ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE
TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Trabajo Fin de Máster

Uso de Azure Functions para lanzar scripts de R en modo "serverless". Aplicación al estudio de tasa de abandono

AUTOR: Víctor Huéscar López
DIRECTOR: Francisco Javier Garrigós Guerrero
CODIRECTOR: Mathieu Kessler
Junio / 2019

ÍNDICE

ÍNDICE DE FIGURAS

CAPÍTULO 1: INTRODUCCIÓN

1.1. Introducción

El desarrollo vertiginoso de la industria tecnológica en los últimos años ha provocado la aparición de nuevas tecnologías y cambios de paradigma con respecto a los utilizados hasta ahora. Las mejoras alcanzadas en materia de capacidad computacional, almacenamiento y en comunicaciones han posibilitado el desarrollo de las tecnologías basadas en *cloud* (nube).

Esta nueva filosofía, que tan común es ya en las aplicaciones que consumimos todos los días, ha supuesto una tendencia a un nuevo modelo basado en servicios que desplazan el almacenamiento y procesado de datos a la nube. Esta evolución desde lo local a lo ubicuo ha tenido su repercusión, no solo en el ámbito del consumo a nivel de usuario, sino también a nivel de desarrolladores y empresas. Bajo la filosofía *cloud* y siguiendo el modelo basados en servicios (XaaS, *Everything as a Service*), aparecen plataformas que ofrecen nuevos modos de desplegar aplicaciones en la nube en contraposición al modelo antiguo de servidor físico. Surge así un nuevo paradigma de desarrollo de aplicaciones llamado *serverless* (sin servidor), que libera a los desarrolladores y empresas de las tareas de adquisición y administración de servidores para sus aplicaciones, facilitando el despliegue de las mismas.

1.2. Objetivos y fases del proyecto

La motivación principal de este trabajo final de máster es el despliegue de una aplicación en la plataforma Azure Functions aplicando la filosofía *serverless*. También se pretende hacer una comparativa de este paradigma con el modelo tradicional de ejecución en un servidor, enumerando las ventajas y desventajas de esta nueva filosofía.

Para probar estas dos filosofías, se ha desarrollado una aplicación sencilla en R que procesa los datos académicos de los alumnos de la UPCT y obtiene estadísticas generales de los mismos. Como complemento a este procesado, se ha desarrollado una aplicación web en Shiny que permite acceder a los datos procesados y representarlos. Como parte de la comparativa arriba mencionada, se ha realizado un despliegue inicial de la aplicación en un servidor propio, para después hacer una migración de la aplicación al entorno de Azure.

Para alcanzar estos objetivos, el trabajo se ha dividido en las siguientes fases:

1. Exploración inicial de los datos.
2. Desarrollo de la aplicación.
3. Implementación de la aplicación con el modelo tradicional.
4. Implementación *serverless* de la aplicación.
5. Redacción de la memoria.

CAPÍTULO 2: DATOS Y APLICACIÓN

2.1. Datos académicos de la UPCT

Como se ha mencionado en el apartado anterior, se ha desarrollado una aplicación sencilla de tratamiento de datos en R como pretexto para realizar el despliegue en Azure Functions. Con este objetivo, se ha dispuesto de un conjunto de datos proporcionado por la Oficina de Prospección y Análisis de Datos (OPADA), extraído del almacén de datos (*data warehouse*) de la UPCT.

El conjunto recoge datos académicos pertenecientes a los alumnos matriculados en las distintas titulaciones ofrecidas por la UPCT entre los cursos 2008-2009 y 2016-2017. Los datos en bruto disponen de 2.149.572 entradas distintas, que incorporan información perteneciente tanto al rendimiento de los alumnos en la UPCT como, en algunos casos, a sus calificaciones en las PAU (Pruebas de Acceso a la Universidad). A continuación, se muestra un resumen acerca del contenido del conjunto:

Entradas totales	2149572
Entradas PAU	176328
Entradas Universidad	1973244
Nº de Expedientes	10281
Nº de Expedientes PAU	7937
Nº de Titulaciones	19
Intervalo de Cursos Comprendidos	2008-09 / 2016-17

Figura 1: Tabla de estadísticas generales del conjunto de datos

Cada una de las entradas dispone de 14 valores que representan, entre otros datos, el número codificado de expediente del alumno, año de matriculación, asignatura, nota obtenida en ella, año de finalización o abandono del plan, etc. El número codificado de expediente aparece como medio para poder agrupar los datos por alumno, pero, aparte de este indicador, no existe ninguna referencia en los datos a la identidad de los alumnos. Se hizo esta selección de variables debido a que, potencialmente, permitiría realizar un estudio sobre los factores determinantes en el abandono de los alumnos.

