



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



Escola Tècnica
Superior d'Enginyeria
Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Implementación de un sitio web para un concurso de programación paralela

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Nicolás Fernando Martini

Tutor: Pedro Alonso Jordá

Curso 2019-2020

Dedicatoria

...

Agradecimientos

A mi tutor Pedro Alonso Jordá por su apoyo y la oportunidad de hacer este proyecto con la esperanza que produzca un impacto positivo en la enseñanza en los cursos venideros.

A la ETSINF y todos sus profesores, gracias a ellos he podido crecer personal y profesionalmente.

Al Ministerio de Educación y Gobierno de España, gracias a sus ayudas he podido acceder los estudios de grado.

Resum

...
...
...

Paraules clau: ?, ?, ?, ?

Resumen

El presente trabajo aborda la implementación de un sitio web para concursos de programación paralela, con el añadido de la gamificación o ludificación, una técnica de aprendizaje que busca recompensar al usuario y aumentar su motivación al sumar elementos y dinámicas propias de los juegos para así ofrecer una experiencia enriquecedora y positiva.

Esta es una tarea compleja, por un lado llevar el proceso de envío de código a un entorno web y la interacción que tendrá con el cluster kahan del DSIC. Y por el otro, transformar las actividades de laboratorio de las asignaturas CPA y LPP con nuevas mecánicas que produzcan al estudiante buscar mejorar sus resultados inclusive después de llegar a una resolución correcta a los ejercicios planteados.

Este proyecto ha sido realizado con el apoyo del *DSIC* de la *UPV* con la finalidad de complementar otras herramientas utilizadas hoy en día en la enseñanza.

Palabras clave: programación paralela, concurso de programación, gamificación, ludificación

Abstract

...
...
...

Key words: parallel programming, programming competition, gamification

Índice general

Índice general	VII
Índice de figuras	IX
Índice de tablas	IX
1 Introducción	1
1.1 Motivación	1
1.2 Objetivo	1
1.3 Estructura de la memoria	2
2 Estado del arte	3
2.1 Crítica al estado del arte	3
2.2 Propuesta	3
3 Análisis del problema	5
3.1 Actores	5
3.2 Historias de usuario	5
3.3 Requisitos funcionales	6
3.4 Requisitos no funcionales	9
4 Diseño de la solución	11
4.1 Software	11
4.2 Arquitectura del sistema	11
5 Desarrollo de la solución	13
5.1 Entregables	13
6 Implantación	15
6.1 Docker y Kubernetes	15
6.2 Despliegue en la Nube con Kubernetes	17
6.2.1 Pasos de un despliegue en GCE	17
6.2.2 Costos asociados	19
7 Mantenimiento	21
7.1 Entorno local de desarrollo	21
7.2 Depuración de errores	22
7.3 Calidad de código	22
8 Extensibilidad	25
8.1 Otros sistemas de Gestión de Tareas	25
8.2 Localización	26
9 Conclusiones	27
9.1 Relación del trabajo desarrollado con los estudios cursados	27
10 Trabajos futuros	29
Bibliografía	31
Apéndices	
A Configuración del sistema	33

A.1	Inicialización	33
A.2	Parámetros	34
B	Licencia	37

Índice de figuras

6.1	Diferencias entre contenedor Docker y máquina virtual	15
6.2	nuevo proyecto en GCE	17
6.3	Calculadora de precios en GCE	20
6.4	Estimación de costos en GCE	20

Índice de tablas

3.1	RF-1 Registro de usuarios	6
3.2	RF-2 Login de usuarios	6
3.3	RF-3 Equipos	7
3.4	RF-4 Dashboard	7
3.5	RF-5 Mis Envíos	7
3.6	RF-6 Envío	8
3.7	RF-7 Visualización de Tareas	8
3.8	RF-8 Visualización de Tarea	8
3.9	RF-9 FAQ	8
3.10	RF-10 Tabla de Posiciones	9
3.11	RF-11 Perfil	9
3.12	RNF-1 Usabilidad	9
3.13	RNF-2 Implantación	10
3.14	RNF-3 Configuración	10
3.15	RNF-4 Localización	10

CAPÍTULO 1

Introducción

En este proyecto se trabajarán dos conceptos importantes, la programación paralela y la gamificación para llevarlos a un terreno en conjunto y así crear una herramienta que motive a los estudiantes en el aprendizaje de la asignatura de *LPP*.

La programación paralela es una rama importante de la computación donde se buscar partir problemas de gran magnitud en pedazos más pequeños donde cada partición es ejecutada de forma simultanea por ...

La gamificación como herramienta para motivar

1.1 Motivación

Desde el inicio de mis estudios en el Grado de Ingeniería Informática me han interesado los concursos de programación como los promovidos desde la propia *ETSINF* así como también los disponibles en plataformas online como UVa¹, HackerRank², CheckIO³ y Project Euler⁴. Estas competiciones ayudan a promover el interés en diferentes ramas de los estudios cursados con el añadido de un marco competitivo.

Normalmente, los concursos de programación abarcan problemas relacionados con algoritmia, y al ver la publicación de la propuesta del *TFG* donde se añade la programación paralela me ha despertado el interés ya que no solo se busca que se encuentre la solución al problema planteado, sino que también el punto más importante es la eficiencia.

Todo esto, además de poder contribuir en crear una herramienta que utilizarán los alumnos de la universidad me ha motivado a realizar esta aplicación que expongo en la memoria.

