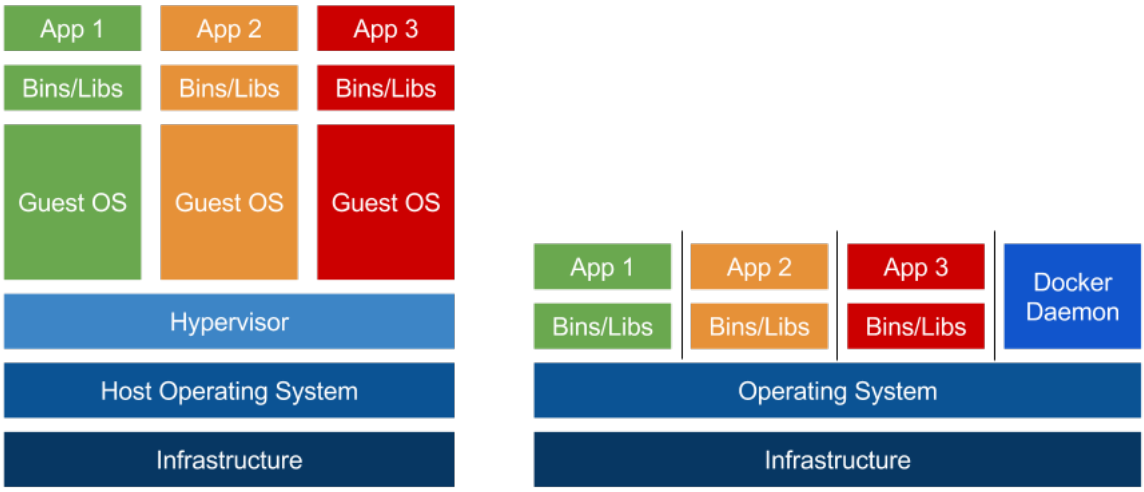


Figura 2.9: Diferencias entre máquinas virtuales y contenedores [4]

Los contenedores permiten mayor portabilidad, permitiendo empaquetar aplicaciones en contenedores y ejecutarlas en diversos entornos sin problemas de compatibilidad. También se pueden subir contenedores a repositorios o incluso trabajar directamente en la nube, permitiendo desplegar contenedores actualizados fácilmente. Esto sería más complicado con máquinas virtuales por su peso.

También tienen ventaja en el mantenimiento. Con los contenedores no hace falta instalar o actualizar los sistemas operativos y aplicaciones necesarias de las máquinas virtuales, ni configurar parámetros o gestionar imágenes y versiones. Con crear un archivo de configuración y actualizar según las necesidades la aplicación virtualizada la aplicación se mantiene actualizada automáticamente.

2.5.1. ¿Cómo Funciona Docker?

Docker virtualiza aplicaciones. Esto significa que crea una capa de ejecución en el ordenador donde la aplicación se ejecuta de manera independiente. La teoría es la misma que con las máquinas virtuales, pero los contenedores no necesitan virtualizar el sistema operativo. Como no necesitan virtualizar un sistema operativo, tampoco necesitan el hypervisor, que es la herramienta del equipo anfitrión para gestionar varios sistemas operativos simultáneamente.

El funcionamiento de Docker se compone de cinco elementos:

Demonio : Es el proceso principal que, como todos los demonios, se ejecuta en segundo plano y es inaccesible de manera directa.

Cliente : Es la aplicación que permite al usuario interactuar con el Demonio.

Imagen : Plantilla de la aplicación que se desea virtualizar.

Contenedor : Unidad de software estandarizada, generada mediante la imagen. Es la aplicación virtualizada.

Registros : Almacenamiento de los datos no perecederos generados en los contenedores.

El Cliente de docker permite al usuario generar mediante las Imágenes, Contenedores de las aplicaciones a virtualizar. Cuando estas aplicaciones generan datos que deben perdurar la ejecución, esa información pasa a Registro.

2.6. Introduccióna a Continuous Delivery with GCP and Jenkins

El despliegue continuo es la extensión de la integración continua y uno de los pilares de DevOps. Hay varias herramientas para poder implementar un sistema de despliegue contínuo, siendo los más habituales Travis CI, GoCD, Bitbucket y Jenkins. A continuación explicaremos Jenkins que es el más conocido [libro] y se puede usar tanto para la parte de integración continua como para la de despliegue contínuo.

Jenkins es de código abierto y está escrito en java. Es compatible con casi todos los tipos de repositorios. Jenkins permite automatizar todos los procesos de CI/CD fácilmente. Algunos de las principales ventajas de Jenkins son su modularidad; su facilidad de instalación, configuración y personalización; y su interfaz limpia y sencilla

Los plug-ins son el concepto más importante de Jenkins. Jenkins se divide en el Core, la parte más esencial para poder funcionar, y los plug-ins, que son funcionalidades que se pueden descargar para especializar la herramienta. Los plug-ins permiten ampliar el entorno de manera modular y mantenerlo sencillo al no tener que compartir la interfaz con opciones irrelevantes para el proyecto. Los plug-ins están divididos por funciones y se pueden encontrar plug-ins para todo tipo de necesidades o incluso crear uno nuevo.

2.6.1. Integración y despliegue contínuo con Jenkins

Jenkins facilita en gran medida la creación de canales para el CI/CD. Los canales deben ejecutarse siempre igual y dar el mismo resultado.

Un canal de CI/CD se puede separar en las siguientes partes: Código, Tests unitarios, Integración de código, Tests de sistema, Publicación por pasos, Aceptación de usuario y Publicación en producción.

Integración continua	
Desarrollo	Cuando se tiene el código hecho y se ha probado localmente, se tiene que subir a un repositorio en la rama correcta. Para esta etapa hay que montar un repositorio y definir una política de commits.
Tests unitarios	Cuando se sube el código al repositorio, este debe pasar por test que se ejecuten automáticamente y detecten los bugs y fallos que impidan un funcionamiento mínimo. Para esta etapa hay que desarrollar estos tests.
Integración de código	Cuando el código de una branch está finalizado, se debe juntar a la branch principal (rama principal). Este paso se puede automatizar, pero generalmente se debe pasar antes una política de revisión del código. Estas políticas de revisión requieren de al menos dos desarrolladores que no hayan escrito el código y sirve para comprobar fallos y problemas y para que todos en el equipo tengan una idea de en qué está trabajando el resto del equipo. Para poder automatizar todo el proceso (y tener un sistema de CD) se puede establecer un análisis de código estático, para no detener el proceso. En esta etapa se debe generar un algoritmo para poder integrar el código y mandar la revisión a quien tenga que hacerla.
Tests de sistema/ Test de integración	Cuando todo el código está integrado, se deben pasar unos test similares a los test unitarios pero con datos reales como los que se van a usar. En estos test se debe probar todo el sistema para detectar si al añadir un nuevo se genera un nuevo fallo. Para esta etapa hay que desarrollar los tests.
Despliegue contínuo	

Publicación por pasos	Cuando los tests de sistema son correctos, Jenkins puede comenzar el proceso de publicación automatizada. Se puede configurar para que ejecute un script personalizado o para envie el código al registro de del servidor de desarrollo al público.
Aceptación de usuario	Esta parte es especialmente importante, consta de varios tests desde el punto de vista del usuarios. Estos test se pueden automatizar si el equipo de control de calidad diseña unos tests en base a entrevistas con los usuarios objetivo. También es posible no automatizar los tests, y dejar que Jenkins simplemente se encargue de enviar las nuevas versiones a los servidores canario, dejando la labor de comprobar la aceptación al equipo de control de calidad.
Publicación en producción	Esta etapa es la más crítica y puede producirse varias veces al día, según se hagan actualizaciones en el código. Para una entrega continua completa, Jenkins puede administrar el código y desplegarlo directamente. Si se prefiere, también se puede hacer una última comprobación en un servidor canario y subir sólo lo que esté completamente probado.

Tabla 2.2: Etapas de un sistema CI/CD

Para usar Jenkins de una manera eficiente (y en general para que cualquier sistema de CI/CD funcione), se debe crear una política de repositorio consistente y eficiente. Concretamente se debe especificar en qué casos y cómo se hacen nuevas ramas de repositorio. La estrategia más popular es crear una rama por cada característica desarrollada para el software. Es una estrategia sencilla y efectiva.

Cada vez que se realiza una unión de ramas (merge), Jenkins realiza las comprobaciones de las que se han visto anteriormente. Si se trabaja en un grupo o empresa grande, lo suficiente como para tener que procesar varios códigos simultáneos, es probable que se necesite usar hilos en Jenkins. Se puede configurar Jenkins mediante un plug-in o desde el Jenkinsfile para que cree un hilo distinto para cada nuevo canal que se use para procesar un código.

Capítulo 3

Metodología y herramientas

En este capítulo se presenta la metodología que se ha seguido para el desarrollo del proyecto, y se exponen las herramientas utilizadas.

En este capítulo se explica la metodología seguida para desarrollar el proyecto y se muestran las herramientas usadas.

3.1. Modelo de desarrollo

El modelo seguido ha sido de Investigación aplicada. Se han establecido tres fases para la investigación.

Describir el modelo de desarrollo empleado durante el proyecto. Por ejemplo el modelo de refinamiento en espiral u otro que se ajuste al desarrollo realizado.

3.2. Planificación

Investigación

En la primera fase se leyó el libro "Pro DevOps with Google Cloud Platform" [1] y se buscó información sobre las tecnologías del proyecto.

Se planificó que esta fase durase un mes aproximadamente.

Aprendizaje

En la segunda fase se realizó un aprendizaje superficial de las tecnologías del proyecto sobre las cuales no se tenían conocimientos previos.

Se planificó que esta fase durase dos meses aproximadamente.

Desarrollo

En la tercera fase se planificó y desarrolló un modelo de ejemplo documentándolo durante el desarrollo. El objetivo de este modelo es asentar los conocimientos de las herramientas y servir como introducción.

Se planificó que esta fase durase dos meses aproximadamente.

Diagrama resumen (Gannt)

La planificación original llevaba el proyecto a una duración de 5 meses con un mes de holgura para finalizar la documentación.

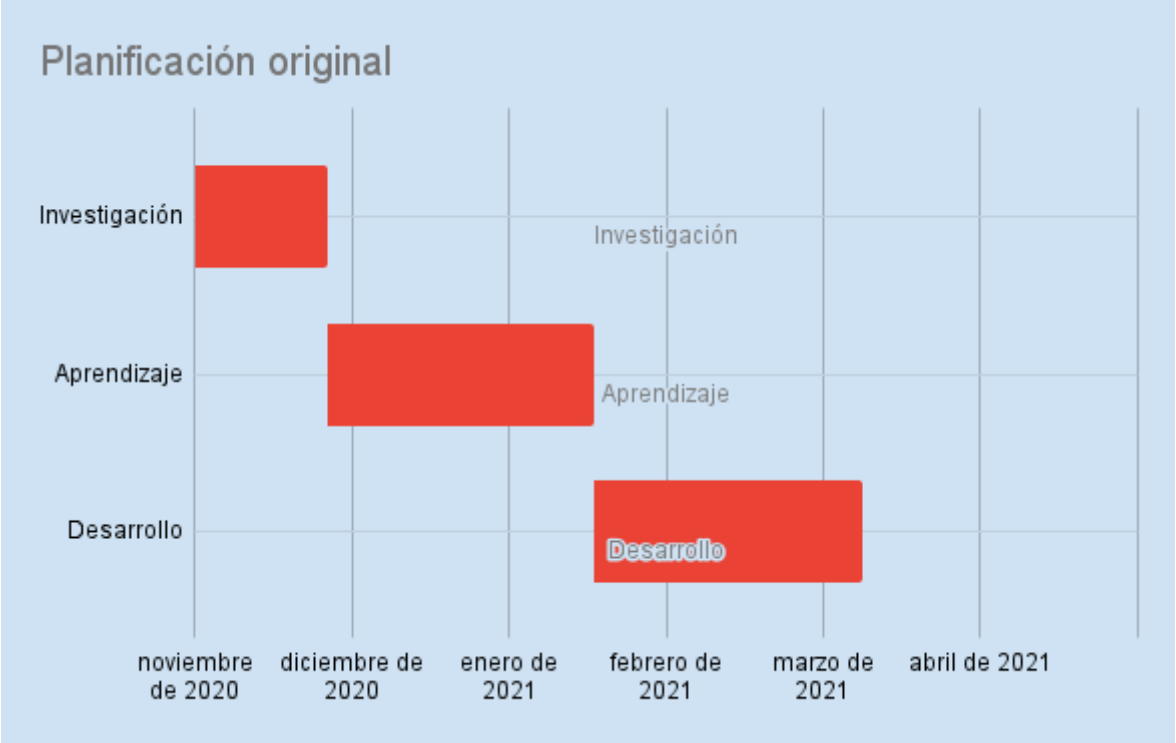


Figura 3.1: Diagrama de Gannt de la planificación original

En la ejecución real hubo problemas ajenos al proyecto que dificultaron avanzar durante dos meses. Además la tercera fase se prolongó más de lo esperado.

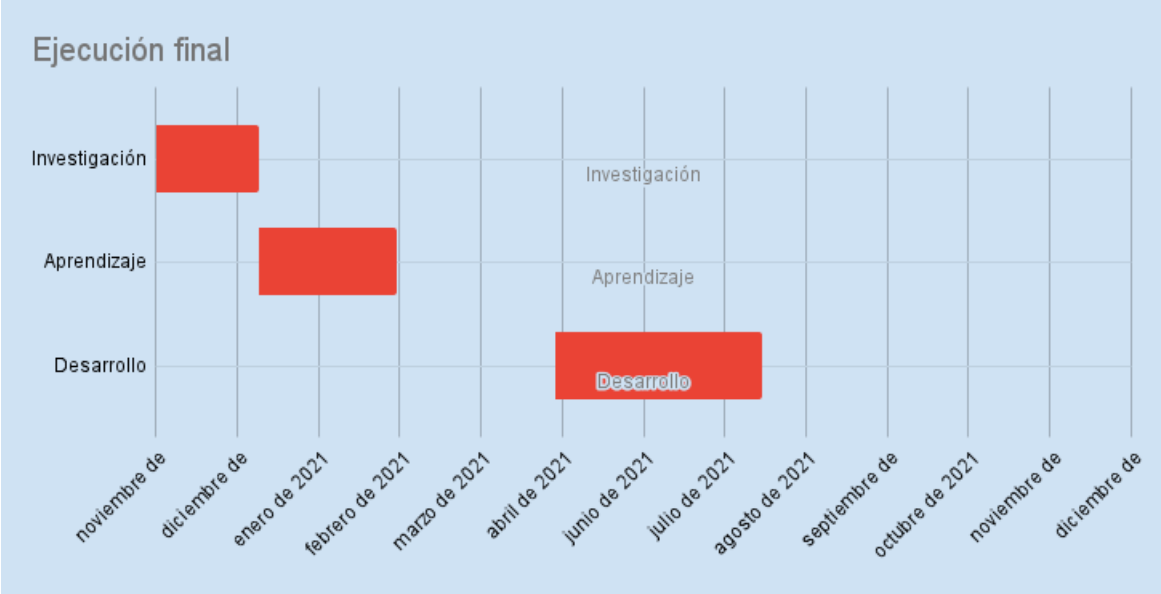


Figura 3.2: Diagrama de Gannt de la ejecución final

3.3. Herramientas

3.3.1. Software

A continuación se dará una breve descripción de las tecnologías software usadas.

Plataforma de desarrollo

El trabajo se ha desarrollado utilizando el sistema operativo Windows 10 Pro.

Visual Studio Code

Se usó el entorno de desarrollo Visual Studio Code para el desarrollo de código general (tanto la aplicación web como los archivos de configuración YAML y el Jenkinsfile).

Con Visual Studio Code se puede programar en una amplia variedad de lenguajes gracias a su modelo de extensiones. Las extensiones permiten añadir modificaciones modulares haciéndolo en un IDE flexible pero ligero.

GitHub

Como repositorio se uso GitHub tanto para probar las tecnologías mencionadas como repositorio del propio proyecto. Se eligió GitHub por la experiencia previa con el repositorio.

Google Cloud Platform

Como servicio de nube para el proyecto se ha elegido Google Cloud Platform (GCP). Tras analizar las tres principales tecnologías se iba a usar Amazon Web Services (AWS), pero debido a un problema de facturación se decidió pasar a la segunda opción: Google Cloud Platform.

Jenkins

Se ha usado Jenkins como administrador de servidor. El principal motivo de la elección de Jenkins fue que la documentación oficial de GCP usaba Jenkins y no otros administradores de servidores.

docker

Se ha usado Docker tanto de manera directa al investigar la herramienta, como indirecta al usar kubernetes.

Se ha elegido Docker frente al resto de tecnologías de contenerización porque es mucho más conocido y está muy bien documentado.

Kubernetes

Se ha usado kubernetes como orquestador de contenedores. No se invirtió mucho tiempo en buscar alternativas porque kubernetes está perfectamente integrado con GCP y Docker.

Node.js

Para la aplicación web de muestra se usó Node.js por su facilidad para crear aplicaciones web sencillas.

Herramientas auxiliares

Citar otras utilizadas o desarrolladas específicamente para el proyecto.

3.3.2. Hardware

Como hardware sólo se ha usado un ordenador portátil Asus. El procesador es un Intel©Core(TM) i7-8750H CPU @ 2.20GHz 2.20 GHz.La RAM es de 8,00 GB. El sistema operativo es de 64 bits.

Técnicamente se ha alquilado un servidor de GCP en europa central para el proyecto.

Capítulo 4

Sistema desarrollado

En este capítulo se explicará de forma más práctica cómo implementar DevOps en un proyecto. Se comenzará explicando cada tecnología se irán uniendo hasta formar un ejemplo funcional.

4.1. Google Cloud Platform (GCP)

Como ya se explicó antes2.2.2, GCP es una

4.1.1. Crear una cuenta

Primero visitamos la dirección https://cloud.google.com/ y hacemos click en "Comenzar gratis". Tras rellenar los datos iniciales, pedirá introducir una tarjeta de crédito o débito, las tarjetas de prepago no son válidas. Esto se necesita para confirmar que no somos un bot y no cobrarán nada salvo que autoricemos el pago.

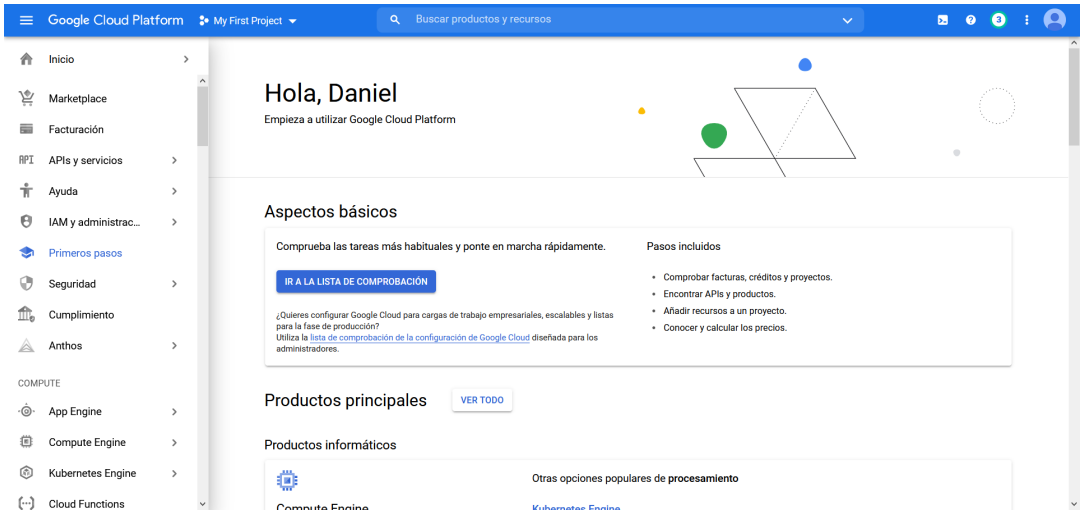


Figura 4.2: Pantalla de inicio

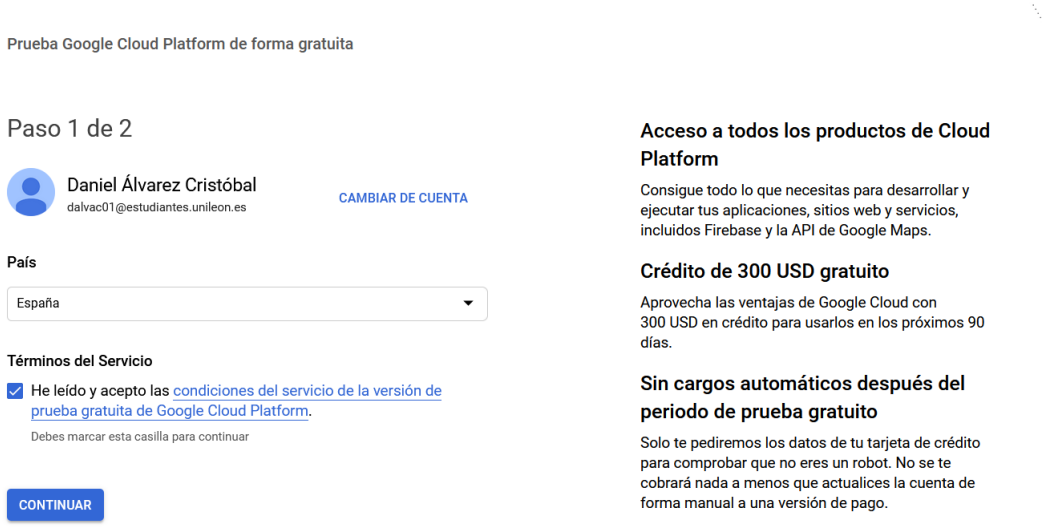


Figura 4.1: Inscripción y ventajas

Una vez registrados tendremos 90 días de acceso gratis y 300 dólares americanos de crédito virtual para usar en ese periodo Al contrario que Amazon Web Services o Azure, tendremos acceso a todos los productos aunque por menos tiempo. Respecto a cuándo se escribió el libro, se ha reducido el periodo gratuito de un año a 90 días.

Cuando completamos el registro nos llevará a la pantalla de "Primeros pasos", donde tendremos acceso a tutoriales y a una lista de tareas para comenzar rápidamente en "Ir a lista de comprobación".

Aunque es opcional, es recomendable visitar la lista si no se ha trabajado previamente con GCP y por su calculadora de costes que es útil para planificar costes en función del servicio y todas las configuraciones. Por ejemplo un servidor de un sistema operativo Devian, durante 18 horas al día, 5 días a la semana y un disco duro en en servidor de EU central; costaría 26,21 USD al mes.

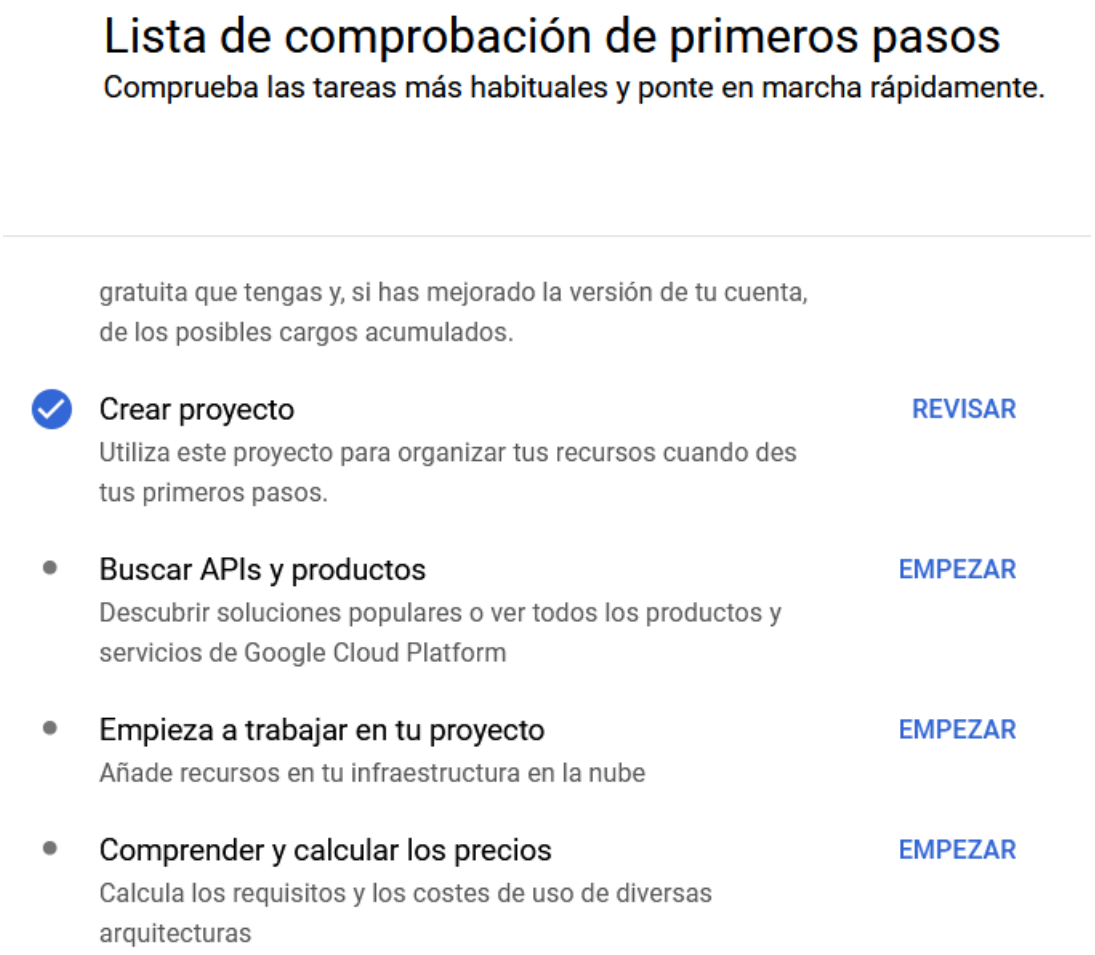


Figura 4.3: Lista de comprobacion de primeros pasos

4.1.2. Crear una organización

Las organizaciones sirven para que si trabajas para varias empresas, sus proyectos no acaben juntos y se mezclen. Todas las cuentas reciben una organización por defecto al aceptar los términos y condiciones de usuarios [5]. En el caso del ejemplo, al usar una cuenta del dominio @unileón.es, se usa el dominio de la universidad automáticamente.

Es necesario disponer de un dominio para usar GCP. Se puede crear una nueva organización siguiendo los siguientes pasos:

1º: Desde el menú principal entramos al menú "IAC y Administración" > "Identidad y organización".

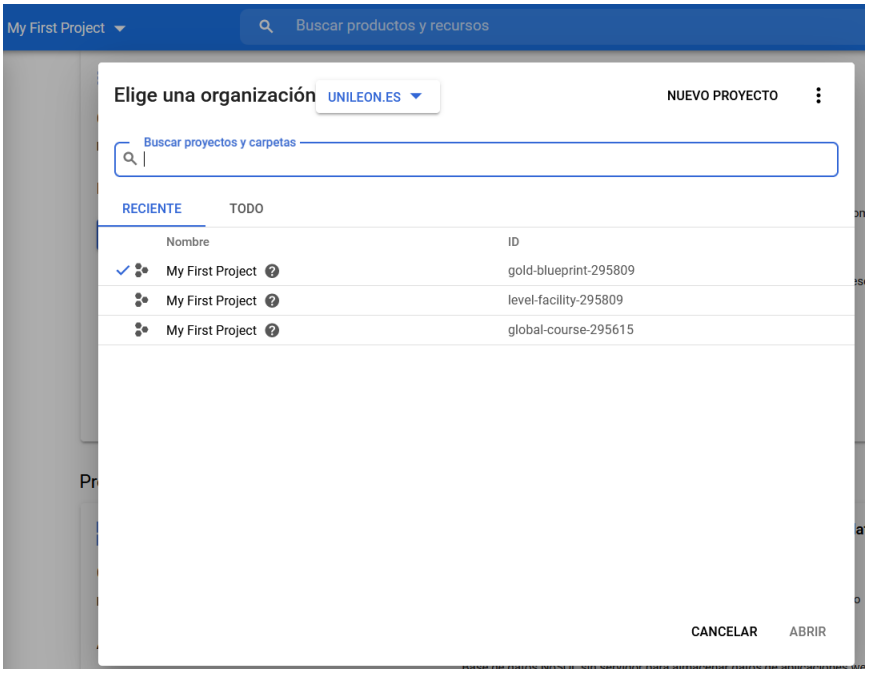


Figura 4.4: Creación de proyecto

2º: Hacemos clic en "Ir a la lista de comprobación"

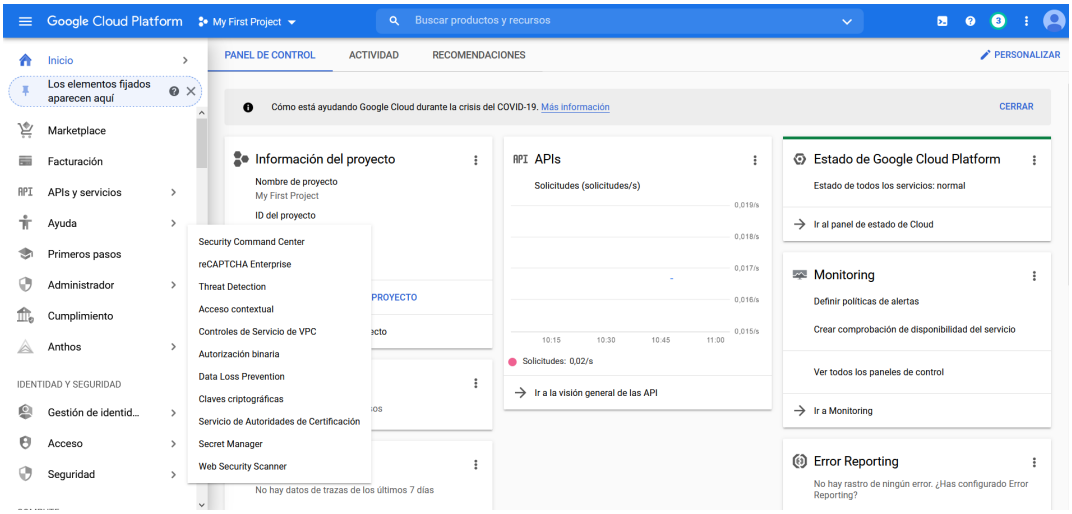


Figura 4.5: Menú principal

3º: Abrimos el menú de "Regístrate en un servicio de gestión de identidades y configura tu dominio y tu recurso de organización".

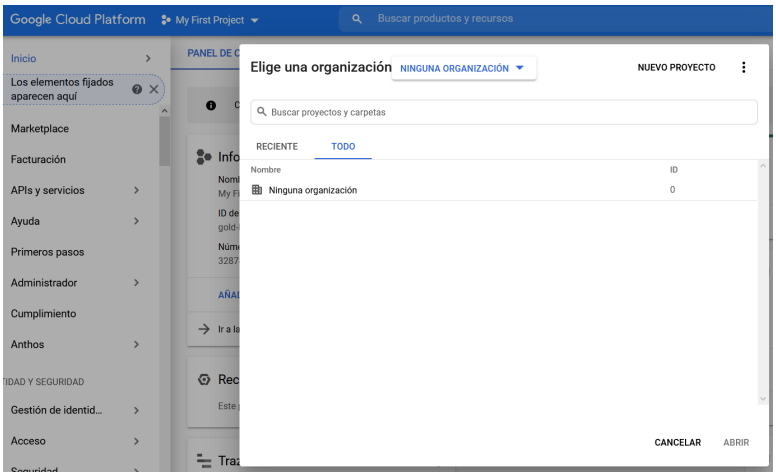


Figura 4.6: Creación de proyecto

4º: Para crear una organización nos hace falta una identidad web, aparecerá un menú en la derecha de la pantalla con instrucciones para crear o configurar una cuenta de Cloud Identity o Google suit. Para completar esta parte sólo hay que seguir los pasos y disponer de un dominio web.

Para este ejemplo vamos a usar el dominio de "unileón.es" que venía por defecto.

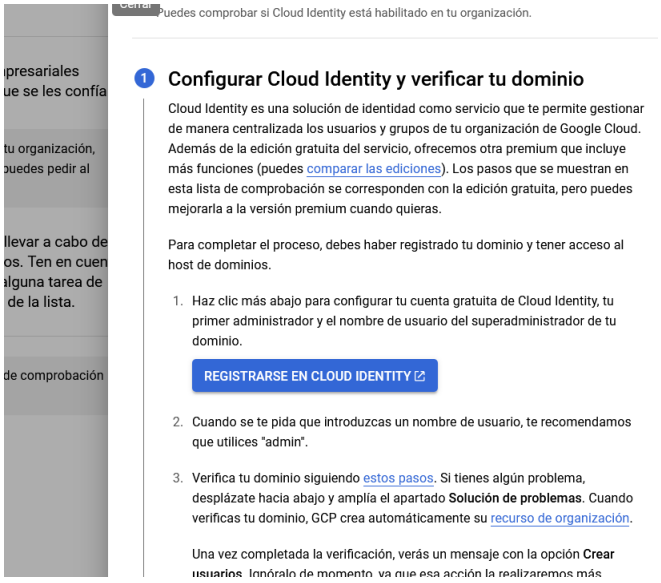


Figura 4.7: Menú de creación de organización

4.1.3. Crear un proyecto

Con la versión gratuita podemos crear hasta 12 proyectos (frente a los 24 que se podía cuando se escribió el libro).