En cuanto a la distribución de expedientes por titulaciones, encontramos las siguientes proporciones en el número de expedientes únicos:

Código	Nombre de la titulación	% de Alumnos	Nº de Alumnos
5011	GRADO EN ARQUITECTURA (BOE 11-12-2012)	6.35%	642
5021	GRADO EN INGENIERÍA DE EDIFICACIÓN (BOE 12-06-2010)	9%	910
5031	GRADO EN TURISMO (BOE 30-11-2011)	1.66%	168
5041	GRADO EN INGENIERÍA EN SISTEMAS DE TELECOMUNICACIÓN (BOE 20-04-2011)	5.09%	515
5051	GRADO EN INGENIERÍA TELEMÁTICA (BOE 20-04-2011)	5.86%	593
5061	GRADO EN INGENIERÍA ELÉCTRICA (BOE 30-11-2011)	4.89%	495
5071	GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA (BOE 30-11-2011)	7.13%	721
5081	GRADO EN INGENIERÍA MECÁNICA (BOE 30-11-2011)	11.34%	1147
5091	GRADO EN INGENIERÍA QUÍMICA INDUSTRIAL (BOE 30-11-2011)	4.11%	416
5101	GRADO EN ADMINISTRACIÓN Y DIRECCIÓN DE EMPRESAS (BOE 20-04-2011)	13.08%	1323
5111	GRADO EN INGENIERÍA EN ORGANIZACIÓN INDUSTRIAL (BOE 21-12-2012)	4.49%	454
5121	GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES (BOE 30-11-2011)	6.76%	684
5131	GRADO EN ARQUITECTURA NAVAL E INGENIERÍA DE SISTEMAS MARINOS BOE 30-11-2011	3.65%	369
5141	GRADO EN INGENIERÍA DE LA HORTOFRUTICULTURA Y JARDINERÍA (BOE 30-11-2011)	1.84%	186
5151	GRADO EN INGENIERÍA DE LAS INDUSTRIAS AGROALIMENTARIAS (BOE 30-11-2011)	1.25%	126
5161	GRADO EN INGENIERÍA CIVIL (BOE 20-04-2011)	7.25%	733
5171	GRADO EN INGENIERÍA DE RECURSOS MINERALES Y ENERGÍA (BOE 20-04-2011)	3%	303
5181	GRADO EN INGENIERÍA AGROALIMENTARIA Y DE SISTEMAS BIOLÓGICOS -BOE12-05-2015	3.16%	320
5191	GRADO EN FUNDAMENTOS DE ARQUITECTURA (BOE 17-04-2017)	1.74%	176

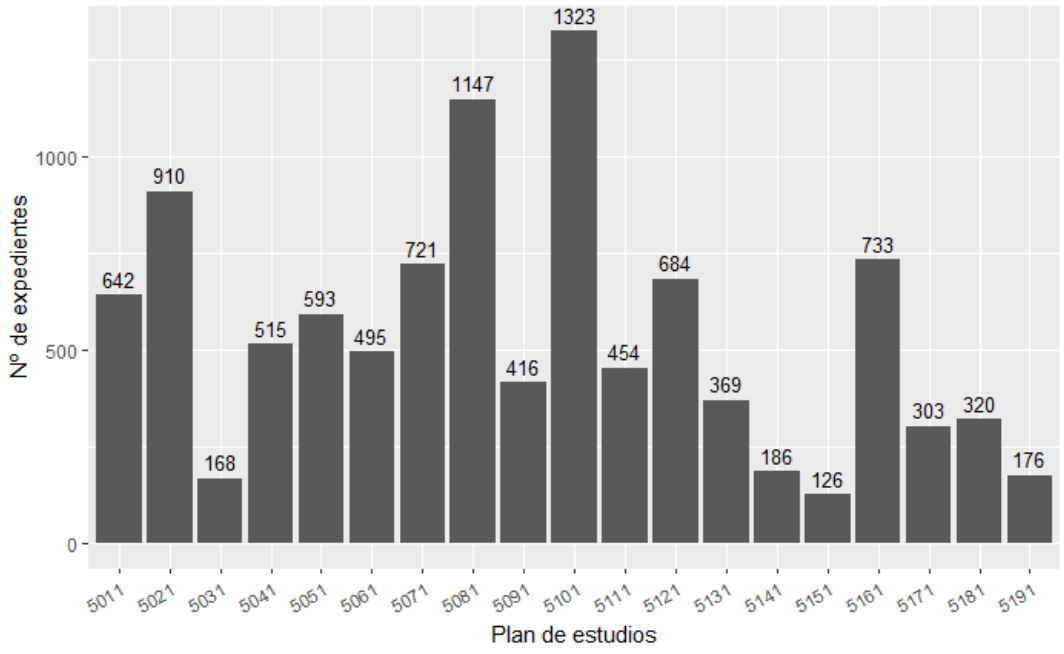

Figura 2: Representación del número de expedientes por titulación

Se observa como la selección de expedientes del conjunto se centra únicamente en las titulaciones de grado, excluyendo los datos académicos relativos a los másteres universitarios. Es relevante conocer el número de expedientes de que se dispone para cada una de las titulaciones para así poder contextualizar mejor los datos que serán representados en la aplicación.

En cuanto a los campos exactos recogidos en cada entrada, se encuentran los siguientes:

- Número de expediente: entero que representa el número expediente codificado del alumno cuyos datos están contenidos en la entrada.
- Código de titulación: entero de cuatro dígitos que representa el identificador de la titulación.
- Nombre de la titulación.
- Año de apertura del expediente.
- Año de cierre de expediente: contiene valor únicamente si el alumno ha finalizado o abandonado la titulación.
- Indicadores de abandono por cursos: se trata de tres indicadores binarios que informan de si el alumno abandonó la titulación en primero, segundo o tercero.
- Año de matriculación.
- Identificador del curso: entero que representa el curso al que pertenece la asignatura. En el caso de las entradas relativas a las calificaciones de las PAU, toma el valor "PAU".
- Identificador de la asignatura: código identificador de la asignatura en cuestión.
- Nombre de la asignatura.
- Descripción del valor: describe la magnitud almacenada en la columna valor. Puede adoptar valores tales como: Nota, Convocatorias acumuladas hasta primera vez que se presenta, Convocatorias presentadas acumuladas, Matriculas acumuladas hasta primera vez que se presenta, Matrículas transcurridas hasta que se aprueba la asignatura y representaciones de las notas en distintos formatos.
- Valor: contiene el valor de la calificación en sí mismo. En función del valor que adopta la columna descripción de valor, puede tener distintos significados y formatos.