1.2 Objetivo

El principal objetivo es crear una herramienta

¹UVa Online Judge website: <https://onlinejudge.org/>

²HackerRank website: <https://www.hackerrank.com/>

³CheckIO website: <https://checkio.org/>

⁴Project Euler website: <https://projecteuler.net/>

1.3 Estructura de la memoria

La memoria está dividida en los siguientes capítulos:

- **Estado del arte:** se analizan las alternativas que existen al trabajo planteado.
- **Análisis del problema:** se recopila los requerimientos del proyecto para así llegar a una propuesta que abarque las necesidades actuales de la asignatura *LPP*.
- **Diseño de la solución:** se describe todos los elementos a nivel de arquitectura que forman parte de la solución.
- **Desarrollo de la solución:** ?
- **Implantación:** se explica los pasos a seguir para desplegar el software.
- **Pruebas:** se describe las pruebas que se han realizado para verificar el correcto funcionamiento.
- **Mantenimiento:** ?
- **Conclusiones:** ?

CAPÍTULO 2

Estado del arte

2.1 Crítica al estado del arte

2.2 Propuesta

CAPÍTULO 3

Análisis del problema

Antes de comenzar a escribir una línea de código, como todo proyecto ingenieril se debe hacer un estudio que involucre los usuarios que utilizarán el sistema donde se recogerán requisitos para poder hacer una propuesta acorde a sus necesidades.

3.1 Actores

El sistema contará con dos perfiles de usuarios que llamaremos actores, estos son el *Estudiante* y *Administrador*. También, veremos que en la memoria se hace mención al *usuario*, esto se hace para referirnos a funcionalidades que se aplican tanto a los *Estudiantes* como *Administradores*.

- **Estudiante:** el rol principal del sistema. Enviará soluciones a los problemas propuestos y tratará de mejorar sus resultados para subir su ranking en la tabla de posiciones de las tareas y la grupal. Su trabajo será recompensado en forma de puntos e insignias.
- **Administrador:** el rol que se encarga de poner el sistema en marcha. Tendrá que tener conocimientos sólidos en la asignatura de *CPA* para poder crear tareas que varíen en dificultad y motive al alumnado a mejorar sus resultados.

3.2 Historias de usuario

Como primer paso se describe el resultado esperado de este proyecto en un lenguaje sencillo que más adelante darán partida a los requisitos funcionales y no funcionales de la aplicación. A esto se lo llaman **Historias de usuario**.

- Como *Estudiante*, quiero ingresar al sistema utilizando mis credenciales personales.
- Como *Estudiante*, quiero poder cambiar mi contraseña.
- Como *Estudiante*, quiero registrarme en el sistema para poder hacer uso del mismo.
- Como *Estudiante*, quiero enviar las resoluciones a mis tareas asignadas.
- Como *Estudiante*, quiero formar parte de un equipo con otros estudiantes.
- Como *Estudiante*, quiero acceder a mi perfil para ver toda mi información.

- Como *Estudiante*, quiero visualizar los resultados de las ejecuciones para ver como han quedado posicionadas en la tabla de clasificación.
- Como *Estudiante*, quiero ver un resumen de mi actividad.
- Como *Administrador*, quiero crear grupos para que los estudiantes puedan formar parte de ellos.
- Como *Administrador*, quiero crear estudiantes subiendo un fichero csv o txt.
- Como *Administrador*, quiero crear y asignar tareas a diferentes grupos.
- Como *Administrador*, quiero crear insignias para asignar a diferentes tareas.
- Como *Administrador*, quiero editar y eliminar grupos, estudiantes, tareas e insignias.
- Como *Administrador*, quiero desplegar el aplicativo en una solución cloud.

3.3 Requisitos funcionales

En el siguiente apartado se describen todos los requisitos funcionales del aplicativo.
... explicación de la tabla ...

RF-1

Nombre	Registro de usuarios
Descripción	El sistema permitirá el registro de usuarios mediante dirección de correo electrónico
Prioridad	Media
Criterio de aceptación	<ul style="list-style-type: none"> • no puede existir más de un usuario con el mismo correo electrónico • se generará un usuario único con el alias del correo electrónico • se podrá deshabilitar el registro de usuarios mediante configuración del sistema • se podrá limitar el registro de usuarios a correos electrónicos que pertenezcan a dominios específicos (eg: @upv.es, @inf.upv.es)

Tabla 3.1: RF-1 Registro de usuarios

RF-2

Nombre	Login de usuarios
Descripción	El sistema permitirá el login de usuarios mediante dirección de correo electrónico o usuario
Prioridad	Más Alta
Criterio de aceptación	<ul style="list-style-type: none"> • solo se podrá ingresar al sistema si la cuenta está activa

Tabla 3.2: RF-2 Login de usuarios

RF-3

Nombre	Equipos
Descripción	Los <i>Estudiantes</i> podrán formar parte de un equipo con otros compañeros de grupo
Prioridad	Baja
Criterio de aceptación	<ul style="list-style-type: none"> • solo se podrá formar parte de un único equipo en un momento determinado de tiempo • si un equipo no tiene integrantes se eliminará del sistema • los equipos contarán con una URL única que permitirá a otros estudiantes unirse a los mismos • un <i>Estudiante</i> no puede unirse un equipo que no forme parte de su grupo o que haya llegado al máximo de integrantes • los equipos tendrán un máximo de integrantes que podrá ser configurado por el <i>Administrador</i>

