



# INNOSOFT - TALLER MEAN STACK

A cargo de: Beny Peña & Juan Alberto Galera





## Requisitos software del taller

- Sistema operativo actual (Windows 10, MacOS, Linux)
- NodeJS v12 LTS (npm incluido)
- CLI de Angular
- Loopback v3
- Docker y una cuenta de Docker Hub
- Crear Cluster de Mongo y cuenta en MongoDB Atlas
- Cuenta de Google Cloud (opcional)
- Google cloud SDK + Kubectl (opcional)
- Gitlab (CI)



# Instalación de NodeJS

Descargamos el instalador desde:

<https://nodejs.org/es/download/>

Descargamos la versión acorde a nuestro sistema, e instalamos.

## Añadir CLI de Angular

Como tenemos npm instalado, lo usamos para instalar paquetes de npm.

Podemos seguir el Getting Started de la página oficial de Angular, con el siguiente comando en el terminal instalamos:

```
$ npm install -g @angular/cli
```

Si usamos Unix/Linux necesitamos ejecutar las instalaciones globales con sudo.

```
$ sudo npm install -g @angular/cli
```

## Añadir CLI de Loopback

Framework NodeJS para la creación de API

Podemos seguir el Getting Started de la página oficial de Loopback, con el siguiente comando en el terminal instalamos:

```
$ npm install -g loopback-cli
```

Si usamos Unix/Linux necesitamos ejecutar las instalaciones globales con sudo.

```
$ sudo npm install -g loopback-cli
```



# Instalar Docker

Para facilitar la compatibilidad y el desarrollo vamos a crear contenedores de Docker con nuestros proyectos.

Para descargar el instalador de Docker necesitamos crearnos una cuenta de Docker Hub, que también nos servirá más adelante para poder subir nuestras imágenes a Docker Hub

<https://www.docker.com/get-started>

Cuando lo tengamos completamente instalado, iniciamos Docker y nos logueamos con nuestra cuenta



## Cluster en MongoDB Atlas(Mlab)

Para crear una BD en MongoDB Atlas es obligatorio crear una cuenta.

<https://www.mongodb.com/cloud/atlas>

Creamos un cluster, añadimos usuarios de conexión con los privilegios que deseemos y modificamos el acceso de red para poder conectar nuestra API.

Nos guardamos la url de acceso.



# Google Cloud + SDK (Kubectl)

Creamos cuenta Google Cloud Platform

<https://console.cloud.google.com/>

Descargamos e instalamos el SDK de Google Cloud

<https://cloud.google.com/sdk/docs/quickstarts>

Instalar Kubectl

```
$ gcloud components install kubectl
```





# Angular

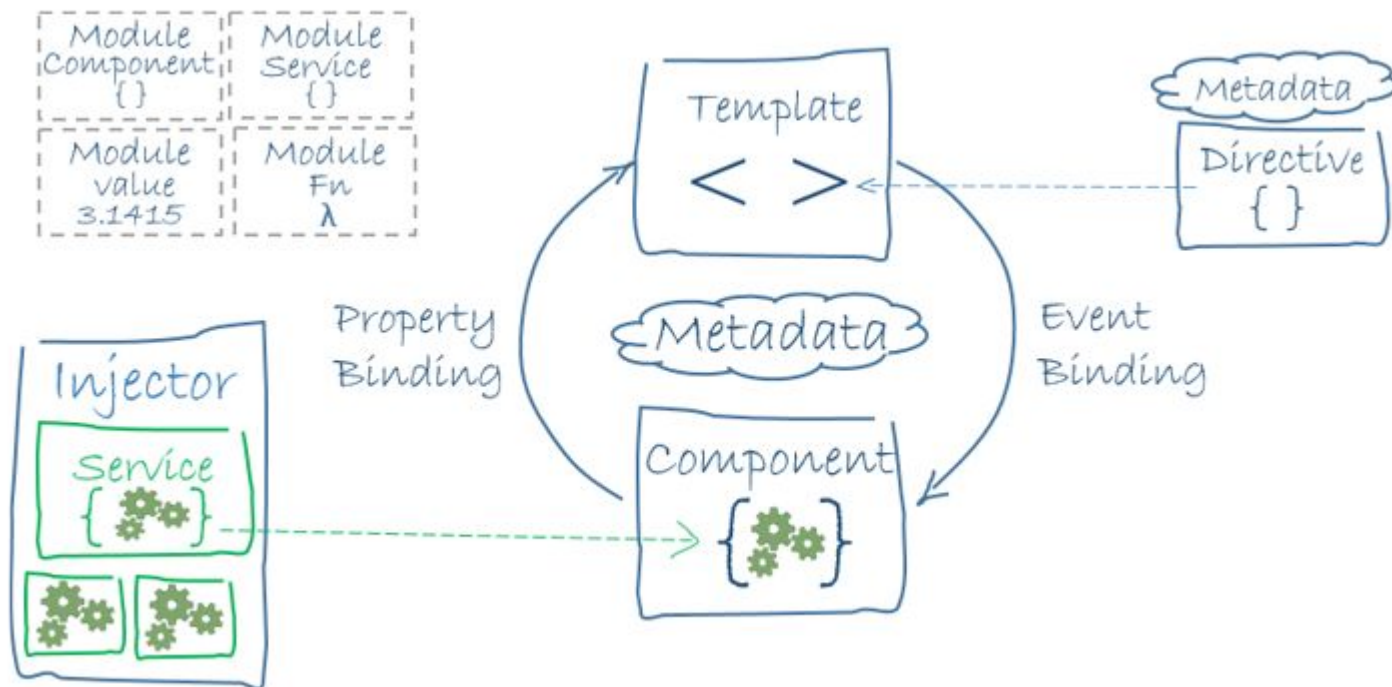
Angular es una plataforma para crear aplicaciones web móviles y de escritorio.