2.2. Aplicación

Como se ha comentado en el capítulo de introducción de este trabajo, se ha desarrollado una aplicación como vehículo para el estudio de la filosofía *serverless* y las pruebas de despliegue en la plataforma Azure Functions. El núcleo del *software* desarrollado se ha llevado a cabo en R, aunque también se ha implementado una parte en PHP.

El diseño de la aplicación se ha hecho intentando dividir los distintos componentes en módulos bien definidos e independientes. Este tipo de implementación permite un gran grado de libertad a la hora de modificar o ampliar alguno de los módulos, siempre que se respete la interfaz y el formato en el intercambio de datos. También permite una distribución de la carga de procesado, ya que cada uno de los módulos puede ejecutarse en sitios distintos.

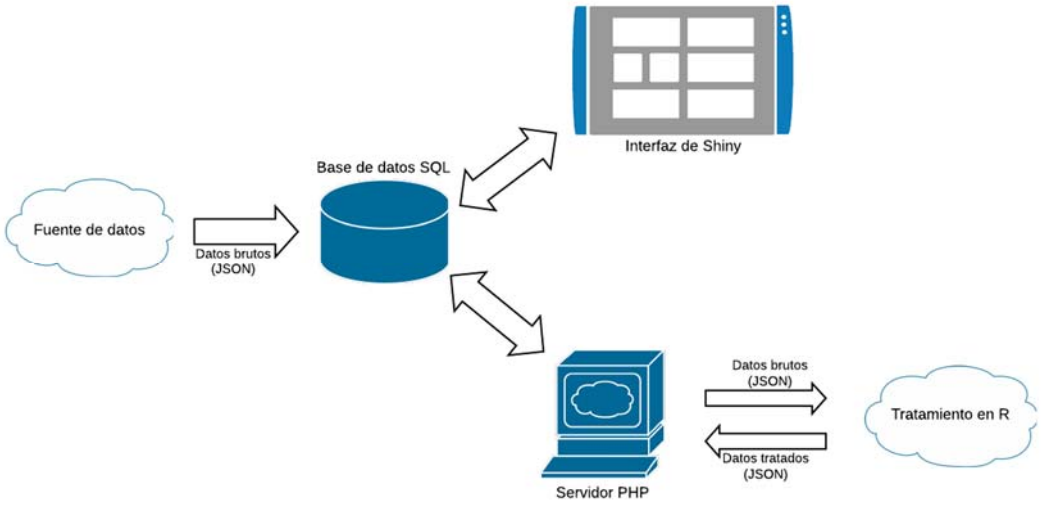

Figura 3: Esquema descriptivo de la aplicación desarrollada

Observando los componentes representados en la Figura 3, podríamos agrupar los distintos elementos en tres módulos principales:

- **Módulo de datos**: está constituido por una base de datos relacional que almacena tanto datos en bruto como datos útiles y una parte de procesado del servidor PHP que gestiona el intercambio de información con el módulo de tratamiento de datos.
- **Módulo de tratamiento de datos**: está formado por el código en R encargado de tratar los datos en brutos y devolver datos útiles.
- **Módulo de representación**: está compuesto por la aplicación Shiny que presenta los datos al usuario final.

2.2.1. Módulo de datos

El módulo de datos está formado por una base de datos relacional, para la implementación local se ha usado MySQL mientras que en Azure Functions se ha usado SQL Server, y un script en PHP que se encarga de hacer la solicitud de datos al módulo de tratamiento de datos y de actualizar la información en la base de datos.

En primer lugar, cabe comentar las suposiciones que se han hecho en cuanto al origen de los datos en bruto. Se ha considerado que los datos van a ser almacenados en formato JSON. Esto es así con el fin de aportar cierta versatilidad y escalabilidad a los datos. También se ha supuesto que los datos son almacenados directamente en la base de datos por el servicio o la aplicación de la UPCT encargada de proveer de información a la aplicación desarrollada. De esta forma, se hace transparente para el sistema el origen de los datos en bruto y se presupone que el formato es el adecuado.

En cuanto a la base de datos, se ha diseñado con una estructura muy básica, ya que la interacción del resto de módulos con ella se limita a la lectura/escritura de datos empaquetados en formato JSON. Por consiguiente, se ha optado por la creación de dos tablas con las siguientes estructuras:

- jsondata: se trata de la tabla que almacena los datos en bruto empaquetados en formato JSON. Los scripts PHP leen de ella la información original que posteriormente envían al módulo de tratamiento de datos. Dispone de dos columnas: fecha, que contiene la fecha en la que se ha introducido la información; y json, que contiene la información propiamente dicha.
- data: constituye la tabla de almacenamiento de datos tratados. Los scripts PHP escriben en ella la información que reciben de la petición al módulo de tratamiento de datos y, además, ofrece la información útil al módulo de representación. Posee una estructura de columnas similar a la tabla jsondata; fecha, que contiene la fecha en la que se ha introducido la información; y json, que contiene la información propiamente dicha.