Tabla 3.3: RF-3 Equipos**RF-4**

Nombre	Dashboard
Descripción	Los <i>Estudiantes</i> al ingresar verán un resumen agregado de su estado general en el sistema
Prioridad	Alta
Criterio de aceptación	<p>El Dashboard o <i>Página de Inicio</i> del <i>Estudiante</i> deberá mostrar la siguiente información:</p> <ul style="list-style-type: none"> • resumen de los últimos envíos • extracto calculado de cantidad de envíos, tareas, cuota disponible y puntaje • la última insignia obtenida, en caso que no tuviere, alentarlos a completar una tarea para conseguir su primera • los integrantes del equipo, en caso que no tuviere, alentarlos a crear uno nuevo o unirse a uno existente

Tabla 3.4: RF-4 Dashboard**RF-5**

Nombre	Mis Envíos
Descripción	Los <i>Usuarios</i> podrán ver un resumen de sus envíos enviados al sistema
Prioridad	Muy Alta
Criterio de aceptación	<p>El resumen de envíos se visualizará en forma de tabla, se podrá filtrar por la información disponible y deberá mostrar las siguientes columnas:</p> <ul style="list-style-type: none"> • ID único en el sistema • Fecha y Hora de envío • Tarea a la que corresponde • Estado del envío • Puntaje obtenido • Tiempo de ejecución

Tabla 3.5: RF-5 Mis Envíos

RF-6

Nombre	Envío
Descripción	Visualización del envío a ejecutar
Prioridad	Alta
Criterio de aceptación	Se deben mostrar los siguientes apartados e información: <ul style="list-style-type: none"> • código fuente enviado • análisis estático del código fuente enviado • output del código ejecutado • tiempo de ejecución y status

Tabla 3.6: RF-6 Envío**RF-7**

Nombre	Tareas
Descripción	Visualización de listado de Tareas
Prioridad	Alta
Criterio de aceptación	Se deben mostrar los siguientes apartados e información: <ul style="list-style-type: none"> • si el usuario es un <i>Estudiante</i> se deben mostrar las tareas asignadas a su grupo • si el usuario es un <i>Administrador</i> se deben mostrar todas las tareas en el sistema • por cada Tarea debe haber un resumen, insignias y últimos envíos • no se deben mostrar las insignias secretas

Tabla 3.7: RF-7 Visualización de Tareas**RF-8**

Nombre	Tarea
Descripción	Visualización de Tarea
Prioridad	Alta
Criterio de aceptación	Se deben mostrar los siguientes apartados e información: <ul style="list-style-type: none"> • descripción completa • insignias a obtener • insignias secretas que el usuario ha obtenido • status de la tarea: abre pronto, abierto, cierra pronto, cerrada • si el usuario ha enviado una ejecución satisfactoria se le debe informar en un mensaje • enlace para hacer un envío con sus credenciales • enlace a la tabla de posiciones

Tabla 3.8: RF-8 Visualización de Tarea**RF-9**

Nombre	FAQ
Descripción	Página con preguntas frecuentes que se pueden hacer los usuarios del sistema y sus respuestas
Prioridad	Baja
Criterio de aceptación	<ul style="list-style-type: none"> • debe estar localizado en al menos 2 idiomas • no debe contener más de 10 preguntas

Tabla 3.9: RF-9 FAQ

RF-10

Nombre	Tabla de Posiciones
Descripción	
Prioridad	Alta
Criterio de aceptación	• ?

Tabla 3.10: RF-10 Tabla de Posiciones**RF-11**

Nombre	Perfil
Descripción	
Prioridad	Media
Criterio de aceptación	• ?

Tabla 3.11: RF-11 Perfil

3.4 Requisitos no funcionales

Los requisitos no funcionales son criterios que se deben cumplir para juzgar la correcta operación del sistema. En contraste con los requisitos funcionales, no definen comportamientos específicos.

... explicación de la tabla ...

RNF-1

Nombre	Usabilidad
Descripción	El sitio web deberá tener una interfaz sencilla y fácil de utilizar.
Prioridad	Muy Alta

Tabla 3.12: RNF-1 Usabilidad

RNF-2

Nombre	Implantación
Descripción	El aplicativo debe poder implantarse de forma automatizada
Prioridad	Media

Tabla 3.13: RNF-2 Implantación**RNF-3**

Nombre	Configuración
Descripción	El aplicativo debe permitir configurar los parámetros de despliegue y uso
Prioridad	Alta

Tabla 3.14: RNF-3 Configuración**RNF-4**

Nombre	Localización
Descripción	El aplicativo debe soportar la localización en diferentes idiomas
Prioridad	Media

Tabla 3.15: RNF-4 Localización

CAPÍTULO 4

Diseño de la solución

?

4.1 Software

?

4.2 Arquitectura del sistema

Resumen de como es el sistema a nivel de arquitectura. Redis, PostgreSQL, etc..

CAPÍTULO 5

Desarrollo de la solución

?

5.1 Entregables

Sprints ¿?

CAPÍTULO 6

Implantación

Pizarra es una herramienta que necesita de otros aplicativos para su funcionamiento. Por un lado una *BD* para la persistencia de datos, Redis para el sistema de colas interno de envío de tareas, un NFS para compartir recursos y Nginx para actuar como servidor web y redirigir las peticiones. En el pasado la implantación hubiera requerido la intervención del equipo de Administración de Sistemas que se encargue de instalar el *SO* de cada máquina, crear usuarios, asignar permisos, abrir puertos y otro sinfín de tareas.

Para evitar todos estos pasos y acelerar la implantación el proyecto incluye la posibilidad de que todos los elementos de la arquitectura se desplieguen en Contenedores. Estos Contenedores están creados con la tecnología Docker y son paquetes que incluyen todo lo necesario, sean librerías y herramientas del sistema para que el software se ejecute.