## Características y Beneficios

- Cross Platform
- Velocidad y rendimiento
- Productividad
- Historial de desarrollo completo

<https://angular.io/features>

# Angular Arquitectura





# NodeJS y Loopback

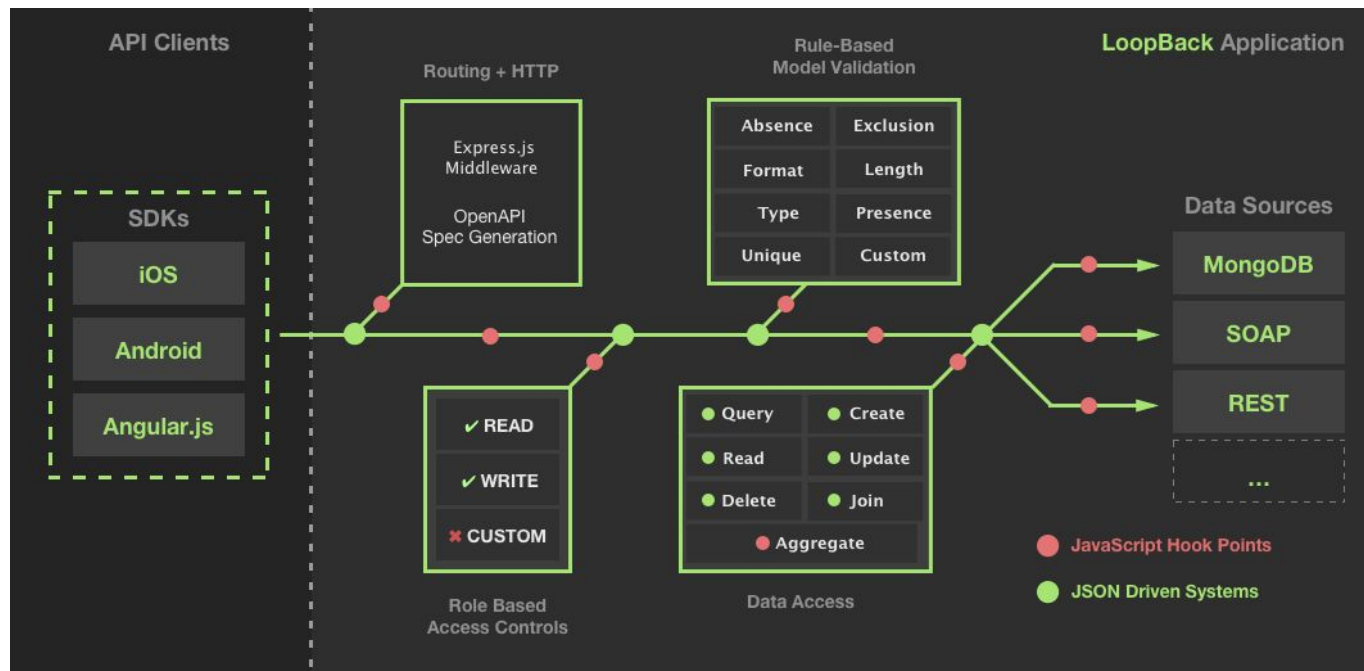
NodeJS es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome

Loopback es un framework open-source para nodejs altamente extensible desarrollado por IBM.

Características:

- Crea rápidamente API REST dinámicas end-to-end
- Conecta dispositivos y navegadores a datos y servicios.
- Componentes complementarios para gestión de archivos, login de terceros y OAuth2

# Flow de Loopback





# Docker

Docker es un proyecto open-source que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos.

Docker, a diferencia de una máquina virtual, no requiere incluir un sistema operativo independiente

Se basa en las funcionalidades del kernel y utiliza el aislamiento de recursos (CPU, la memoria, el bloque E / S, red, etc.) y namespaces separados para aislar la vista de una aplicación del sistema operativo.

# Docker y Docker Hub

Mediante el uso de contenedores, los recursos pueden ser aislados, los servicios restringidos, y se otorga a los procesos la capacidad de tener una visión casi completamente privada del sistema operativo con su propio identificador de espacio de proceso, la estructura del sistema de archivos, y las interfaces de red. Contenedores múltiples comparten el mismo núcleo, pero cada contenedor puede ser restringido a utilizar solo una cantidad definida de recursos como CPU, memoria y E / S.

**Docker Hub** es un repositorio público en la nube, similar a Github, para distribuir los contenidos. Está mantenido por la propia Docker y hay multitud de imágenes, de carácter gratuito, que se pueden descargar y así no tener que hacer el trabajo desde cero al poder aprovechar “plantillas”. También podemos crear nuestros propios repositorios privados e, incluso, dispone de tienda.



# Diferencias entre Docker y Máquinas Virtuales

Aunque, por su naturaleza, se asemejan a las clásicas máquinas virtuales, estamos hablando de algo más avanzado porque nos ofrecen una mayor eficiencia y sencillez.