Para acceder a los proyectos hacemos click en (1) que pondrá el nombre del proyecto que se muestra actualmente (en este caso"My First Project") y aparecerá un menú con todos los proyectos. Los proyectos están divididos por organizaciones, se puede elegir la organización que queramos usar en la parte superior izquierda de la ventana.

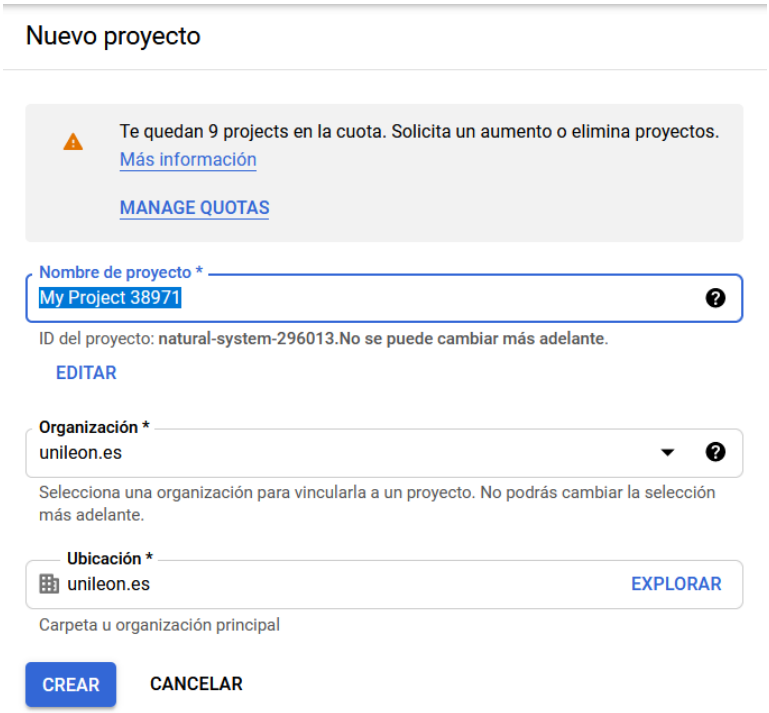


Figura 4.8: Menú de creación de proyecto

Ahora hacemos clic en "Nuevo proyecto". Ahora ponemos un nombre, seleccionamos la organización utilizada y la ubicación web. Como este es un ejemplo, dejaremos el nombre "My Project 38971". Cuando creemos el proyecto y seleccionemos la organización "unileon.es" aparecerá en la lista.

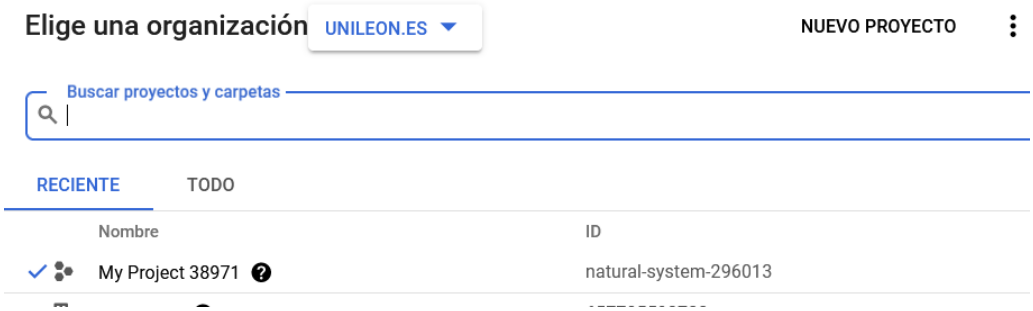


Figura 4.9: Seleccionar proyecto

Si seleccionamos el proyecto aparecerá toda la información en el panel de control, dentro del menú de inicio. Nos aparecerá entre otra información:

1. El nombre del proyecto, que podremos editar libremente.
2. ID del proyecto, que es el identificador global del proyecto. Sólo lo podremos editar una vez.
3. Número del proyecto, que es la ID auto-generada por GCP. No la podremos editar.

También aparecerá más información relevante como las solicitudes de API por segundo en tiempo real (para ver cuánto se está usando el servidor).

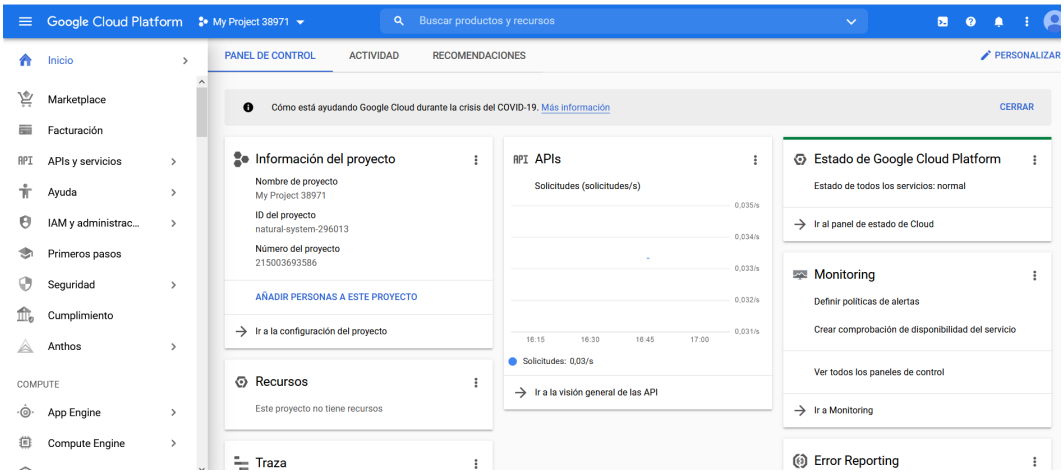


Figura 4.10: Menú de inicio con un proyecto seleccionado

4.1.4. Planes de pago

Esta parte es especialmente importante a la hora de evaluar si interesa contratar estos servicios o crear un servidor propio. También hay que analizar qué plan es el más adecuado para cada tipo de negocio.

Desde inicio hacemos click en "Facturación" y nos saldrá una vista general donde aparecen la información de pago básica, los costes para el mes en curso y los costes estimados para final de mes. Este menú se puede configurar para que muestre proyectos concretos o información de pago.

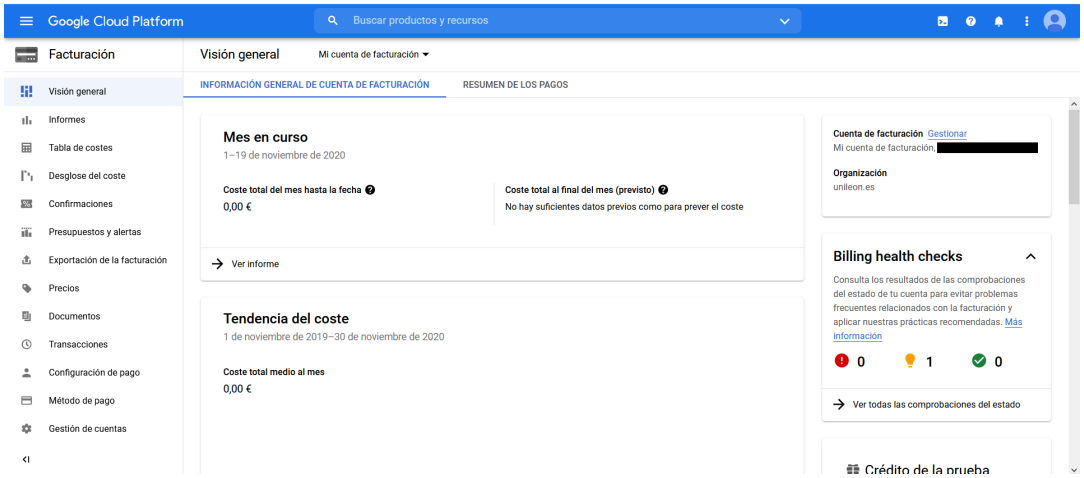


Figura 4.11: Menú de facturación

Desde Facturación se pueden realizar varias tareas en cada menú:

Informes : Muestra gráficas de costes y el presupuesto utilizado en el mes por los proyectos seleccionados. Permite ver también descuentos y el coste de los impuestos.

Tabla de costes : Muestra el resumen de costes de un mes concreto.

Desglose de costes : Muestra las gráficas de costes de un mes concreto.

Confirmaciones : Calcula los descuentos por uso continuado que se obtienen al contratar GCP de manera continuada.

Presupuestos y alertas : Permite limitar el dinero que puede usar GCP y configurar una alarma cuando se alcance un límite seleccionado.

Exportación de la facturación : Tiene varias herramientas para exportar los gastos en forma de factura. Esto es especialmente útil para empresas.

Precios : Muestra el precio de los servicios de GCP. Actualiza el precio y permite ver sólo los que se usan.

Documentos : Aquí se encuentran los documentos relacionados a los servicios utilizados. También muestra los documentos relacionados a la facturación (extractos, facturas con IVA, etc).

Transacciones : Muestra exclusivamente los documentos relacionados a la facturación.

Configuración de pago : Muestra la información relacionada al pago como el número de cuenta o el tipo de pago y permite editar alguna de esta información.

Método de pago : Añade y elimina tarjetas de crédito o débito.

Gestión de cuentas : Vincula proyectos de distintas cuentas para poder unificar el pago.

Existen descuentos para ciertos tipos de uso que ayudan a Google a distribuir sus recursos eficientemente.

Uso continuado : Contratas un servicio fijo a un precio reducido independientemente de si lo usas o no. Existen penalizaciones por cambiar el servicio.

Uso confirmado : Contratas una cantidad fija de servicio a un precio reducido. Si acabas usando más, los servicios que sobrepasen lo contratado se cobrarán a precio normal y sin descuento.

Máquina interrumpible : Contratas el servicio de forma mucho más barata a condición de que a veces Google te apagará la máquina cuando necesite el servicio. Este descuento se basa en que trabajas con el excedente de los recursos de GCP, pero cuando se necesitan esos recursos se apagarán las máquinas de este plan. Cuando se te va a apagar la máquina te llega una notificación a la máquina que puedes utilizar para preparar la máquina, a los 30 segundos llegará la señal de apagado. Este descuento es útil para aplicaciones que no necesitan un servicio contínuo.

4.1.5. Recursos de GCP

En GCP no sólo se alquila el software, también el hardware de sus servidores. Google tiene servidores dispersos en varias regiones para reducir el tiempo de respuesta y para evitar que se corte el servicio si ocurre algo en uno de los servidores.

Google divide sus servidores en cuatro regiones: Norteamérica, Sudamérica, Europa y APAC (Incluye Asia y Australia)

Cada región tiene varios servidores y también tiene servidores fuera de estas regiones como el de Sudáfrica y está expandiendo su oferta de servidores. Ya ha anunciado la próxima apertura de servidores en Catar, India, Toronto, Varsovia y Australia.

Por zona : El proyecto tiene acceso a todos los servidores en una zona dentro de una región. Es el servicio más pequeño que se puede contratar.

Regional : El proyecto tiene acceso a todos los servidores de la región (Norteamérica, Europa, ...) y a todas las zonas que lo comprenden.

Global : El proyecto tiene acceso a todos los servidores ofertados por Google en el mundo. Este tipo de recurso se utiliza para servicios de cobertura mundial si no se puede dividir por zonas.

Es importante elegir el recurso adecuado para tener suficiente cobertura sin contratar de más. También se debe escoger un servidor cercano a la zona donde se va a usar la nube para reducir la latencia. Por ejemplo, si desde España se crea una aplicación de administración para una empresa en Australia, lo ideal es que el servidor en la Zona de Australia en la región de APAC.

Podemos ver una lista de regiones y servidores en el siguiente enlace: https://cloud.google.com/abou

4.1.6. SDK de GCP

Para comunicarse con los servidores de Google es necesario usar la consola de comandos. Se puede usar la consola web o el SDK de GCP. Para esta configuración se mostrará el uso e instalación del SDK, pero para el resto del proyecto se usará la consola de comandos.

Primero descargamos la última versión del archivo "GoogleCloudSDKInstaller.exe" desde https://cloud.google.com/sdk/docs/quickstart. Instalamos el programa y finalmente abrirá una pantalla. Por defecto empieza usando el comando gcloud init, este comando configura el SDK y preguntará si queremos conectarnos con una cuenta. Escribimos "Y" para confirmar y nos abrirá una ventana en el navegador para que iniciemos sesión en google.

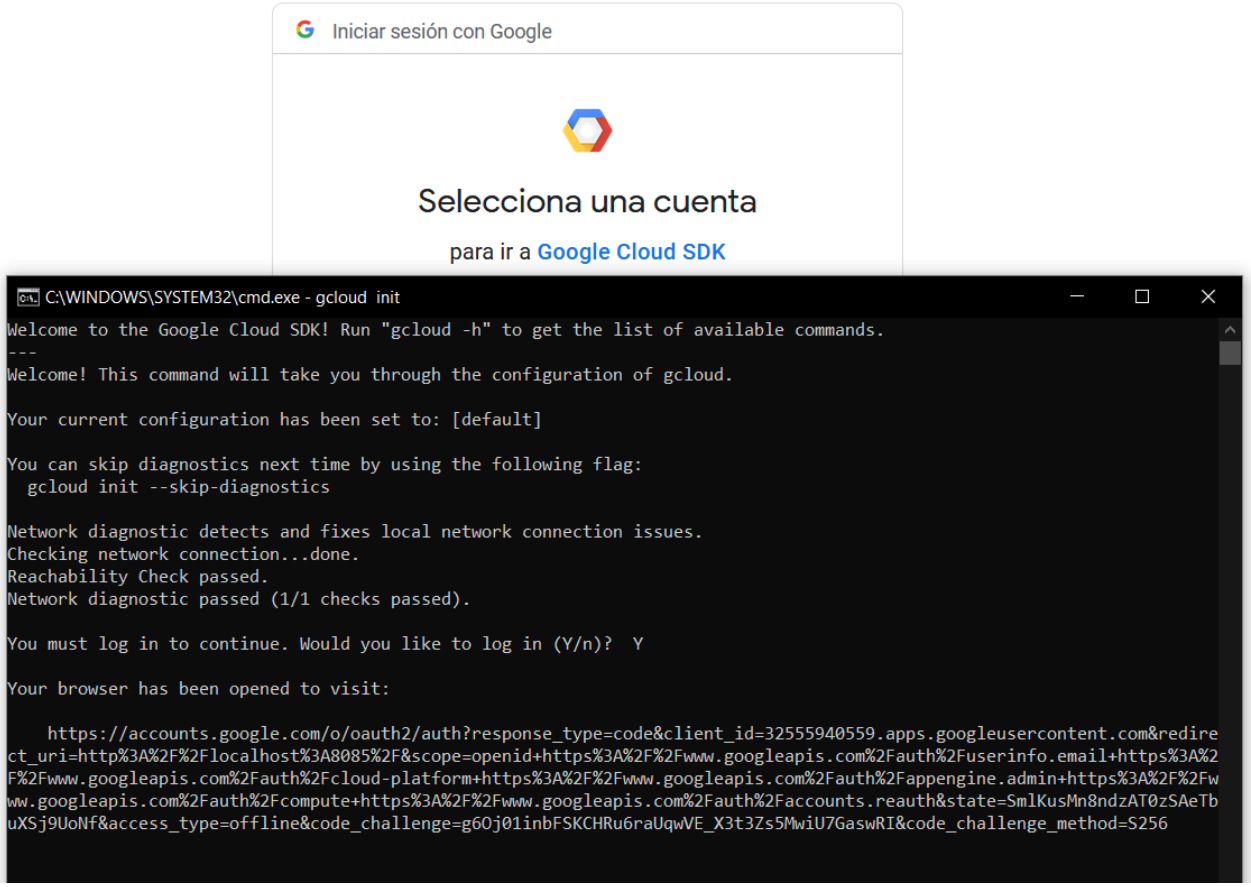


Figura 4.12: Conexión con el SDK

Tras conectarnos con nuestra cuenta por primera vez, nos pedirá acceso a ciertos permisos que necesita para funcionar. Si estamos de acuerdo con las condiciones, le damos a Permitir y la sesión de SDK estará autenticada.

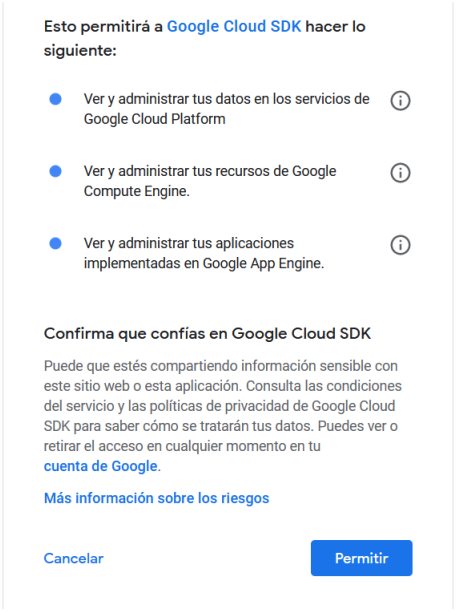


Figura 4.13: Permisos del SDK

A continuación nos pedirá elegir un proyecto. Se debe elegir por ID, asique si no nos la sabemos de memoria deberemos volver a la página web de GCP.

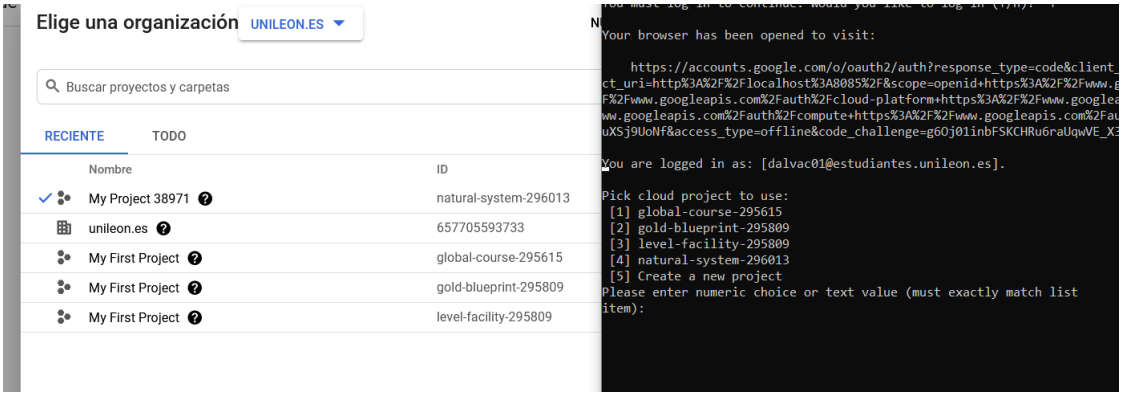


Figura 4.14: Selección de proyecto

Una vez llegados a este punto, nos pedirá que seleccionemos una región por defecto para el proyecto. En nuestro caso elegiremos el servidor de Bélgica (europe-west1) porque tiene casi todos los servicios y está cerca. Con esto ya tendríamos el proyecto configurado.

Para mostrar la ayuda de SDK se necesita el comando **gcloud compute -h** que muestra la lista de varias herramientas, funciones o temas relevantes que pueden necesitar de guía. Si se escribe **gcloud compute -help <nombre del tema>**, saldrá información sobre el tema elegido.

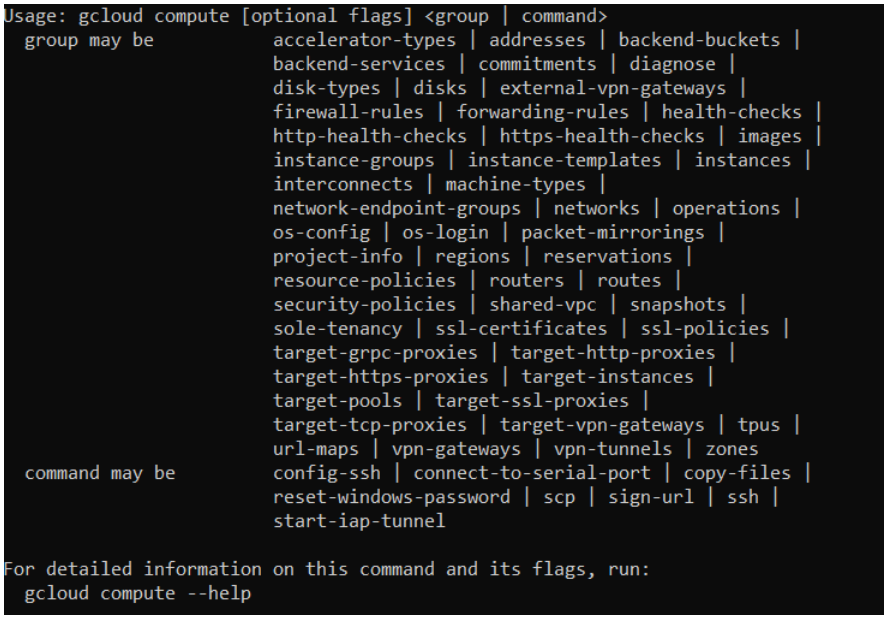


Figura 4.15: Resultado del comando **gcloud compute -h**

Se pueden usar los comandos de gcloud en otros lenguajes de script para automatizar tareas. Esto es necesario para mantener un modelo de DevOps con integración y entrega continuos, además de servir para otras funcionalidades o ayudas.

Si gcloud se queda corto, también se puede usar la API de Rest o RCP para acceder diréctamente a la nube. Con estas APIs se puede usar cualquier comando en la nube. La versión de Rest usa llamadas de JSON y la versión gRCP las usa de RCP.

Los lenguajes de programación actualmente compatibles con la API de REST son: Java, Python, PHP, .NET, JavaScript, Objetive-C, Dart, Ruby, Node.js y Go.

Los lenguajes de programación actualmente compatibles con la API de gRCP [12] son: C / .NET, C++, Dart, Go, Java, Kotlin/JVM, Node.js, Objective-C, PHP, Python y Ruby.

4.2. Instancias de GCP

En este apartado se explica cómo trabajar a un nivel básico con las instancias de GCP. Se usará de ejemplo la creación de un grupo de instancias para usar docker en ellas.

4.2.1. Crear y configurar un proyecto

Primero empezamos creando un proyecto de Google Cloud Platform y configurandolo. Este paso ya los realizamos en el anexo [Introducción a Google Cloud Platform > Crear un proyecto] y no es necesario repetirlo.

Si se quiere un proyecto nuevo, entonces usamos en la consola de comandos de GCP el comando **gcloud init** para crear y empezar a configurarla el proyecto.

```
Welcome! This command will take you through the configuration of gcloud.

Settings from your current configuration [default] are:
accessibility:
  screen_reader: 'False'
compute:
  region: europe-west1
  zone: europe-west1-b
core:
  account: dalvac01@estudiantes.unileon.es
  disable_usage_reporting: 'True'
  project: natural-system-296013

Pick configuration to use:
 [1] Re-initialize this configuration [default] with new settings
 [2] Create a new configuration
Please enter your numeric choice:  1
```

Figura 4.16: Configuración de servidor

Una vez conectados nos pedirá que elijamos un proyecto, que será establecido como proyecto por defecto. Si no tenemos proyectos nos preguntará si queremos crear uno nuevo. Si aceptamos nos pedirá introducir un nombre.

```
You can skip diagnostics next time by using the following flag:
  gcloud init --skip-diagnostics

Network diagnostic detects and fixes local network connection issues.
Checking network connection...done.
Reachability Check passed.
Network diagnostic passed (1/1 checks passed).

Choose the account you would like to use to perform operations for
this configuration:
 [1] dalvac01@estudiantes.unileon.es
 [2] Log in with a new account
Please enter your numeric choice:  1
```

Figura 4.17: Selección de proyecto

Si ya teníamos proyectos, tras elegir el proyecto nos preguntará si queremos establecer la región y zona.

```
You are logged in as: [dalvac01@estudiantes.unileon.es].

Pick cloud project to use:
 [1] global-course-295615
 [2] gold-blueprint-295809
 [3] level-facility-295809
 [4] natural-system-296013
 [5] Create a new project
Please enter numeric choice or text value (must exactly match list
item):  4

Your current project has been set to: [natural-system-296013].

Do you want to configure a default Compute Region and Zone? (Y/n)?
```

Figura 4.18: Con proyecto

```
You are logged in as: [dalvac01@estudiantes.unileon.es].

This account has no projects.

Would you like to create one? (Y/n)?  Y

Enter a Project ID. Note that a Project ID CANNOT be changed later.
Project IDs must be 6-30 characters (lowercase ASCII, digits, or
hyphens) in length and start with a lowercase letter. pruebas-gcp-docker
Waiting for [operations/cp.6560039325081803533] to finish...done.
Your current project has been set to: [pruebas-gcp-docker].
```

Figura 4.19: Sin proyecto

La región y zona es el servidor donde se aloja nuestro proyecto, lo ideal es escoger uno cercano como ya se explicó anteriormente. Nosotros elegiremos "europe-west1-b". Si no se elige una zona o es el primer proyecto, se establecerá la zona por defecto.

```
Do you want to configure a default Compute Region and Zone? (Y/n)?  Y

Which Google Compute Engine zone would you like to use as project
default?
If you do not specify a zone via a command line flag while working
with Compute Engine resources, the default is assumed.
 [1] us-east1-b
 [2] us-east1-c
 [3] us-east1-d
 [4] us-east4-c
 [5] us-east4-b
 [6] us-east4-a
 [7] us-central1-c
 [8] us-central1-a
 [9] us-central1-f
 [10] us-central1-b
 [11] us-west1-b
 [12] us-west1-c
 [13] us-west1-a
 [14] europe-west4-a
 [15] europe-west4-b
 [16] europe-west4-c
 [17] europe-west1-b
 [18] europe-west1-d
 [19] europe-west1-c
 [20] europe-west3-c
```

Figura 4.20: Elegir región del proyecto

Una vez tenemos una máquina configurada podremos cambiar la región y zona con el comando **gcloud config set compute/zone <Nombre del servidor>** (por ejemplo, "europe-west-b") para la zona y **gcloud config set compute/region <Nombre del servidor>** (por ejemplo, "europe-west") para la región

```
C:\Users\Daniel\AppData\Local\Google\Cloud SDK>gcloud config set compute/zone europe-west1-b
Updated property [compute/zone].
```

Figura 4.21: Uso de **gcloud config set compute/zone <Nombre del servidor>**

También podemos establecer el proyecto por defecto con el comando **gcloud config set project <ID del proyecto>**, esto es útil si vamos a trabajar principalmente con ese proyecto.

```
C:\Users\Daniel\AppData\Local\Google\Cloud SDK>gcloud config set project pruebas-gcp-docker
Updated property [core/project].
```

Figura 4.22: Uso de **gcloud config set project <ID del proyecto>**

4.2.2. Habilitar la API

Una vez tenemos un proyecto, vamos a crear una instancia de máquina virtual en él. Las instancias son máquinas virtuales ejecutadas dentro del servidor, podemos introducir nuestras propias imágenes, instancias de máquinas virtuales o una máquina virtual por defecto.

Para poder usar instancias en un proyecto necesitamos haber dado permiso a la API en ese proyecto. Al intentar crear una instancia, nos saldrá un mensaje de error con una URL (`https://console.developers.google.com/apis/api/compute.googleapis.com/overview?project=<IDdelproyecto>`), este enlace nos llevará a una página web donde podremos habilitar la API.

```
C:\Users\Daniel\AppData\Local\Google\Cloud SDK>gcloud compute instances create instaciacos --image-project cos-cloud --i
mage-family cos-stable
ERROR: (gcloud.compute.instances.create) Could not fetch resource:
 - Project 1058642792777 is not found and cannot be used for API calls. If it is recently created, enable Compute Engine
 API by visiting https://console.developers.google.com/apis/api/compute.googleapis.com/overview?project=1058642792777 th
en retry. If you enabled this API recently, wait a few minutes for the action to propagate to our systems and retry.
```

Figura 4.23: Mensaje de error por no tener la API habilitada

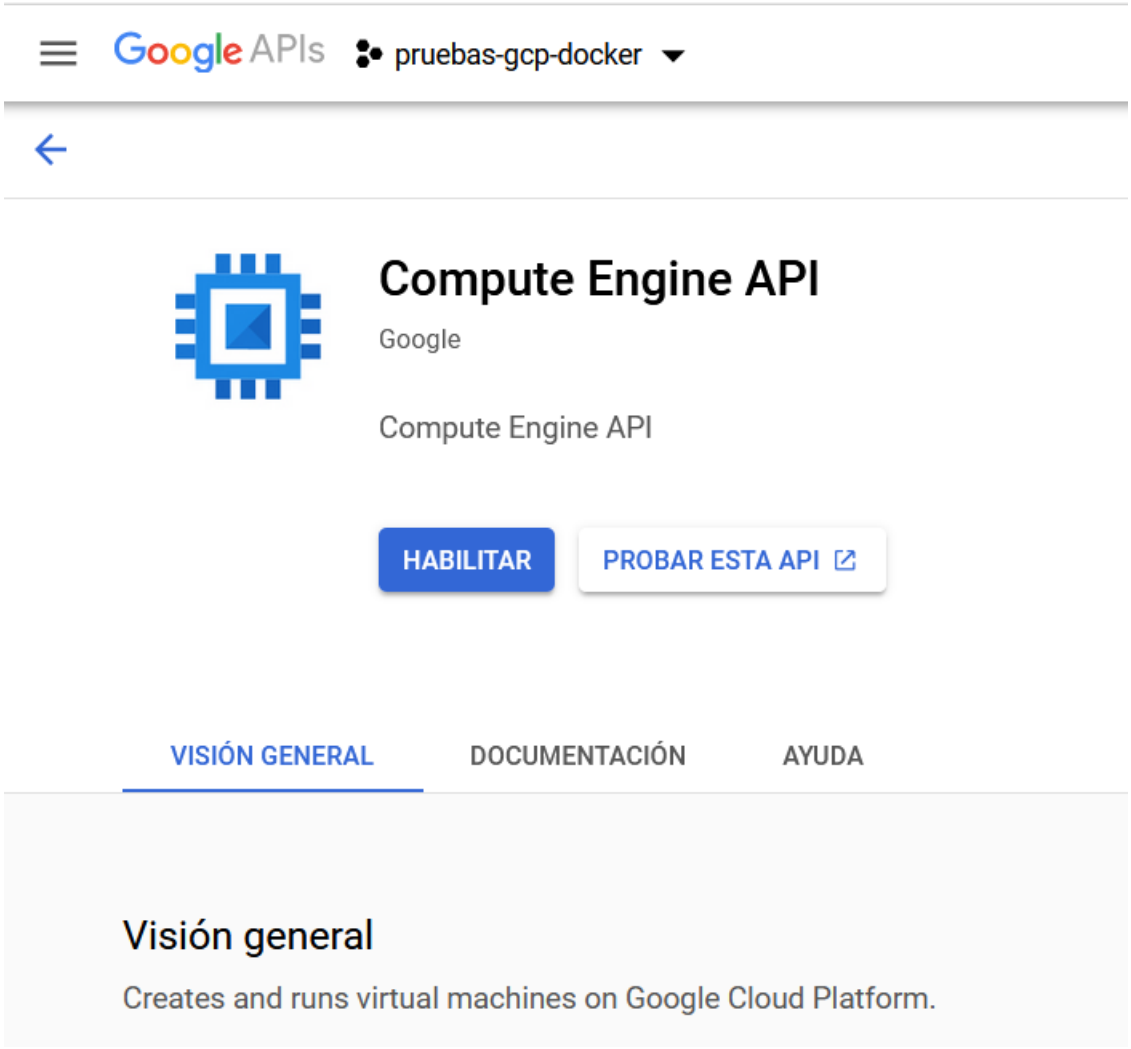


Figura 4.24: dirección de la URL

Para poder habilitar la API será necesario conectarse con la cuenta de GCP y seleccionar una cuenta de facturación.

Figura 4.25: Advertencia de facturación de la API

4.2.3. Crear una instancia de máquina virtual

Antes de crear una instancia, debemos saber qué imagen vamos a utilizar. La imagen se puede cambiar después de crear la máquina, pero es recomendable planificar de antemano para qué se va a utilizar y qué se necesita.