Como ya se ha mencionado, para posibilitar la interacción entre el módulo de datos y el de tratamiento de datos ha sido necesario desarrollar dos scripts en PHP. El primero de ellos, dbconnection.php, contiene la información necesaria para establecer la conexión con la base de datos. Su contenido es el siguiente:

```php
<?php
	$servername = "localhost";
	$username = "root";
	$password = "dashpss10";
	$dbname = "dashboardtfm";
?>
```

Figura 4: Script dbconnection.php del módulo de datos

El segundo script, postrequest.php, contiene el grueso de la funcionalidad. Primero lee los datos brutos de la base de datos. Tras esto, prepara la petición y la ejecuta a través de curl. Por último, escribe la información contenida en la respuesta de la petición en la base de datos.

```php
<?php
	header("Content-Type:text/plain; charset=utf-8");

	require "dbconnection.php";
// Establecemos el tiempo máximo de ejecución en 15 mins debido a que el procesamiento de los ficheros es lento por la cantidad potencial de datos
	ini_set('max_execution_time', 900);

// Generamos la cadena de JSON que se va a enviar en la petición POST a OpenCPU
	$sql = "SELECT * FROM jsondata ORDER BY fecha LIMIT 20;";
	// Creamos la conexión con la BBDD
	$conn = new mysqli($servername, $username, $password, $dbname);
	// Comprobamos que la conexión es correcta
	if ($conn->connect_error) {
	    die("Connection failed: " . $conn->connect_error);
	}
	$jsondata = "";
	$flag = false;
	// Realizamos la petición SQL
	$result = $conn->query($sql);
	if ($result->num_rows > 0) {
	    while($row = $result->fetch_assoc()) {
	      if($flag){
	            $jsondata = $jsondata.",";
	      }
	       $jsondata = $jsondata.substr($row["json"], 1, -3);
	       $flag = true;
	    }
	}
	else {
	      echo "0 results";
	}

	$conn->close();
	$jsondata = utf8_encode($jsondata);
	$jsondata = str_replace('"', '\"', $jsondata);
	$jsondata = "[{\"datosAlumnosJSON\":\"[".$jsondata."]\"}]";
```

```php
// URL de la función a la que hacemos la llamada. NOTA: especificamos al final de la URL que los datos que debe devolver deben estar en formato JSON
      $url = "https://vic-hl.ocpu.io/dashboardTFM/R/input/json";

      $curl = curl_init();
      curl_setopt_array($curl, array(
        CURLOPT_URL => $url,
        CURLOPT_RETURNTRANSFER => 1,
        CURLOPT_MAXREDIRS => 10,
        CURLOPT_TIMEOUT => 1500,
        CURLOPT_CUSTOMREQUEST => "POST",
        CURLOPT_POST => 1,
        CURLOPT_POSTFIELDS => $jsondata,
        CURLOPT_HTTPHEADER => array(
          "Content-Type: application/json",
          "Charset:utf-8",
          "Content-Length: " . strlen($jsondata)
        ),
      ));
      $response = curl_exec($curl);
      $info = curl_getinfo($curl);
      curl_close($curl);

// INTRODUCCIÓN DE LOS DATOS RECIBIDOS EN LA BBDD
      $today = date("Y-m-d H:i", time());
      $sql = "INSERT INTO data (fecha, json)".
                " VALUES ('".$today."', '". $response ."')";
// Creamos la conexión con la BBDD
      $conn = new mysqli($servername, $username, $password, $dbname);
// Comprobamos que la conexión es correcta
      if ($conn->connect_error) {
          die("Connection failed: " . $conn->connect_error);
      }

      if ($conn->query($sql) === TRUE) {
          echo "Se han introducido los datos correctamente.";
      } else {
          echo "Error: " . $sql . "<br>" . $conn->error;
      }
      $conn->close();

    exit();
?>
```

Figura 5: Script postrequest.php del módulo de datos

Cabe mencionar que este módulo se ha implementado con estas herramientas por familiaridad con el lenguaje PHP y la sintaxis SQL, pero el diseño modular del sistema permitiría cualquier otra implementación siempre que se respetase el formato y el empaquetado JSON de los datos.

2.2.2. Módulo de tratamiento de datos

El módulo de tratamiento de datos se ha implementado en R. El código consta de un único script en R llamado input.R. Este script esencialmente se encarga de extraer los datos del empaquetado JSON, calcular estadísticas generales para cada una de las