- Para empezar, los contenedores de Docker **comparten recursos con el sistema operativo** sobre el que se ejecutan. De esta manera podemos arrancar o parar el contenedor rápidamente, mientras que máquinas virtuales, como Vmware, Citrix o Virtualbox, se aíslan del sistema operativo sobre el que trabajan y se comunican a través del **hypervisor**.
- La portabilidad de los contenedores hace que los problemas causados por cambiar el entorno donde está corriendo la aplicación se reduzcan a la mínima expresión.
- Si las máquinas virtuales quieren simular el entorno diferente al nuestro, el container de Docker se centra en crear la aplicación y que se pueda portar todo el contenido de manera sencilla.

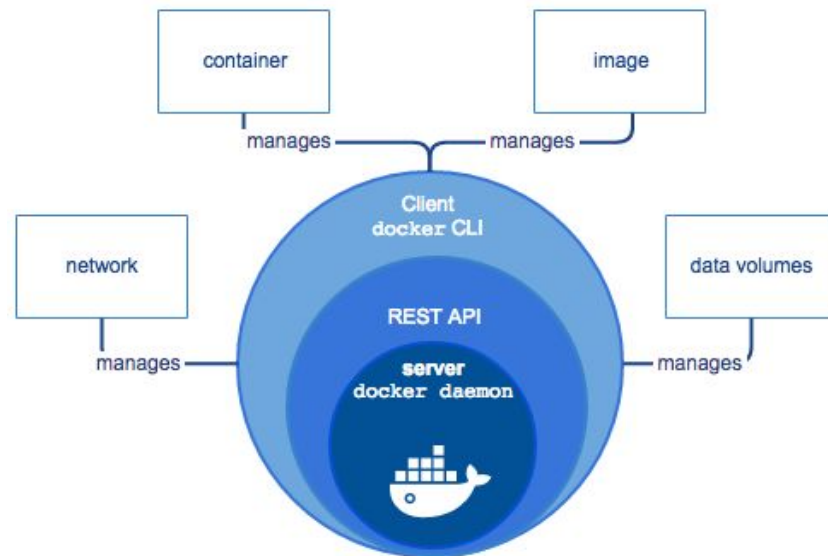
# Docker Engine

*Docker Engine* es una aplicación cliente-servidor con estos componentes principales:

- Un servidor que es un tipo de programa de larga duración llamado proceso daemon (comando `dockerd`).
- Una API REST que especifica las interfaces que los programas pueden usar para hablar con el demonio e indicarle qué hacer.
- Un cliente de interfaz de línea de comandos (CLI) (comando `docker`).

El CLI utiliza la API REST de Docker para controlar o interactuar con el demonio Docker a través de scripts o comandos directos de la CLI. Muchas otras aplicaciones de Docker usan la API y la CLI subyacentes.

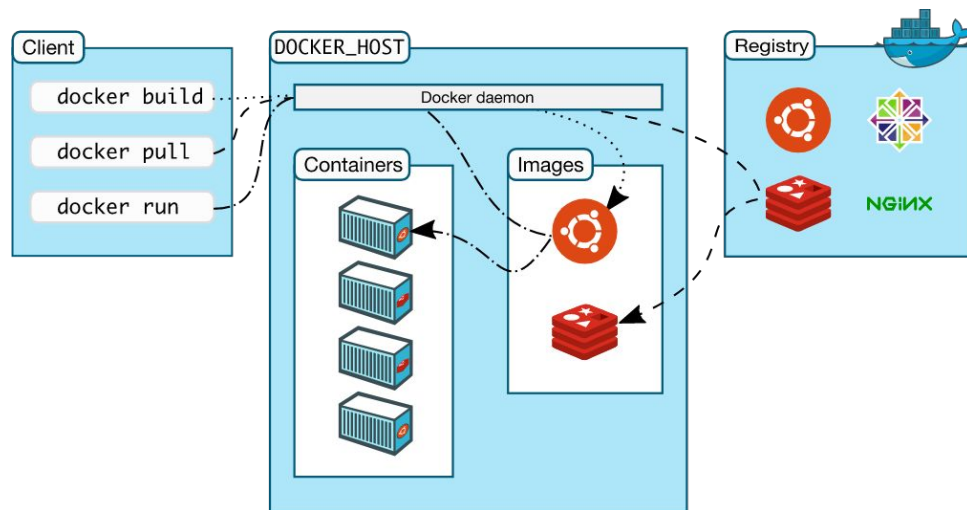
El daemon crea y gestiona *objetos* Docker , como imágenes, contenedores, redes y volúmenes.





# Docker Arquitectura

Docker utiliza una arquitectura cliente-servidor. El *cliente* Docker habla con el *demonio* Docker, que hace el trabajo pesado de construir, ejecutar y distribuir sus contenedores Docker. El cliente Docker y el demonio *pueden* ejecutarse en el mismo sistema, o puede conectar un cliente Docker a un demonio Docker remoto. El cliente Docker y el daemon se comunican mediante una API REST, a través de sockets UNIX o una interfaz de red.





# MongoDB Atlas

MongoDB Atlas es el servicio global de base de datos en la nube para aplicaciones modernas. Implemente MongoDB totalmente administrado en AWS, Azure o GCP. La mejor automatización y prácticas comprobadas de su clase garantizan la disponibilidad, la escalabilidad y el cumplimiento de los estándares de seguridad y privacidad de datos más exigentes. Use el robusto ecosistema de controladores, integraciones y herramientas de MongoDB para construir más rápido y pasar menos tiempo administrando su base de datos.