Para obtener una lista de máquinas virtuales por defecto de GCP se puede usar el comando de **gcloud gcloud compute images list**. Si se quiere una lista más específica se puede usar **gcloud compute images list –filter <Nombre de la imagen/proyecto/familia>** para una lista más comprensiva.

```
C:\Users\Daniel\AppData\Local\Google\Cloud SDK>gcloud compute images list
NAME                                                  PROJECT              FAMILY
 STATUS
centos-6-v20201112                                    centos-cloud         centos-6
 READY
centos-7-v20201112                                    centos-cloud         centos-7
 READY
centos-8-v20201112                                    centos-cloud         centos-8
 READY
cos-77-12371-1096-0                                   cos-cloud            cos-77-lts
 READY
cos-81-12871-1218-0                                   cos-cloud            cos-81-lts
 READY
cos-85-13310-1041-28                                  cos-cloud            cos-85-lts
 READY
cos-beta-85-13310-1041-1                              cos-cloud            cos-beta
 READY
cos-dev-88-15480-0-0                                  cos-cloud            cos-dev
 READY
debian-10-buster-v20201112                            debian-cloud         debian-10
 READY
debian-9-stretch-v20201112                            debian-cloud         debian-9
 READY
fedora-coreos-32-20201104-3-0-gcp-x86-64              fedora-coreos-cloud  fedora-coreos-stable
 READY
fedora-coreos-33-20201116-2-0-gcp-x86-64              fedora-coreos-cloud  fedora-coreos-testing
```

Figura 4.26: gcloud gcloud compute images list

```
C:\Users\Daniel\AppData\Local\Google\Cloud SDK>gcloud compute images list --filter cos-cloud
NAME                          PROJECT    FAMILY      DEPRECATED  STATUS
cos-77-12371-1096-0           cos-cloud  cos-77-lts              READY
cos-81-12871-1218-0           cos-cloud  cos-81-lts              READY
cos-85-13310-1041-28          cos-cloud  cos-85-lts              READY
cos-beta-85-13310-1041-1      cos-cloud  cos-beta                READY
cos-dev-88-15480-0-0          cos-cloud  cos-dev                 READY
cos-stable-85-13310-1041-28   cos-cloud  cos-stable              READY
```

Figura 4.27: gcloud compute images list –filter cos-cloud

Las imágenes se dividen por proyectos, cada proyecto tiene varias familias y cada familia varias versiones. Los proyectos son una imagen general, las familias son las variaciones de ese proyecto (incluyendo las versiones de prueba) y cada imagen de una familia es la versión. Lo habitual es escoger del proyecto que queramos, la última versión de la familia de imágenes elegida.

En total hay decenas de imágenes, pero vamos a recortar nuestra selección a las más apropiadas para Docker.

Container-Optimized OS : Es una versión del proyecto Chromium OS creada por google específicamente para el uso de Docker. El sistema operativo está optimizado para el uso de contenedores y vienen preinstalados Docker y Kubernetes. El proyecto de la imagen es *cos-cloud* y la familia que nos interesa es *cos-stable*.

CoreOS : Es un sistema operativo basado en fedora (linux), optimizado para el uso de contenedores como el de Docker. Incluye Docker, rkt (Una alternativa a Docker) y Kubernetes. El proyecto es *fedora-coreos-cloud* y la familia de imágenes que nos interesa es *fedora-coreos-stable*

Ubuntu : Está basada en la versión más actual de Ubuntu (18.04 a fecha de este proyecto), incluye LXD. El proyecto es *ubuntu-os-cloud* y la familia que nos interesa en este caso es *ubuntu-2010*

Windows : Es una versión simplificada y ligera de windows server diseñada para usar Docker, el cual lleva incluido. El proyecto es *windows-cloud* y la familia que nos interesa es *windows-2019-core-for-containers*, o la versión más actual si la hubiera. Las máquinas de windows no se pueden usar con la licencia de pruebas o gratuita.

Una vez tenemos la API activada podremos crear una máquina virtual con el comando **gcloud compute instances create <Nombre de la Instancia>** o mediante la plataforma de GCP en el menú "Compute Engine". El nombre sólo puede contener letras minúsculas y números.

Con el comando de gcloud creará una instancia por defecto en la zona del proyecto con una imagen de debian-10 de la familia de proyectos de debian-cloud. También se puede crear directamente con la imagen elegida añadiendo los argumentos de **–image-proyect <Nombre del proyecto>** y **–image-family <Nombre de la familia>**. Se puede sustituir **–image-family** por **–image <nombre de la imagen>** si se quiere una versión específica que no sea la más actual de esa familia de imágenes. La lista completa de argumentos se encuentra en la documentación de GCP: `https://cloud.google.com/sdk/gcloud/reference/compute/instances/create`.

```
C:\Users\Daniel\AppData\Local\Google\Cloud SDK>gcloud compute instances create instanciacos --image-project
cos-cloud --image-family cos-stable
Created [https://www.googleapis.com/compute/v1/projects/natural-system-296013/zones/europe-west1-b/instances
/instanciacos].
NAME          ZONE           MACHINE_TYPE   PREEMPTIBLE  INTERNAL_IP  EXTERNAL_IP     STATUS
instanciacos  europe-west1-b  n1-standard-1               10.132.0.5   35.195.204.212  RUNNING

C:\Users\Daniel\AppData\Local\Google\Cloud SDK>
```

Figura 4.28: gcloud compute instances create instanciacos –image-project cos-cloud –image-family cos-stable

4.2.4. Comandos básicos de instancias

Los comandos más básicos para para trabajar con instancias son los siguientes: **gcloud compute instances ...**

1. **... create** para crear una instancia.
2. **... delete <Nombre de instancia>** para borrar una instancia.
3. **... list** para ver las instancias actuales.
4. **... start <Nombre de instancia>** para arrancar la máquina virtual.
5. **... stop <Nombre de instancia>** para parar la máquina virtual. Esto es de vital importancia porque las instancias paradas no tienen coste de mantenimiento.
6. **... reset <Nombre de instancia>** reinicia la instancia.
7. **... update <Nombre de instancia>** para actualizar la imágen a la última versión estable de su familia.
8. **... attach-disk <Nombre de instancia>** vincula un disco virtual a la instancia.
9. **... dettach-disk <Nombre de instancia>** desvincula un disco virtual a la instancia.

Se pueden hacer más cosas con **gcloud compute intances** como añadir/eliminar tags y etiquetas o cambiar la configuración. La lista completa de comandos se encuentra en la documentación: `https://cloud.google.com/sdk/gcloud/reference/compute/instances`.

gcloud config ...

1. **... list** muestra las configuraciones activas de la cuenta y el proyecto activo.
2. **... set project <Id del Proyecto>** cambia el proyecto activo.

4.2.5. Conectar con una instancia

Una vez se tienen las instancias, se puede conectar a ellas mediante SSH. Se puede hacer mediante la consola de GCP en "Compute Engine/Instancias VM" haciendo click en el botón "SSH" de la instancia deseada. También se puede hacer mediante gcloud con el siguiente comando.

gcloud compute ssh <Nombre de instancia> Permite conectarse a la instancia mediante ssh. Es el comando que nos permite trabajar con la instancia.

La primera vez que se ejecuta el comando detectará que no existe un par de claves SSH y te preguntará si quieres generar una nueva.

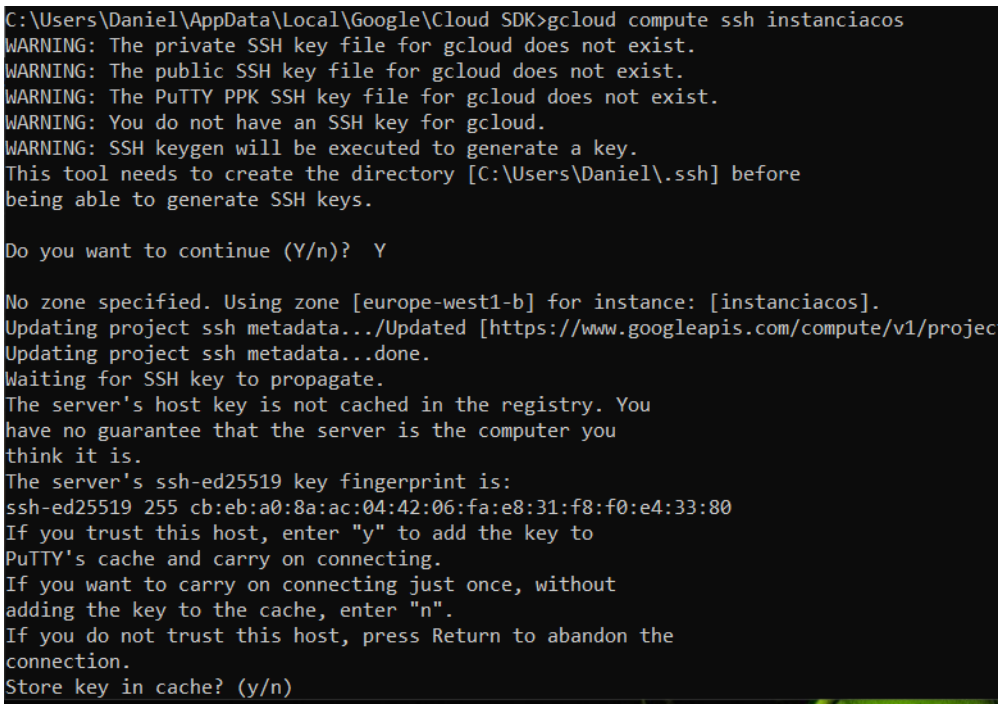


Figura 4.29: Primera ejecución de **gcloud compute ssh instanciacos**

Una vez creada la clave se abrirá una ventana con la consola de comandos de la instancia.

Figura 4.30: Consola de comandos de la instancia de COS

4.3. Docker

En este apartado se explicará el funcionamiento básico de docker y cómo usar docker a un nivel básico en GCP. Se debe aclarar desde el principio que sólo vale para cosas pequeñas y sus limitaciones.

4.3.1. ¿Qué es Docker?

Docker es una plataforma abierta para desarrollar, enviar o usar aplicaciones. Funciona empaquetando y usando aplicaciones en un entorno individual y aislado del equipo, que se llama contenedor.

Funciona de manera similar a una máquina virtual pero en vez virtualizar un ordenador al completo, se virtualiza sólo lo mínimo y necesario para la ejecución. No sustituye por completo a las máquinas virtuales, pero sí en muchos casos de uso. Docker tiene principalmente tres ventajas: Ligereza, Portabilidad y Sencillez.

Los contenedores son más ligeros tanto computacionalmente como en el disco duro, en relación a una máquina virtual. No necesita de un Hypervisor (que es una plataforma que permite usar varios sistemas operativos simultáneamente) y usan directamente el kernel del host. Esto significa un descenso de consumo de memoria de alrededor de un 80 % [13] respecto a máquinas virtuales. Esto también significa que no es necesario instalar un sistema operativo por cada contenedor.

Docker permite trabajar con mayor portabilidad. Puede empaquetar una aplicación en un contenedor y pasarla a otro entorno sin problemas de compatibilidad. También puede subir un contenedor a un repositorio o trabajar directamente en la nube.

Como las máquinas de docker se ejecutan por separado, también añaden seguridad a un sistema (tanto de datos como frente a ciberataques). Si por ejemplo, una máquina tuviera una vulnerabilidad o sufriera un error crítico, no afectaría a la máquina original.

Docker es una buena adicción a DevOps. En el caso de este proyecto se usará principalmente para probar las compilaciones sin poner en riesgo al sistema y para encapsular el código de forma que reduzca la probabilidad de error cada vez que un código cambia de manos.

4.3.2. Trabajando con Docker

Docker consta de tres partes: El servidor, la API de REST y el cliente. El servidor contiene al demonio de docker, que es un proceso no iterativo que está en pausa hasta que se le necesita. La API de REST administra la red y el volumen de datos. El cliente es el que contiene el registro de imágenes y los contenedores.

Las recursos que se usan principalmente en Docker son:

Imágenes : Las imágenes son capturas de un sistema operativo, que pueden usarse para crear contenedores de Docker idénticos.

Contenedores : Los contenedores son unidades de software estandarizadas que permiten ejecutar una imagen de forma similar a una máquina virtual.

Registro de imágenes : Permite almacenar imágenes en la nube para poder recuperarlas. El más habitual es Dockerhub, pero existen otros registros públicos como el de Google (Google Cloud Registry). También es posible crear un repositorio privado.

Dockerfile : Un archivo de configuración que permite generar y configurar un contenedor directamente. Descarga la imágen directamente del registro de imágenes.

Grupos (Clusters) : Agrupaciones de máquinas de Docker conectados configurados para trabajar como uno sólo. Para administrar un grupo se usa un orquestador de contenedores como Kubernetes o Docker swarm.

El ejemplo más básico del uso de docker sería crear un contenedor mediante una imagen, compilarlo y ejecutarlo. Las acciones más usadas son las siguientes:

docker search : Busca una imagen en Docker Hub o un repositorio de imágenes.

docker run : Ejecuta un comando en un nuevo contenedor.

docker start : Inicializa uno o más contenedores.

docker stop : Detiene uno o más contenedores.

docker build : Compila una imagen desde un archivo Dockerfile.

docker pull : Descarga una imagen de un repositorio o registro.

docker push : Carga una imagen a un repositorio o registro.

docker export : Exporta el sistema de ficheros de un contenedor como un archivo tar.

docker exec : Ejecuta un comando en una contenedor en ejecución.

docker attach : Añade un contenedor en ejecución.

docker commit : Crea una nueva imagen de un contenedor, guardando los cambios realizados.

4.3.3. Docker en GCP

A continuación se mostrará como usar docker a un nivel básico dentro de una máquina de GCP. No obstante, GCP limita los contenedores dentro de la instancia y no permite ejecutarlos en segundo plano (limitando a un sólo contenedor). Esto inhabilita las instancias de GCP para la mayoría de proyectos con docker. Si se quiere trabajar con docker en GCP lo ideal es usar kubernetes.

En este apartado se explicará igualmente como breve introducción a Docker y para los proyectos para los que pueda ser beneficioso. Concretamente los proyectos que sólo necesiten una instancia de Docker activa al mismo tiempo.

Para mostrar cómo usar Docker a un nivel básico, vamos a crear un contenedor de una imagen de ubuntu.

Conectamos a una de las instancias que hemos creado (aunque los comandos que vamos a usar son válidos para cualquier máquina con docker instalado).

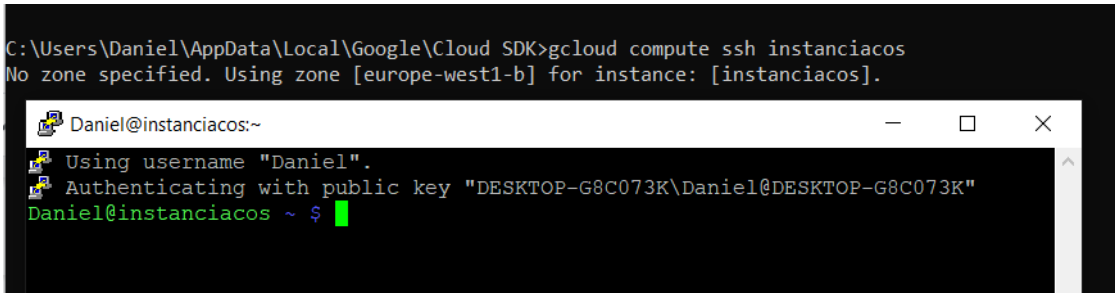


Figura 4.31: gcloud compute ssh

El primer paso es instalar docker en la máquina si no está ya. Se puede comprobar mediante el comando **docker –version**. Como estamos usando la instancia de COS, Docker ya está instalado.

```
Daniel@instanciacos ~ $ docker --version
Docker version 19.03.9, build 9d98839
```

Figura 4.32: docker –version

Con docker instalado, nos hace falta decidir qué imagen queremos descargar. Para buscar las imágenes oficiales de docker, podemos usar el comando **docker search <Nombre de la imagen>** para buscar imágenes en el docker hub (donde se alojan las imágenes oficiales de docker).

También se puede usar el comando Git (si se tiene instalado git en la máquina) o curl para descargar la imágen de un repositorio de Git.

Es posible borrar imágenes con el comando **docker images rm <nombre de la imágen>**.

```
Daniel@instanciacos ~ $ docker search ubuntu
NAME                                                      DESCRIPTION
                                STARS               OFFICIAL            AUTOMATED
ubuntu                                                    Ubuntu is a Debian-bas
ed Linux operating sys…   11601               [OK]
dorowu/ubuntu-desktop-lxde-vnc                            Docker image to provid
e HTML5 VNC interface …   480                                     [OK]
websphere-liberty                                         WebSphere Liberty mult
i-architecture images …   264                 [OK]
rastasheep/ubuntu-sshd                                    Dockerized SSH service
, built on top of offi…   250                                     [OK]
consol/ubuntu-xfce-vnc                                    Ubuntu container with
"headless" VNC session…   228                                     [OK]
ubuntu-upstart                                            Upstart is an event-ba
sed replacement for th…   109                 [OK]
neurodebian                                               NeuroDebian provides n
euroscience research s…   76                  [OK]
1and1internet/ubuntu-16-nginx-php-phpmyadmin-mysql-5      ubuntu-16-nginx-php-ph
pmyadmin-mysql-5          50                                      [OK]
ubuntu-debootstrap                                        debootstrap --variant=
minbase --components=m…   44                  [OK]
open-liberty                                              Open Liberty multi-arc
```

Figura 4.33: docker search ubuntu

Ahora vamos a descargar la primera imagen, la de ubuntu base. Para ello usaremos el comando **docker pull <Nombre de imagen>**. Este paso se ejecuta automáticamente la primera vez que se ejecuta un contenedor.

```
Daniel@instanciacos ~ $ docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
da7391352a9b: Pull complete
14428a6d4bcd: Pull complete
2c2d948710f2: Pull complete
Digest: sha256:c95a8e48bf88e9849f3e0f723d9f49fa12c5a00cfc6e60d2bc99d87555295e4c
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
```

Figura 4.34: docker pull ubuntu

Para ejecutar un contenedor usamos el comando **docker run <Nombre de la imagen>**. Veremos que nos aparece la consola de comandos de ubuntu.

```
Daniel@instanciacos ~ $ docker run -it ubuntu
root@17b4e7aebeb4:/# ls
bin   dev  home  lib32  libx32  mnt  proc  run   srv  tmp  var
boot  etc  lib   lib64  media   opt  root  sbin  sys  usr
root@17b4e7aebeb4:/#
```

Figura 4.35: docker run ubuntu

Se puede ver cuántas máquinas de docker se están ejecutando en el momento con el comando **docker ps** o **docker containers ls**. Así mostrará todos los contenedores activos, pero si añadimos el argumento **-a** mostrará todos los contenedores.

Se puede pausar un contenedor mediante el comando **docker stop <ID del contenedor arrancado>**. Si se quiere borrar un contenedor específico se usaría el comando **docker rm <ID del contenedor en pausa>** que requiere que el contenedor esté en pausa. También se puede usar **docker prune** para formatear docker en la máquina.

```
Daniel@instanciacos ~ $ docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
            PORTS               NAMES
789109a1d49a        ubuntu              "/bin/bash"         6 seconds ago       Exited (0) 5 s
econds ago                      sad_ritchie
28142d015def        ubuntu              "/bin/bash"         9 seconds ago       Exited (0) 8 s
econds ago                      nervous_villani
Daniel@instanciacos ~ $ docker container rm 789109a1d49a
789109a1d49a
Daniel@instanciacos ~ $ docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
            PORTS               NAMES
28142d015def        ubuntu              "/bin/bash"         27 seconds ago      Exited (0) 26
seconds ago                     nervous_villani
Daniel@instanciacos ~ $
```

Figura 4.36: Borrando un contenedor mediante **docker container rm <ID>**

Normalmente podemos ejecutar comandos en un contenedor sin entrar en la máquina, pero como ya se advirtió al principio GCP no permite ejecutar en segundo plano contenedores dentro de sus instancias.

El proceso de poderse (si lo estás probando fuera de GCP) se empieza arrancando el contenedor con el comando run añadiendo el argumento **-d** en el comando run (hace que se ejecute en segundo plano). Si ya se había creado el contenedor se usaría el comando **docker start <nombre o ID del contenedor>** para iniciarlo. A continuación se usaría el comando **docker exec <argumento>** que requiere que la máquina esté arrancada.

Un ejemplo del comando sería **docker exec apt-get install php5**, este comando ejecutaría el comando apt-get para instalar php5 por consola dentro del contenedor.

```
Daniel@instanciacos ~ $ docker start nervous_villani
nervous_villani
Daniel@instanciacos ~ $ docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
      PORTS               NAMES
28142d015def        ubuntu              "/bin/bash"         7 minutes ago       Exited (0) 2 seconds
 ago                       nervous_villani
Daniel@instanciacos ~ $
```

Figura 4.37: No se pueden usar contenedores en segundo plano en GCP

Los contenedores de docker pueden almacenar datos, pero no deberían usarse como memoria porque son particularmente susceptibles a ser borrados. Para guardar los datos se debería regularmente exportar mediante el comando el comando **docker export -output="<Ruta o dirección de repositorio donde se quiere guardar>" <ID o Nombre del contenedor>** y volverlos a importar mediante el comando **docker import <Ruta o dirección de repositorio de origen>**. Export/import exporta o importa todo el sistema de ficheros del contenedor a un archivo local o a una dirección del repositorio.

```
Daniel@instanciacos ~ $ mkdir DockerSaves
Daniel@instanciacos ~ $ docker export --output="DockerSaves/nervousVillani.tar" nervous_villani
Daniel@instanciacos ~ $ ls
.bash_history  .bash_profile  .ssh/          DockerSaves/
.bash_logout   .bashrc        Docker/
Daniel@instanciacos ~ $ ls DockerSaves/
nervousVillani.tar
```

Figura 4.38: Exportación de contenedor.

Por último se puede guardar el contenedor como una imágen y subirlo a un repositorio con los siguientes comandos:

docker commit <ID o Nombre del contenedor> <Dirección de la imagen> : El commit se realizaría sobre la imagen seleccionada. Sólo quedaría subirla mediante un push al repositorio.

docker push <Nombre del host>/<ID del proyecto>/<Dirección de la imagen>:<Tag : El tag es opcional.

4.3.4. Dockerfile

Todo el proceso de instalación se puede automatizar de dos formas: Usando una imagen preparada a mano mediante los comandos import/export, o usando un dockerfile.

Los dockerfile son documentos de texto con una serie de instrucciones que se ejecutan ordenadamente al compilar el dockerfile. Los dockerfile tienen los siguientes argumentos:

\# : Es el símbolo para comentarios, lo vaya detrás de la almohadilla no se ejecuta. Ejemplo: # **Este texto no se ejecutará**

FROM <imagen> : Este debe ser el primer comando ejecutado del dockerfile. Crea un contenedor en base a una imagen de Docker Hub. Puede usar los argumentos para escoger una versión específica, por defecto se escoge "latest", la última estable: **FROM <imagen>:<tag>** o **FROM <imagen>@<digest>**. Ejemplo: **FROM ubuntu:latest**

MAINTAINER <nombre> : Sirve para establecer el autor del contenedor. Ejemplo: **MANTAINER Daniel**

RUN <comando> <parametro1> <parametro2> : Ejecuta el comando en el sistema de la imágen (generalmente el host). Ejemplo: **RUN npm install**

CMD <comando> <parametro1> <parametro2> : Ejecuta el comando dentro del contenedor. Ejemplo: **CMD mkdir carpeta**

LABEL <clave>=<valor> : Añade un nuevo metadato a la imagen con el valor indicado. Ejemplo: **LABEL version=v1.1.2**

EXPOSE <puerto> : Indica a Docker que el contenedor va a escuchar a ese puerto durante la ejecución. Ejemplo: **EXPOSE 8080**

ENV <clave>=<valor> : Cambia el valor de un metadato existente. Ejemplo: **ENV version=v1.1.3**

ADD <archivo> <destino> : Añade un archivo a la dirección especificada dentro del contenedor. El archivo puede ser una ruta o una URL. Aunque se pueda usar una URL es más correcto usar curl/wget y COPY. ADD se usa frente a COPY para auto-descomprimir archivos .tar . Ejemplo: **ADD /usr/file.tar carpeta**

COPY <archivo> <destino> : Añade un archivo a la dirección especificada dentro del archivo. El archivo debe ser una ruta local. En términos generales es mejor usar COPY antes que ADD, salvo que se necesite una de las características de ADD. Ejemplo: **COPY /usr/file.tar carpeta**

ENTRYPOINT <ejecutable> <parametro1> <parametro2> : Permite configurar un contenedor como un ejecutable. Al usar el ejecutable con el comando docker run <ejecutable> se ejecutarán el dockerfile desde el último ENTRYPOINT. Ejemplo: **ENTRYPOINT ["EjecutableUbuntu"]**

VOLUME ["<ruta>"]: Crea un punto de montaje.La carpeta creada será tratada como si estuviera montado un dispositivo externo conectado directamente con el host. El objetivo de este comando es designar las zonas del contenedor que van a albergar bases de datos, configuraciones o cualquier tipo de dato que más adelante se desee extraer. Ejemplo: **VOLUME ["/data"]**

USER <nombre de usuario o UID> : Sirve para conectarse dentro del contenedor como un usuario específico. Se puede especificar el grupo de usuarios o el GID. Ejemplo: **USER Manuel**

WORKDIR <"ruta"> : Establece desde dónde se ejecutan los comandos. Ejemplo: **WORKDIR /src/**

ARG <nombre>=<valor por defecto> : Crea una variable y se le puede asignar un valor por defecto. Ejemplo: **ARG build**

ONBUILD <Otra instrucción de dockerfile> : Realiza una instrucción de dockerfile desde el contenedor creado, tras finalizar la compilación. Si se crea un contenedor mediante FROM, el contenedor se crea dentro del contenedor. Ejemplo: **ONBUILD COPY /usr/file.tar carpeta**

STOPSIGNAL <señal> : Manda la señal "exit" a la señal elegida. Si se elige el número de su posición en la tabla de syscall del kernel o el SIGNAME de la señal que se quiere detener. Ejemplo: **STOPSIGNAL 9**

HEALTHCHECK <opciones> CMD <comandos> : Indica cómo se debe comprobar si el contenedor sigue activo. Ejemplo: **HEALTHCHECK –interval=5m –timeout=3s**

SHELL [<ejecutable> <parámetros>]: Permite ejecutar comandos con otro tipo de terminal. Se pueden usar shells como la de windows ["cmd", "/S", "/C"] o linux ["/bin/sh", "-c"]. Ejemplo: **SHELL ["cmd", "/S", "/C"]**

A continuación crearemos un archivo Dockerfile para la aplicación web del proyecto:

```
# Descargar desde Docker Hub, la imagen pública de Node versión 10.
FROM node:10
# Establecer como directorio de trabajo (el directorio por defecto) la ruta
/usr/src/app
WORKDIR /usr/src/app
# Copia en la máquina de Docker el archivo 'server.js'. El segundo argumento
es la ruta
#  donde se encuentra el archivo respecto al Dockerfile (en la misma carpeta).
COPY server.js ./
# Copia todos los archivos que comiencen por 'package' y acaben en '.json'
COPY package*.json ./
# Copia la carpeta './www' dentro de la máquina con el nombre 'www'.
ADD www ./www
# Ejecuta los comandos en la máquina al crearse. En este caso instala npm y
express.
RUN npm install
RUN npm install --save express
# Abre el puerto 8081 para poder desplegar la aplicación.
EXPOSE 8081
# Al arrancar la máquina, ejecuta el siguiente comando de terminal (arranca el
servidor)
CMD [ "node", "server.js" ]
```

Figura 4.39: Ejemplo de Dockerfile

4.4. Kubernetes

En este apartado se explicará qué es la herramienta kubernetes y cómo se trabaja con ella. También se añadirá un pequeño ejemplo de cómo usar kubernetes en GCP.

4.4.1. ¿Qué es Kubernetes?

Kubernetes es una plataforma portable de código abierto que sirve para administrar y escalar (comúnmente referido como orquestar en este ámbito) contenedores simultáneamente. Fue desarrollado por Google y es administrado por la fundación "Cloud Native Computing Foundation". [6]

La estructura interna de Kubernetes se divide en Nodos y Clusters. Los nodos son contenedores y pueden ser nodos maestros o nodos trabajadores. Los nodos maestros se encargan de crear, administrar y borrar los nodos trabajadores. Los nodos trabajadores realizan las tareas. Los nodos se agrupan en clusters. Todos los clusters están formados de al menos un nodo maestro y una cantidad indefinida de nodos trabajadores.

Los recursos con los que se trabajan principalmente en Kubernetes [1] son:

Pods : Bloques de contenedores con la misma IP. Son la herramienta del usuario para ejecutar tareas.

Etiquetas : Pares de clave-valor que sirven para separar y organizar las diferentes tareas de un cluster.

Selectores de etiquetas : Una versión más avanzada de las etiquetas que permite agrupar diferentes tipos de objetos.

Controladores : Procesos indefinidos que administran el resto de componentes.

Servicios : Abstracción usada para definir un grupo de Pods como un servicio en red. Generalmente se define dentro de un selector de etiquetas.

4.4.2. Trabajando con kubernetes

Vamos a analizar kubernetes desde un punto de vista más práctico. Una vez tenemos un cluster funcional podemos empezar a usar kubernetes.

Pods

La unidad más pequeña que podemos manejar en Kubernetes son los pods, que son grupos de uno o más contenedores. Se pueden agrupar los pods para crear un **stack**. Los pods se crean y se destruyen según sea necesario. Es habitual ejecutar en los pods NGINX, un tipo de servidor ligero.

Servicios

Son una forma abstracta de establecer la estructura de un grupo de pods como un único servicio web. Cada pod tienen su propia IP, pero comparte el nombre de DNS con el resto de su grupo. Usar servicios permite administrar más fácilmente los pods, que se crean y destruyen dinámicamente. [6]

Para crear un servicio se debe crear un archivo YAML que identifique el servicio y establezca qué puertos va a usar. La estructura básica sería la siguiente:

```
kind: Service
apiVersion: v1
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
    port: 80
    targetPort: 8080
```

Figura 4.40: Ejemplo de archivo YAML definiendo un servicio.

"Metadata" indica información sobre el propio servicio. "Spec" configura los pods elegidos por "selector" y les aplica la configuración web de "ports".

Con este código se establece que es un servicio (kind: Service), llamado jenkins-service que se comunica con todos los pods con la etiqueta "jenkinsApp". Los pods a los que llama el servicio usan un protocolo TCP, escuchan al puerto 8080 y exponen el puerto 80.

Es posible no usar un selector, pero en ese caso se debe crear un archivo "Endpoint" indicando la IP y el puerto de manera manual. Si se usa el selector, el endpoint se genera automáticamente.

Cuotas

Las cuotas son formas de limitar el tráfico en el cluster de Jenkins. Se modifica con el archivo YAML del Deployment. Se activa añadiendo en el API server, en

el argumento "–enable-admission-plugins=", ResourceQuota como uno de sus argumentos.

Es importante establecer cuotas ajustadas a las necesidades para evitar una sobrecarga en el servidor. Teóricamente no debería dar problemas si todos los usuarios usaran de manera normal el servicio, pero siempre hay que tomar precauciones ante un error humano o contra un ataque de denegación de servicio. También permite distribuir equitativamente los recursos del servidor entre todos los equipos.

Las cuotas pueden limitar el uso de CPU, la memoria y el almacenamiento usado por los pods de un mismo servicio.

```
resources:
  limits:
    cpu: 500m
    memory: 1500Mi
  requests:
    cpu: 500m
    memory: 1500Mi
```

Figura 4.41: Establecer cuotas de un servicio en un archivo YAML.

Despliegues En los despliegues se establece el estado de los pods y de los ReplicaSet (que se encarga de sincronizar a los pods). En el documento de "Deployment" se establece el estado deseado de ejecución y el controlador de despliegue actualiza kubernetes para alcanzar ese estado de una forma controlada y segura.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

Figura 4.42: Estructura básica de un archivo de despliegue en un archivo YAML.

Este es un deployment del grupo nginx (metadata.labels.app). El deployment crea tres réplicas de pods (.spec.replicas), concretamente del pod "nginx" (.spec.selector.matchLabels.app). Los pods creados serán como se indica en .template.spec.container, contenedores llamados "nginx" con la imágen de dockerHub "ngnix" versión 1.14.2 y el puerto 80 abierto.

4.5. Ejemplo práctico de Kubernetes en GCP

Para ver cómo se usa la herramienta usaremos un proyecto de demostración. Ejecutaremos la aplicación web 'hello-server' con Go en la API de GCP.

Para ver esta herramienta en más profundidad recomiendo buscar la página web de Kubernetes, los tutoriales están muy bien explicados y recomiendo usarlos para

aprender a usar la herramienta. Incluyen ejemplos interactivos paso a paso y esquemas visuales que permiten entender el funcionamiento interno de la herramienta.

4.5.1. Preparando un proyecto para usar Kubernetes

Primero vamos a crear un proyecto para Kubernetes como vimos en el apartado [Trabajar con instancias>Crear un proyecto]. Llamaremos a este proyecto "Proyecto-Kubernetes". Activaremos este proyecto con el comando **gcloud config set project proyecto-kubernetes-298809** y seleccionaremos su zona en "europe-west1-b" mediante el comando **gcloud config set compute/zone europe-west1-b**. También deberemos habilitar la API de "Compute Engine API".