titulaciones, extraer porcentajes de alumnos presentados/no presentados para las distintas asignaturas de cada titulación y devolver los datos tratados.

```
input <- function(datosAlumnosJSON){

  listDatos <- jsonlite::fromJSON(datosAlumnosJSON, simplifyDataFrame =
TRUE)
  listaFinal <- list()
  dfDatos <- as.data.frame(listDatos)

  titulos <- dfDatos$PLAN_ID %>% factor() %>% levels()

  # Cálculo de datos generales
  listaDatos <- list()
  i = 1
  for(titulo in titulos){
    datosTabla <- dfDatos %>% filter(PLAN_ID == titulo)
    datosGenerales <- list()
    datosGenerales[["Entradas totales"]] <- nrow(datosTabla)

    nEntPAU <- datosTabla %>% filter(ALU_ASS_CURSO_ID == "PAU") %>%
select(idExpediente)
    datosGenerales[["Entradas PAU"]] <- nrow(nEntPAU)

    nExp <- datosTabla %>% select(idExpediente)
    nExp <- nExp[!duplicated(nExp),] %>% length()
    datosGenerales[["Numero de Expedientes"]] <- nExp

    nExpPAU <- nEntPAU[!duplicated(nEntPAU),] %>% length()
    datosGenerales[["Numero de Expedientes PAU"]] <- nExpPAU

    anios <- datosTabla$ANYACA_MAT_ID %>% factor() %>% levels()
    datosGenerales[["Intervalo de cursos comprendidos"]] <-
c(paste(min(anios), max(anios), sep="/"))


    listaDatos[[titulo]] <- datosGenerales
  }
  listaFinal[["Datos Generales"]] <- listaDatos

  # Cálculo de presentados por curso
  listaDatos <- list()
  for(titulo in titulos){
    listaPorcentajes <- list()
    for(curso in c("1", "2", "3", "4")){
      datosGrafica <- dfDatos %>% filter(PLAN_ID == titulo)
      temp2 <- datosGrafica %>% filter(descValor == "CALIFICACION_DESC" &
ALU_ASS_CURSO_ID == curso) %>%
        select(idExpediente, ASS_ASS_DESC, valor) %>% na.omit()

      temp2 <- temp2 %>% mutate(PRESENTADO = ifelse(temp2$valor != "NO
PRESENTADO", "PRESENTADO", "NO PRESENTADO"))

      temp2 <- temp2[!duplicated(temp2), ]

      tabla <- as.data.frame(table(temp2 %>% select(ASS_ASS_DESC,
PRESENTADO)))
      porcentajes <- data.frame("ASS_ASS_DESC" = character(),
"PRESENTADO" = character(), "Freq" = numeric(), stringsAsFactors = FALSE)
```

```
for(asig in temp2$ASS_ASS_DESC %>% factor() %>% levels()){
        t <- tabla %>% filter(ASS_ASS_DESC == asig) %>% select(Freq)
%>% sum()
        if(length(as.numeric(tabla[tabla$ASS_ASS_DESC == asig &
tabla$PRESENTADO == "PRESENTADO", "Freq"]/t)) != 0){
          porcentajes[length(porcentajes$ASS_ASS_DESC)+1, ] <- c(asig,
"PRESENTADO", as.numeric(tabla[tabla$ASS_ASS_DESC == asig &
tabla$PRESENTADO == "PRESENTADO", "Freq"]/t))
        }
        else{
          porcentajes[length(porcentajes$ASS_ASS_DESC)+1, ] <- c(asig,
"PRESENTADO", 0)
        }
        if(length(as.numeric(tabla[tabla$ASS_ASS_DESC == asig &
tabla$PRESENTADO == "NO PRESENTADO", "Freq"]/t)) != 0){
          porcentajes[length(porcentajes$ASS_ASS_DESC)+1, ] <- c(asig,
"NO PRESENTADO", as.numeric(tabla[tabla$ASS_ASS_DESC == asig &
tabla$PRESENTADO == "NO PRESENTADO", "Freq"]/t))
        }
        else{
          porcentajes[length(porcentajes$ASS_ASS_DESC)+1, ] <- c(asig,
"NO PRESENTADO", 0)
        }
      }
      listaPorcentajes[[curso]] <- porcentajes
    }
    listaDatos[[titulo]] <- listaPorcentajes
  }
  listaFinal[["Datos Presentados"]] <- listaDatos
  return(listaFinal)
}
```

Figura 6: Script input.R del módulo de tratamiento de datos

El resultado de este tratamiento es una cadena JSON que contiene dos elementos principales: Datos Generales y Datos Presentados. El primero contendría datos generales para cada titulación, tales como número de entradas correspondientes a esa titulación, entradas con datos de PAU, número expedientes, etc. En cuanto a Datos Presentados, se compondría de los porcentajes de alumnos presentados y no presentados para cada una de las asignaturas correspondientes a las distintas titulaciones de grado ofertadas por la UPCT.

2.2.3. Módulo de representación

El módulo de representación de datos tiene como fin último extraer la información ya tratada de la base de datos y representarla en una interfaz disponible para el usuario final. Para ello, se ha hecho uso del paquete Shiny de R.

El desarrollo de este módulo se ha hecho a través de un paquete que dispone de tres scripts principales: ui.R, server.R y functions.R.

El primero de ellos, ui.R, contiene la definición de los distintos controles de que dispondrá la interfaz, así como los contendores donde se representarán los datos.

```
source("R/functions.R")

fluidPage(

  titlePanel("Panel de Control"),

  sidebarLayout(
    sidebarPanel(
      selectInput("planEstudios", h4("Selecciona un plan de estudios:"),
        choices = list("5011 - Grado en Arquitectura" = "5011",
        "5021 - Grado en Ingeniería de Edificación" = "5021",
        "5031 - Grado en Turismo" = "5031",
        "5041 - Grado en Ingeniería de Sistemas de Telecomunicaciones" =
"5041",
        "5051 - Grado en Ingeniería Telemática" = "5051",
        "5061 - Grado en Ingeniería Eléctrica" = "5061",
        "5071 - Grado en Ingeniería Electrónica Industrial y Automática"
= "5071",
        "5081 - Grado en Ingeniería Mecánica" = "5081",
        "5091 - Grado en Ingeniería Química Industrial" = "5091",
        "5101 - Grado en Administración y Dirección de Empresas" =
"5101",
        "5111 - Grado en Ingeniería de Organización Industrial" = "5111",
        "5121 - Grado en Ingeniería de Tecnologías Industriales" =
"5121",
        "5131 - Grado en Arquitectura Naval e Ingeniería de Sistemas
Marinos" = "5131",
        "5141 - Grado en Ingeniería de la Hortofruticultura y Jardinería"
= "5141",
        "5151 - Grado en Ingeniería de las Industrias Agroalimentarias" =
"5151",
        "5161 - Grado en Ingeniería Civil" = "5161",
        "5171 - Grado en Ingeniería de Recursos Minerales y Energía" =
"5171",
        "5181 - Grado en Ingeniería Agroalimentaria y de Sistemas
Biológicos" = "5181",
        "5191 - Grado en Fundamentos de Arquitectura" = "5191"),
       selected = 5041),
      h4("Opciones de visualización"),
      fluidRow(
        column(6, checkboxInput("showTablaDatosGenerales", "Resumen de
datos", value = TRUE)),
        column(6, checkboxInput("showGrafica", "Gráfica", value = FALSE))
      ),
      conditionalPanel("input.showGrafica",
        selectInput("cursoID", h4("Curso"),
         choices = list("1er Curso" = "1",
                        "2do Curso" = "2",
                        "3er Curso" = "3",
                        "4to Curso" = "4")))
    ),
    mainPanel(
      conditionalPanel("input.showTablaDatosGenerales",
        tableOutput("tabla")),
      conditionalPanel("input.showGrafica",
        plotOutput("grafica", height = "600px"))
    )
  )
)
```