# Google Cloud

**Google Cloud** (Nube de Google), es una plataforma que ha reunido todas las aplicaciones de desarrollo web que Google estaba ofreciendo por separado. Es utilizada para crear ciertos tipos de soluciones a través de la tecnología almacenada en la nube y permite por ejemplo destacar la rapidez y la escalabilidad de su infraestructura en las aplicaciones del buscador.<sup>1</sup>

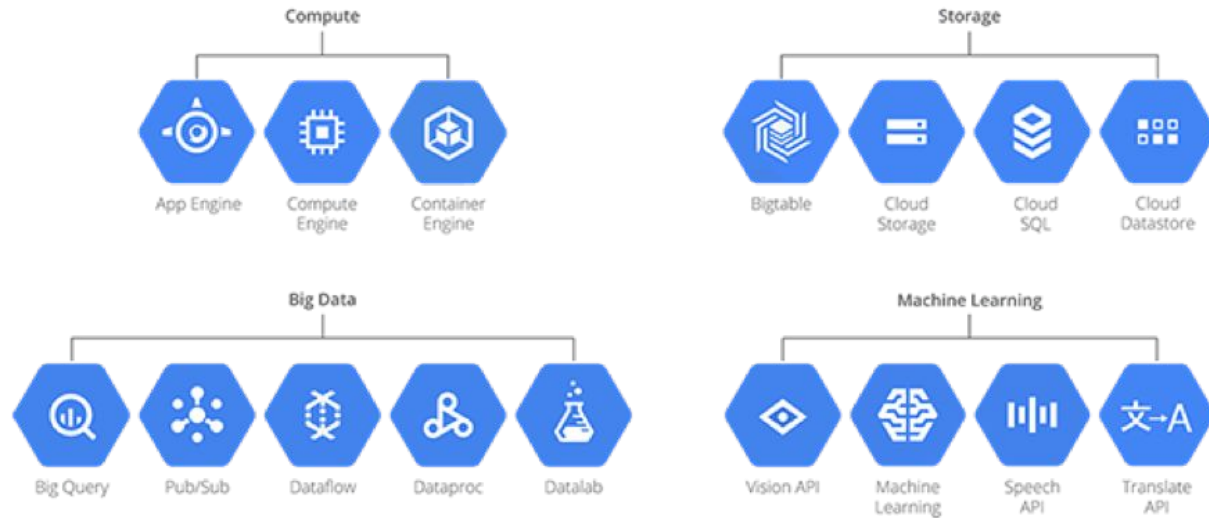
Google Cloud se refiere al espacio virtual a través del cual se puede realizar una serie de tareas que antes requerían de hardware o software y que ahora utilizan la nube de Google como única forma de acceso, almacenamiento y gestión de datos.

# Google Cloud

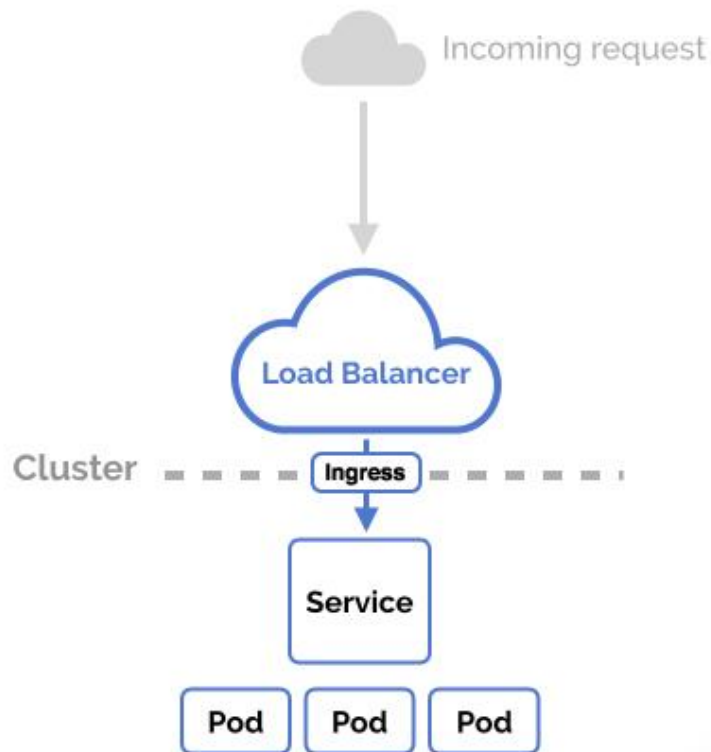


# Google Cloud

## Google Cloud Platform



# Nuestro Google Cloud



# Comandos básicos de Angular

- Crear proyecto

```
$ ng new <nombre proyecto>
```

- Ejecutar proyecto (cd al proyecto)

```
$ ng serve
```

- Generar archivos basados en una esquemática (cd al proyecto)

```
$ ng generate <schematic>
```