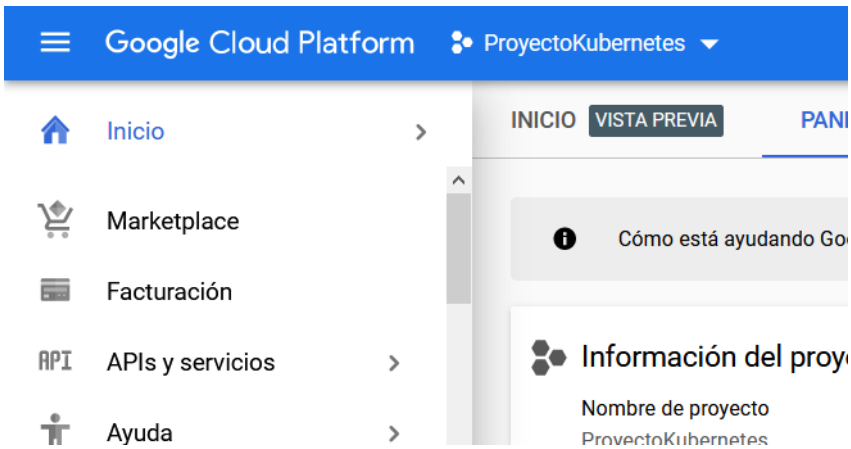
```
C:\Users\Daniel\AppData\Local\Google\Cloud SDK>gcloud config set project proyecto-kubernetes-298809
Updated property [core/project].

C:\Users\Daniel\AppData\Local\Google\Cloud SDK>gcloud config set compute/zone europe-west1-b
Updated property [compute/zone].
```

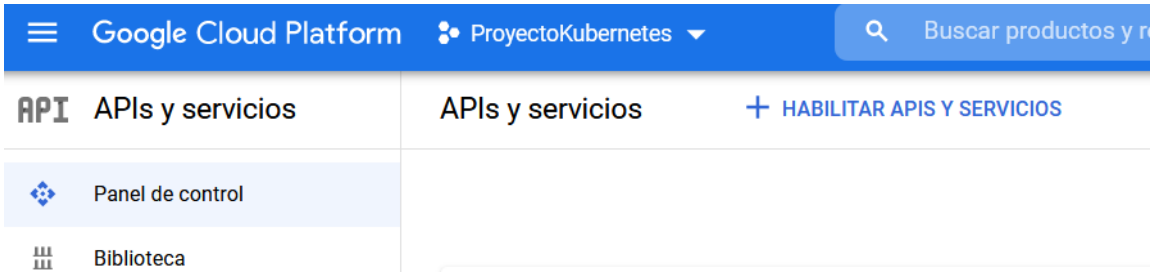
Figura 4.43: Configuración del nuevo proyecto

Para usar Kubernetes también hay que activar la API. Primero nos aseguramos de que estamos en el proyecto correcto y seguimos los siguientes pasos:

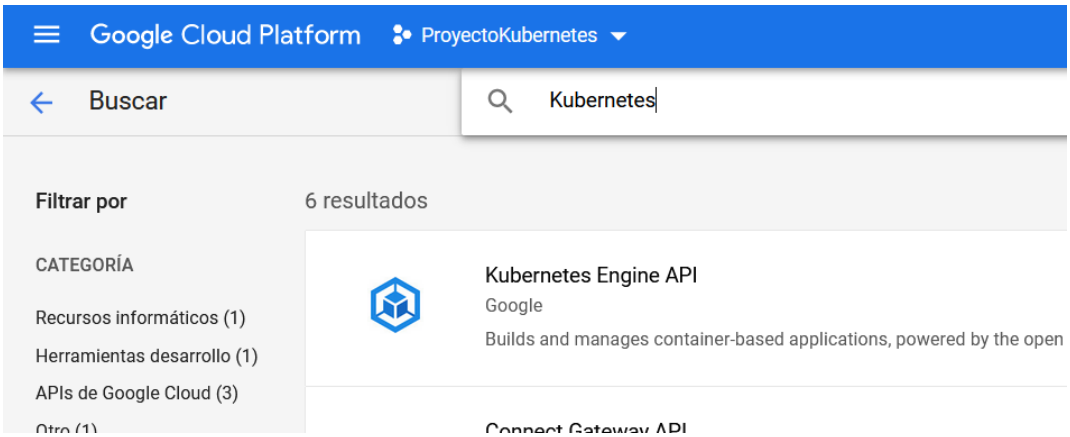
En la aplicación web de GCP vamos a "APIs y servicios".



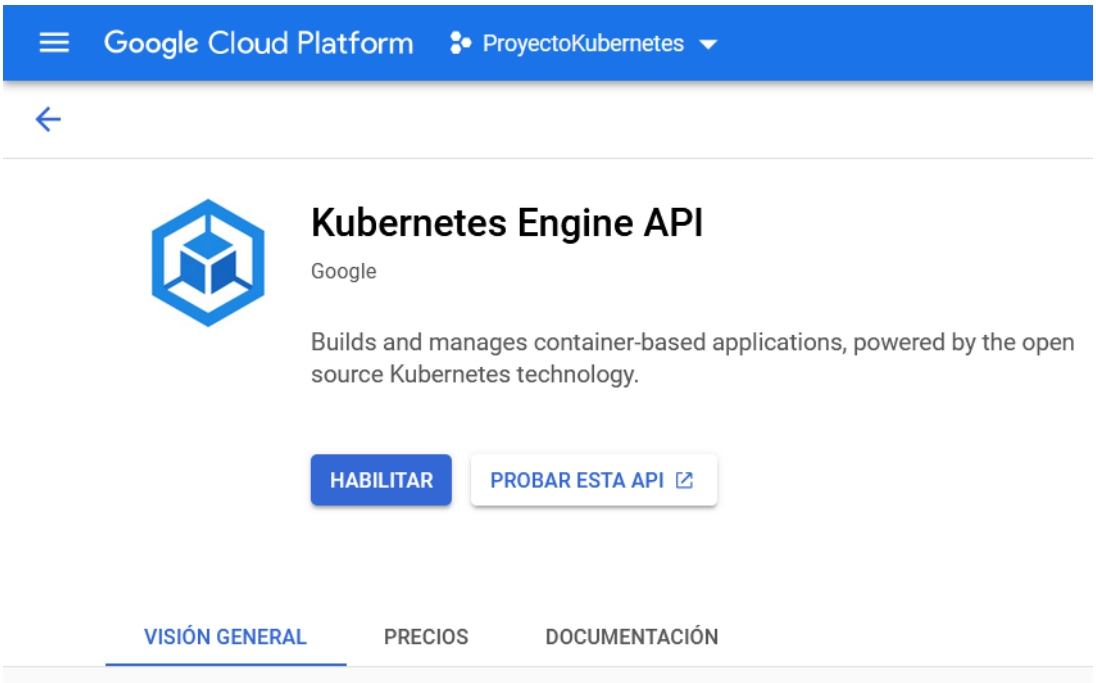
Seleccionamos HABILITAR APIS Y SERVICIOS.

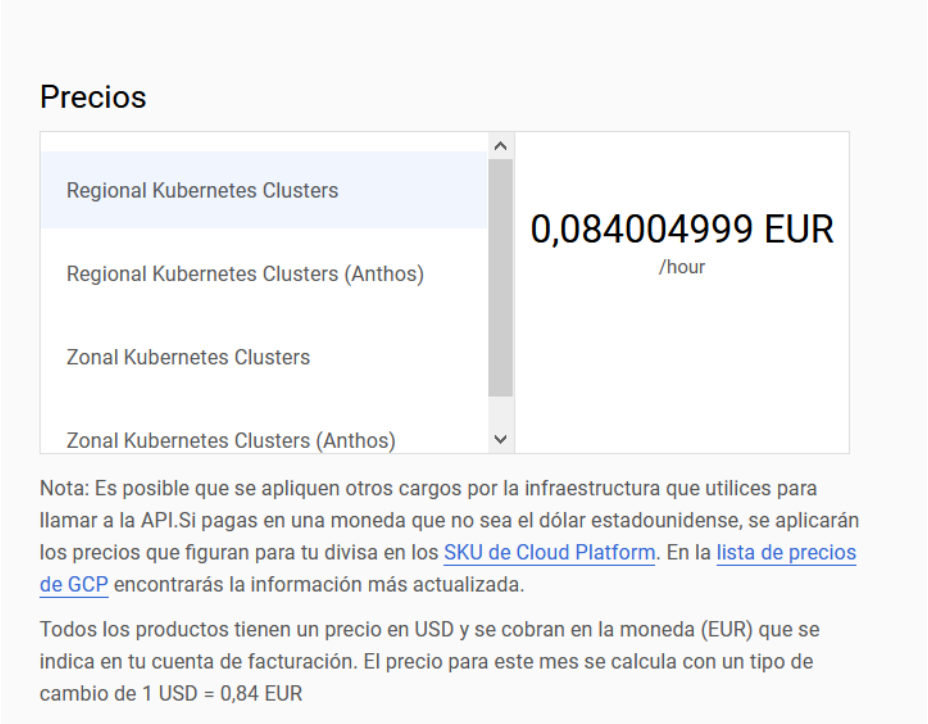


Tecleamos en el buscador "kubernetes" y seleccionamos la API de "Kubernetes Engine API".

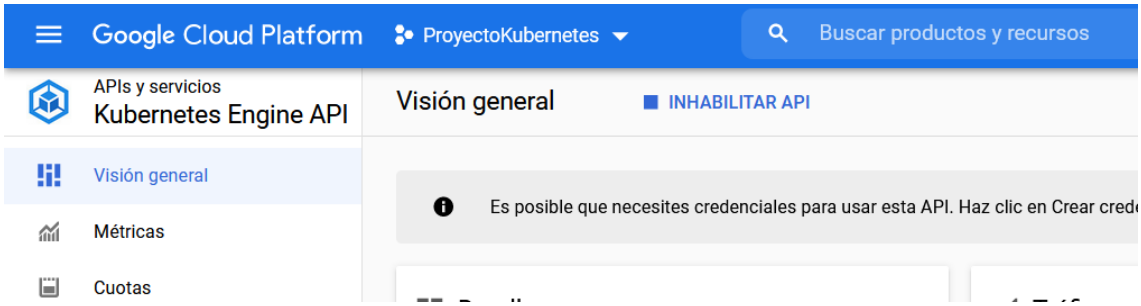


En la pestaña que nos saldrá podremos ver la descripción, el precio y la documentación, Podremos activar la API en el botón "Habilitar".





Más adelante podremos inhabilitarla en la pantalla de "APIs y servicios> Kubernetes Engine API" en Inhabilitar API.



Para poder usar Kubernetes hay que instalar Kubernetes en la máquina de GCP, para ello usaremos el comando del SDK **gcloud components install kubectl**. Antes y después se puede usar el comando **kubectl version** para comprobar si Kubernetes está debidamente instalado.

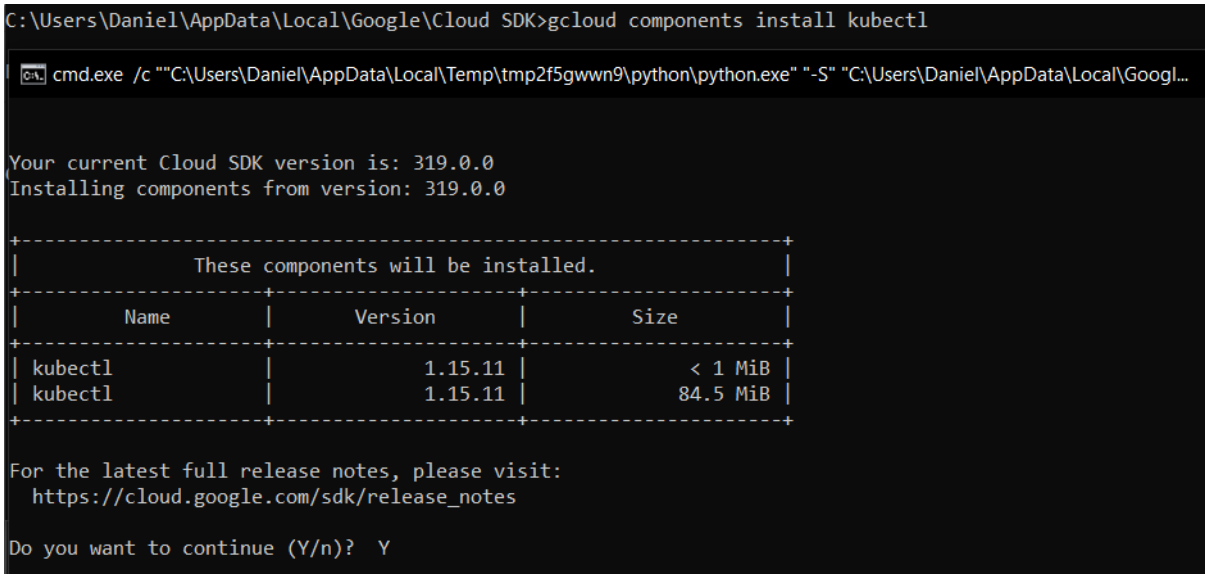


Figura 4.44: gcloud components install kubectl

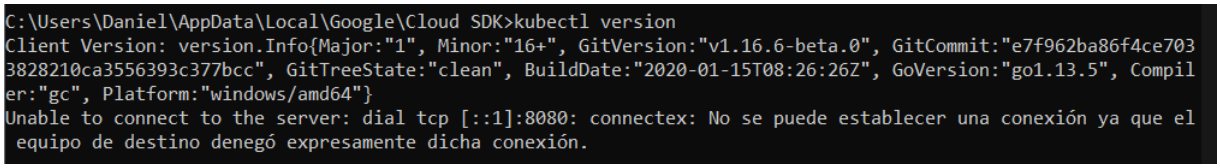


Figura 4.45: kubectl version

El comando **version** uestra un problema de conexión que trataremos más adelante. Es un problema de credenciales que se solucionará al asignar credenciales a un cluster (no es importante).

4.5.2. Crear un cluster

Una vez nos hemos cerciorado de que el proyecto es el correcto y está debidamente configurado, podemos crear un cluster. Los clusters son agrupaciones nodos y cada nodo una máquina virtual.

El comando para crear un cluster es: **gcloud container clusters create <Nombre del cluster>**. Si no añadimos más parámetros el cluster maestro será una máquina COS con todo lo necesario para usar kubernetes preinstalado.

```
C:\Users\Daniel\AppData\Local\Google\Cloud SDK>gcloud container clusters create pruebacluster
WARNING: Warning: basic authentication is deprecated, and will be removed in GKE control plane versions 1.19 and newer.
For a list of recommended authentication methods, see: https://cloud.google.com/kubernetes-engine/docs/how-to/api-server
-authentication
WARNING: Currently VPC-native is not the default mode during cluster creation. In the future, this will become the defau
lt mode and can be disabled using `--no-enable-ip-alias` flag. Use `--[no-]enable-ip-alias` flag to suppress this warnin
g.
WARNING: Newly created clusters and node-pools will have node auto-upgrade enabled by default. This can be disabled usin
g the `--no-enable-autoupgrade` flag.
WARNING: Starting with version 1.18, clusters will have shielded GKE nodes by default.
WARNING: Your Pod address range (`--cluster-ipv4-cidr`) can accommodate at most 1008 node(s).
WARNING: Starting with version 1.19, newly created clusters and node-pools will have COS_CONTAINERD as the default node
image when no image type is specified.
Creating cluster pruebacluster in europe-west1-b... Cluster is being health-checked (master is healthy)...done.
Created [https://container.googleapis.com/v1/projects/proyecto-kubernetes-298809/zones/europe-west1-b/clusters/pruebaclu
ster].
To inspect the contents of your cluster, go to: https://console.cloud.google.com/kubernetes/workload_/gcloud/europe-west
1-b/pruebacluster?project=proyecto-kubernetes-298809
kubeconfig entry generated for pruebacluster.
NAME          LOCATION       MASTER_VERSION    MASTER_IP       MACHINE_TYPE   NODE_VERSION      NUM_NODES  STATUS
pruebacluster  europe-west1-b  1.16.15-gke.4901  35.205.240.165  n1-standard-1  1.16.15-gke.4901  3          RUNNING
```

Figura 4.46: gcloud container clusters create pruebacluster

Ahora que tenemos un cluster instalado podemos crear credenciales. Para ello usaremos el comando **gcloud container clusters get-credentials <Nombre del cluster>**.

```
C:\Users\Daniel\AppData\Local\Google\Cloud SDK>gcloud container clusters get-credentials pruebacluster
Fetching cluster endpoint and auth data.
kubeconfig entry generated for pruebacluster.
```

Figura 4.47: gcloud container clusters get-credentials pruebacluster

Ahora probaremos a desplegar un servidor de ejemplo con kubernetes llamado "Hello server" (un servidor sencillo en lenguaje Go). El comando para desplegar un contenedor es **kubectl create deployment <Nombre del nodo> –image=<Dirección de imágen>**.

En el parámetro –imagen pondremos la imágen de máquina virtual que queramos desplegar mediante Kubernetes. La dirección del servidor hello-server en el repositorio es "gcr.io/google-samples/hello-app:1.0". Se pueden usar los repositorios de Docker Hub y de Google Repository (en este caso el de Google Repository). Se puede especificar una versión usando ":" tras el nombre de la imagen, si no, usará la última versión estable. También es posible introducir una imagen personalizada en forma de archivo YAML o un directorio del que desplegará todos los archivos YAML. El comando para ejecutar Hello-server sería **kubectl create deployment <server> <image>**

```
C:\Users\Daniel\AppData\Local\Google\Cloud SDK>kubectl create deployment hello-server --image=gcr.io/google-samples/hello-app:1.0
deployment.apps/hello-server created

C:\Users\Daniel\AppData\Local\Google\Cloud SDK>kubectl get deployment
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
hello-server   1/1     1            1           73s
```

Figura 4.48: kubectl create deployment hello-server –image=gcr.io/google-samples/hello-app:1.0

Si nos equivocamos o cuando ya no lo necesitemos podremos usar **kubectl delete deployment <Nombre de deployment>** para borrarlo.

```
C:\Users\Daniel\AppData\Local\Google\Cloud SDK>kubectl get deployment
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
hello-server   1/1     1            1           3m31s

C:\Users\Daniel\AppData\Local\Google\Cloud SDK>kubectl delete deployment hello-server
deployment.apps "hello-server" deleted
```

Figura 4.49: kubectl delete deployment hello-server

Ahora ya tenemos una aplicación corriendo en kubernetes, pero para que pueda ser accedida desde internet debemos configurar el "load balancer" (Equilibrador de carga). El load balancer creará un equilibrio entre los recursos del cluster y las conexiones entrantes.

Para crear una conexión abierta a internet usaremos el comando **kubectl expose deployment <Nombre del deployment> –port=<Número de puerto>** y para que tenga un equilibrador de carga añadiremos el argumento **–type "LoadBalancer"**.

```
C:\Users\Daniel\AppData\Local\Google\Cloud SDK>kubectl expose deployment hello-server --type=LoadBalancer --port=8080
service/hello-server exposed
```

Figura 4.50: kubectl expose deployment hello-server –type=LoadBalancer –port=8080

Ahora el servidor está publicado en la red y el load balancer se encargará de administrar las llamadas. Podremos comprobar la IP del servidor con el comando **kubectl get service <Nombre del servicio>** o ver la IP de todos los servicios publicados con **kubectl get service**.

```
C:\Users\Daniel\AppData\Local\Google\Cloud SDK>kubectl get service hello-server
NAME           TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)          AGE
hello-server   LoadBalancer   10.79.247.83   34.77.189.149   8080:30244/TCP   107s
```

Figura 4.51: kubectl get service hello-server

Si ahora vamos a nuestro navegador e introducimos la IP con el puerto, podremos ver que nos lleva a la página que hemos creado.

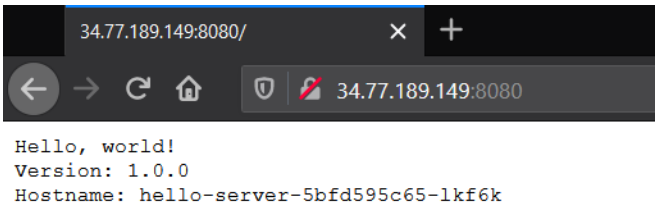


Figura 4.52: Muestra del servidor funcionando desde otro equipo.

4.6. Aplicación web con Node.js

Node.js es un entorno de ejecución para javascript asíncrono y escalable. Sirve para crear aplicaciones web y destaca por no usar los hilos del sistema operativo para su controlador. [14]

Vamos a usar Node.js para crear una aplicación de ejemplo para probar en Jenkins. La aplicación sumará y restará números de manera dinámica, ignorando letras.

4.6.1. La aplicación

La aplicación necesitará cuatro archivos:

AddWebpage.HTML : Será la parte visible de la página web.

AddScript.js : Javascript para realizar las sumas de AddWebpage.

AddScriptTest.js : Test unitario que más adelante deberá pasar Jenkins automáticamente.

server.js : Archivo que generará y publicará la aplicación web.

"AddWebpage.HTML" se crearía en una carpeta "www" y "AddScript.js" en una carpeta "js" dentro de "www". "server.js" iría en el directorio raíz y AddScriptTest.js en una carpeta "test".

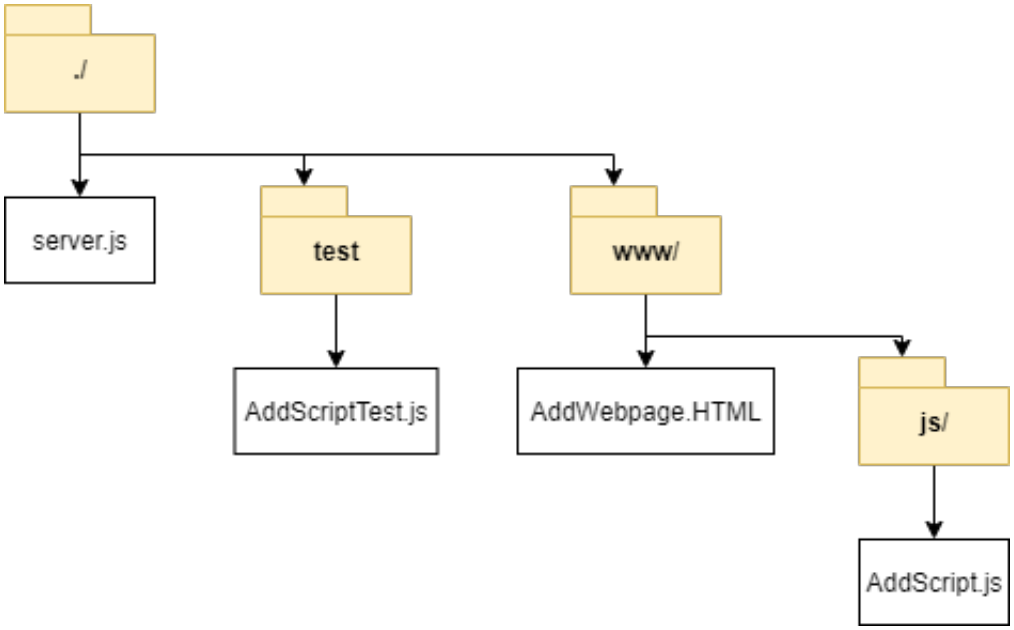


Figura 4.53: Esquema de la estructura de aplicaciones de la aplicación web "AddWebpage"

AddWebpage.HTML A.1.1 no tiene demasiada importancia en este ejemplo. Vale con saber que genera un textarea (un cuadro de texto rellenable) que al introducir cualquier valor llama a un script.

```
<!DOCTYPE html>
<meta charset="UTF-8">
<html>
    <script type="text/javascript" src="/js/AddScript.js"></script>
    <body>...
                                    ...
      <form >
            <textarea id="input" name="TextBox" rows="4" cols="50"
oninput="output.value=add(this.value)">
            </textarea>
                                    ...
       </form>
    </body>
</html>
```

Figura 4.54: Código de Addwebpage.html

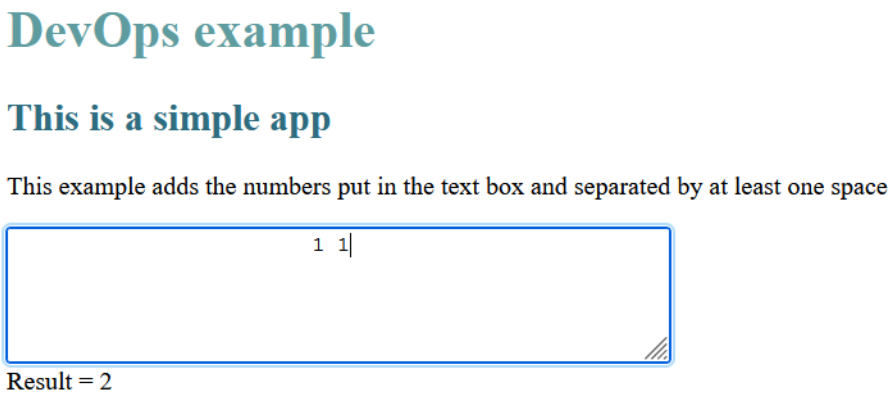


Figura 4.55: Aspecto de Addwebpage.html

AddScriptTest.js A.1.3 es un test unitario de AddScript.js. Como se ha explicado en el apartado de DevOps, es una buena costumbre crear primero los test unitarios. Estos test no están tan elaborados, pero sirven para hacerse una idea.

AddScript.js A.1.2 es sólo un script que elimina los caracteres que no sean números y suma los números separados por al menos un espacio. En este caso dejaremos comentado una alternativa que no funciona para más adelante (cuando probemos Jenkins).

server.js A.1.4 es donde está la parte interesante para este proyecto. Primero importamos la herramienta "express" que es una utilidad de node.js para desplegar aplicaciones web. Después establecemos que devuelva en la dirección de la app el archivo AddWebpage.html, se pueden crear diferentes archivos según la dirección. Cargamos el directorio y comenzamo a escuchar en el puerto 8081, creando una dirección y puerto para poder recibir y enviar. Usamo el puerto 8081 específicamente porque Jenkins usará el puerto 8080.

```
var express = require('express');
 var app = express();

 /*Cuando se reciba una petición dentro del directorio "/" (el raíz)
   Ejecutará el código dentro.*/
 app.get('/', function(req, res) {
     //Devuielve el archivo AddWebpage.html
     res.sendFile(__dirname+'/www/AddWebpage.html');
 });

 //Carga el directorio www
 app.use(express.static(__dirname + '/www'));

 /*Escucha en el puerto localhost:8081
    Crea una dirección y un puerto*/
 var server = app.listen(8081, function(req,res) {
  var host = server.address().address
  var port = server.address().port

  console.log("Example app listening at http://%s:%s", host, port)
 });
```

Figura 4.56: Código de server.js

4.6.2. Ejecutar el servidor en local

Ahora simplemente navegamos con la consola de comandos al directorio donde hemos creado la aplicación.

El primer paso es instalar npm y node si no lo habíamos instalado antes. Después ejecutamos "npm init" dentro del directorio para que cree los archivos de configuración de node necesarios (un archivo "package.json" y una carpeta "node_modules"). También usaremos mocha y express. Todo se instala introduciendo los siguientes comandos en orden.

1. **npm install**
2. **npm init**
3. **npm install –global mocha**
4. **npm install express –save**

Con todo instalado podemos pasar los test introduciendo el comando **npm test**

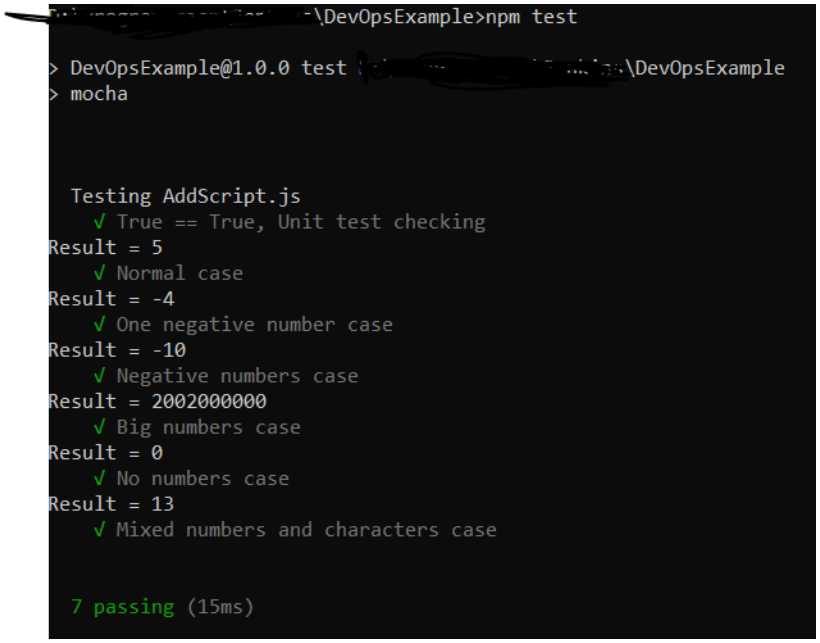


Figura 4.57: npm test

Por último podemos ejecutar el servidor simplemente introduciendo el comando **node server.js**

Figura 4.58: npm test

Si ahora en el navegador introducimos la dirección http://localhost:8081/ cargará la aplicación web. Se puede cerrar pulsando Ctrl+C en la consola de comandos.

Figura 4.59: Localhost 8081

4.7. Ejecutar en Kubernetes desde GCP

Vamos a publicar este servicio web usando Kubernetes y GCP. Para ello vamos a necesitar ir a nuestro proyecto y abrir el dashboard.

El primer paso será descargar la carpeta que creamos en el anterior apartado. Se puede realizar fácilmente con un pull de Git si lo subimos a un repositorio.

4.7.1. Archivos de configuración

Para ejecutar esta aplicación desde Kubernetes, deberemos crear antes los archivos YAML de configuración. Normalmente los servicios como este usan un archivo de servicio, uno de despliegue y uno de ingress. Se pueden dividir y juntar en un sólo archivo, pero por claridad vamos a crear un archivo por cada uno. Vamos a crear los siguientes archivos en una carpeta llamada “deploy”.

service.YAML

A.2.1El archivo de servicio se encarga de definir y orquestar el resto de archivos y definiendo la aplicación. También sirve como puente entre el puerto del servidor y el puerto del front-end.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: addwebpage-service
  labels:
    app: addwebpage
spec:
  selector:
    app: addwebpage
  type: LoadBalancer
  ports:
  - protocol: TCP
    port: 5000
    targetPort: 8081
    nodePort: 31110
```

Figura 4.60: service.YAML

Primero este archivo se define como un archivo de servicio llamado addwebpage-service con la etiqueta "addwebpage". Selecciona a todos los archivos de configuración con la etiqueta addwebpage. Crea un puerto con protocolo TCP que escucha al puerto 8080 y exponen el puerto 80. nodePort expone este servicio en ese puerto a todos los nodos hijos.

deployment.YAML A.2.2 El archivo de despliegue define qué se entrega o qué ocurre cada vez que alguien accede al servicio.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: addwebpage-deploy
  labels:
    app: addwebpage
spec:
  replicas: 1
  selector:
    matchLabels:
      app: addwebpage
  template:
    metadata:
      labels:
        app: addwebpage
    spec:
      containers:
      - name: addwebpage
        image: eu.gcr.io/proyectokubernetes-301509/addwebpage:latest
        ports:
        - containerPort: 8081
```

Figura 4.61: deployment.YAML

Primero se nombra y se añade la etiqueta del despliegue. A continuación en spec, definimos qué se va a crear. “replicas” define el número de nodos que se crean por defecto. Selector se usa para establecer el servicio al que pertenece.

Template es un campo obligatorio que especifica la plantilla de lo que devuelve (normalmente un pod). En este caso crea un pod desde una imágen en un depósito de imágenes. Normalmente se usa una cuenta de Docker Hub, pero como estamos en GCP vamos a usar su propio repositorio de imágenes. usamos la imagen que hemos construido. Ahí es donde vamos a subir la imagen compilada con Docker más adelante.

Ingress.YAML

A.2.3 El archivo Ingress es una API que se encarga de administrar el acceso de la red al exterior. En este caso se encarga de manejar el servicio HTTP. En un ejemplo más local no hace falta, pero como queremos publicar esta aplicación lo necesitaremos.