Docker funciona de forma similar a una maquina virtual, con algunas ventajas como la asignación de recursos de forma dinámica, espacio en disco reducido ya que podemos evitar la instalación de un *SO* en los contenedores y la portabilidad.

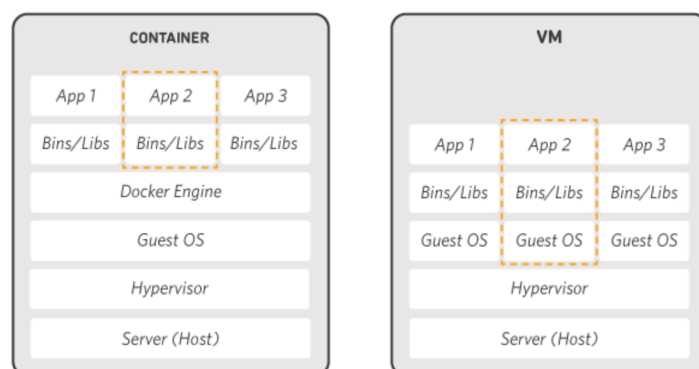


Figura 6.1: Diferencias entre contenedor y máquina virtual

6.1 Docker y Kubernetes

Para nuestro entorno de desarrollo local utilizamos contenedores públicos como el de Postgres que con un fichero de configuración nos permite tener el servicio ejecutándose en cuestión de minutos.

A continuación, un extracto del fichero **docker-compose.yml** del repositorio¹ con una breve explicación de los parámetros.

¹Docker yaml: <https://github.com/nimar3/pizarra/blob/master/docker-compose.yml>

```
1 version: "3.1"
2 services:
3   postgres:
4     image: postgres: 9.6-alpine
5     container_name: postgres
6     volumes:
7       - /data/pizarra/postgres:/var/lib/postgresql/data
8     ports:
9       - 5432: 5432
10    environment:
11      - POSTGRES_USER=pizarra
12      - POSTGRES_PASSWORD=pizarra
13    restart: unless-stopped
```

- **version:** versión de la API de Docker que utilizaremos.
- **services:** listado de servicios que desplegaremos.
- **image:** contenedor que utilizaremos, en este caso el oficial de postgres² sobre Linux Alpine.
- **container_name:** nombre con el que comenzará el contenedor que se ejecute.
- **volumes:** mapeo de directorios de nuestro entorno de desarrollo local al contenedor, esto se hace para evitar perder la información de la *BD* al ser los contenedores stateless.
- **ports:** mapeo de puertos del *SO* a los contenedores, esto permite acceder al servicio de Postgres por un puerto local.
- **environment:** variables de entorno del *SO*, en nuestro caso definimos un usuario y contraseña para acceder a la *BD*
- **restart:** con el parámetro `unless-stopped` nuestro contenedor se reiniciará de forma automática en caso de errores a menos que se envíe un mensaje de finalización de ejecución.

Con los parámetros previos y siguiendo la documentación de variables de entorno disponibles, al iniciarse el contenedor se crearán los schemas y directorios necesarios en caso de que no existan y un usuario con privilegios con el nombre y contraseña proporcionados.

Para un entorno de desarrollo local y/o de pruebas Docker puede ser una herramienta que cumpla con nuestras necesidades, pero para un entorno de producción es necesario contar con un servicio que realice la implantación de forma automatizada, que sea eficiente en el manejo de recursos y resiliente en caso de problemas. Para ésta tarea utilizaremos Kubernetes, un *orquestador* de contenedores que permite auto escalado de aplicaciones, automatización de despliegues y configuración de forma declarativa.

Debido a una limitación de las máquinas donde está alojado kahan no es posible la instalación del servicio de Kubernetes y se ha optado por un despliegue de Pizarra en la nube.

²Postgres Official Docker Image: https://hub.docker.com/_/postgres

6.2 Despliegue en la Nube con Kubernetes

Existen varios proveedores de *K8s* en Internet pudiendo consultar el listado de todos los partners en el sitio oficial³. Siendo los más importantes Google Cloud Engine, Amazon AWS y Microsoft Azure se ha escogido por desplegar la aplicación en *GCE* ya que dispone de buena documentación y además \$300 de crédito promocional que són más que suficientes para tener el aplicativo corriendo durante meses.

6.2.1. Pasos de un despliegue en GCE

Debemos crearnos una cuenta en Google o utilizar una existente que nunca ha activado *GCE* de forma previa ya que el crédito promocional de bienvenida tiene una caducidad de 12 meses desde la activación.

Con nuestra cuenta accedemos a la Consola Web de GCE y creamos nuestro proyecto, en este caso lo llamaremos Pizarra. Nuestro aplicativo utiliza Kubernetes Engine y el Container Registry que son visibles desde el menú lateral, entraremos a cada uno de ellos para habilitar su API.

La imagen muestra dos capturas de pantalla de la interfaz de Google Cloud Platform. La parte superior muestra el 'Dashboard' con un mensaje que indica que se debe seleccionar un proyecto para ver la página, y un botón 'CREATE PROJECT' resaltado con un recuadro rojo. La parte inferior muestra la pantalla 'New Project' con el nombre del proyecto 'pizarra' ingresado en el campo 'Project name *'. Debajo, se muestra el 'Project ID' como 'pizarra-279707' y una advertencia de que no puede cambiarse más tarde. El campo 'Location *' está configurado en 'No organization'. Al final, hay botones 'CREATE' y 'CANCEL'.

Figura 6.2: Creación de un nuevo proyecto en Google Cloud Engine.