más usados : component, service, module, library, interface

- Deploy produccion (cd al proyecto)

```
$ ng build --prod
```



## Comandos básicos de Loopback v3

- Generar proyecto

```
$ lb
```

- Añadir modelos

```
$ lb model
```

- Añadir datasources

```
$ lb datasource
```

- Añadir relacion

```
$ lb relation
```

- Ejecutar proyecto

```
$ node .
```





# Comandos básicos de Docker

- Construir una imagen desde un Dockerfile

```
$ docker build -f Dockerfile
```

- Manejar containers

```
$ docker container
```

- Manejar el docker engine

```
$ docker engine
```

- Manejar imagen

```
$ docker image
```



# Comandos básicos de Docker

- Manejar redes

```
$ docker network
```

- Manejar plugin

```
$ docker plugin
```

- Listar imagenes

```
$ docker images
```

- Listar containers

```
$ docker ps
```

```
$ docker ps -a
```



# Comandos básicos de Docker

- Pull una imagen

```
$ docker pull <image_name>
```

- Push una imagen

```
$ docker push <image_name>
```

- Borrar container

```
$ docker rm <container>
```

- Borrar imagenes

```
$ docker rmi <image_name>
```

# Comandos básicos de Docker

- Ejecutar container

```
$ docker start <container>
```

- Ejecutar comando en un container que está corriendo

```
$ docker exec -it <container> bash
```

- Reiniciar, apagar y encender containers

```
$ docker start <container>
```

```
$ docker stop <container>
```

```
$ docker restart <container>
```

# Comandos básicos de Gcloud SDK + Kubectl

- Login cuenta

```
$ gcloud auth login
```

- Cambiar contexto
- Listar namespaces

```
$ kubectl get namespace
```

- Listar pods

```
$ kubectl get pods -n <namespace>
```

- Listar services

```
$ kubectl get service -n <namespace>
```

- Listar Deployments

```
$ kubectl get deploy -n <namespace>
```

- Editar recursos  
(namespace, pod, deploy, service, etc...)

```
$ kubectl edit <recurso> -n <namespace>
```

- Borrar recursos  
(namespace, pod, deploy, service, etc...)

```
$ kubectl delete <recurso> -n <namespace>
```



# Manos a la Obra

Para seguir el taller podeis descargar todo el código necesario desde el siguiente repositorio de Github

<https://github.com/guadaltech/tallerUS>

- Un poco de código (Full-Stack)
- Arquitectura
- Subida a cloud
- Ejecución
- CI