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: addwebpage-ingress
  labels:
    app: addwebpage
spec:
  rules:
  - http:
      paths:
      - pathType: Prefix
        path: "/"
        backend:
          service:
            name: addwebpage-service
            port:
              number: 5000
```

Figura 4.62: Ingress.YAML

Primero se establece el archivo y en spec se aplican las reglas. El archivo Ingress puede especificar reglas. Las reglas son las acciones que se ejecutarán dependiendo de dónde se acceda. Por ejemplo en este archivo si se accede a la dirección raiz ("/") entonces ejecutará el servicio de back-end llamado "addwebpage.

Siempre hay una regla por defecto. Si no se especifica será elegido por el controlador de Ingress, pero esto no se decide desde este archivo. Se puede especificar mediante este archivo.

4.7.2. Compilar y subir la imagen

Ahora debemos subir la imágen al repositorio que mencionamos anteriormente. "eu.gcr.io/proyectokubernetes-301509/addwebpage:latest". Se puede subir a un repositorio como Docker Hub o un repositorio propio. "eu.grc.io" es el repositorio europeo de imágenes de google. Después va el ID del proyecto y por último el nombre de la imagen seguida de dos puntos y la etiqueta (normalmente la versión)

Para no hacerlo tan largo de escribir exportamos el nombre del proyecto a una variable global con el comando export. Es importante recordar que en la consola de GCP esta variable no persistirá cuando la cerremos.

export PROJECT_ID=proyectokubernetes-301509

Ahora usaremos el comando build de docker para compilar la imagen a partir de la ruta desde donde se hace el comando hasta donde está el Dockerfile. El Dockerfile debería estar en la raíz de la carpeta. Después comprobaremos que se ha subido correctamente mostrando la lista de imágenes con el comando docker images.

docker build -t eu.gcr.io/$PROJECT_ID/addwebpage:latest .

docker images

Opcionalmente podemos arrancar la máquina como una máquina de docker local para probar que funciona correctamente. Para ello se usa el comando de "docker run".

docker run –rm -p 8081:8081 eu.gcr.io/$PROJECT_ID/addwebpage:latest

```
dalvac01@cloudshell:~ (proyectokubernetes-301509)$ export PROJECT_ID=proyectokubernetes-301509
dalvac01@cloudshell:~ (proyectokubernetes-301509)$ docker build -t eu.gcr.io/${PROJECT_ID}/addwebpage:v1 .
unable to prepare context: unable to evaluate symlinks in Dockerfile path: lstat /home/dalvac01/Dockerfile:
dalvac01@cloudshell:~ (proyectokubernetes-301509)$ docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
dalvac01@cloudshell:~ (proyectokubernetes-301509)$ docker run --rm -p 8081:8081 eu.gcr.io/${PROJECT_ID}/addw
Unable to find image 'eu.gcr.io/proyectokubernetes-301509/addwebpage:v1' locally
v1: Pulling from proyectokubernetes-301509/addwebpage
76b8ef87096f: Pull complete
2e2bafe8a0f4: Pull complete
b53ce1fd2746: Pull complete
84a8c1bd5887: Pull complete
7a803dc0b40f: Pull complete
b800e94e7303: Pull complete
0da9fbf60d48: Pull complete
04dccde934cf: Pull complete
73269890f6fd: Pull complete
3b490236987c: Pull complete
dd626c74096c: Pull complete
f206ef804425: Pull complete
a84ae941f674: Pull complete
6fc4d8ec305c: Pull complete
3fe8373aef3c: Pull complete
b4343857523e: Pull complete
Digest: sha256:8aa8c2022f8d0cc4576b51ff559ffbc8c3a5cd900e41941d33171498b3bcfd95
Status: Downloaded newer image for eu.gcr.io/proyectokubernetes-301509/addwebpage:v1
Example app listening at http://:::8081
```

Figura 4.63: Crear servidor en kubernetes

Si no ha habido ningún problema, la imagen se debería haber cargado en `http://localhost:8081`. En GCP para acceder al localhost se debe usar una dirección especial a la que se puede acceder en el menú de arriba a la derecha de la consola de comandos de GCP. Si no se usa el puerto 8080 se puede acceder a este y cambiar manualmente en la url el puerto.

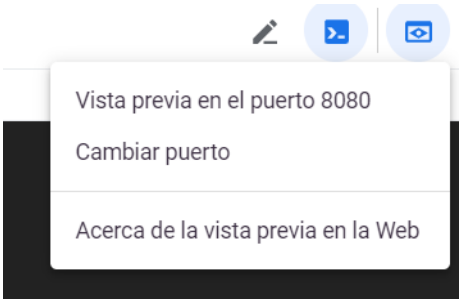


Figura 4.64: Acceso a localhost 8081

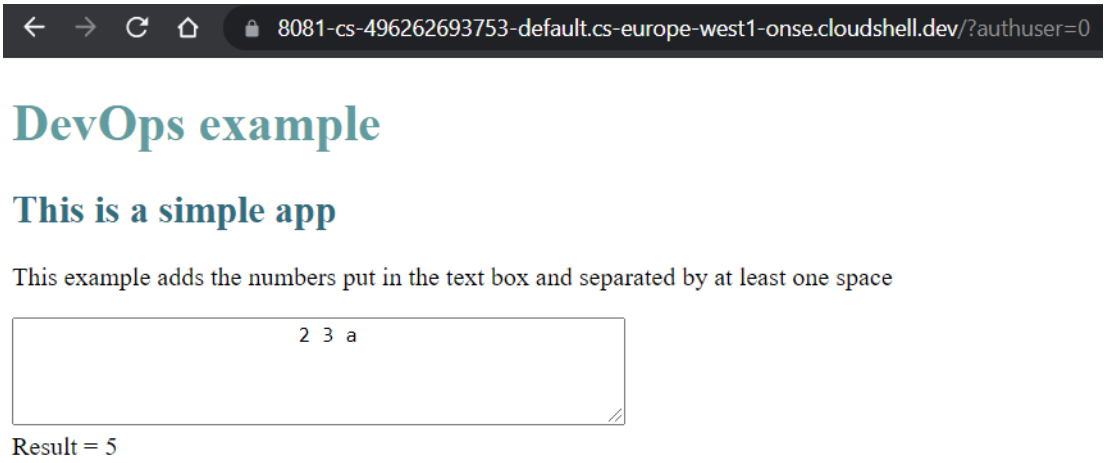


Figura 4.65: Puerto 8081

Como la imágen funciona correctamente podemos subirla al repositorio con el comando push de docker

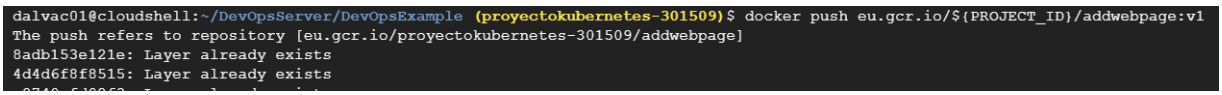


Figura 4.66: docker push eu.gcr.io/$PROJECT_ID/addwebpage:latest

Ahora podremos acceder a esa imágen y compilarla con la dirección "eu.gcr.io/proyectokubernetes-301509/addwebpage:latest". Se puede descargar la imagen con el comando docker pull.

4.7.3. Publicar la imagen con Kubernetes

Con los archivos de configuración ya creados en la carpeta "deploy", desplegar la aplicación sólo necesita del comando "kubectl apply -f deploy". Para hacerlo más

limpio y controlado creamos antes el espacio de nombre "producción". Cuando hayamos desplegado la aplicación, haremos un comando "get services" en el espacio de nombres para ver que se ha creado. La dirección IP tardará entre treinta segundos y un minuto y medio en crearse.

kubectl create ns production

kubectl –namespace=production apply -f deploy/

kubectl –namespace=production get services

```
dalvac01@cloudshell:~/DevOpsServer/DevOpsExample (proyectokubernetes-301509)$ kubectl create
 ns production
namespace/production created
dalvac01@cloudshell:~/DevOpsServer/DevOpsExample (proyectokubernetes-301509)$ kubectl --name
space=production apply -f deploy/
deployment.apps/addwebpage-deploy created
ingress.networking.k8s.io/addwebpage-ingress created
service/addwebpage-service created
dalvac01@cloudshell:~/DevOpsServer/DevOpsExample (proyectokubernetes-301509)$ kubectl --name
space=production get services
NAME                TYPE           CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
addwebpage-service  LoadBalancer   10.3.241.100   <pending>     5000:31110/TCP   1s
dalvac01@cloudshell:~/DevOpsServer/DevOpsExample (proyectokubernetes-301509)$
```

Figura 4.67: Despliegue de un servicio de Kubernetes

Una vez desplegado podemos hacer algunos cambios en los parámetros. Por ejemplo, cambiar el número de réplicas de producción.

kubectl –namespace=production get deployments

kubectl –namespace=production scale deployment addwebpage-deploy –replicas=4

kubectl –namespace=production get deployments

```
dalvac01@cloudshell:~/DevOpsServer/DevOpsExample (proyectokubernetes-301509)$ kubectl --name
space=production get deployments
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
addwebpage-deploy   1/1     1            1           98s
dalvac01@cloudshell:~/DevOpsServer/DevOpsExample (proyectokubernetes-301509)$ kubectl --name
space=production scale deployment addwebpage-deploy --replicas=4
deployment.apps/addwebpage-deploy scaled
dalvac01@cloudshell:~/DevOpsServer/DevOpsExample (proyectokubernetes-301509)$ kubectl --name
space=production get deployments
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
addwebpage-deploy   1/4     1            1           98s
```

Figura 4.68: Escalado del servicio de Kubernetes

Finalmente se generará la IP externa del servidor. Para acceder a la aplicación web debemos ir a esa IP en el puerto que indicamos en el ingress (en este caso el 5000). Esto funcionará desde cualquier navegador y es accesible desde todo internet.

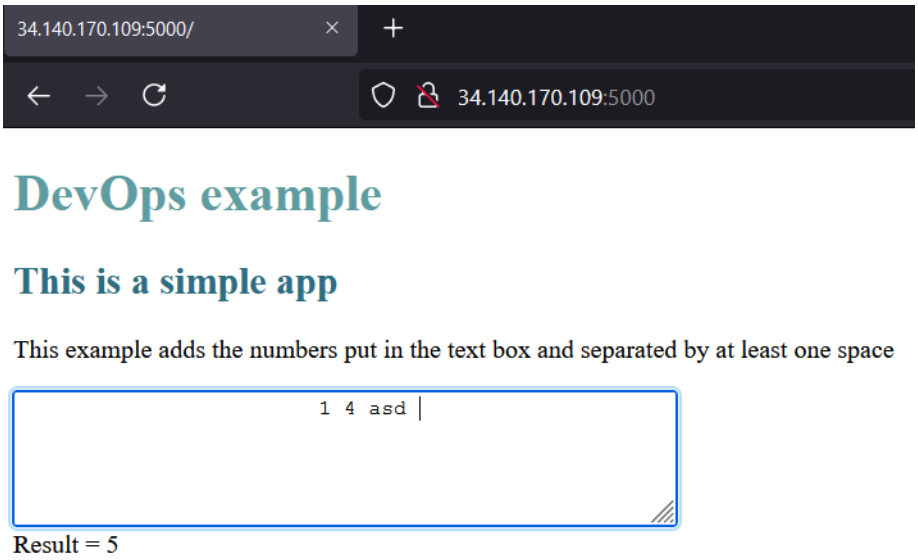


Figura 4.69: Servidor desplegado desde otro navegador

4.8. Jenkins

Jenkins es un servidor automatizado para implementar el proceso de CI/CD. Es de código abierto y muy popular en DevOps junto a otras alternativas como Travis, Bitbucklet o GitLab. Una de las características principales es que es modular, es decir, de base es una herramienta muy sencilla pero se puede agrandar con plug-ins.

4.8.1. Instalando Jenkins en local

Jenkins se puede montar en un servidor privado o equipo fácilmente. Se puede descargar el instalador desde la página web `https://www.jenkins.io/download/`. El proceso de instalación depende del sistema operativo y sólo requiere seguir la guía de instalación en la documentación oficial `https://www.jenkins.io/doc/book/installing/`.

Durante el proceso se generará un archivo con la contraseña del usuario que creemos y podremos acceder desde el puerto de LocalHost que hayamos elegido.

Es útil para probar Jenkins antes de crear todo un sistema. Aunque una instalación así de sencilla sirva para probar la herramienta y para equipos pequeños con acceso a un servidor, no es la mejor opción para proyectos más grandes.

Problemas de Jenkins local

De manera nativa Jenkins usa agentes generados del propio nodo maestro. Esto conlleva un fallo de seguridad: Si una ejecución bloquea al agente, entonces bloquea al maestro. Este problema no es grave en un sistema pequeño, pero escala rápidamente si se usa con un equipo grande o varios equipos. Otro problema es que si hay un fallo de ciberseguridad durante la ejecución de un agente y es explotado durante su ejecución, tendría acceso al sistema.

Estos dos problemas se pueden solucionar añadiendo una capa intermedia con kubernetes. Se puede crear Jenkins como un servicio que genera pods de kubernetes como agentes.

Debido a este problema, en este proyecto veremos directamente cómo se crea desde GCP con Kubernetes. Es interesante saber que existe una opción más sencilla para empezar. Si es la primera vez que se trata con Jenkins tecnologías, es recomendable empezar poco a poco y probar Jenkins local para familiarizarse con la herramienta primero. De esta forma es más fácil entender la herramienta y no se hace uso del servidor mientras tanto, en especial si es un servidor externo como GCP que se paga por uso.

4.8.2. Configuración de Jenkins

En este apartado procederemos a configurar Jenkins. Este proceso se puede automatizar mediante un archivo Yaml, pero vamos a hacerlo a mano para no complicar la explicación. Vamos a instalar tres plugins que nos serán útiles en el próximo apartado.

Instalar plugin de Blue Ocean

Esta herramienta fue creada por CloudBees (desarrolladores de Jenkins) para que el proceso de creación de pipelines sea más sencillo. En un futuro Blue Ocean reemplazará completamente a la UI tradicional [15]. A día de hoy todavía no puede ajustarse a proyectos muy específicos, pero en esos casos se pueden crear los archivos de Jenkinsfile a mano.

En este proyecto usaremos Blue Ocean. No es una herramienta imprescindible pero facilita la conexión del pipeline entre Jenkins y GitHub. Además la interfaz es más clara que la original.

En el menú de la izquierda de la interfaz de Jenkins seleccionamos “Administrar Jenkins”>”Administrar plugins”.

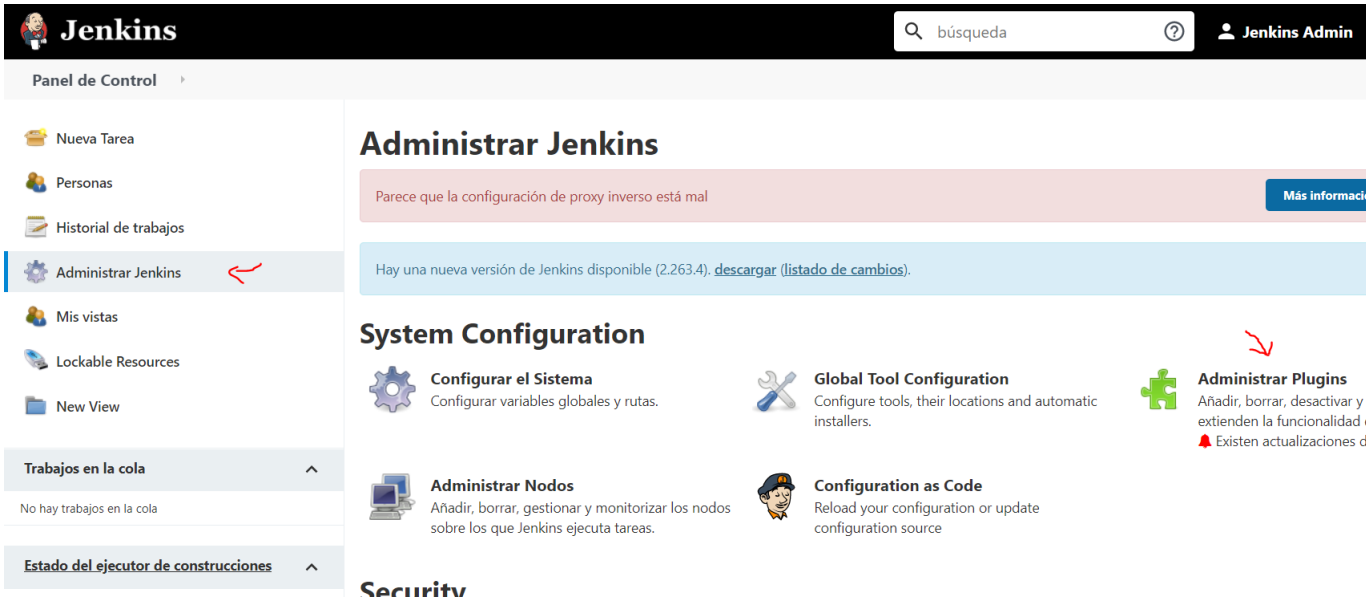


Figura 4.70: Menú de administración de Jenkins

Actualizamos los plugins de necesitarlo y seleccionamos la pestaña “Todos los plugins” y filtramos en la casilla de búsqueda por “Blue Ocean”. Tenemos que encontrar el plug in que se llama sólo “Blue Ocean”. Marcamos la casilla del plugin y hacemos click en “Descargar ahora e instalar después de instalar”

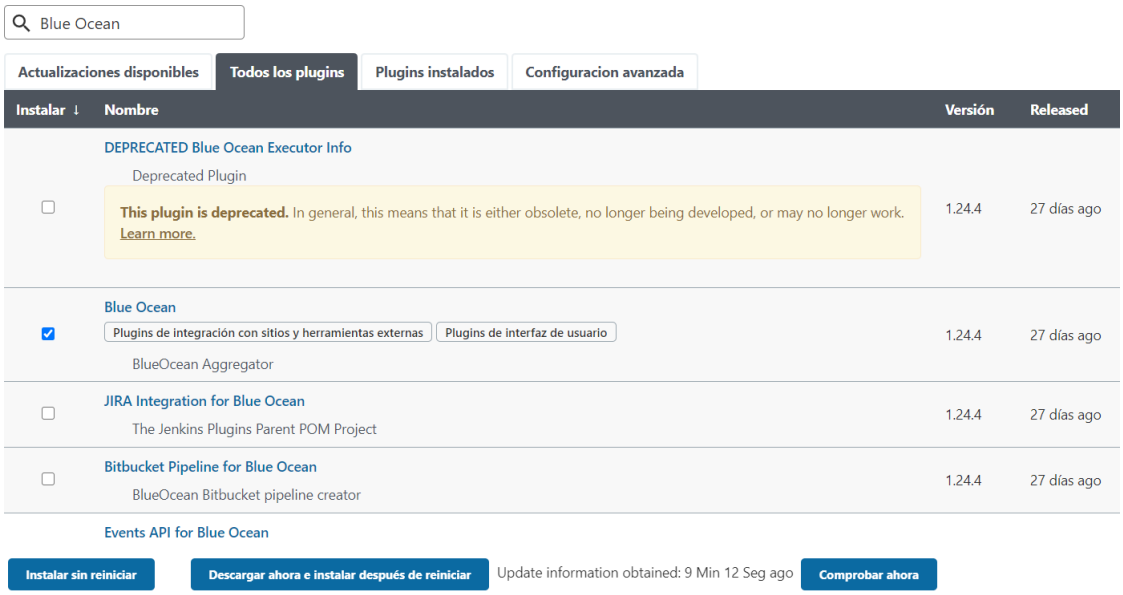


Figura 4.71: Menú de plugins de Jenkins

Ahora nos llevará a una página donde podremos ver todos los plugins en lista de descarga. Aparecen más de los que hemos puesto porque añade los plugins necesarios para poder funcionar.

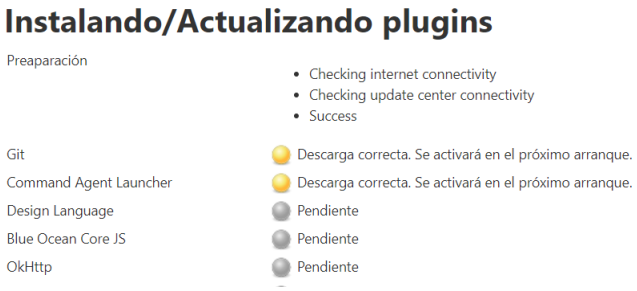


Figura 4.72: Instalación de plugins en Jenkins, parte I

La pantalla no actualiza sólo, así que vamos a ir al fondo de la página y seleccionaremos la opción de que se reinicie cuando acabe de descargar todo. Técnicamente se puede instalar y usar, pero es una buena práctica reiniciar tras instalar en un servidor.

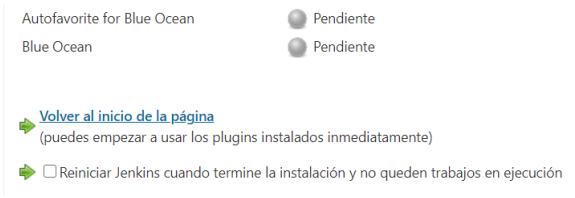


Figura 4.73: Instalación de plugins en Jenkins, parte II

Figura 4.74: Reinicio de Jenkins

Cuando termine, se reiniciará Jenkins y todo estará instalado. Se puede comprobar yendo otra vez a Administrar Jenkins>administrar plugins>Plugins instalados. Buscamos Blue Ocean para ver que ha sido correctamente instalado a la última versión.

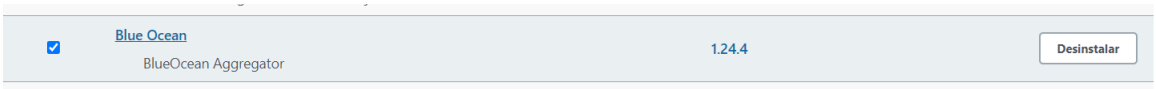


Figura 4.75: Blue Ocean en plugins instalados

En el menú principal también nos saldrá un nuevo icono que nos llevará a la herramienta de Blue Ocean.

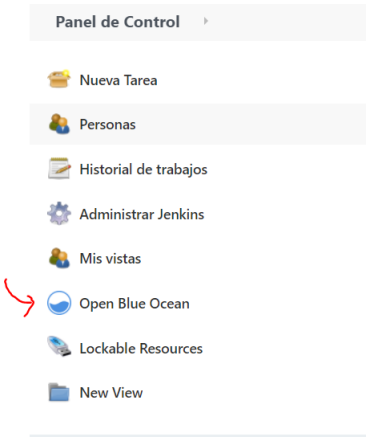


Figura 4.76: Blue Ocean en el menú principal de Jenkins

Adicionalmente podemos configurar en "Administrar Jenkins>Configurar el sistema" y buscamos el apartado

Instalar plugin de Nodejs

Para ejecutar la aplicación web de ejemplo que creamos, necesitamos poder ejecutar node en los canales de Jenkins.

Volvemos a "Administrar Jenkins>Administrar plugins>Todos los plugins", buscamos el plugin "nodejs" y lo instalamos.

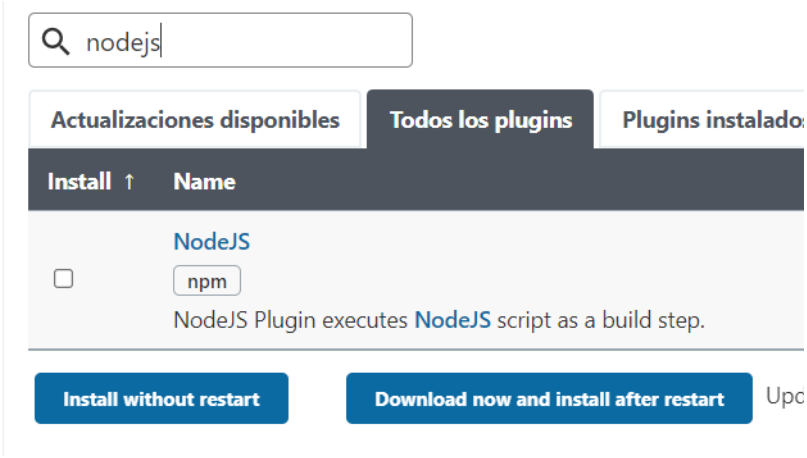


Figura 4.77: Plugin de Node.js

Cuando acabe de instalar vamos a "Administrar Jenkins>Administrar plugins>Global Config Tool" y vamos a la pestaña de Nodejs. Ahí creamos una instalación automática a la que llamaremos "nodejs" con la última versión.

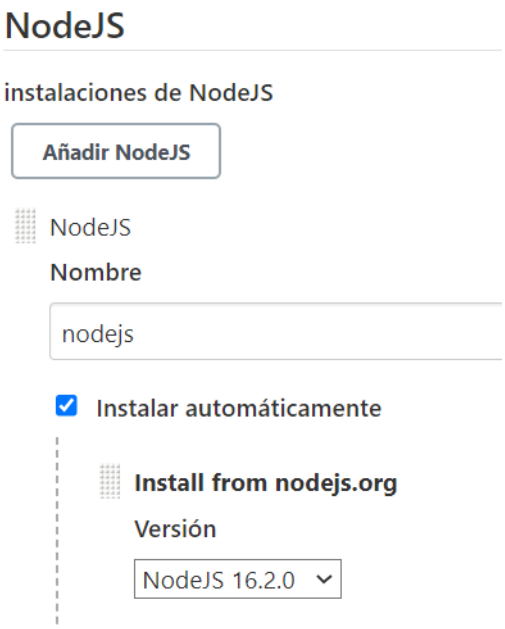


Figura 4.78: Configuración de Node.js

Instalar plugins de mail extension

Este plugin ayudará a mejorar el plugin base mailer de Jenkins. Entre otras cosas facilitará enviar mensajes y nos permitirá enviarlos con formato HTML para poder usar saltos de línea.

En "Administrar Jenkins>Administrar plugins>Todos los plugins", buscamos el plugin "mail extension" y lo instalamos.

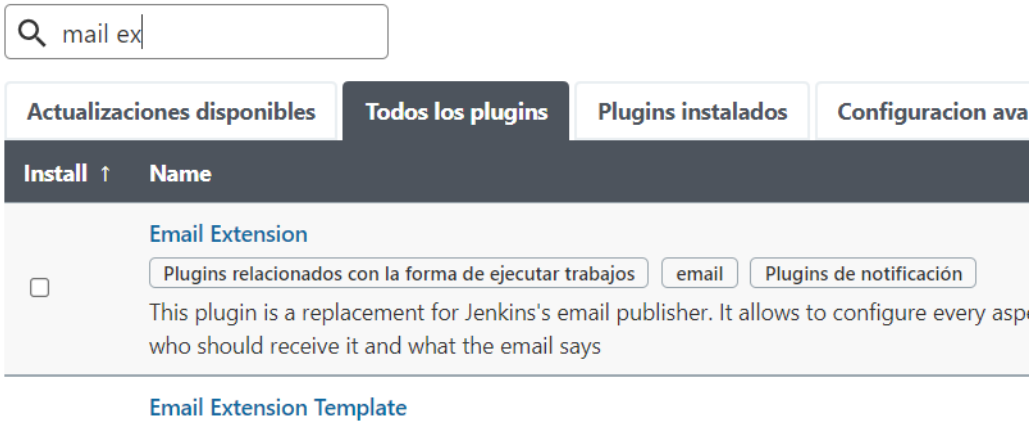


Figura 4.79: Plugin de Mail extension

Ahora hay que configurar un servidor SMTP para Jenkins. Para ello necesitaremos una cuenta de google. Para seleccionar la cuenta de google necesitaremos

una contraseña de app. Para este ejemplo usaremos la dirección de correo correo.de.desarrollador.ficticio@gmail.com

Para obtener una contraseña de app, vamos a un navegador en el que nos hayamos conectado con la cuenta de google que vamos a usar y vamos a la dirección `https://myaccount.google.com/`

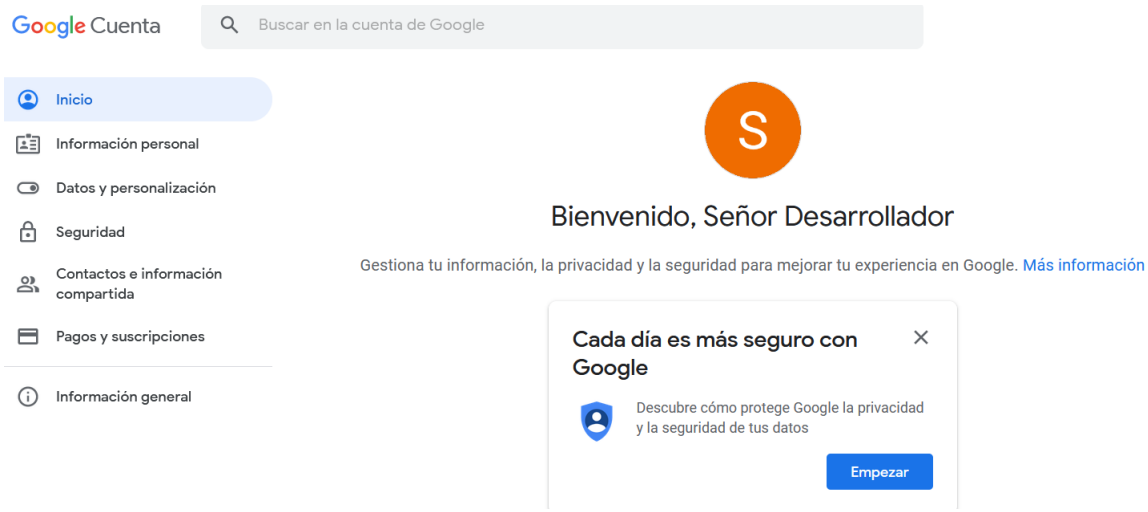


Figura 4.80: Creación de servidor STMP

Esta dirección permite varias configuraciones. Lo primero que vamos a hacer es activar la verificación en dos pasos para poder crear una contraseña de app.

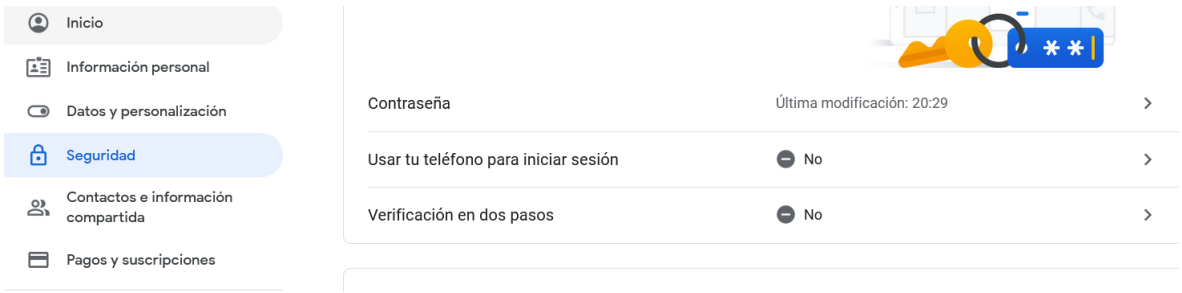


Figura 4.81: Menú de opciones de seguridad de la cuenta de Google

Seguimos los pasos para activar la autenticación en dos pasos y aparecerá en el mismo menú una nueva opción.

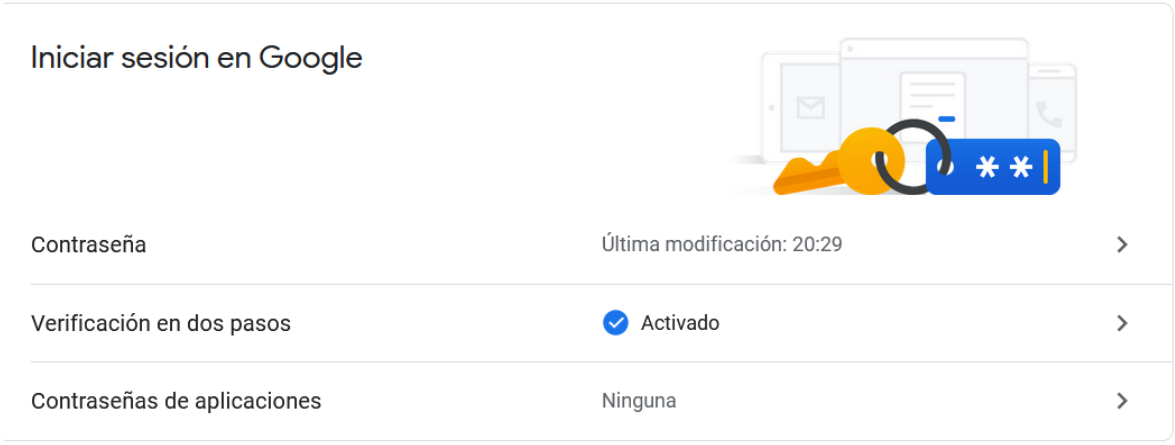


Figura 4.82: Verificación en dos pasos

Hacemos clic en contraseñas de aplicaiones y seguimos los pasos. Elegimos de tipo de aplicación "Otra (especificar)" y escribimos un identificador para recordar cuál es (en este caso Jenkins).

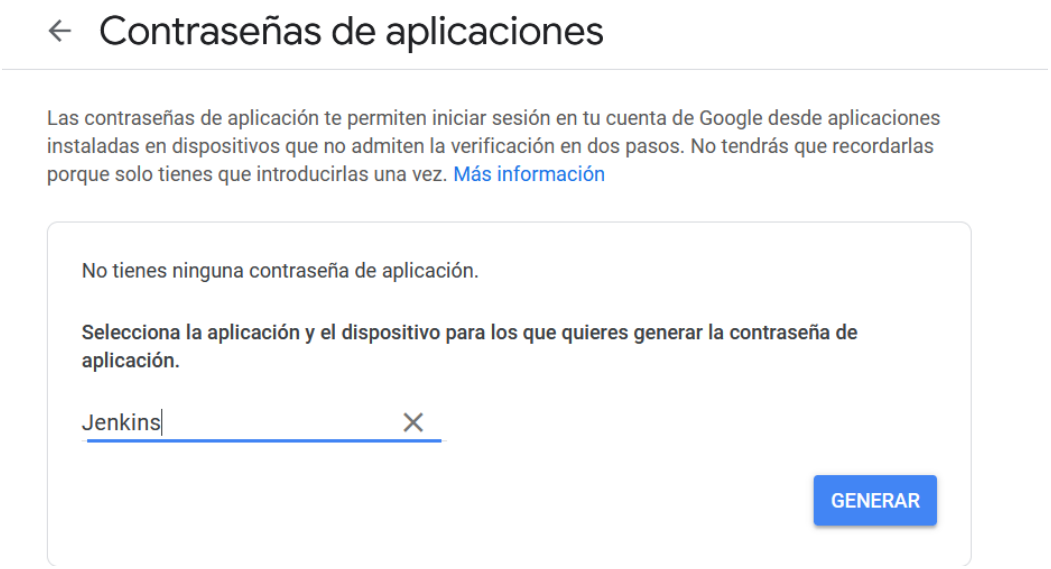


Figura 4.83: Creación de contraseña de app

Hacemos clic en generar y copiamos la clave temporalmente (preferiblemente de manera analógica).

Ahora vamos a "Administrar Jenkins>Configurar el sistema". Buscamos el apartado "Jenkins Location" y añadimos en System Admin e-mail address la cuenta de correo de administrador desde donde se enviarán los correos.

Figura 4.84: Establecer e-mail de administrador.

A continuación buscamos el apartado "Extended E-mail Notification". Rellenamos los siguientes campos:

SMTP server	smtp.gmail.com
Use SMTP Authentication	Check
Nombre de usuario	Cuenta de google que preparamos
Contraseña	Contraseña app que creamos
Usar seguridad SSL	Check
Puerto de SMTP	465 (distinto según el servidor SMTP usado)
Default Content Type	HTML (text/html)
Juego de caracteres	ISO-8859-1
(opcional) Dirección para la respuesta	noreply@gmail.com o un correo

Tabla 4.1: Caption

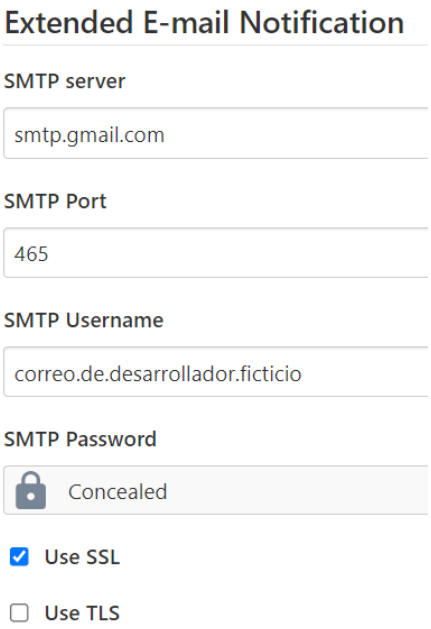


Figura 4.85: Configuración del correo, parte I

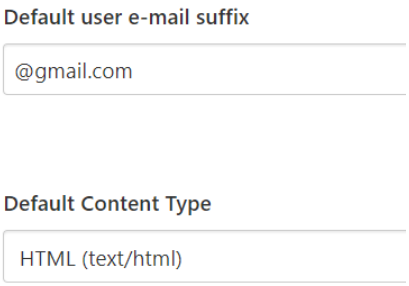


Figura 4.86: Configuración del correo, parte II

Bajamos un poco más en la página y buscamos el apartado "Notificación por correo electrónico". Rellenamos los mismo campos con el mismo contenido, aunque están en otro orden. También añadimos el campo Juego de caracteres para que los correos puedan contener tildes.

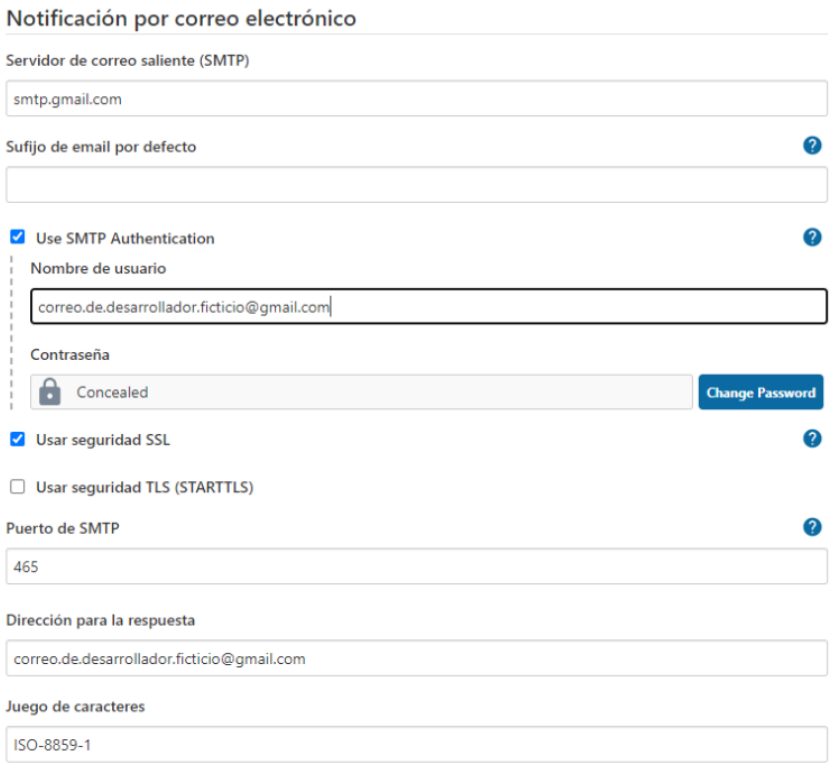


Figura 4.87: Configuración del correo, parte III

Por último debemos añadir la llamada a este plugin en la sección de post del Jenkinsfile