Una vez creado el proyecto y habilitadas las API's hay que descargar⁴ e inicializar⁵ Google Cloud SDK para hacer uso de la consola de forma local.

Todos los pasos descritos a continuación se ejecutan de forma automática con un script⁶ incluido en el repositorio de código. Lo único que se necesita de forma previa es la configuración de algunos parámetros de despliegue como variables de entorno del SO donde se ejecuta.

³Kubernetes - Partners: <https://kubernetes.io/es/partners>

⁴Cómo instalar el SDK de Google Cloud: <https://cloud.google.com/sdk/install?hl=es-419>

⁵Cómo inicializar el SDK de Cloud: <https://cloud.google.com/sdk/docs/initializing?hl=es-419>

⁶Script Despliegue GCE: <https://github.com/nimar3/pizarra/blob/master/gce/commands.sh>

Las variables de entorno necesarias son el ID del proyecto en GCE y las credenciales de la DB que serán guardados como un secreto en Kubernetes. Podemos exportarlas antes de lanzar nuestro script ejecutando los siguientes comandos en nuestra consola.

```
1 $ export PROJECT_ID=pizarra-id
2 $ export DB_USERNAME=pizarra
3 $ export DB_PASSWORD=pizarra
```

Éstas son las líneas que ejecuta el script, donde todos los ficheros que forman parte de cada comando se encuentran en el repositorio⁷. Al ser la sintaxis similar a la previamente explicada y evitando que éste documento se extienda de sobremanera al mencionar cada ítem, invitamos al lector a revisar la documentación oficial de Kubernetes⁸.

1. Seteo de proyecto y zona geográfica donde desplegaremos el aplicativo. Por cercanía geográfica se ha escogido el centro de datos de Países Bajos.

```
1 $ gcloud config set project ${PROJECT_ID}
2 $ gcloud config set compute/zone europe-west4
```

2. Credenciales para la DB para ser obtenidas a posterior por los contenedores de Kubernetes

```
1 $ kubectl create secret generic pizarra-credentials \
2 --from-literal db_username=${DB_USERNAME} \
3 --from-literal db_password=${DB_PASSWORD}
```

3. Subida de Contenedores de Nginx y Pizarra al Registro de GCE. Los otros Contenedores utilizarán imágenes públicas.

```
1 $ docker push eu.gcr.io/${PROJECT_ID}/pizarra
2 $ docker push eu.gcr.io/${PROJECT_ID}/nginx
```

4. Creación de cluster en Kubernetes con 2 nodos. Una vez creado se descargan las credenciales para trabajar con él.

```
1 $ gcloud container clusters create pizarra --num-nodes=2
2 $ gcloud container clusters get-credentials pizarra
```

5. Creación de Almacenamiento de 10 GiB para la persistencia de datos que será utilizado por el Sistema de archivos de red. Replicado en varias zonas ya que el almacenamiento no es parte de Kubernetes.

```
1 $ kubectl apply -f storage.yaml
```

6. Creación del servicio de NFS

```
1 $ kubectl apply -f nfs.yaml
```

7. Creación del servicio de Redis para el sistema de colas local

```
1 $ kubectl apply -f redis.yaml
```

8. Creación del servicio de Base de datos

```
1 $ kubectl apply -f postgres.yaml
```

⁷Ficheros de despliegue en GCE: <https://github.com/nimar3/pizarra/tree/master/gce>

⁸Documentación Kubernetes: <https://kubernetes.io/docs>

9. Despliegue de Pizarra y 1 Worker que se encargará de encolar las las Tareas en kahan

```
1 $ kubectl apply -f pizarra.yaml
```

10. Despliegue del servicio de NGinx con una IP pública para el acceso externo de usuarios

```
1 $ kubectl apply -f nginx.yaml
```

Una vez completados todos los pasos se puede obtener la IP pública de Pizarra ejecutando el siguiente comando.

```
1 $ kubectl get services
```

También se puede obtener en la web de GCE sobre la pestaña Services e Ingress.

IMAGEN

Hemos completado el despliegue y ya podemos navegar a Pizarra para comenzar a interactuar con el aplicativo.

IMAGEN

6.2.2. Costos asociados

GCE cuenta con una herramienta⁹ para estimar el gasto mensual en el que podemos incurrir con la utilización de los recursos contratados. Se ha realiado una estimación con los siguientes parámetros:

- Kubernetes
 - Número de nodos: 2
 - Tipo de Instancia: n1-standard-n1 (vCPUs: 1, RAM: 3.75 GB)
 - Centro de Datos: Países Bajos
 - Uso estipulado: 24 horas al día los 7 días a la semana.
- Persistencia
 - Tamaño disco: 10 GiB
 - Centro de Datos: Países Bajos

En resumen, el número de nodos es 2 para tener servicio en caso de que haya una caída de uno de los nodos del cluster, el centro de datos se ha escogido Países Bajos por cercanía geográfica y una instancia con los recursos mínimos necesarios para los contenedores. El disco es compartido entre la *BD* y Pizarra con suficiente margen para evitar problemas de espacio.

En total llegamos a un costo estimado de €49.00 mensuales. Como dato adicional, se puede variar de Centro de Datos y otros parámetros como el uso estipulado para reducir los costos aún más. Podemos guardar la estimación o enviárnosla por correo electrónico.

⁹Calculadora de precios de Google: <https://cloud.google.com/products/calculator/>

Kubernetes Engine

Number of nodes *
2

Number of Zonal clusters - excluding Anthos GKE clusters ^{1,2}

Number of Regional clusters - excluding Anthos GKE clusters ¹

What are these nodes for?