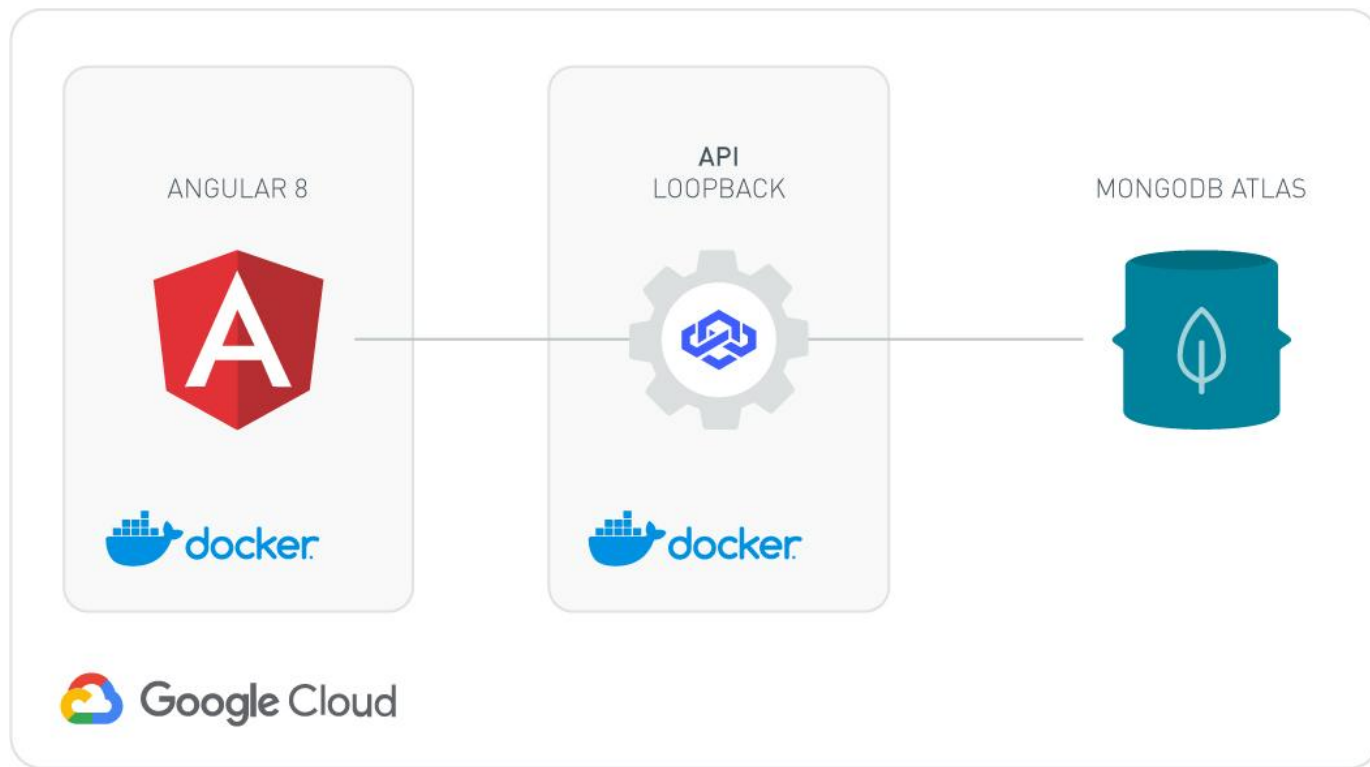
# Un poco de código

Revisamos código creado tanto con Angular como con NodeJS

```
hero-detail.component.ts
34 export class HeroDetailComponent implements AfterViewInit, OnInit {
35   @Input() hero: Hero;
36   @Output() unselect = new EventEmitter<string>();
37   @Output() heroChanged = new EventEmitter<{ mode: string; hero: Hero }>();
38
39   @ViewChild('id') idElement: ElementRef;
40   @ViewChild('name') nameElement: ElementRef;
41
42   addingHero = false;
43   editingHero: Hero;
44
45   ngAfterViewInit() {
46     if (this.addingHero && this.editingHero) {
47       this.idElement.nativeElement.focus();
48     } else {
49       this.nameElement.nativeElement.focus();
50     }
51   }
52
53   ngOnInit() {
54     this.addingHero = !this.hero;
55     this.editingHero = this.cloneIt();
56   }
57
58   addHero() {
59     const hero = this.editingHero;
60     this.emitRefresh('add');
61   }
62
63   clear() {
64     this.unselect.emit();
65     this.editingHero = null;
66   }
67 }
```

```
HeroDetail.vue
25 <script>
26 export default {
27   props: { hero: { type: Object } },
28
29   data() {
30     return {
31       addingHero: !this.hero,
32       editingHero: this.cloneIt()
33     };
34   },
35
36   watch: {
37     hero() {
38       this.editingHero = this.cloneIt();
39     }
40   },
41
42   mounted() {
43     if (this.addingHero && this.editingHero) {
44       this.$refs.id.focus();
45     } else {
46       this.$refs.name.focus();
47     }
48   },
49   methods: {
50     addHero() {
51       const hero = this.editingHero;
52       this.emitRefresh('add');
53     },
54
55     clear() {
56       this.$emit('unselect');
57       this.editingHero = null;
58     },
59   }
60 }
```

# Arquitectura







# Ejecucion en cloud

Revisamos código:

- Dockerfile
- Makefile

# Gitlab CI

Archivo gitlab-ci.yml

## **variables:**

GCP\_PROJECT\_ID: heineken-181810

IMAGE\_NAME:

`${CI_PROJECT_PATH_SLUG}/${CI_COMMIT_REF_SLUG}:${CI_COMMIT_REF_SLUG}_${CI_PROJECT_ID}.${CI_JOB_ID}`

IMAGE\_NAME\_LATEST: `${CI_PROJECT_PATH_SLUG}/${CI_COMMIT_REF_SLUG}:latest`

## **stages:**

- quality
- build-and-push
- delete-pod

# Gitlab CI

Archivo gitlab-ci.yml

```
#####OPERACIONES DE FRONTEND
```

**quality:**

```
  stage: quality
```

```
  image: hiorgserver/gitlab-sonar-scanner
```

**variables:**

```
  SONAR_URL: "http://sonar.guadaltech.xyz"
```

```
  SONAR_TOKEN: "token"
```

```
  SONAR_PROJECT_KEY: "servicios-digitales-frontend"
```

```
  SONAR_PROJECT_NAME: "servicios-digitales-frontend"
```

```
  SONAR_GITLAB_COMMENT: "true"
```

```
  SONAR_PUBLISH: "true"
```

**script:**

```
  - sonar-scanner -Dsonar.ws.timeout=9000 -Dsonar.projectKey=$SONAR_PROJECT_KEY -Dsonar.projectName=$SONAR_PROJECT_NAME
```

```
-Dsonar.host.url=$SONAR_URL -Dsonar.login=$SONAR_TOKEN -Dsonar.sources=. -Dsonar.exclusions=documentation/*,e2e/*,src/assets/*
```

```
-Dsonar.sourceEncoding=UTF-8
```

**only:**

```
  - test
```

```
  - master
```

# Gitlab CI

## Archivo gitlab-ci.yml

### build-and-push:

image: docker

stage: build-and-push

### services:

- docker:dind

### variables:

DOCKER\_DRIVER: overlay2

DOCKER\_HOST: tcp://localhost:2375

### script:

- # Install CA certs, openssl to https downloads, python for gcloud sdk

- apk add --update make ca-certificates openssl python

- update-ca-certificates

- # Write our GCP service account private key into a file

- echo \$GCLLOUD\_SERVICE\_KEY | base64 -d > \${HOME}/gcloud-registry-key.json

- # Export gcloud path

- export PATH=\$PATH:google-cloud-sdk/bin/gcloud

- # Download and install Google Cloud SDK

- wget https://dl.google.com/dl/cloudsdk/release/google-cloud-sdk.tar.gz

- tar zxvf google-cloud-sdk.tar.gz && ./google-cloud-sdk/install.sh --usage-reporting=false --path-update=true

- google-cloud-sdk/bin/gcloud --quiet components update

- google-cloud-sdk/bin/gcloud auth activate-service-account --key-file=gcloud-registry-key.json

- # Create our image. Expected to create an image 'image\_id'