```
emailext(
          body: "Compilación fallida ${env.BUILD_URL}. Por favor, revise el
código.",
          to: 'email@gmail.com',
          recipientProviders: [developers(), requestor()],
${APP_NAME}:${env.BRANCH_NAME}.${env.BUILD_NUMBER}"
          subject: "Error en build
${APP_NAME}:${env.BRANCH_NAME}.${env.BUILD_NUMBER}"


)
```

Figura 4.88: Configuración del correo en el jenkinsfile

En 'subject' va el título del correo y en 'body' va el cuerpo del correo. Se pueden añadir variables del Jenkinsfile usando '$'. Para establecer a quien se le manda el

correo se pueden añadir correos específicos con 'to' o de manera automática con 'recipientProviders'. 'recipienProviders' usa parámetros que busca directamente la dirección de correo. [16]

developers() : Busca el correo de todos los usuarios que revisaron el código desde la última compilación.

requestor() : Busca el correo del usuario que inició la build.

culprits() : Junto con 'developers' manda un correo a todos los usuarios que hayan subido un commit desde la última compilación exitosa.

4.8.3. Actualizar plugins

Es importante hacer este paso el último por si hay problemas de compatibilidad. En "Administrar Jenkins>Administrar plugins>Actualizaciones disponibles" vamos al fondo de la página.

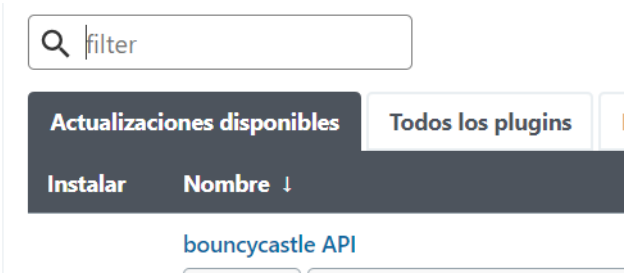


Figura 4.89: Actualizar plugins, parte I

Seleccionamos el botón "Compatible" y seleccionará todos los plugins que no causen problemas de compatibilidad. Los actualizamos.

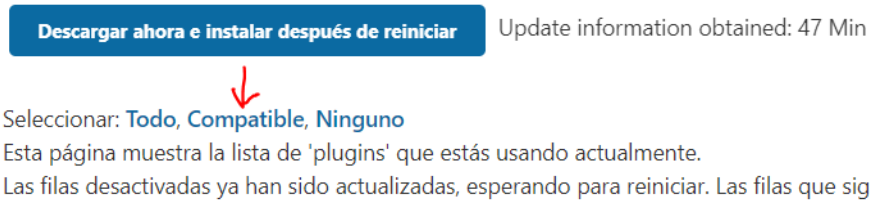


Figura 4.90: Actualizar plugins, parte II

4.8.4. Crear usuarios

Para poder interactuar entre Jenkins y el equipo, es necesario crear a cada uno un usuario y contraseña. Vamos a crearlos de manera más manual, pero una empresa

más grande podría usar plugins para que sea el usuario el que cree su cuenta. Primero vamos a “Administrar Jenkins/>Gestión de usuarios”

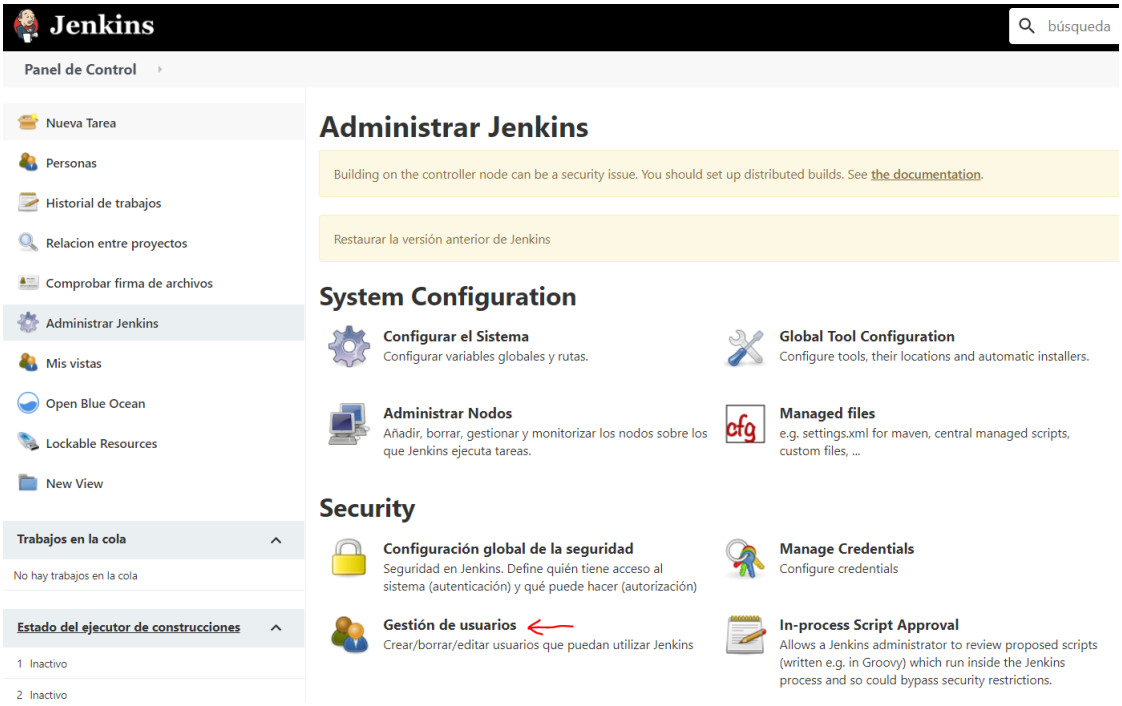


Figura 4.91: Crear usuarios, parte I

En este menú aparecerá una lista con todos los usuarios registrados. El administrador puede acceder y editar la información de todos los usuarios y restablecer contraseña. Continuamos haciendo clic en “Crear un usuario"

Figura 4.92: Crear usuarios, parte I

Rellenamos todos los campos incluyendo la contraseña y el correo. El correo debe ser el mismo que usa en Git.

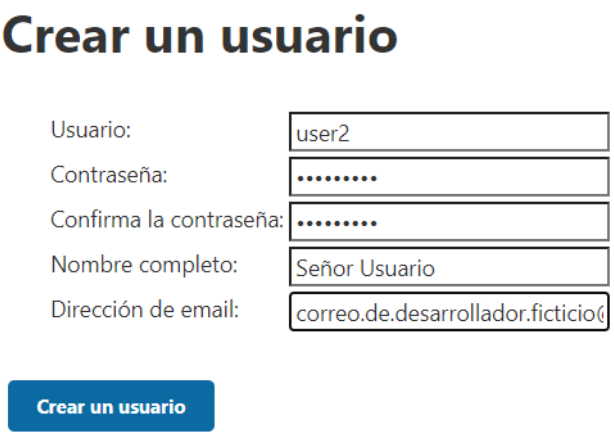


Figura 4.93: Crear usuarios, parte II

Una vez aceptado nos aparecerá en la lista de usuarios y podrá conectarse con el usuario y contraseña establecidos. También podrá configurarse la información del usuario en el símbolo del engranaje a la derecha del nombre de usuario.

Figura 4.94: Crear usuarios, parte III

Una vez el usuario está creado, si no lo ha hecho anteriormente, el usuario deberá vincular su correo electrónico en la rama del repositorio de git. Para ello se usa el comando de git **Config user.email [Correo electrónico]** en la consola de comandos.

```
D:\Programacion\Jenkins\DevOpsExample>git config user.email correo.de.desarrollador.ficticio@gmail.com

D:\Programacion\Jenkins\DevOpsExample>git config user.email
correo.de.desarrollador.ficticio@gmail.com
```

Figura 4.95: Vinculación de correo a git

Añadir correos

Jenkins no mandará notificaciones automáticas por defecto a correos electrónicos que no estén registrados en el repositorio. Se puede desactivar la restricción, pero es una apertura de seguridad que es mejor cubrir.

Para agregar un correo electrónico en GitHub vamos a la página del proyecto en la página web de GitHub y vamos a la sección de "Settings>Manage access".

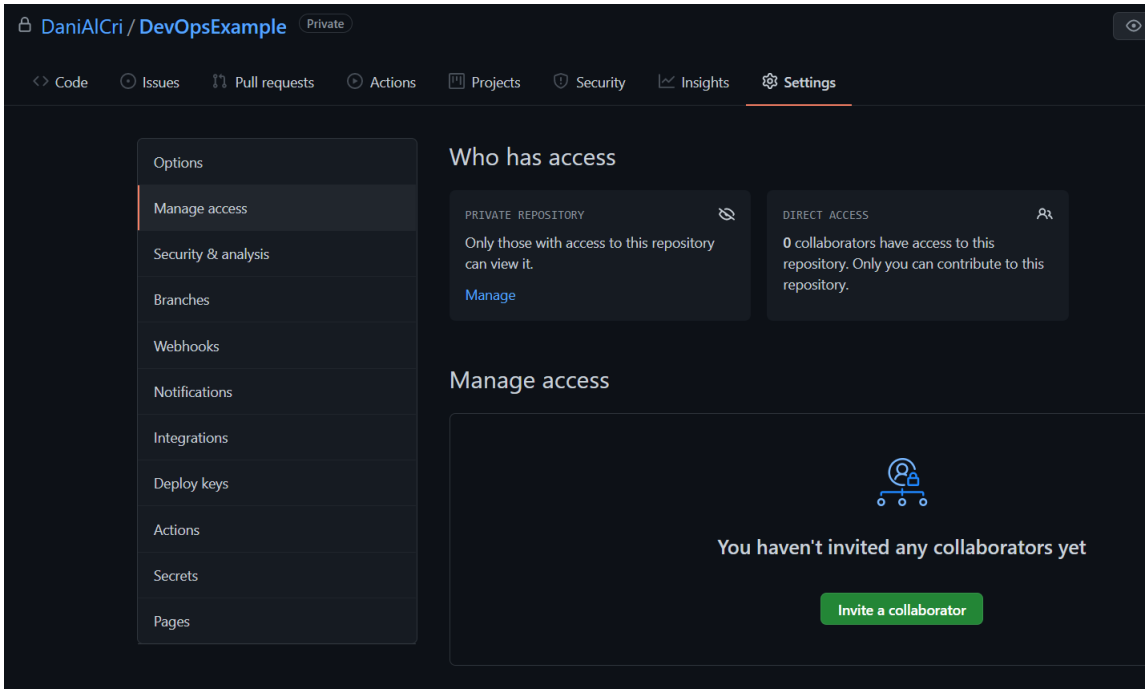


Figura 4.96: Menú de configuración de GitHub

Hacemos clic en el botón de invitar colaborador y añadimos uno de los emails que queramos añadir. Ese email recibirá un correo de confirmación y cuando lo acepte podrá recibir notificaciones. La cuenta de correo electrónico debe estar vinculada a una cuenta de GitHub para poder ser invitada.

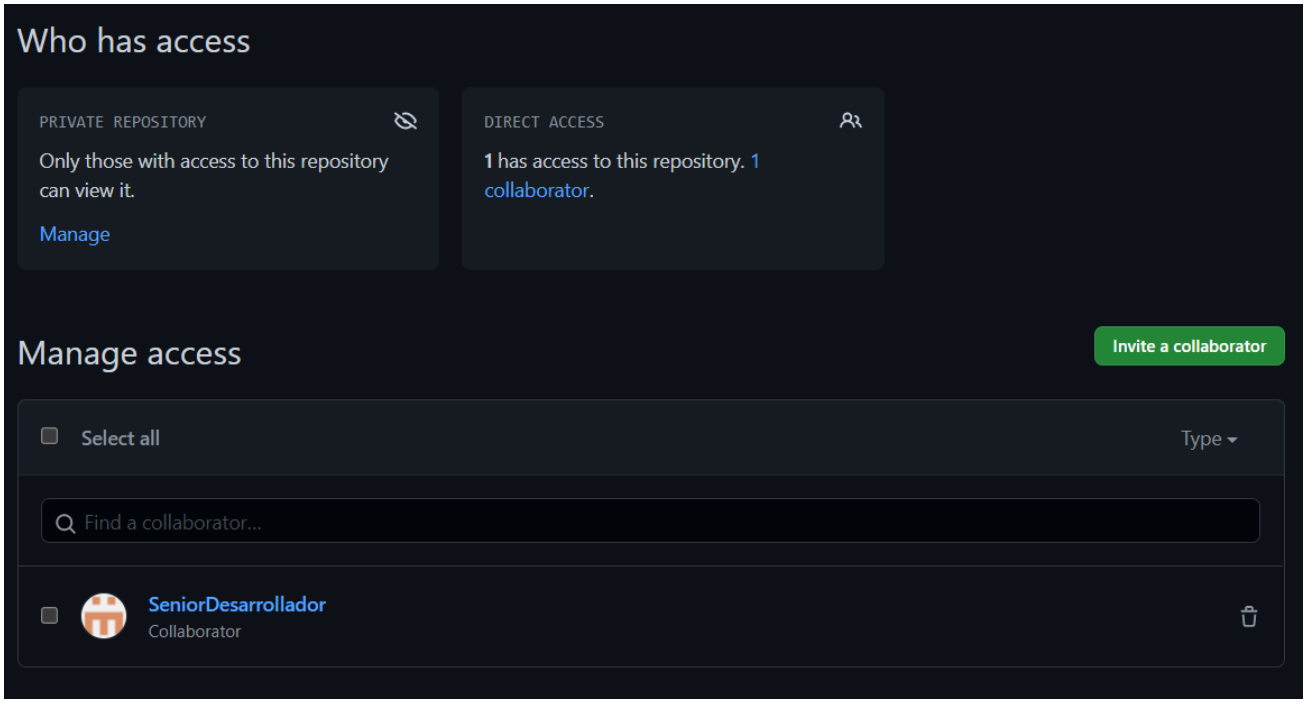


Figura 4.97: Invitación de participación de GitHub aceptada

4.8.5. Crear un canal a GitHub

Jenkins tiene un maestro y agentes. El maestro detecta cambios en el repositorio y crea instancias de agentes según necesite. Los agentes ejecutan el archivo Jenkinsfile del repositorio, que es un archivo de configuración que define el pipeline o canal. El canal es un proceso abstracto de automatización de tareas.

Jenkins puede automatizar prácticamente cualquier proceso, aunque tiene algunas limitaciones. El canal más habitual compila y prueba el código del repositorio en cada commit, mandando un mensaje al desarrollador que realizó el commit si hubo algún error.

El Jenkinsfile se debe subir a la raíz del repositorio del proyecto para que lo encuentre Jenkins. Hay tres formas para crear el Jenkinsfile: Blue Ocean, Jenkins UI o escribirlo a mano.

Blue Ocean es un nuevo plugin creado por CloudBees (desarrolladores de Jenkins) para que el proceso de creación de pipelines sea más sencillo. En un futuro Blue Ocean reemplazará completamente a la UI tradicional. A día de hoy todavía no puede ajustarse a proyectos muy específicos.

Jenkins UI es la herramienta básica y se accede desde el propio Jenkins. Es completamente funcional, pero algo más incómoda de usar respecto a otros editores

por UI. También se pueden escribir los Jenkinsfile a mano como un archivo de YAML. Es una manera fiable e instructiva de programar jenkins.

Conectar un canal a GitHub

Para crear el canal usaremos Blue Ocean. En el menú de Jenkins hacemos click en "Open Blue Ocean". Para empezar hacemos click en "Create New Pipeline". Esto nos permitirá conectar Jenkins a un repositorio y crear un archivo Jenkinsfile si no existe ya uno en el proyecto. Es más rápido y sencillo conectarse mediante Blue Ocean que manualmente.

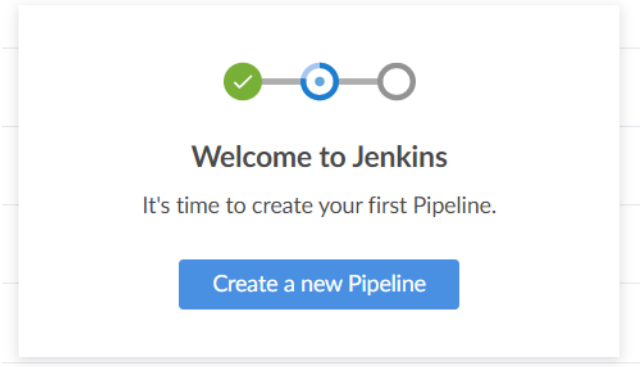


Figura 4.98: Creación de Jenkinsfile por BlueOcean, Parte I

Elegimos el tipo de repositorio usado, en nuestro caso github.

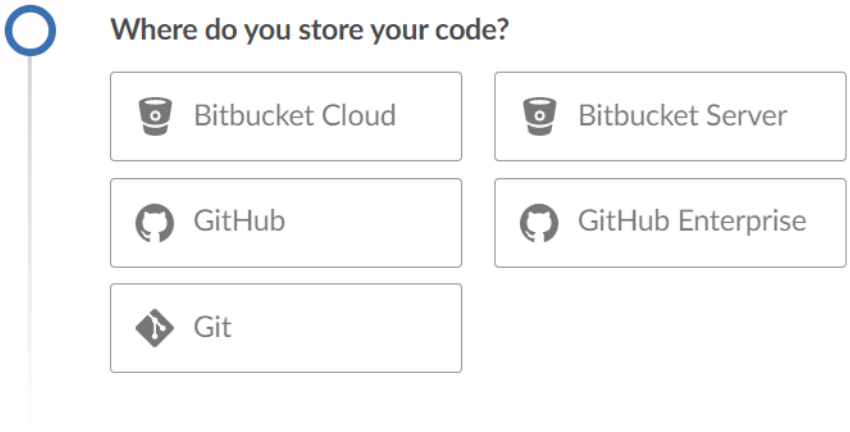


Figura 4.99: Creación de Jenkinsfile por BlueOcean, Parte II

Nos pedirá un token de acceso a GitHub para autorizar los cambios. Si no se tiene uno se puede crear uno haciendo click en "Create an access token here". Eso

nos llevará a GitHub donde tendremos que identificarnos con la contraseña y seguir los pasos que se indican.

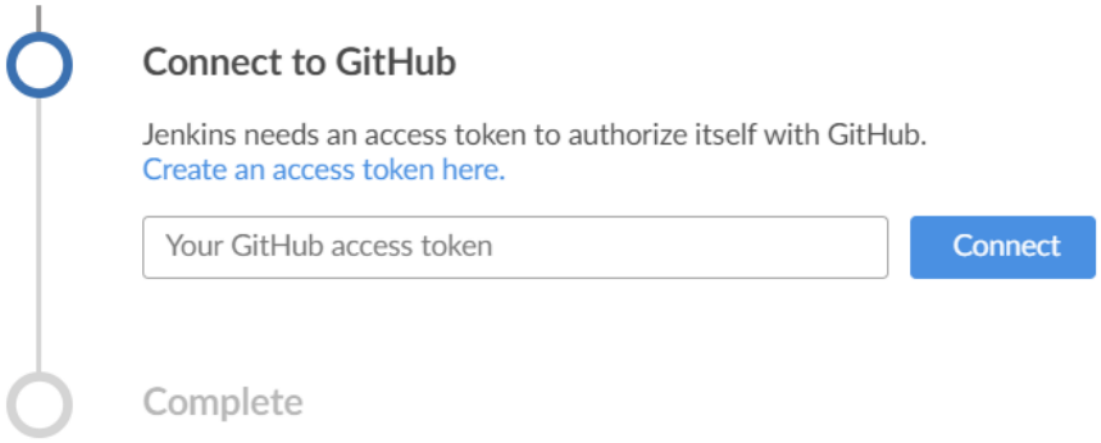


Figura 4.100: Creación de Jenkinsfile por BlueOcean, Parte III

Una vez identificados podremos seleccionar a qué características se tendrá acceso con ese token. Lo dejaremos por defecto. Añadimos en "Note" para qué vamos a usar el token (Para organizar los tokens). Cuando terminamos hacemos click en "Generate token".

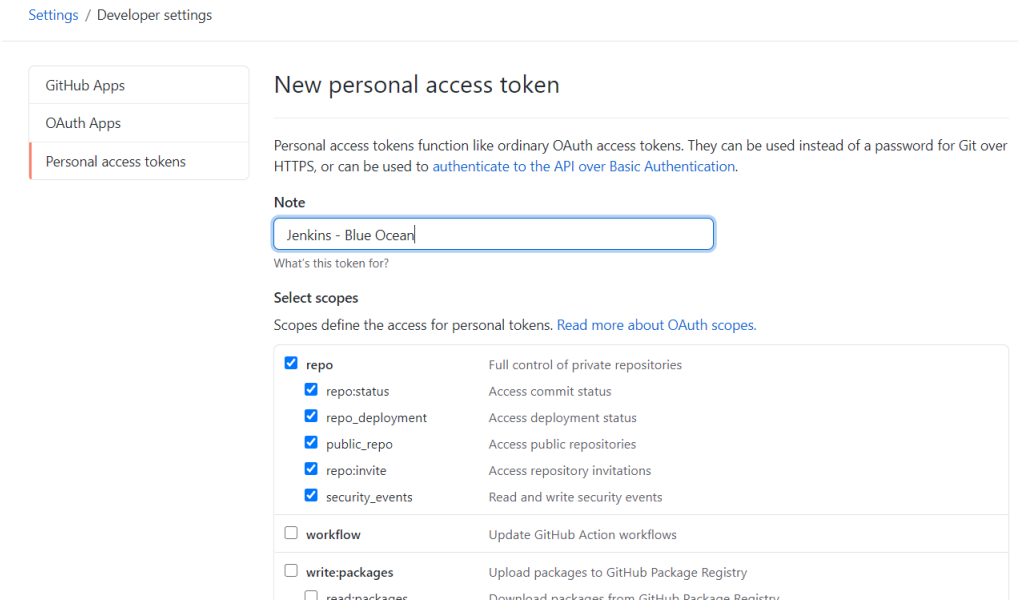


Figura 4.101: Creación de Jenkinsfile por BlueOcean, Parte IV

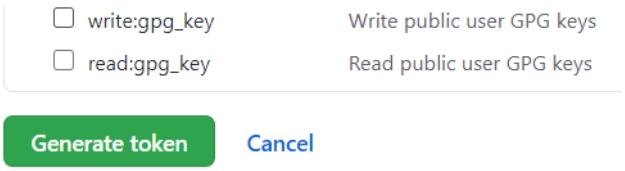


Figura 4.102: Creación de Jenkinsfile por BlueOcean, Parte V

Ahora nos aparecerá en GitHub el nuevo token con un código. Ese código se debe guardar en un lugar seguro porque será borrado al abandonar la página y no se podrá volver a ver. Si se pierde el código y vuelve a hacer falta se deberá crear un nuevo token. Se pueden (y deben) borrar los tokens que ya no se necesiten haciendo click en "Delete".

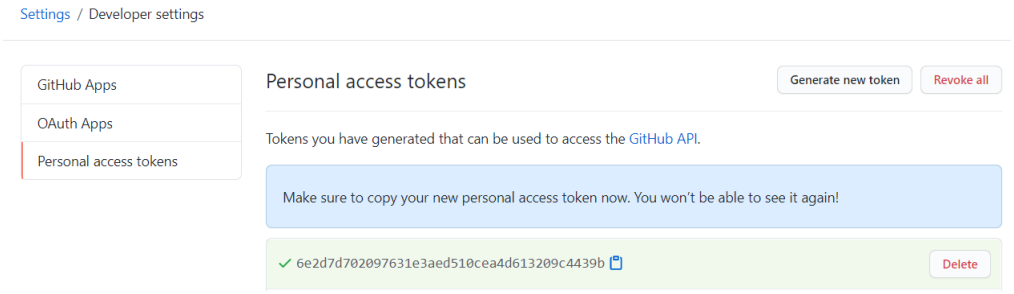


Figura 4.103: Creación de Jenkinsfile por BlueOcean, Parte VI

Volvemos a Blue Ocean y hacemos click en conectar. Elegimos la organización, es este caso sólo tenemos uno.

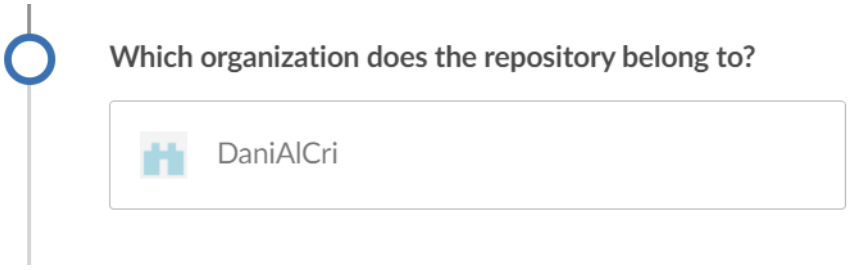


Figura 4.104: Creación de Jenkinsfile por BlueOcean, Parte VII

Elegimos el repositorio deseado. Vamos a elegir el repositorio "DevOpsExample" y hacemos click en "Create Pipeline".

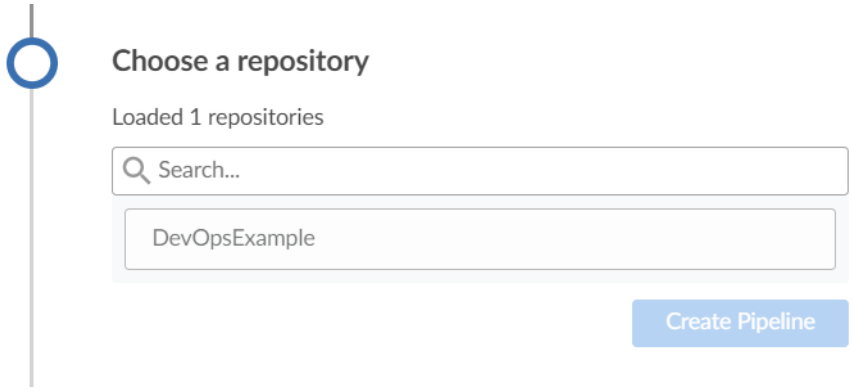


Figura 4.105: Creación de Jenkinsfile por BlueOcean, Parte VIII

Para este proyecto usaremos un Jenkinsfile escrito a mano. Si no lo tuviéramos Blue Ocean crearía uno por defecto y nos llevaría directamente a la pantalla de modificación/creación de Jenkinsfile de Blue Ocean. También podemos acceder a este menú para editar el Jenkinsfile existente.

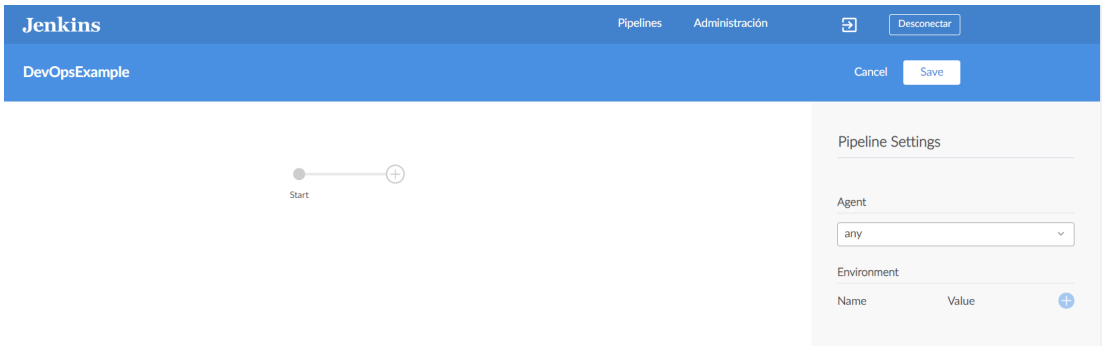


Figura 4.106: Creación de Jenkinsfile por BlueOcean, Parte IX

4.8.6. Componentes de un canal

El Jenkinsfile puede definir en gran profundidad cómo lo procesa Jenkins, pero en este apartado vamos a identificar sólo los recursos clave.

Agent

El agente [8] es el tipo de máquina que ejecutará el código procesado. Hay que declarar el tipo de agente (o que no hay agente) al principio del documento, este será el agente superior (top agent) que se ejecutará por defecto. En cada stage (etapa) se puede declarar opcionalmente un agente de etapa (stage agent) que sólo se aplicará durante esa etapa. Los distintos tipos de agente disponibles de manera nativa son:

any : Usará cualquier agente disponible.

none : Sólo es válido como agente superior, implica que cada etapa especificará individualmente su propio agente.

label : Usará como agente una imagen de docker identificada mediante una etiqueta.

node : Es similar a "label" pero con más opciones, como asignar un workspace concreto.

docker : El agente será un contenedor de Docker, para el cual podremos elegir la etiqueta. Se pueden añadir varias opciones como pasar argumentos o registryUrl y registryCredentialsId que sirven para acceder a un repositorio privado.

dockerfile : Igual a un agente docker, pero se carga la imágen de un dockerfile alojado en el repositorio fuente al que está conectado jenkins.

kubernetes : El agente es un pod de kubernetes. Se debe definir la plantilla del pod justo después en el bloque "kubernetes ". Se puede usar un archivo YAML.

Environment

Aquí van las variables de entorno. Las variables se pueden declarar al comienzo para todo el Jenkinsfile o en un stage para que sólo tenga efecto ahí. Es útil para guardar rutas, versiones, credenciales, etc. Los valores de las variables se declaran entre comillas simples <'>.

Stages

Aquí se indican el número de etapas por el que va a pasar todo código procesado por Jenkins y el nombre de la etapa. Dentro de cada etapa se puede añadir una serie de opciones, comandos y scripts.

Post

Aquí se establece qué ocurre cuando acabe el proceso. Se pueden usar condicionales como que ejecute un script si la compilación ha sido exitosa o que mande un correo electrónico en caso contrario.