Figura 7: Script ui.R de módulo de representación

Como se aprecia en la Figura 7, los elementos de control principales son un selector, que permite elegir la titulación, y dos *checkbox* que permiten habilitar o deshabilitar alguna de las dos representaciones. Aparte, existe un segundo selector, cuya visibilidad viene condicionada al hecho de si está habilitada la vista de gráfica o no, que permite seleccionar el curso de cuyas asignaturas se van a representar las tasas de presentados/no presentados. En cuanto a elementos de representación, existe una tabla que contendrá los datos generales y una gráfica en la que se representaran los porcentajes.

Figura 8: Controles de representación de la interfaz

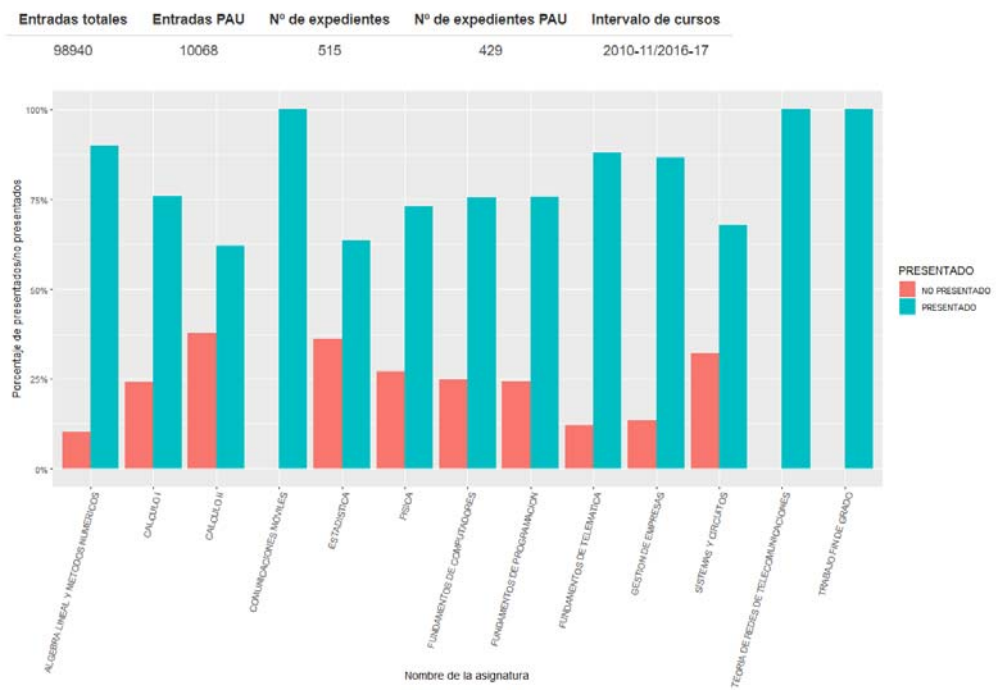

Figura 9: Elementos de representación de la interfaz

El segundo script, server.R, se encarga de definir el comportamiento de los elementos definidos en ui.R. En esencia, su propósito es, leer los estados de los controles, hacer una llamada a las funciones ubicadas en functions.R, y actualizar los datos de los contenedores con las respuestas de dichas llamadas.

```
function(input, output) {
  output$tabla <- renderTable({

as.data.frame(jsonlite::fromJSON(datos_generales(input$planEstudios)))
%>% rename("Entradas totales" = Entradas.totales,
         "Entradas PAU" = Entradas.PAU,
         "Nº de expedientes" = Nº.de.Expedientes,
         "Nº de expedientes PAU" = Nº.de.Expedientes.PAU,
         "Intervalo de cursos" = Intervalo.de.cursos.comprendidos)
    }, align = 'c')
  output$grafica <- renderPlot({
    porcentajes <-
as.data.frame(jsonlite::fromJSON(datos_presentados(input$planEstudios,
input$cursoID)))
    ggplot(data = porcentajes, aes(x = ASS_ASS_DESC)) +
      geom_col(aes(y = as.numeric(Freq), fill = PRESENTADO), position =
position_dodge()) +
      scale_y_continuous(labels=percent) +
      labs(x = "Nombre de la asignatura", y = "Porcentaje de
presentados/no presentados") +
      theme(axis.text.x = element_text(angle = 75, hjust = 1))
  })
}
```