- make pull

- make build

- make push

# Gitlab CI

## Archivo gitlab-ci.yml

```
delete-pod:
  stage: delete-pod
  image: alpine:3.7
  services:
    - docker:dind
  variables:
    DOCKER_DRIVER: overlay2
    DOCKER_HOST: tcp://localhost:2375
  script:
    - apk update && apk add --no-cache curl
    - curl -X GET https://api-repositorioapp.guadaltech.xyz/api/plists/generatePlist?token=hvgnvgvgcfdd465
    - curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubect1
    - chmod +x ./kubect1 && mv ./kubect1 /usr/local/bin/kubect1
    - mkdir -p $HOME/.kube
    - apk add --update make ca-certificates openssl python
    - update-ca-certificates
    - echo -n $KUBE_CONFIG | base64 -d > $HOME/.kube/config
    - echo $GCLOUD_SERVICE_KEY | base64 -d > ${HOME}/gcloud-registry-key.json
    - wget https://dl.google.com/dl/cloudsdk/release/google-cloud-sdk.tar.gz
    - tar zxvf google-cloud-sdk.tar.gz && ./google-cloud-sdk/install.sh --usage-reporting=false --path-update=true
    - google-cloud-sdk/bin/gcloud --quiet components update
    - google-cloud-sdk/bin/gcloud auth activate-service-account --key-file=gcloud-registry-key.json
    - google-cloud-sdk/bin/gcloud container clusters get-credentials guadaltech-tools --zone europe-west1-b --project heineken-181810
    - POD=$(kubectl get pod -n servicios-digitales -o name | grep node-servicios-digitales-frontend | sed "s/^.{4}\\//")
    - kubectl delete pod $POD -n 'servicios-digitales'
```

# Integración Continua con Gitlab (CI)

La **integración continua** es una práctica de desarrollo de software mediante la cual los desarrolladores combinan los cambios en el código en un repositorio central de forma periódica, tras lo cual se ejecutan versiones y pruebas automáticas. La integración continua se refiere en su mayoría a la fase de creación o integración del proceso de publicación de software y conlleva un componente de automatización (p. ej., CI o servicio de versiones) y un componente cultural (p. ej., aprender a integrar con frecuencia). Los objetivos clave de la integración continua consisten en encontrar y arreglar errores con mayor rapidez, mejorar la calidad del software y reducir el tiempo que se tarda en validar y publicar nuevas actualizaciones de software.



# Integración Continua con Gitlab (CI)

Voy a tratar de ilustrar las principales bondades de la herramienta y demostrar lo fácil que es empezar a usarla sin ningún tipo de experiencia cómo *sysadmin* o molestar a tus devops. **Todo gira en torno al fichero `.gitlab-ci.yml`** que contiene la definición de las diferentes **stages** (o pasos) que contienen diferentes trabajos que tendrán que completarse para que el proyecto pueda ser desplegado correctamente. La estructura del archivo es de lectura natural y una vez hayas visto un par de ejemplos, podrás empezar a escribir los tuyos sin apenas esfuerzo.



# Integración Continua con Gitlab (CI)

## Configuración de Integración/Despliegue Continuo (CI/CD)

La página de configuración da una idea de cómo de sencillo es configurar todo aquello necesario para que se obre la magia de la integración continua:

### Runners

Qué pasa si te digo que por fin **te puedes olvidar de configurar y gestionar *slaves***: el cómo se conectan al servidor de Integración Continua, cómo balancear la carga entre todos o cualquier otros detalles que son tanto tediosos como complicados en muchos casos...





# Integración Continua con Gitlab (CI)

Te presento a los Runners! **Su instalación se reduce a algo tan sencillo como seguir los tres pasos** descritos en la configuración de CI/CD de tu proyecto: instalar el binario correspondiente a tu sistema operativo, pega la URL de tu servidor y el token que aparece en esta página y listo. Además, **puedes etiquetar a tus Runners en base a sus capacidades** para seleccionarlos en trabajos específicos desde el fichero `.gitlab-ci.yml` cuando sea necesario.

Los Runners pueden utilizar diferentes executors, que no son más que formas distintas de ejecutar tus scripts/código en ellos; desde el más básico, sobre ssh, pasando por un servicio de contenedores, hasta el cluster kubernetes más grande que se te pueda ocurrir; soportando incluso ejecuciones sobre Powershell/Batch en sistemas Windows.

# Integración Continua con Gitlab (CI)

## Gestión de secrets

En la era de los microservicios, en la que es muy probable que tu proyecto esté integrado con docenas de APIs que utilizan *tokens*, *secrets*, contraseñas de aplicación o cualquier otra forma de autorizar ese diálogo, **se convierte en prioridad alguna forma elegante de manejar toda esta complejidad**. Un auténtico *bad smell* más frecuente de lo que creeríamos es almacenar todos estos datos en ficheros de configuración en una máquina remota o incluso dejarlos colgando en alguna parte del código. Para poner énfasis en la importancia de esto, simplemente considera cuántos servicios y proyectos están surgiendo en los últimos años para gestionar esta problemática. Los proyectos de GitLab también incluyen un sencillo *keystore* en su página de settings que pueden ser accedidos desde los scripts de Integración Continua para que los administradores del proyecto los gestionen.