Instance type
n1-standard-1 (vCPUs: 1, RAM: 3.75 GB)

☐ Add GPUs.

Local SSD
0

Datacenter location
Netherlands (europe-west4)

Committed usage
None

Average hours per day each node is running *
24 hours per day

Average days per week each node is running *
7


¹ Anthos GKE clusters are exempted from GKE Cluster Management Fee.
² One Zonal cluster is free per billing account.

Persistent Disk

Location
Netherlands (europe-west4)

Persistent disk storage
10 GiB

Figura 6.3: Configuración de la calculadora de precios para el despliegue en Kubernetes de Pizarra.



Google Cloud

Nicolas Martini,

Your Estimated Bill *

Estimated Monthly Cost: EUR 49.00



 2 x	n1-standard-1	1460 total hours per month	EUR 48.60
 Persistent disk	Storage	10 GiB	EUR 0.40
Total Estimated Monthly Cost			EUR 49.00

Figura 6.4: Ejemplo de correo de GCE con una estimación de los costos asociados por mes para la implantación.

CAPÍTULO 7

Mantenimiento

El desarrollo de software es un proceso evolutivo y requiere de un mantenimiento continuo para asegurar el correcto funcionamiento, como así también la posibilidad de añadir mejoras y nuevas funcionalidades. Para esto se han creado los siguientes apartados y así ayudar con éste proceso.

7.1 Entorno local de desarrollo

Para configurar un entorno de desarrollo local debemos cumplir con los siguientes requisitos.

- Tener instalado Python³¹ y Docker² en nuestra máquina
- Clonado el repositorio de Pizarra

1. Creación de directorio

```
1 $ mkdir pizarra
2 $ cd pizarra/
```

2. Inicialización de un repositorio nuevo y la dirección remota

```
1 $ git init .
2 Initialized empty Git repository in /Users/nmartini/pizarra/.git/
3 $ git remote add upstream git@github.com:nimar3/pizarra.git
```

3. Descarga el código fuente

```
1 $ git fetch upstream
2 remote: Enumerating objects: 589, done.
3 remote: Counting objects: 100% (589/589), done.
4 remote: Compressing objects: 100% (364/364), done.
5 remote: Total 6891 (delta 396), reused 393 (delta 221), pack-
   reused 6302
6 Receiving objects: 100% (6891/6891), 19.14 MiB | 675.00 KiB/s,
   done.
7 Resolving deltas: 100% (1748/1748), done.
```

4. checkout a la rama de desarrollo

¹Descarga oficial de Python: <https://www.python.org/downloads/>

²Descarga oficial de Docker: <https://docs.docker.com/get-docker/>

```

1 $ git checkout master
2 Checking out files: 100% (4858/4858), done.
3 Branch 'master' set up to track remote branch 'master' from '
  upstream'.
4 Already on 'master'

```

- Creación de un entorno virtual e instalación de todos los Paquetes de Python necesarios

```

1 $ python3 -m venv env
2 $ source env/bin/activate
3 (env) $ pip3 install -r requirements_dev.txt

```

Si hemos realizado cambios en el código fuente se debe ejecutar el siguiente comando para generar una nueva versión de nuestro Contenedor de Docker de Pizarra

```

1 $ docker build -t pizarra .

```

Por último, para ejecutar Pizarra con todos los componentes introducimos el siguiente comando en la raíz del repositorio. Todos los componentes deben mostrar que se han creado correctamente con el mensaje *done*

```

1 (env) $ docker-compose up
2 Creating network "pizarra_default" with the default driver
3 Creating redis ... done
4 Creating postgres ... done
5 Creating pizarra ... done
6 Creating worker ... done
7 Creating nginx ... done

```

7.2 Depuración de errores

Como todo software, Pizarra no estará exento de errores o bugs que no se han identificado en la etapa de desarrollo y como muchos otros lenguajes Python ofrece la ejecución de código en modo Debug, lo que nos permite poner puntos de ruptura y hacer una ejecución *paso a paso* del código para identificar el problema.

En el entorno de desarrollo local, el modo Debug siempre está habilitado. Si queremos habilitarlo cuando ejecutamos el aplicativo en modo *Production* debemos especificarlo explícitamente en las variables de entorno del SO, en este caso revisar el Apéndice A: Configuración del Sistema para más información.

7.3 Calidad de código

Para una mejor lectura del código fuente se ha utilizado el Paquete *flake8*³ que nos obliga a seguir las guías de estilo oficiales de Python.

En el directorio raíz del repositorio de código ejecutamos el siguiente comando y podemos ver un listado de errores a corregir.

```

1 $ flake8 app/
2 app/__init__.py:8:1: F401 'logging.DEBUG' imported but unused
3 app/__init__.py:71:5: E722 do not use bare 'except'

```

³Sitio Web de Flake8: <https://flake8.pycqa.org/en/latest/>


```
4 app/home/routes.py:59:101: E501 line too long (106 > 100 characters)
5 app/home/routes.py:210:5: E722 do not use bare 'except'
6 app/home/routes.py:216:101: E501 line too long (108 > 100 characters)
7 app/admin/forms.py:32:28: N805 first argument of a method should be named 'self'
```

La configuración de *flake8* se encuentra en el fichero **setup.tfg** en la raíz del repositorio.

```
1 [flake8]
2 max-line-length=100
3 ignore=E402,E266
4 exclude=./migrations
```

Otro punto importante es la cobertura del código fuente, aunque el repositorio incluye las librerías de *coveralls* y *pytest* por cuestiones de tiempo no se han escrito tests unitarios y se ha dejado como una mejora futura.