Figura 10: Script server.R del módulo de representación

El tercer script, functions.R, se encarga de solicitar la información a la base de datos y de seleccionar un elemento u otro. Se compone de dos funciones, datos_generales y datos_presentados. Cada una de estas funciones realiza una petición SQL a la base de datos y devuelve la información indicada por su propio nombre.

```
my_db <- dbPool(
  RMySQL::MySQL(),
  dbname = "dashboardtfm",
  host = "127.0.0.1",
  username = "root",
  password = "dashpss10"
)

datos_generales <- function(planEstudios){
  dfDatos <- dbGetQuery(my_db, "SELECT json FROM data ORDER BY
fecha DESC LIMIT 1;")
  jsonTmp <- jsonlite::fromJSON(dfDatos$json)
  datos <- jsonTmp$`Datos Generales`
  json <- jsonlite::toJSON(datos[[planEstudios]])
  return(json)
}

datos_presentados <- function(planEstudios, curso){
  dfDatos <- dbGetQuery(my_db, "SELECT json FROM data ORDER BY
fecha DESC LIMIT 1;")
  jsonTmp <- jsonlite::fromJSON(dfDatos$json)
  datos <- jsonTmp$`Datos Presentados`
  json <- jsonlite::toJSON(datos[[planEstudios]][[curso]])
  return(json)
}
```

Figura 11: Script functions.R del módulo de representación

La apariencia de la interfaz al completo puede observarse en la siguiente figura:

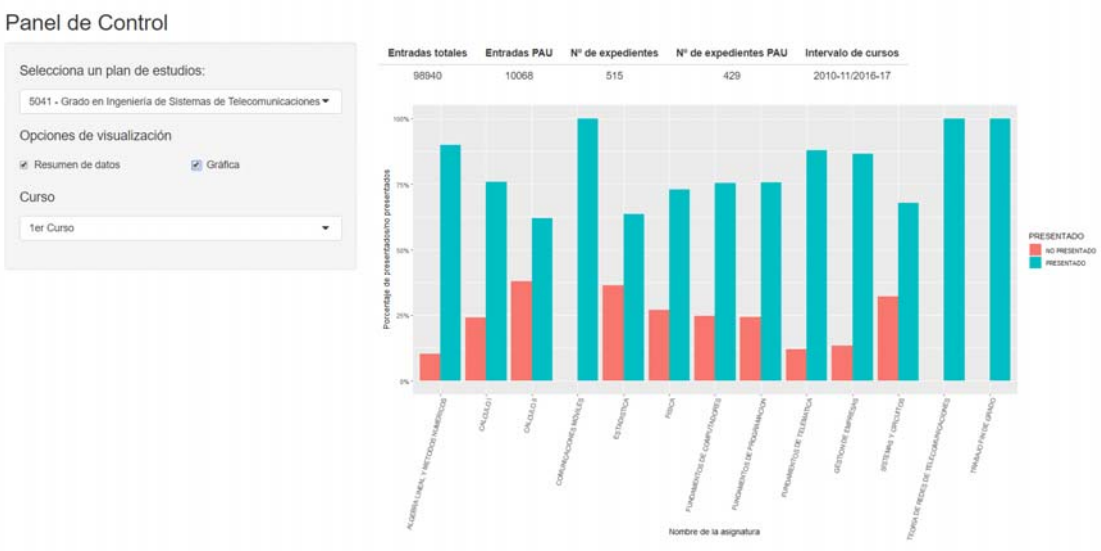

Figura 12: Interfaz completa

CAPÍTULO 3: FILOSOFÍA “SERVERLESS”

La filosofía *serverless computing*, o computación sin servidor, hace referencia a aquellos modelos de ejecución surgidos como extensión del *cloud computing* en los que se releva a los desarrolladores de la tarea de aprovisionar, escalar y administrar los servidores. Con este paradigma, por tanto, se desplaza la carga al desarrollo propio de la aplicación, minimizando las tareas de administración y mantenimiento, y agilizando el proceso de despliegue inicial.

3.1. “On-premises computing” vs “Cloud computing”

Tradicionalmente, el modelo de computación *on-premises* ha sido el vigente en el desarrollo de cualquier aplicación en internet. Las limitaciones tecnológicas obligaban a las empresas a abastecerse de servidores en los que poder alojar sus aplicaciones. Esto conllevaba una gran inversión inicial, fruto de la compra del equipamiento y el despliegue. Además, el mantenimiento y gestión posterior también podía suponer una inversión de recursos importante. Este modelo se caracterizaba por una fuerte rigidez a la hora de escalar los recursos, ya que suponía la compra de nuevo equipamiento, por lo que ciclos de actividad irregulares en la aplicación podían suponer saturación en la infraestructura o infrautilización. En resumen, las características principales del modelo *on-premises* serían:

- **Infraestructura propia**: la infraestructura y los recursos necesarios para la aplicación son adquiridos por la empresa.
- **Mantenimiento y actualización**: la responsabilidad y el coste del mantenimiento y la actualización de los recursos recae en la empresa.
- **Baja escalabilidad**: la ampliación de la estructura pasa por la adquisición de más recursos, por lo que el proceso es lento y difícil.
- **Baja utilización**: dado que en muchos casos es necesario sobredimensionar los equipos para soportar picos de carga, cuando hay periodos de menos demanda, la utilización es menor y por tanto se desperdician.