```
---
# Comentario
pipeline {
  agent any
  environment {
    ENV_VARIABLE = 'valor'
  }
  stages {
    stage('Etapa1') {
      steps {
        // Aquí se declara qué ocurre en la etapa por cada paso
      }
    }
  }

  post {
    /* No es obligatorio, pero es muy útil. Aquí se declara que ocurre cuando
    acaba la ejecución (sea porque ha fallado o completado exitosamente).*/

    failure {
      //Lo habitual es que aquí se mande un correo al commiter
    }
    // success -> Si finaliza con éxito
    // failure -> Si algún paso ha fallado
    // always -> Siempre al acabar la ejecución
  }
}
```

Figura 4.107: Ejemplo de Jenkinsfile básico

4.8.7. Estructura del canal

Una vez sabemos cómo funcionan las partes del canal podemos crear una estructura interna. Esto se hace mediante las etapas. El modelo más habitual para un entorno completamente automatizado usa cuatro etapas (las dos últimas se pueden juntar):

Build : Prepara el entorno. Descarga y compila el código.

Tests : Ejecuta los test unitarios del código. Si fallan la etapa da error y devuelve el log de errores.

Push : Hacer un push de docker para subir la nueva imágen al repositorio.

Deploy : Se actualiza el servidor, o el servidor canario de existir.

Para este ejemplo vamos a crear sólo las etaaps de build y test. También explicaremos cómo funcionaría una tercera rama de push/deploy.

También debemos elegir el agente que usaremos y las variables de entorno que necesitaremos. Si necesitamos uno de estos recursos específicamente para una etapa, lo declararemos dentro de la etapa antes de los pasos. Así sólo se ejecutará para esa etapa.

```
...
stage('Etapa1') {
  agent {
    label 'custom-agent'
  }
  steps {
    // Aquí se declara qué ocurre en la etapa por cada paso
  }
}
```

Figura 4.108: Agente de Jenkins en el canal

Dentro de cada etapa hay una sección de pasos. Dentro de pasos, el agente ejecutará todos los comandos que se le pidan. Por defecto se usan los comandos de Jenkins, pero se puede hacer que ejecute comandos nativos del agente con los comandos sh (para linux) o bat (para windows).

4.8.8. Trabajo desde el punto de vista del usuario

Cuando el usuario sube un cambio al repositorio, Jenkins lo capta, lo prueba y manda los correos pertinentes. Si hay un fallo, el usuario puede entrar en la dirección de Jenkins con su usuario y buscar el registro de la compilación. Para ello se puede usar Blue Ocean o el registro de la herramienta.

Se puede entrar al menú de Blue Ocean desde la página de inicio. Seleccionamos el proyecto y buscamos el código del cambio que falló.

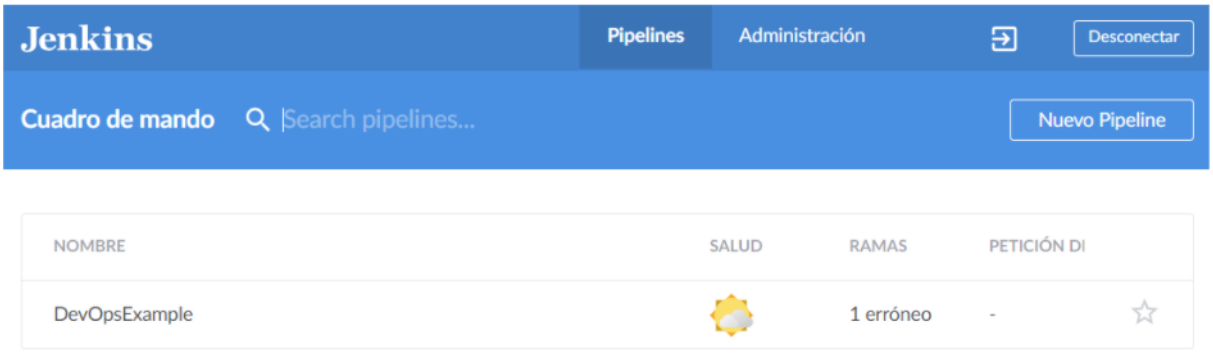


Figura 4.109: Lista de ramas del repositorio

Figura 4.110: Lista de compilaciones

Cuando lo abramos podremos ver todos los pasos del canal y sus notificaciones. Si el código falló, nos mostrará la salida de la ejecución con sus errores.

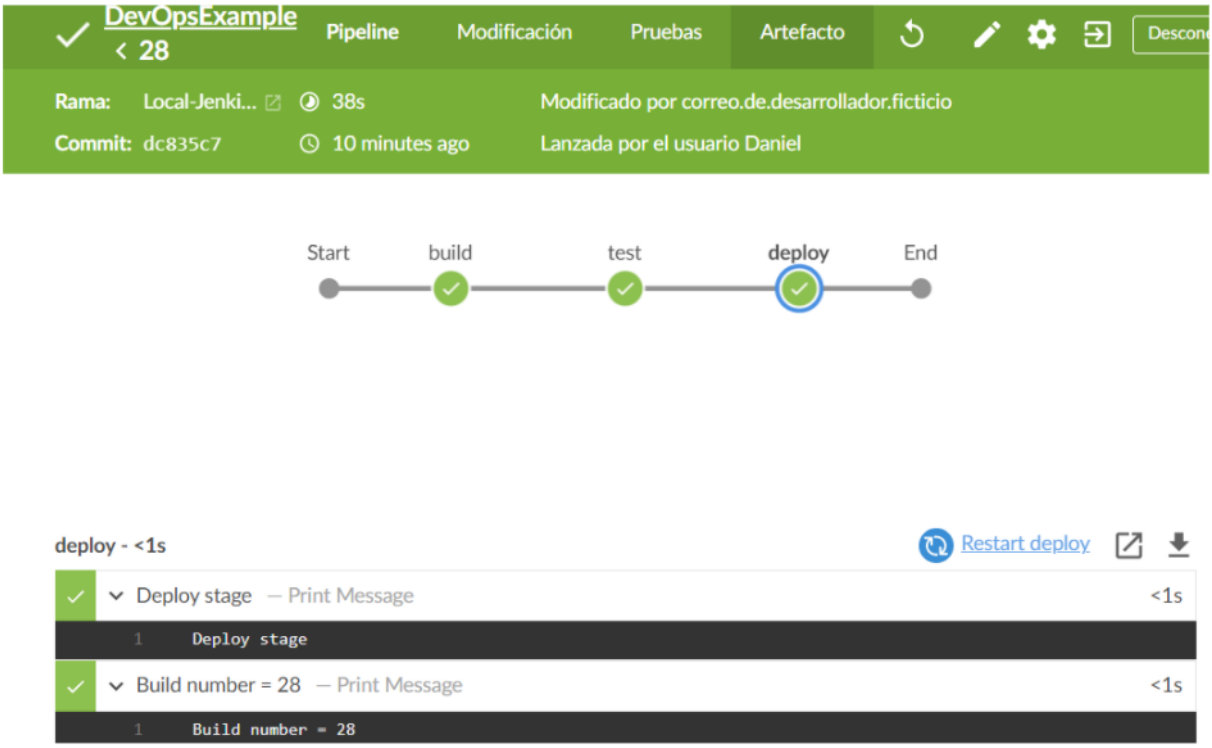


Figura 4.111: Compilación exitosa

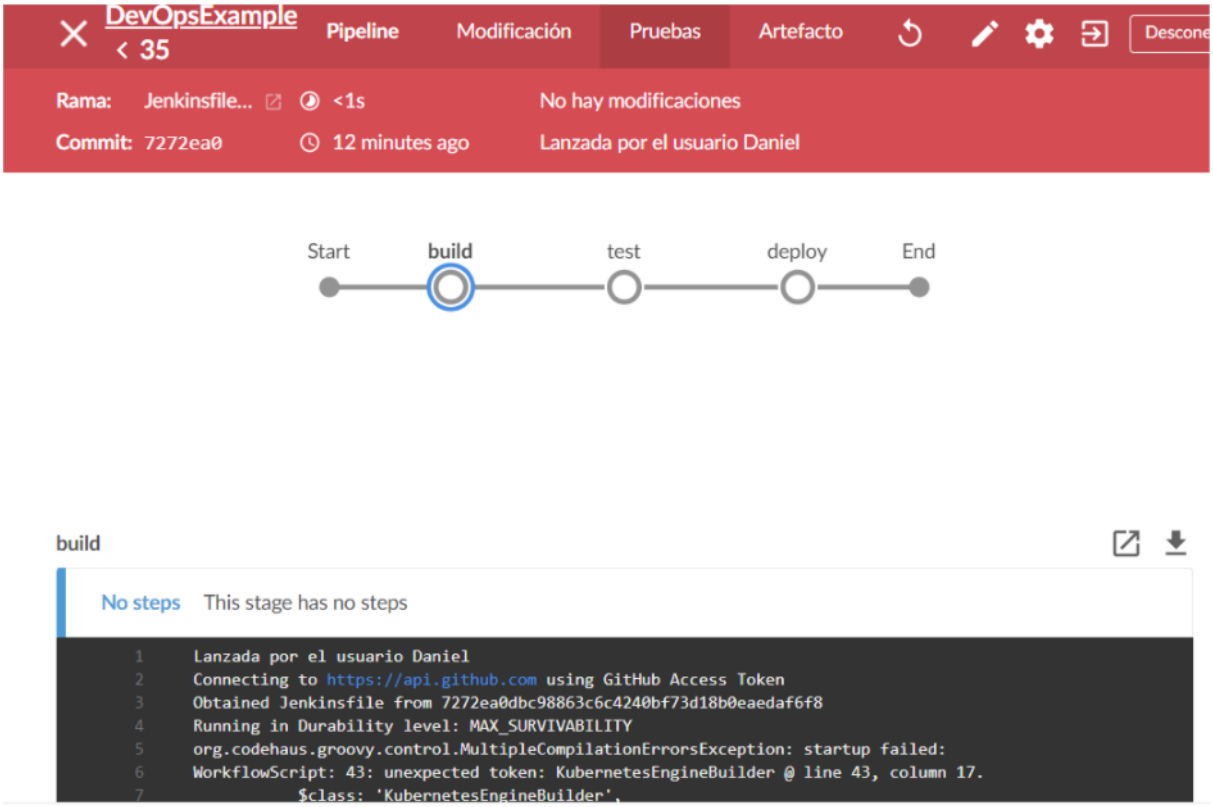


Figura 4.112: Compilación errónea

4.9. Servidor de Integración Continua

En este apartado aplicaremos las tecnologías que hemos revisado para crear un servidor de DevOps para integración contínua. El resultado final será un servidor de Jenkins montado sobre kubernetes y conectado a un repositorio de GitHub. Para ver mejor el proceso se usará el ejemplo de node que vimos anteriormente.

Este proyecto no va a usar un CI/CD completamente automatizado porque el sistema está orientado como introducción y no tiene las restricciones adecuadas para proteger el repositorio de errores humanos (uno de los objetivos del CI/CD). Por ello, este sistema no automatiza las tareas críticas, tal y como se debería hacer en un sistema que no está en un estado de desarrollo lo suficientemente avanzado como para interactuar directamente con estas tareas críticas sin supervisión.

4.9.1. Desplegar Jenkins en Kubernetes con GCP

De manera nativa Jenkins puede ejecutar varios agentes desde el propio servicio para procesar los Jenkinsfile. Para un caso pequeño es suficiente, pero en proyectos

grandes es un problema de seguridad que el nodo maestro ejecute el código por sí mismo. Esto se debe a que si la compilación congela o hace caer al agente, se lleva consigo a todo el servidor. Para solventar este problema usaremos Kubernetes para generar pods que actúen como agentes dinámicamente.

Planificar la arquitectura

Lo primero es planificar la estructura que vamos a utilizar. Lo habitual con Jenkins es configurar una arquitectura de maestro/esclavo. En este tipo de arquitectura una de las instancias (Pod maestro) organiza al resto de instancias (Pod ejecutor) para que realicen las tareas de manera eficiente. Es posible establecer cómo se mandan las tareas. Las dos estrategias más habituales son restringir qué tipo de tarea va a qué nodo para establecer prioridades, o crear una cola de tareas y usar todos los nodos libres.

En este ejemplo vamos a replicar la estructura del ejemplo en el libro [1] en la que hay un maestro y tres esclavos en dos nodos o instancias.

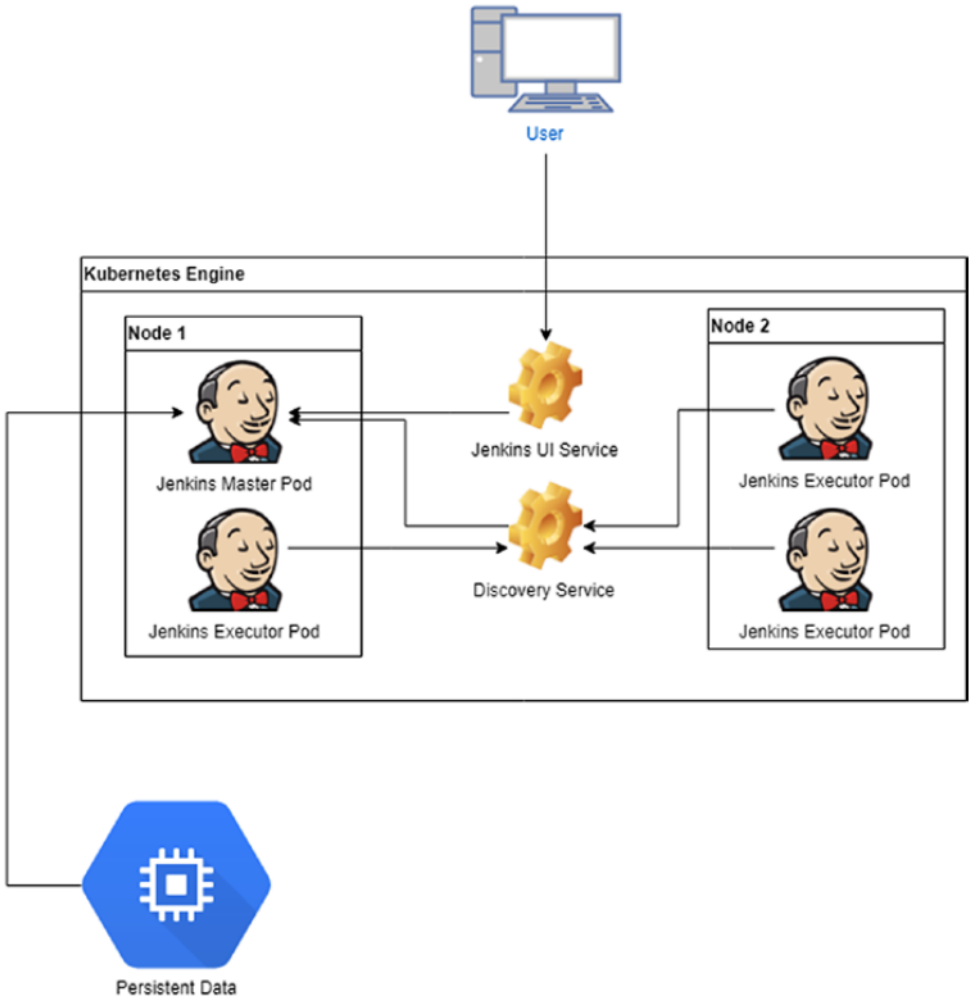


Figura 4.113: Esquema de servidor de CI/CD en Jenkins

En la imagen **??** podemos ver que el flujo de trabajo es el siguiente:

1. El usuario se comunica con el servicio "Jenkins UI" mediante el puerto 8080 .
2. El servicio "Jenkins UI" envía una tarea al pod "Jenkins Master Pod" que organiza la orden y la reparte entre los nodos esclavos "Jenkins Execution Pod".
3. Los "Jenkins Execution Pod" envían los resultados al servicio "Discovery Service" que a su vez redirecciona otra vez al pod maestro para que lo almacene en la memoria persistente y responda al usuario.

Creación de archivo de configuración

Esta estructura se puede establecer con un archivo de configuración de YAML. Vamos a tomar como referencia el archivo YAML de los ejemplos oficiales de Google Cloud, que también fué usado en el libro "Pro DevOps with Google Cloud Platform" [1] que se usa como referencia en este proyecto. El archivo de configuración creará los servicios “Jenkins UI” y “Discovery”. Se pueden unir ambos archivos en uno para simplificar la estructura de carpetas.

En el archivo de configuración se inicia con tres guiones seguidos. Cada elemento debe empezar por tres guiones y separa un archivo de otro. Es recomendable delimitar también mediante comentarios (usando ‘’) dónde empieza y donde termina cada archivo.

En los archivos primero se determina qué tipo de archivo es mediante “kind”. En este caso es un servicio con la versión de API v1.

```
---
kind: Service
apiVersion: v1
```

Figura 4.114: Archivo de configuración de ejemplo, Parte I

Con metadata se pueden asignar identificadores como el nombre del servicio, etiquetas o el espacio de nombre. Las etiquetas se añaden con “labels” y escribiendo en el siguiente nivel de indentación los pares de etiqueta-valor. El nombre del servicio creado se establece con “name” y su espacio de nombre con “namespace”.

```
metadata:
  name: jenkins-ui
  namespace: jenkinsgcp
labels:
  role: master
  label1: value1
  label2: value2
```

Figura 4.115: Archivo de configuración de ejemplo, Parte II

En spec se introduce la información de cómo es por dentro el servicio. Se puede seleccionar un recurso con “selector”, designar el tipo de servicio con “type”, asignar

los puertos con “ports”, crear contenedores con “container” o crear plantillas con “template”.

El tipo sirve para configuraciones internas, podemos ver cuál es el correcto en la documentación oficial de Kubernetes [6]. El tipo Nodeport expone todos los puertos que creemos en este spec como puertos estáticos a los que se pueden acceder. Seleccionar un recurso sirve para vincular el servicio con el resto de servicios y que funcionen conjuntamente, seleccionamos la etiqueta “app: master” para conectar los dos servicios. Sólo nos falta crear puertos, para ello añadimos el protocolo, el puerto, puerto de destino y el nombre del puerto. Usamos el puerto 8080 porque es por donde accederemos a la interfaz gráfica de Jenkins.

```
spec:
  type: NodePort
  selector:
    app: master
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
      name: ui
```

Figura 4.116: Archivo de configuración de ejemplo, Parte III

Para el servicio jenkins_discovery es prácticamente igual, pero en spec no lleva tipo, usa el puerto 50000 y al puerto se le pone el nombre “slaves”.

```
spec:
  selector:
    app: master
  ports:
    - protocol: TCP
      port: 50000
      targetPort: 50000
      name: slaves
```

Figura 4.117: Archivo de configuración de ejemplo, Parte IV

El resultado final sería el siguiente archivo yaml o dos archivos de yaml separados por donde marcan los comentarios.

```yaml
#[START jenkins_service_ui]
---
kind: Service
apiVersion: v1
metadata:
  name: jenkins-ui
  namespace: jenkinsgcp
spec:
  type: NodePort
  selector:
    app: master
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
      name: ui
# [END jenkins_service_ui]
```

Figura 4.118: Jenkins UI Service

```
# [START jenkins_service_discovery]
---
kind: Service
apiVersion: v1
metadata:
  name: jenkins-discovery
  namespace: jenkinsgcp
spec:
  selector:
    app: master
  ports:
    - protocol: TCP
      port: 50000
      targetPort: 50000
      name: slaves
# [END jenkins_service_discovery]
```

Figura 4.119: Jenkins Discovery Service

Preparar un cluster para Jenkins

Antes de empezar creamos una carpeta (por ejemplo "DevOpsServer") para no ensuciar la cloud shell. Dentro usamos el comando git clone con nuestro proyecto para poder acceder a los archivos.

El primer paso es crear un cluster. Usaremos 2 nodos, en la zona "europe-west1-b" y con la imágen de una api. Hay cuatro apis de gcp para repositorios que estan listadas en `https://developers.google.com/identity/protocols/oauth2/scopes#sourcerepo`

`https://www.googleapis.com/auth/cloud-platform` : Esta API es para utilizar los datos de los repositorios entre los distintos servicios de GCP.

`https://www.googleapis.com/auth/source.full_control` : Esta API es para administrar repositorios. Esta API tiene permisos para configurar el repositorio y es más adecuada para un servicio de administración. Sería un problema de seguridad dar acceso a esta API a los desarrolladores porque se les daría más control que el que necesitan.

`https://www.googleapis.com/auth/source.read_only` : Esta API permite leer repositorios, pero no modificar el contenido. Sería apropiada para trabajos de control de calidad.

`https://www.googleapis.com/auth/source.read_write` : Esta API permite leer y escribir en repositorios. Es lo ideal para un trabajador que debe subir código al repositorio.

Vamos a usar la API de source.read_write, porque este es el sistema que utilizarían los desarrolladores. Sería posible crear servicios web más especializados para otros roles, pero eso también aumentaría el precio.

```
dalvac01@cloudshell:~/DevOpsServer (proyectokubernetes-301509)$ gcloud container clusters create jenkins-devops
  --machine-type n1-standard-2 --zone europe-west1-b --num-nodes 2   --scopes "https://www.googleapis.com/auth/
source.read_write,cloud-platform"
WARNING: Starting in January 2021, clusters will use the Regular release channel by default when `--cluster-ver
sion`, `--release-channel`, `--no-enable-autoupgrade`, and `--no-enable-autorepair` flags are not specified.
WARNING: Currently VPC-native is not the default mode during cluster creation. In the future, this will become
the default mode and can be disabled using `--no-enable-ip-alias` flag. Use `--[no-]enable-ip-alias` flag to su
ppress this warning.
WARNING: Starting with version 1.18, clusters will have shielded GKE nodes by default.
WARNING: Your Pod address range (`--cluster-ipv4-cidr`) can accommodate at most 1008 node(s).
WARNING: Starting with version 1.19, newly created clusters and node-pools will have COS_CONTAINERD as the defa
ult node image when no image type is specified.
Creating cluster jenkins-devops in europe-west1-b... Cluster is being health-checked (master is healthy)...done
.
Created [https://container.googleapis.com/v1/projects/proyectokubernetes-301509/zones/europe-west1-b/clusters/j
enkins-devops].
To inspect the contents of your cluster, go to: https://console.cloud.google.com/kubernetes/workload_/gcloud/eu
rope-west1-b/jenkins-devops?project=proyectokubernetes-301509
kubeconfig entry generated for jenkins-devops.
NAME            LOCATION        MASTER_VERSION   MASTER_IP     MACHINE_TYPE   NODE_VERSION     NUM_NODES  STATU
S
jenkins-devops  europe-west1-b  1.19.9-gke.1900  35.187.49.46  n1-standard-2  1.19.9-gke.1900  2          RUNNI
NG
```

Figura 4.120: gcloud container clusters create jenkins-devops –zone europe-west1-b –num-nodes 2 –scopes "https://www.googleapis.com/auth/source.read_write,cloud-platform"

Comprobamos que el cluster se ha creado apropiadamente y que tiene conexión. para ello usamos el comando kubectl cluster-info.

```
dalvac01@cloudshell:~/DevOpsServer (proyectokubernetes-301509)$  gcloud container clusters list
NAME            LOCATION        MASTER_VERSION   MASTER_IP     MACHINE_TYPE   NODE_VERSION     NUM_NODES  STATUS
jenkins-devops  europe-west1-b  1.19.9-gke.1900  35.187.49.46  n1-standard-2  1.19.9-gke.1900  2          RUNNING
dalvac01@cloudshell:~/DevOpsServer (proyectokubernetes-301509)$  kubectl cluster-info
Kubernetes control plane is running at https://35.187.49.46
GLBCDefaultBackend is running at https://35.187.49.46/api/v1/namespaces/kube-system/services/default-http-backend:http/proxy
KubeDNS is running at https://35.187.49.46/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy
Metrics-server is running at https://35.187.49.46/api/v1/namespaces/kube-system/services/https:metrics-server:/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

Figura 4.121: kubectl cluster-info

Instalar Jenkins

Para facilitar la instalación de Jenkins, vamos a instalar el administrador de paquetes para kubernetes: Helm. Descargamos Helm del repositorio mediante wget y lo extraemos. Se debe comprobar que la versión de Helm es la última estable.

```
dalvac01@cloudshell:~/DevOpsServer (proyectokubernetes-301509)$ wget https://get.helm.sh/helm-v3.2.1-
linux-amd64.tar.gz seguido de tar zxfv helm-v3.2.1-linux-amd64.tar.gz y
--2021-07-04 16:53:14--  https://get.helm.sh/helm-v3.2.1-linux-amd64.tar.gz
Resolving get.helm.sh (get.helm.sh)... 152.199.21.175, 2606:2800:233:1cb7:261b:1f9c:2074:3c
Connecting to get.helm.sh (get.helm.sh)|152.199.21.175|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 12927632 (12M) [application/x-tar]
Saving to: 'helm-v3.2.1-linux-amd64.tar.gz'

helm-v3.2.1-linux-amd64.tar.gz                  100%[=============================================
============================================================>]  12.33M  --.-KB/s    in 0.1s

2021-07-04 16:53:14 (106 MB/s) - 'helm-v3.2.1-linux-amd64.tar.gz' saved [12927632/12927632]
```

Figura 4.122: wget https://get.helm.sh/helm-v3.2.1-linux-amd64.tar.gz || tar zxfv helm-v3.2.1-linux-amd64.tar.gz || cp linux-amd64/helm .

A continuación nos daremos permisos de administrador para poder otorgar permisos a Jenkins.

```
dalvac01@cloudshell:~ (proyectokubernetes-301509)$ kubectl create clusterrolebinding cluster-admin-binding
--clusterrole=cluster-admin \
>         --user=$(gcloud config get-value account)
Your active configuration is: [cloudshell-21026]
clusterrolebinding.rbac.authorization.k8s.io/cluster-admin-binding created
```

Figura 4.123: kubectl create clusterrolebinding cluster-admin-binding –clusterrole=cluster-admin –user=$(gcloud config get-value account)

Actualizamos los repositorios de Helm.

```
dalvac01@cloudshell:~ (proyectokubernetes-301509)$ ./helm repo add jenkinsci https://charts.jenkins.io
"jenkinsci" has been added to your repositories
dalvac01@cloudshell:~ (proyectokubernetes-301509)$ ./helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "jenkinsci" chart repository
Update Complete. ⎈ Happy Helming!⎈
```

Figura 4.124: ./helm repo add jenkinsci https://charts.jenkins.io || ./helm repo update

Creamos el archivo YAML del apartado “Creación del archivo de configuración”4.9.1 o lo descargamos de un repositorio de git. Metemos el archivo en una carpeta “Jenkins“ por organizarlo y lo usamos para instalar jenkins con helm.

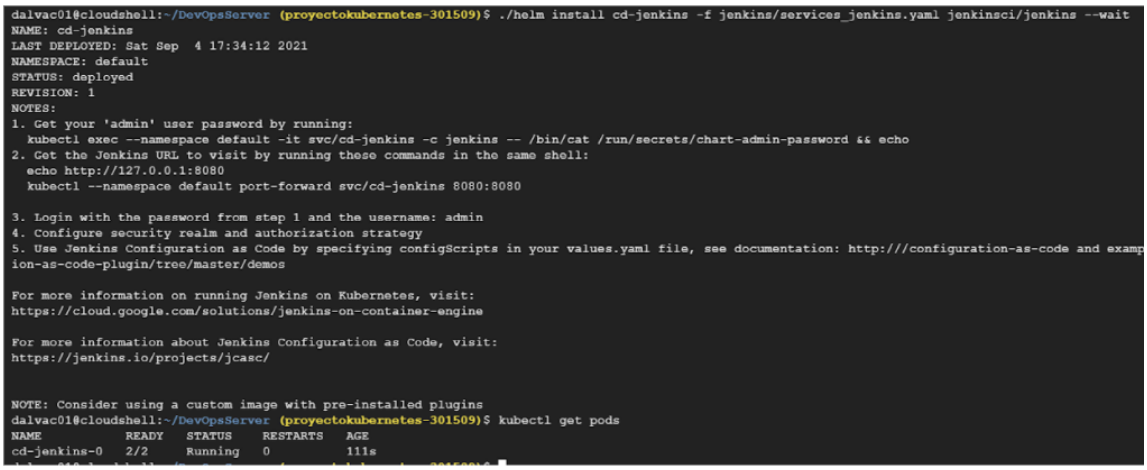


Figura 4.125: ./helm install cd-jenkins -f jenkins/services_jenkins.yaml jenkinsci/jenkins –wait || kubectl get pods

Conectar con el servidor Jenkins

Conectamos el pod de jenkins con el puerto 8080. Primero seleccionamos todos los pods en el espacio de nombre default, si hubiera más pods en ese espacio tendríamos que cambiar el namespace o introducir los pods manualmente. Después conectamos el puerto con el pod seleccionado. Podemos comprobar que la configuración es exitosa si al usar el comando **kubectl get svc** se muestran un cluster de jenkins abierto al puerto 8080, otro al puerto 50000 y otro cluster de kubernetes.

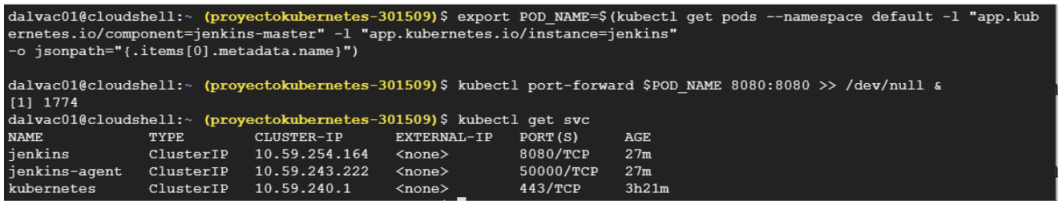


Figura 4.126: export POD_NAME=$(kubectl get pods –namespace default -l "app.kubernetes.io/component=jenkins-master" -l "app.kubernetes.io/instance=jenkins" -o jsonpath="{.items[0].metadata.name}") || kubectl port-forward $POD_NAME 8080:8080 » /dev/null kubectl get svc

Ahora que Jenkins está instalado, podremos conectarnos mediante la consola de comandos.