CAPÍTULO 8

Extensibilidad

Uno de los puntos más importantes es permitir la colaboración y que Pizarra funcione con otros sistemas de tareas que no sean una Cola local o la de kahan así como también que pueda ser localizado en diferentes idiomas.

8.1 Otros sistemas de Gestión de Tareas

La ejecución de una tarea es diferente en cada Cola y para esto se ha implementado un autómata que va cambiando de estados en cada ejecución hasta finalizarla. Esto permite que cada nuevo tipo de Cola tenga su propia definición, sea flexible y compatible con el funcionamiento del aplicativo.

La definición de estas colas se encuentra en **models_tasks.py**¹ siendo el objeto a extender para las diferentes colas que queramos implementar *Task*

```
1 class Task:
2
3     def __init__(self, user_request):
4         self.user_request = user_request
5         self.output = ""
6         self.binary_file_location = ""
7         self.return_code = 0
8         self.run_time = 0.0
9         self.points_earned = 0
10        # automata
11        self.task_process = {}
```

El autómata cuenta con las siguientes transiciones

```
1 class StepResult(enum.Enum):
2     START = 0
3     OK = 1
4     NOK = 2
5     WAIT = 3
6     END = 4
```

Para el sistema de Colas de kahan tenemos el siguiente extracto. Donde creamos la nueva clase *KahanTask* que extiende de *LocalTask* y hereda todos sus atributos y métodos.

```
1 class KahanTask(LocalTask):
2
3     def __init__(self, user_request):
```

¹Definición de Colas: https://github.com/nimar3/pizarra/blob/master/app/base/models_tasks.py

```
4  super().__init__(user_request)
5  self.task_process = {
6      RequestStatus.CREATED: {
7          'f': self.start,
8          'steps': {
9              StepResult.OK: RequestStatus.VERIFYING,
10             }
11      },
12      RequestStatus.VERIFYING: {
13          'f': self.verify,
14          'steps': {
15              StepResult.OK: RequestStatus.COMPILING,
16              StepResult.NOK: RequestStatus.ERROR
17          }
18      },
19      RequestStatus.COMPILING: {
20          'f': self.compile,
21          'steps': {
22              StepResult.OK: RequestStatus.DEPLOYING,
23              StepResult.NOK: RequestStatus.ERROR
24          }
25      },
26      ...
```

En la definición del *task_process* tenemos un diccionario con los diferentes estados que puede encontrarse el autómata, la función a ejecutar *f* y para cada ejecución el diccionario *steps* con las transiciones.

8.2 Localización

CAPÍTULO 9

Conclusiones

...

9.1 Relación del trabajo desarrollado con los estudios cursados

?

CAPÍTULO 10

Trabajos futuros

?

Bibliografía

- [1] Jennifer S. Light. When computers were women. *Technology and Culture*, 40:3:455–483, juliol, 1999.
- [2] Georges Ifrah. *Historia universal de las cifras*. Espasa Calpe, S.A., Madrid, sisena edició, 2008.
- [3] Comunicat de premsa del Departament de la Guerra, emés el 16 de febrer de 1946. Consultat a <http://americanhistory.si.edu/comphist/pr1.pdf>.

APÉNDICE A

Configuración del sistema

Para una correcto funcionamiento de Pizarra el sistema debe inicializarse siguiendo los pasos descritos a continuación. Aunque se dispone de un script que realiza el despliegue de forma automatizada hay algunos pasos que requieren de intervención manual.

A.1 Inicialización

Antes de comenzar con la inicialización del sistema, debemos exportar las variables de entorno del SO que utiliza el script. Abrimos una consola y ejecutamos los siguientes comandos, donde *pizarra-id* es el ID de proyecto en Google Cloud Engine (GCE).

```
1 $ export PROJECT_ID=pizarra-id
2 $ export DB_USERNAME=pizarra
3 $ export DB_PASSWORD=pizarra
```

Si quisiéramos cambiar algún parámetro por defecto de Pizarra, debemos modificar el fichero **pizarra.yaml**¹ añadiendo los nuevos parámetros. En el siguiente extracto se ha cambiado los puntos de penalización al obtener un *KO* y *TIMEWALL* en el envío de una Tarea y la carga de data de ejemplo.

```
1 ...
2 spec:
3   containers:
4     - image: eu.gcr.io/pizarra-279100/pizarra
5       name: pizarra
6       env:
7         - name: TIMEWALL_PENALTY
8           value: "-20"
9         - name: "KO_PENALTY"
10          value: "-30"
11         - name: "IMPORT_SAMPLE_DATA"
12          value: "True"
13 ...
```

A continuación se ejecuta el script.

```
1 $ sh commands.sh
```

¹Fichero YAML de Pizarra: <https://github.com/nimar3/pizarra/blob/master/gce/pizarra.yaml>

Al terminar la ejecución deberíamos tener Pizarra con todos los componentes desplegados. Podemos obtener el estado de los contenedores y servicios con los siguientes comandos.

```
1 $ kubectl get pods
2 $ kubectl get services
```

A.2 Parámetros

Al iniciar un Contenedor utilizamos variables de entorno del SO para configurar su funcionamiento. El Contenedor de Pizarra soporta estas variables para dar diferentes posibilidades en cómo queremos que se comporte el aplicativo. Los mencionados a continuación son relevantes en un despliegue pero existen adicionales que pueden ser consultados en el fichero de configuración²