# Integración Continua con Gitlab (CI)

## Pipelines

Esta es la **funcionalidad central** de cualquier sistema de Integración Continua y, a la vez, un concepto realmente simple que se traduce en conectar todos los pasos (*stages*) que deben seguirse desde el momento en el que dispone de todo el código fuente al punto en el que la aplicación puede ser desplegada. Entre medias, puedes incluir todo aquello que se te ocurra para comprobar que: el código tiene buena pinta (*linting*), puede compilarse sin errores, funciona de forma esperada (*unit tests*), se integra con otros sistemas... y cualquier cosa que nos lleve a desplegar con seguridad nuestras aplicaciones

The screenshot shows a GitLab CI pipeline interface. At the top, it indicates the pipeline is 'passed' and was triggered about an hour ago by 'Martin Jenkovski'. The main title is 'Merge branch '3440-remove-hsts-header-from-rails-app' into 'master'', with a subtitle 'Stop setting Strict-Transport-Security header from within the app' and a link to 'See merge request !5341'. Below this, it shows '67 jobs from master in 103 minutes 32 seconds (queued for 4 minutes 46 seconds)' and a commit hash '9ec83887'. The pipeline is visualized as a series of stages: 'Prepare' (with jobs 'knapack' and 'setup-test-env'), 'Test' (with jobs 'bundler:audit', 'bundler:check', 'cache:gems', and 'license\_finder'), 'Post Test' (with jobs 'coverage', 'lint:javascript', 'notify:slack', and 'trigger\_docs'), 'Pages' (with job 'pages'), and 'Deploy' (with job 'pages:deploy'). All jobs in the 'Prepare', 'Test', 'Post Test', and 'Pages' stages are marked as successful with green checkmarks. The 'Deploy' stage job 'pages:deploy' is marked as failed with a red 'x'.



# Integración Continua con Gitlab (CI)

## *Container Registry*

Imagina tener tu propio *docker hub* privado en el que almacenar las imágenes de tu proyecto, actualizarlas en cualquier momento que sea necesario sin tener que exponerlas al público y con la capacidad de utilizarlas desde cualquier sitio con sólo autenticarse. Puedes tener una imagen preparada para cada *stage* de tu *pipeline* y abstraer cantidad de información de *build* a los *runners* de forma transparente e increíblemente rápida. Esto se vuelve **fundamental a la hora de inicializar los entornos**, reduciendo el tiempo necesario para ejecutar nuestros trabajos.



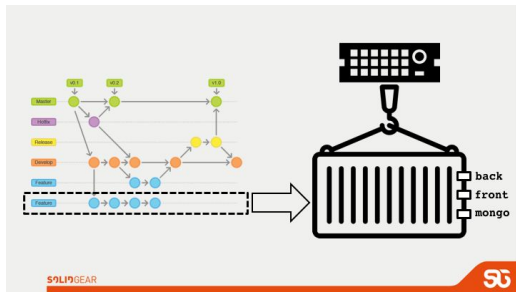
# Integración Continua con Gitlab (CI)

## Entornos y review apps

A todos nos encanta gitflow, ¿verdad? y es que hay unas cuantas razones para ello. Se construye sobre las premisas de que **si cada rama es desarrollada de forma aislada, las nuevas funcionalidades no interfieren con el resto** o con la estabilidad de toda la aplicación hasta que son mergeadas en master. Esto nos ayuda tanto a desarrollar funcionalidades como a testarlas.




# Integración Continua con Gitlab (CI)

Cuando los contenedores irrumpieron en nuestras vidas, era obvio cómo el proceso de desplegar entornos diferentes e independientes con configuraciones individuales podía agilizar nuestros despliegues. Y ahora es justo decir que utilizamos los contenedores como estándar de facto en lo que a despliegues se refiere. La mayor parte de los sistemas de Integración Continua se idearon mucho antes de que el primer contenedor de la historia fuese creado, y por tanto, sin Docker en mente. Con GitLab CI ocurre todo lo contrario: fomentan el uso de los contenedores basándose en todos los beneficios que aportan al flujo de trabajo de trabajo






# Integración Continua con Gitlab (CI)

Por ejemplo, un escenario que se repite en cada *sprint*: estás a punto de *mergear* la rama de una *user story* que tiene que ser testada por el equipo de QA y que a su vez, introduce nuevas dependencias con librerías y con un nuevo servicio (p.ej. Redis, mongoDB...) para ser desplegada. Solo tienes que actualizar tu Dockerfile para incluir la/las nuevas capas que gestionan estas dependencias y hacer *push* de dicha imagen a tu *Docker registry* local. Una vez se suban los cambios de la rama al repositorio, tendrá todo en su lugar para ser desplegada con un click en el entorno de pruebas.

Environment	Last deployment	Build	Commit
production			No deployments yet
staging	#9 by 	#1182	 5832d6c9  Update .gitlab-ci.yml

5 days ago

   Re-deploy



# Integración Continua con Gitlab (CI)

Aquí es donde entran en juego las review apps, que no es más que una forma pija de llamar a **entornos creados dinámicamente por rama, para verificar los cambios y verlos desplegados**. Cada rama tiene la capacidad de ser desplegada en cuanto se hace el push contra el repositorio. Este proceso puede agilizarse mediante herramientas como **dpl** que abstraen muchos de los detalles de implementación de las principales plataformas de despliegues en la nube. O en cambio, llamar manualmente a tu script de despliegue personalizado desde ahí.





# Agradecimiento

Dedicamos esta ponencia a todos los que nos escucháis y en **especial al gran equipo Mob&Web de Guadaltech Soluciones Tecnológicas** que sin ellos esto no habría sido posible.

- Juan Martin
- Ernesto Robles
- Fernando Garduño
- Sergio Cuesta
- Ventura Pino
- Fco. Javier Belloso

Gracias por  
atender