Primero recuperamos la contraseña generada por Helm con el comando **printf $(kubectl get secret cd-jenkins -o jsonpath="{.data.jenkins-admin-password}"**

| base64 –decode);echo Podemos cambiar la contraseña modificando el archivo jenkins
config.xml

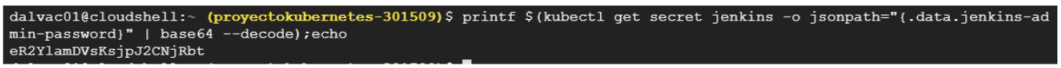


Figura 4.127: printf $(kubectl get secret jenkins -o jsonpath="{.data.jenkins-admin-password}" | base64 –decode);echo

Una vez tenemos la contraseña vamos a la consola web y seleccionamos el icono de "Vista previa web > Obtener vista previa en puerto 8080".

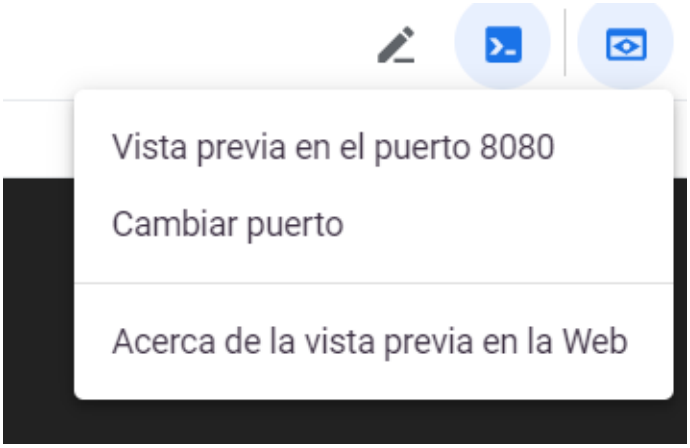


Figura 4.128: Conexión a Jenkins mediante el puerto 8080 de GCP

Esto nos llevará al login de Jenkins. El usuario es "admin" y la contraseña la anterior.

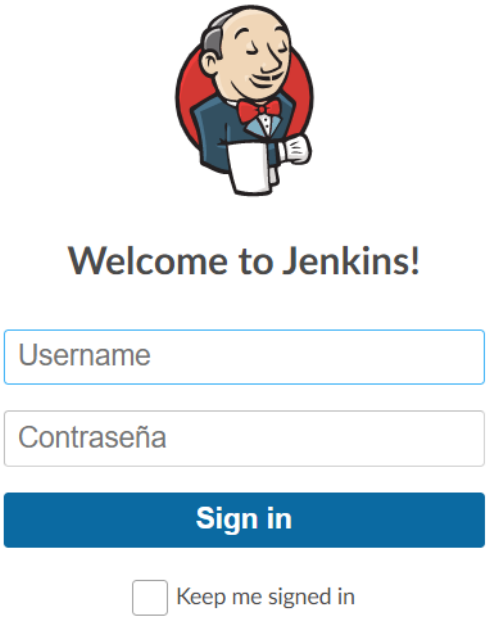


Figura 4.129: LogIn de Jenkins

Finalmente acabamos en la UI de jenkins donde podremos realizar más operaciones.

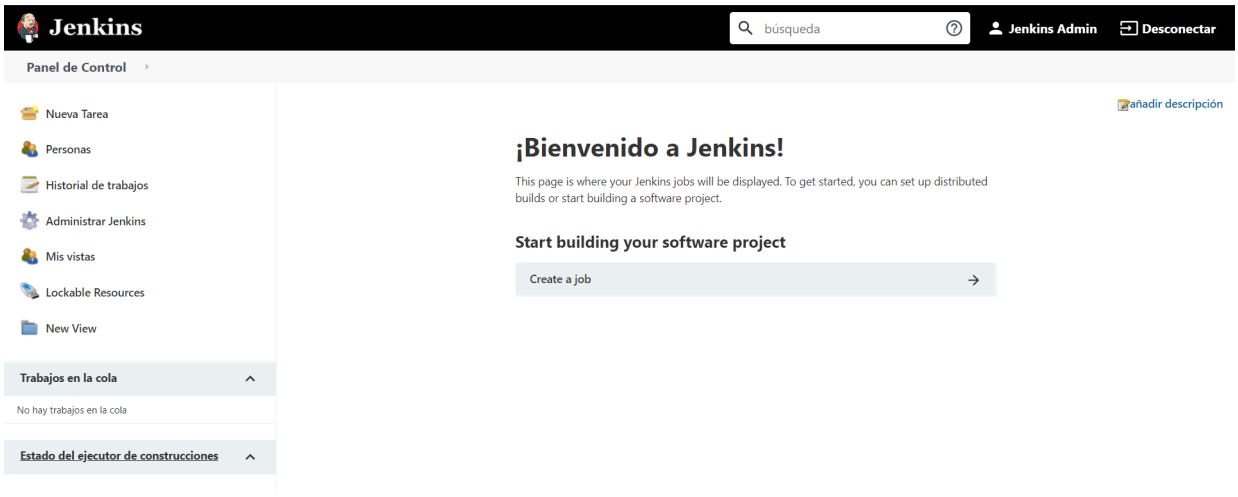


Figura 4.130: Menú de Jenkins

4.9.2. Configuración básica de Jenkins

En este punto tenemos un cluster de kubernetes con jenkins instalado y un repositorio de github. Gran parte de la configuración de jenkins viene ya hecha por el archivo Yaml. Lo más importante es que ha configurado automáticamente la nube,

la creación de pods como agentes y las credenciales. También vienen algunos plugins preinstalados como el plugin de kubernetes y GKE.

Con la parte más compleja así, debemos acabar de configurar Jenkins tal y como vimos en el apartado "Jenkins", dentro del subapartado "Configuración de Jenkins"[REF referencia interna].

La principal diferencia en la configuración respecto hacerlo local o en kubernetes es el agente. Al usar Jenkins como un enjambre de kubernetes, no se usan los ejecutores del maestro, sino pods de Kubernetes. Esto significa que en el canal debemos definir como agente un pod de kubernetes o, si tenemos una plantilla para el pod, la etiqueta de esta.

Se puede ver el cambio en la configuración del sistema de Jenkins en el número de ejecutores. Cuando se usa en local por defecto hay dos activos, pero cuando se usa en kubernetes por defecto hay cero. Es más, si se cambia de cero el propio Jenkins mostrará una advertencia.

Figura 4.131: Diferencia en ejecutores de local a GCP

Crear el jenkinsfile

A continuación vamos a crear un jenkinsfile para Jenkins. Se podría hacer mediante blue ocean, pero vamos a escribirlo a mano para verlo mejor.

Primero se abre "pipeline ...". Todo el jenkinsfile se debe encontrar dentro. Además, todo lo que configuremos antes de las etapas, será aplicado de manera global a todo el canal y será usado por defecto. Vamos a añadir el agente de jenkins cd-jenkins-jenkins-agent. También añadimos unas variables locales que servirán para más adelante.

```
pipeline {
  agent {
    label 'cd-jenkins-jenkins-agent'
  }
  environment {
    PROJECT_ID  = 'proyectokubernetes-301509'
    APP_NAME = "addwebpage"
    IMAGE_TAG =
"eu.gcr.io/${PROJECT_ID}/${APP_NAME}:${env.BRANCH_NAME}.${env.BUILD_NUMBER}"

  }
```

Figura 4.132: Jenkinsfile, Parte I

Para la etapa de build sólo nos hace falta instalar en el nodo de javascript npm, mocha y mocha junit reporter. Usamos un nodo de nodejs porque node no es compatible de manera nativa con este contenedor. En otro tipo de programa se probaría a compilar aquí el código para las siguientes fases.

```
stages {
  stage('build') {

    steps {
      echo 'Build stage'

      nodejs(nodeJSInstallationName: 'nodejs') {
        sh 'npm config ls'
        sh 'npm install --global mocha'
        sh 'npm install -g mocha-junit-reporter'
      }
    }
  }
```

Figura 4.133: Jenkinsfile, Parte II

En la etapa de test usamos otra vez el nodo de nodejs para pasar los tests. Es importante hacerlo aparte para saber si ha fallado en la compilación o en los tests.

```
  stage('test') {
    steps {
      echo 'Test stage'
      nodejs(nodeJSInstallationName: 'nodejs') {
        sh 'npm test'
      }
    }
  }
```

Figura 4.134: Jenkinsfile, Parte III

Finalmente en post añadimos una salida para si el código es correcto y otra por si hay un fallo. Usamos emailext, que es el plugin "email extension" que configuramos anteriormente. Si el código es correcto devuelve un correo con el código para ejecutarlo. Si el código falla devuelve el fallo y un link al test fallido.

```
  post {
    success {
      echo 'Succesfull test'

      emailext(
        body: "Compilación exitosa en build ${env.BUILD_URL}. Por favor,
actualice la versión del repositorio de ${IMAGE_TAG} \
          con los siguientes comandos. \
          <br> docker build -t ${IMAGE_TAG} . <br> docker build -t
eu.gcr.io/${PROJECT_ID}/addwebpage:latest . <br>   \
          kubectl --namespace=production apply -f deploy/ <br> kubectl
--namespace=production scale deployment \
          addwebpage-deploy --replicas=4",
        recipientProviders: [developers(), requestor()],
        subject: "Compilación exitosa
${APP_NAME}:${env.BRANCH_NAME}.${env.BUILD_NUMBER}"
      )
    }
```

Figura 4.135: Jenkinsfile, Parte IV

```
    failure {
      echo 'Failed test, sending mail to developer'

      emailext(
          body: "Compilación fallida ${env.BUILD_URL}. Por favor, revise el
código.",
          recipientProviders: [developers(), requestor()],
          subject: "Error en build
${APP_NAME}:${env.BRANCH_NAME}.${env.BUILD_NUMBER}"

      )

    }
  }
}
```

Figura 4.136: Jenkinsfile, Parte V

Podemos ver el código completo en el anexo A.2.4.

4.9.3. CI con Jenkins

Una vez está todo configurado, si realizamos un cambio en el proyecto y los subimos a git, cuando jenkins inicialice la tarea. La tarea se puede inicializar manualmente, cada cierto tiempo o automáticamente al realizar un cambio usando los hooks de git. En nuestro caso vamos a hacerlo manualmente haciendo click en el símbolo del reloj de la tarea en el panel de control.

El motivo para hacerlo manualmente es que nuestro pipeline manda un correo haya tenido éxito o no, por tanto un refresco periodico crearía una gran cantidad de correos. Los hooks de git son la mejor opción, pero se salen de una explicación sencilla del funcionamiento de Jenkins.

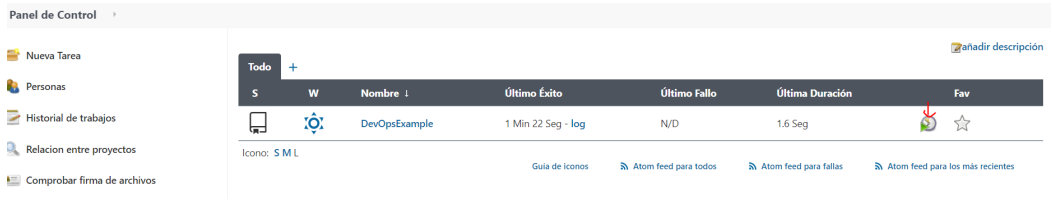


Figura 4.137: Panel de control de Jenkins

Si quisiéramos configurar la tarea para que se ejecute automáticamente, se puede realizar desde la configuración de la tarea, a la que se accede desde el panel de control.

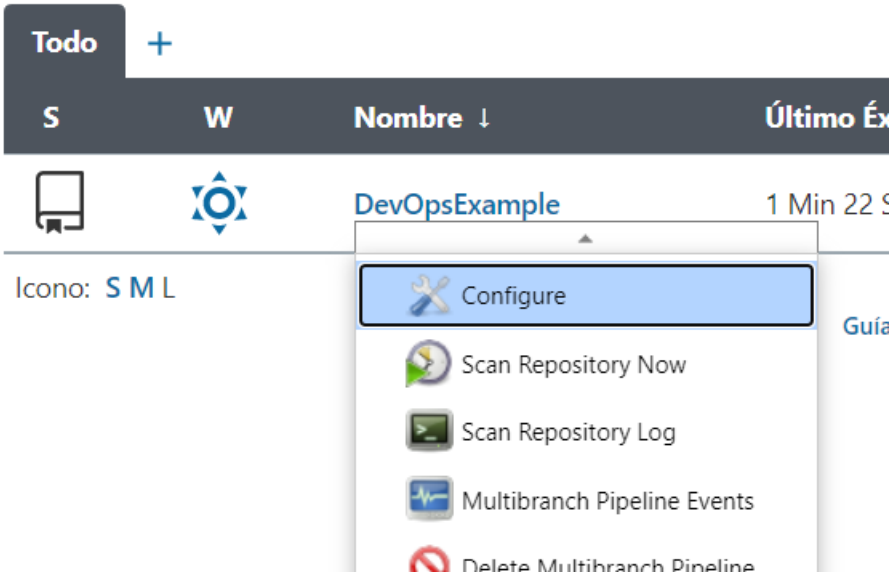


Figura 4.138: Configuración del panel de control de Jenkins

Si entramos en Blue Ocean, podremos ver todas las ramas del repositorio que conectamos. Están la rama maestra, la rama "Deployment_preparation" con unos tests que realicé para aprender a configurar el proyecto y la rama "Local Jenkins" que sólo funciona en el servidor local que cree para aprender a usar la herramienta.

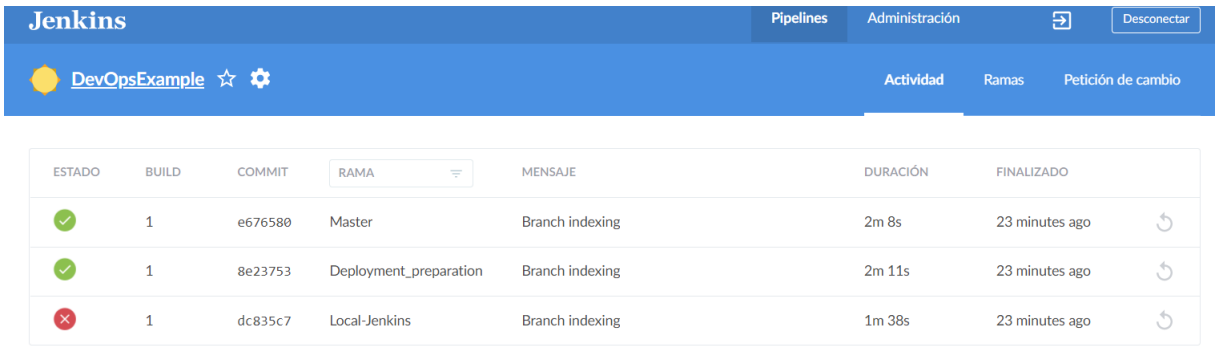


Figura 4.139: Configuración del panel de control de Jenkins

Si hacemos clic en una de las ramas, nos mostrará los resultados de la tarea. Podemos ver el output de cada etapa y ver más detalles en cada comando de la etapa. También se puede volver a ejecutar haciendo clic en "Restart Deploy".

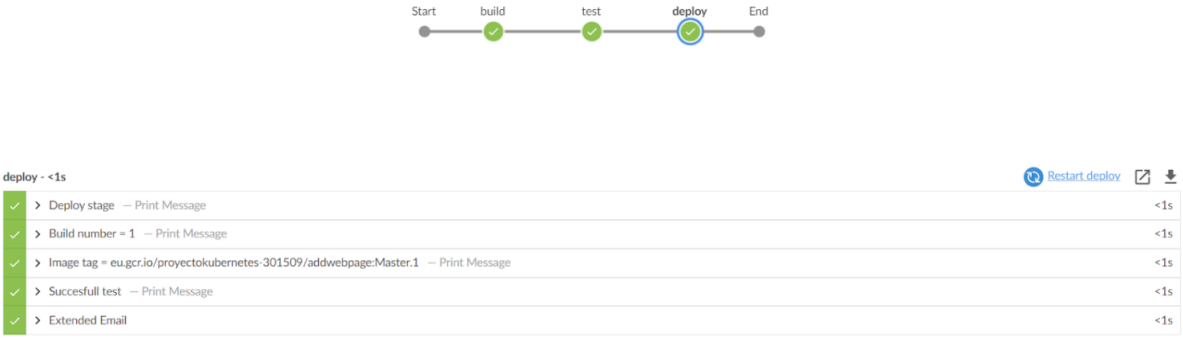


Figura 4.140: Resultado de compilación

Al abrir la tarea cuando aún está ejecutándose, se puede ver en tiempo real el proceso. Haciendo clic en cada apartado muestra el texto de la consola de comandos.

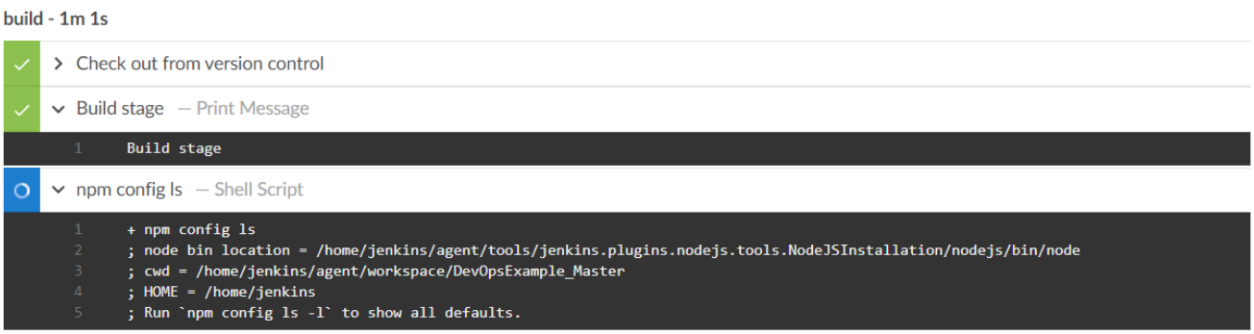


Figura 4.141: Detalles de la compilación

Tal y como configuramos en el pipepline, se envía un correo electrónico al desarrollador cuando la compilación es exitosa. Se muestra el proyecto, la rama y el número de compilación.

Figura 4.142: Correo de compilación exitosa

Para mostrar qué pasaría si la compilación fallará, vamos a añadir un error en el código del script "AddScript.js" en la rama "Deployment_preparation" para que no pase los test. Este fallo hará que no se filtren los caracteres no numéricos.

```
6
7       //Correct code. This is the valid solution
8       /*
9       for (var x of numbers)
10          if(Number.isInteger(Number.parseInt(x)))
11              result += Number.parseInt(x);
12      */
13      //Wrong code. This is expected to fail unit tests.
14
15      for (var x of numbers)
16          result += Number.parseInt(x);
17
```

Figura 4.143: Fallo inducido en el código

Activamos la tarea otra vez desde el panel de control de Jenkins y actualizamos el menú de Blue Ocean. Podemos ver en el panel de control cómo se añade el trabajo en la cola para que lo ejecute un agente.

Figura 4.144: Cola de trabajos de Jenkins

En el log de Blue Ocean podemos ver que ha fallado el apartado de test. Haciendo clic en la fase podemos ver todos los detalles del log.

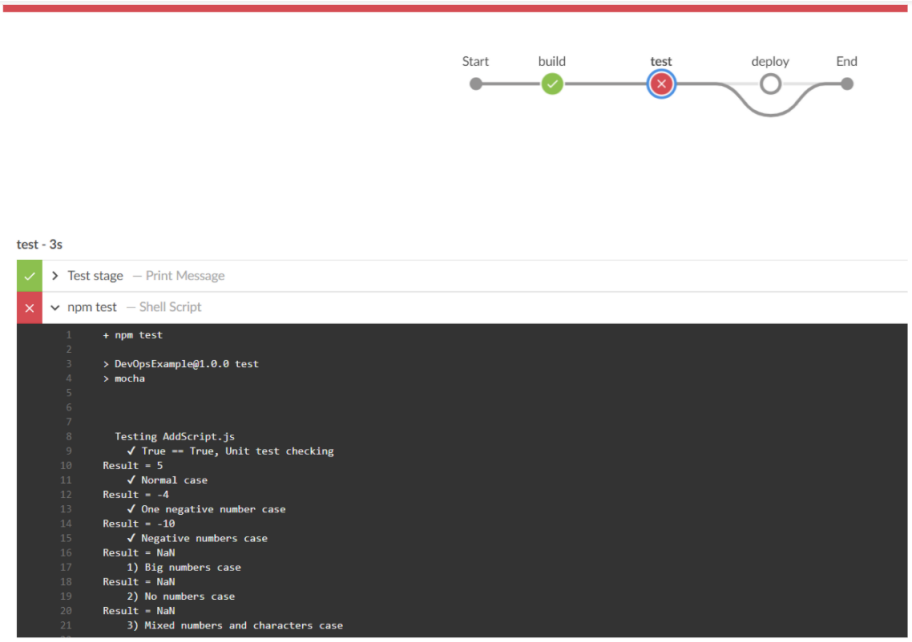


Figura 4.145: Compilación errónea

```
      1) Big numbers case
Result = NaN
      2) No numbers case
Result = NaN
      3) Mixed numbers and characters case


  4 passing (11ms)
  3 failing

  1) Testing AddScript.js
       Big numbers case:

      AssertionError [ERR_ASSERTION]: Expected values to be strictly equal:
+ actual - expected

+ NaN
- 2002000000
      + expected - actual

      -NaN
      +2002000000

      at Context.<anonymous> (test/AddScriptTest.js:32:12)
      at processImmediate (node:internal/timers:464:21)

  2) Testing AddScript.js
       No numbers case:
```

Figura 4.146: Log de la compilación errónea

Si vamos al correo con el que iniciamos la tarea, recibiremos el correo que establecimos en el jenkinsfile en caso de error. Hemos añadido un link al log.

Figura 4.147: Correo de compilación errónea

En este caso la Integración contínua no es absoluta porque el pipeline no hace rollback automáticamente. Para hacerlo así, se debería añadir en el jenkinsfile, en el apartado postfailure, el comando "git reset –soft HEAD 1" que revierte el último commit. Esto no se ha añadido al jenkinsfile porque no hay ningún control sobre cuántos commits se realizan simultáneamente (pudiendo borrar el commit que no es), además de que no es seguro añadir esta funcionalidad en la rama maestra.

El procedimiento con este jenkinsfile sería que para cada tarea se use una rama distinta y que sólo se pueda mezclar con la rama maestra si todo funciona apropiadamente.

4.9.4. CD con Jenkins

Esta parte no es exactamente CD, porque no se automatiza el proceso. En realidad depende más de GCP que de Jenkins.

El problema aquí es que estamos usando Jenkins desde kubernetes y queremos lanzar la aplicación como un pod. Esto significa lanzar un contenedor Docker desde dentro de otro contenedor de Docker. Esto provoca problemas por la arquitectura interna de Docker. Concretamente, el mayor problema es que Docker necesita acceso exclusivo a un archivo interno que no puede ser replicado.

Es posible usar Docker desde dentro de Docker con el uso de módulos como "Docker in Docker" de jpetazzo, pero no soluciona todos los problemas. Es más, el propio desarrollador lanzó un aviso desde su repositorio para no usar la herramienta salvo por casos de extrema necesidad [17]. También hay una recomendación habitual de montar un socket en Jenkins con el Docker del anfitrión, pero ya no es una solución tan viable porque Docker ya no se distribuye como librerías estáticas.

Existen soluciones más elegantes, pero incrementan la complejidad del sistema.

Por problemas de tiempo no se han podido implementar en este proyecto, pero es posible implementarlo de varias formas. Entre ellas, algunas de las más populares son módulos de Jenkins como el de Docker o configurando el repositorio de GitHub. Sin automatizar el despliegue, funciona como vimos en el apartado "Aplicación Web con Node.js>Ejecutar en Kubernetes desde GCP"

Si el repositorio fuera local, podría usarse simplemente con un comandos docker o los comandos de terminal para compilar y subir el commit.

Capítulo 5

Conclusiones y trabajos futuros

5.1. Conclusiones

La metodología de DevOps es muy beneficiosa y eficiente para equipos grandes. Para equipos más pequeños no estoy convencido de que aporte tanto en relación coste de implementación-beneficio, aunque se pueden aplicar algunas cosas.

Respecto a las tecnologías de nube no me convencen a nivel de usuario independiente. Como ya se explicó para empresas emergentes es muy beneficioso, pero para usuarios independientes me pareció caro.

Veo mucho potencial en Jenkins pero lo noté algo inestable. Durante el desarrollo fue común que diera errores y tuviera que reiniciarlo. La documentación era también confusa, incompleta y en una ocasión errónea.

Las tecnologías de contenedores las conocía pero no las había usado a este nivel. Efectivamente funcionan de forma ligera y eficiente. La documentación ha sido destacablemente buena y ha sido sencillo aprender estas tecnologías.

5.2. Trabajos futuros

Por restricciones de tiempo al proyecto le falta una implementación completa de CI/CD. Hay un problema a la hora de aplicar el despliegue continuo 4.9.4. Se pudo solventar de una forma incorrecta usando una máquina de Docker dentro de Kubernetes, pero no me gustó la idea ya que parte de este proyecto es ser una guía de inicio.

A nivel personal investigaré más de Docker y Kubernetes. Jenkins me pareció también interesante, pero por mi experiencia con Jenkins investigaré alternativas.

5.3. Agradecimientos

Quiero agradecer a mis padres por ayudarme tanto a nivel emocional durante este tiempo.

También quiero agradecer a mi tutor Isaías por estar ahí pese a los retrasos y problemas que han surgido en el proyecto.

Bibliografía

[1] Riti, P.: *Pro DevOps with Google Cloud Platform.* (2018)

[2] Guerriero, M., Garriga, M., Tamburri, D.A., Palomba, F.: *Adoption, Support, and Challenges of Infrastructure-as-Code: Insights from Industry.* Proceedings - 2019 IEEE International Conference on Software Maintenance and Evolution, ICSME 2019. pp. 580–589. (2019)

[3] Shahin, M., Babar, M.A., Zhu, L.: *Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices.* IEEE Access. Volume 5. pp. 3909–3943. (2017)

[4] blog, m.: *Docker is not a hypervisor*

[5]

[6] : *Documentación de Kubernetes*

[7] (Dec 2021)

[8]

[9] Kamal, M.A., Raza, H.W., Alam, M.M., Mohd, M.: *Highlight the Features of AWS, GCP and Microsoft Azure that Have an Impact when Choosing a Cloud Service Provider.* International Journal of Recent Technology and Engineering. Volume 8. pp. 4124–4232. (2020)

[10] Mell, P., Grance, T., et al.: *The NIST definition of cloud computing.* (2011)

[11] Kamal, M.A., Raza, H.W., Alam, M.M., Su'ud, M.M.: *Highlight the features of AWS, GCP and Microsoft Azure that have an impact when choosing a cloud service provider.* International Journal of Recent Technology and Engineering (IJRTE). (2020)

[12] : *Supported languages*

[13] JMAlarcon: *7 motivos para utilizar Docker en general y con ASP.NET Core en particular*

[14]

[15]

[16]

[17]

Anexo A

Anexo

A.1. AddWebpage

A.1.1. AddWebpage.HTML

```
<!DOCTYPE html>
<meta charset="UTF-8">
<html>
    <!--
     Script de javascript que permite sumar los valores que se pasen
    como un string.
    -->
    <script type="text/javascript" src="/js/AddScript.js"></script>

    <body>

        <h1 style="color: #5e9ca0;">DevOps example</h1>
        <h2 style="color: #2e6c80;">This is a simple app</h2>
        <p>This example adds the numbers put in the text box and separated by at
            least one space</p>

        <form >
            <textarea id="input" name="TextBox" rows="4" cols="50"
                oninput="output.value=add(this.value)">
            </textarea>
            <br>
            Result =
```

```
        <output type="number" id="output" name="result">
        <br>
    </form>


</body>
</html>
```

A.1.2. AddScript.js

```
function add(text) {
    var result = 0
    var numbers

    numbers = text.split(" ");

    //Correct code. This is the valid solution
    for (var x of numbers)
        if(Number.isInteger(Number.parseInt(x)))
            result += Number.parseInt(x);

    //Wrong code. This is expected to fail unit tests.
    /*
    for (var x of numbers)
        result += Number.parseInt(x);
    */

    console.log("Result = " + result);
    return result;
}

// If we're running under Node
if(typeof exports !== 'undefined') {
    exports.add = add;
}
```

A.1.3. AddScriptTest.js

```
//run with "npm test"

const assert = require('assert');
const AddScript = require ('../www/js/AddScript.js');

var result = 5;

describe('Testing AddScript.js', function(){

  it('True == True, Unit test checking', function() {
    assert.strictEqual(1,1, "There is a problem in the Unit test")
  })

  it('Normal case', function() {
    var result = AddScript.add("2 3");
    assert.strictEqual(result, 5)
  })

  it('One negative number case', function() {
    result = AddScript.add("-7 3");
    assert.strictEqual(result, -4)
  });

  it('Negative numbers case', function(){
    result = AddScript.add("-7 -3");
    assert.strictEqual(result, -10)

  });

  it('Big numbers case', function(){
    result = AddScript.add(" 1000000 1000000000 1001000000 ");
    assert.strictEqual(result, 2002000000);
  });

  it('No numbers case', function(){
    result = AddScript.add(" asd e - * ");
    assert.strictEqual(result, 0);
  });
```

```
  it('Mixed numbers and characters case', function() {
    result = AddScript.add("5 g--s ad asd -r - 12 -4 @3~# asd");
    assert.strictEqual(result, 13);
  });
})
```

A.1.4. server.js

```
var express = require('express');
var app = express();

/*Cuando se reciba una peticin dentro del directorio "/" (el raz )
  Ejecutar el cdigo dentro.*/
app.get('/', function(req, res) {
    //Devuielve el archivo AddWebpage.html
    res.sendFile(__dirname+'/www/AddWebpage.html');
});

//Carga el directorio www
app.use(express.static(__dirname + '/www'));

/*Escucha en el puerto localhost:8081
  Crea una direccin y un puerto*/
var server = app.listen(8081, function(req,res) {
  var host = server.address().address
  var port = server.address().port

  console.log("Example app listening at http://%s:%s", host, port)
});
```

A.2. deploy

A.2.1. service.YAML

```
apiVersion: v1
kind: Service
metadata:
  name: addwebpage-service
```

```
  labels:
    app: addwebpage
spec:
  selector:
    app: addwebpage
  type: LoadBalancer
  ports:
  - protocol: TCP
    port: 5000
    targetPort: 8081
    nodePort: 31110
```

A.2.2. deployment.YAML

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: addwebpage-deploy
  labels:
    app: addwebpage
spec:
  replicas: 1
  selector:
    matchLabels:
      app: addwebpage
  template:
    metadata:
      labels:
        app: addwebpage
    spec:
      containers:
      - name: addwebpage
        image: eu.gcr.io/proyectokubernetes-301509/addwebpage:latest
        ports:
        - containerPort: 8081
```

A.2.3. Ingress.YAML

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: addwebpage-ingress
  labels:
    app: addwebpage
spec:
  rules:
  - http:
      paths:
      - pathType: Prefix
        path: "/"
        backend:
          service:
            name: addwebpage-service
            port:
              number: 5000
```

A.2.4. Jenkinsfile

```
pipeline {
  agent any

  environment {
    PROJECT_ID = 'proyectokubernetes-301509'
    APP_NAME = 'addwebpage'
    IMAGE_TAG =
        "eu.gcr.io/${PROJECT_ID}/${APP_NAME}:${env.BRANCH_NAME}.
        ${env.BUILD_NUMBER}"
  }

  stages {
    stage('build') {

      steps {
        echo 'Build stage'

        nodejs('nodejs') {
          bat '''
```

```
            npm config ls
            npm install --global mocha
            npm install -g mocha-junit-reporter
          '''
        }


      }
    }


    stage('test') {
      steps {
        echo 'Test stage'
        nodejs('nodejs') {
          bat 'npm test'
        }
      }
    }

    stage('deploy') {
      steps {
        echo 'Deploy stage'
        echo "Build number = ${env.BUILD_NUMBER}"
        echo "Image tag = ${IMAGE_TAG}"

        //docker build -t eu.gcr.io/${PROJECT_ID}/addwebpage:latest .
        //docker build -t
            eu.gcr.io/${PROJECT_ID}/addwebpage:${env.BUILD_NUMBER} .

      }
    }
  }

post {
    success {
      echo 'Succesfull  test'

        emailext(
```

```
            body: "Compilacion exitosa en build ${env.BUILD_URL}. Por favor,
                actualice la version del repositorio de ${IMAGE_TAG} \
              con los siguientes comandos. \
              <br> docker build -t ${IMAGE_TAG} . <br> docker build -t
                  eu.gcr.io/${PROJECT_ID}/addwebpage:latest . <br> \
              kubectl --namespace=production apply -f deploy/ <br> kubectl
                  --namespace=production scale deployment \
              addwebpage-deploy --replicas=4",
            recipientProviders: [[$class: 'DevelopersRecipientProvider'], [$class:
                'RequesterRecipientProvider']],
            subject: "Compilacion exitosa
                ${APP_NAME}:${env.BRANCH_NAME}.${env.BUILD_NUMBER}"
        )
    }
}
```