■ APP_MODE

- Descripción: ejecución del aplicativo web o un worker
- Opciones
 - Pizarra: aplicativo web
 - Worker: worker
- Valor por defecto: *Pizarra*

■ CONFIG_MODE

- Descripción: modo en el cual queremos que se ejecute Pizarra
- Opciones
 - Debug: modo Debug habilitado y Base de datos (DB) SQLite
 - Production: modo Debug deshabilitado y Base de datos (DB) Postgres
- Valor por defecto: *Debug*

■ REMOTE_HOST

- Descripción: hostname o IP del sistema externo a conectarnos para las colas de kahan
- Valor por defecto: *kahan.dsic.upv.es*

■ REMOTE_USER

- Descripción: usuario con el que nos conectaremos por SSH y SCP a kahan
- Valor por defecto: *pizarra*

■ REMOTE_PATH

- Descripción: directorio remoto donde se copiarán los ficheros de cada ejecución en kahan
- Valor por defecto: */pizarra*

²Fichero de Configuración: <https://github.com/nimar3/pizarra/blob/master/config.py>

■ SSH_FILE_PATH

- Descripción: ubicación de la clave privada para conectarnos por SSH y SCP a kahan
- Valor por defecto: *app/data/keys/id_rsa*

■ LOG_LEVEL

- Descripción: establece el nivel de mensajes que deben mostrarse al ejecutarse el aplicativo
- Opciones
 - CRITICAL
 - ERROR
 - WARNING
 - INFO
 - DEBUG
- Valor por defecto: *INFO*

■ SECRET_KEY

- Descripción: clave secreta con la que se encriptarán las contraseñas, no se puede cambiar una vez desplegada la aplicación y por seguridad nunca debe utilizarse el valor por defecto
- Valor por defecto: *pUdos1KbNyLYUvb4P7MvHWmuWSGH0Au*

■ IMPORT_SAMPLE_DATA

- Descripción: borrado de información de la Base de datos e importación de data de ejemplo.
- Valor por defecto: *False*

■ TIME_BETWEEN_REQUESTS

- Descripción: tiempo mínimo en segundos que debe esperar un Alumno entre cada envío de Tareas
- Valor por defecto: *60*

■ TEAM_MAX_SIZE

- Descripción: tamaño máximo de Alumnos que puede tener un Equipo
- Valor por defecto: *3*

■ REGISTRATION_ENABLED

- Descripción: habilita el registro de usuarios en el aplicativo
- Valor por defecto: *False*

■ TIMEWALL_PENALTY

- Descripción: puntos de penalización al obtener un resultado de *TIMEWALL* en un envío de Tarea

- Valor por defecto: *-10*

■ **KO_PENALTY**

- Descripción: puntos de penalización al obtener un resultado de *KO* en un envío de Tarea
- Valor por defecto: *-15*

■ **DEBUG**

- Descripción: inicia el aplicativo en modo Debug, útil para identificar errores.
- Valor por defecto: *False*

APÉNDICE B

Licencia

Como este es un proyecto de código abierto, se ha escogido la Licencia MIT¹ ya que es una de las más permisivas para fomentar el rehúso y colaboración. Este tipo de licencia permite la libre utilización de forma privada o comercial, la modificación y distribución del código sin ningún tipo de garantía donde solo se pide que se mantengan las menciones a la licencia y derechos de autor.

¹Licencia MIT: <https://opensource.org/licenses/MIT>

Acrónimos

- API** Interfaz de programación de aplicaciones. 17, 35, 37
- CPA** Computación Paralela. 35, 37
- DB** Base de datos. 18, 32, 33, 35, 37
- DSIC** Departamento de Sistemas Informáticos y Computación. 35, 37
- ETSINF** Escuela Técnica Superior de Ingeniería Informática. 35, 37
- GCE** Google Cloud Engine. 17–19, 31, 35, 37
- GiB** Gibibyte. 18, 35, 37
- K8s** Kubernetes. 35, 37
- LPP** Lenguajes y Entornos de la Programación Paralela. 35, 37
- NFS** Sistema de archivos de red. 15, 18, 35, 37
- RAM** Memoria de acceso aleatorio. 35, 37
- RF** Requisito funcional. 35, 37
- RNF** Requisito no funcional. 35, 37
- SCP** Secure Copy. 32, 33, 35, 37
- SO** Sistema operativo. 15, 17, 31, 32, 35, 37
- SSH** Secure Shell. 32, 33, 35, 37
- TFG** Trabajo Final de Grado. 35, 37
- UPV** Universitat Politècnica de València. 35, 37
- vCPUs** Unidades centrales de procesamiento virtuales. 35, 37

Términos

alumno description. 33, 35, 37

contenedor imagen ejecutable ligera y portátil que contiene el software y todas sus dependencias. 15, 18, 32, 35, 37

Entorno de producción lugar donde se ejecuta el aplicativo para los usuarios finales. 35, 37

equipo description. 33, 35, 37

Equipo de Estudiantes Grupo de uno o más estudiantes formado para resolver Tareas de forma conjunta. 35, 37

formula A mathematical expression. 35, 37

GCD Greatest Common Divisor. 35, 37

Grupo de Estudiantes Clase a la que pertenece el estudiante para el año lectivo. 35, 37

Insignia Trofeo o Marcador que se obtiene al realizar cierta acción en la aplicación. 35, 37

latex Is a mark up language specially suited for scientific documents. 35, 37

LCM Least Common Multiple. 35, 37

mathematics Mathematics is what mathematicians do. 35, 37

Tarea Problema a resolver asignado a un estudiante o equipo. 19, 31, 33–35, 37