En contraposición al modelo tradicional encontramos los modelos basados en *cloud computing*. El NIST (*National Institute of Standars and Technology*) define *cloud computing* como "un modelo que permite el acceso ubicuo, adecuado y bajo demanda a un conjunto compartido de recursos informáticos configurables (por ejemplo, redes, servidores, almacenamiento, aplicaciones y servicios) que pueden aprovisionarse y liberarse con un mínimo esfuerzo de gestión o interacción con el proveedor de servicios" (Mell & Grance, 2011).

En esta definición ya encontramos algunas de las principales diferencias que existen entre ambos modelos. La primera de ellas se localiza en el tercer adjetivo con que el NIST define el acceso en *cloud computing*. El acceso bajo demanda nos indica que el cliente tiene la capacidad de reservar recursos de forma unilateral y sin prácticamente interacción con el proveedor en base a sus necesidades. La segunda, se encuentra en la referencia a "recursos compartidos". Esto alude al hecho de que los recursos pertenecen al proveedor de la infraestructura, pero son compartidos por los clientes del proveedor. La tercera gran diferencia que encontramos en esta definición tiene que ver con el aprovisionamiento de recursos. Mientras que en el modelo *on-premises* encontrábamos que el aprovisionamiento de nuevos recursos conllevaba la adquisición de estos y su posterior configuración e integración en el sistema ya existente, observamos que, en los modelos basados en *cloud computing*, los recursos son asignados y liberados de forma dinámica y en base a la demanda de que disponga la aplicación.

Además de las características que se deducen de la propia definición, el NIST define de forma explícita cinco propiedades esenciales que debe poseer un sistema para que se considere una implementación del modelo *cloud*:

- **Autoservicio bajo demanda**: el cliente debe tener la posibilidad de proveerse de recursos según su necesidad, unilateralmente y sin interacción humana por parte del proveedor.
- **Acceso a través de la red**: debe existir acceso a los recursos a través de la red, mediante el uso de mecanismos estándar y distintos clientes.
- **Compartición de recursos**: los recursos del proveedor son compartidos por los distintos clientes, siendo estos asignados o liberados de forma dinámica en base a las necesidades de cada uno de ellos.
- **Elasticidad**: las capacidades deben poder aprovisionarse o liberarse rápidamente, a veces de forma automática, para hacer frente a las variaciones de demanda del cliente.
- **Servicio medido**: los sistemas *cloud* deben controlar y optimizar los recursos de forma automática. El uso de los recursos por parte del cliente debe ser monitorizado, controlado y estar disponible para su consulta.

A pesar de que el modelo de *cloud computing* resuelve algunas de las principales carencias del modelo *on-premises*, también es importante mencionar que el modelo tradicional dispone de algunas fortalezas frente a la nube. La primera de ellas es la capacidad de configuración y personalización. El disponer de la propiedad de la infraestructura, permite tener control absoluto sobre la configuración del sistema. Esto, aunque los proveedores de *cloud* ofrezcan más o menos flexibilidad en sus servicios, nunca será alcanzable por una infraestructura contratada. En segundo lugar, está relacionada con la seguridad. Dado que los clientes no tienen acceso físico a los recursos que están contratando, tienen que depender del proveedor para la gestión de la seguridad de su aplicación. Esto, sumado al hecho de que los recursos suelen ser compartidos con otros clientes, puede suponer un déficit de seguridad frente al control absoluto de la seguridad de que dispone el modelo *on-premises*. Por último, existe un escenario en el que disponer de la propiedad de la infraestructura puede ser rentable, frente a la contratación de la misma. A pesar de que *cloud computing* ofrece una menor inversión inicial y un pago por uso de los recursos consumidos, en aquellos casos en que se adquiere la infraestructura, se optimiza adecuadamente y se hace un uso de ella prolongado en el tiempo, la rentabilidad del modelo tradicional puede ser mejor. Esto es así debido a que, a pesar de la fuerte inversión inicial que requiere la compra del equipamiento, el gasto mensual de mantenimiento de la infraestructura en general será menor que el pago por

esa misma infraestructura en un servicio *cloud*. Por tanto, si se el uso del equipo se prolonga lo suficiente en el tiempo como para rentabilizar la inversión inicial, a la larga el modelo *on-premises* terminaría siendo más barato.

3.2. Modelos de negocio

Cloud computing utiliza un modelo de negocio basado en servicios. Bajo las siglas XaaS (*Everything as a Service*), los proveedores ofrecen su infraestructura a distintos niveles en función del modelo escogido. En base al nivel de abstracción del servicio que se ofrece, surgen tres categorías distintas:

- ***Infrastructure as a Service* (IaaS)**: se trata del modelo con el nivel de abstracción más bajo. En él, los proveedores ofrecen recursos de procesamiento, almacenamiento, red, etc. en los que el cliente es capaz de desplegar y ejecutar el *software* que quiere. El cliente no tiene control sobre la infraestructura subyacente que soporta el *software* que despliega o ejecuta, aunque sí lo tiene sobre sistemas operativos, recursos de almacenamiento y control limitado sobre los recursos de red.
- ***Platform as a Service* (PaaS)**: en este modelo, los clientes despliegan sus propias aplicaciones o *software* a través del uso de lenguajes, librerías, herramientas o servicios suportados por la plataforma del proveedor. De esta manera, el cliente no dispone ningún tipo de control sobre la infraestructura en la que se ejecuta, pero sí sobre el código que despliega.
- ***Software as a Service* (SaaS)**: es el modelo que dispone de un mayor nivel de abstracción. El proveedor ofrece la capacidad de usar un cierto *software* ejecutado sobre su nube. El cliente no dispone de control sobre nada más allá de las configuraciones inherentes al propio *software* que está usando.

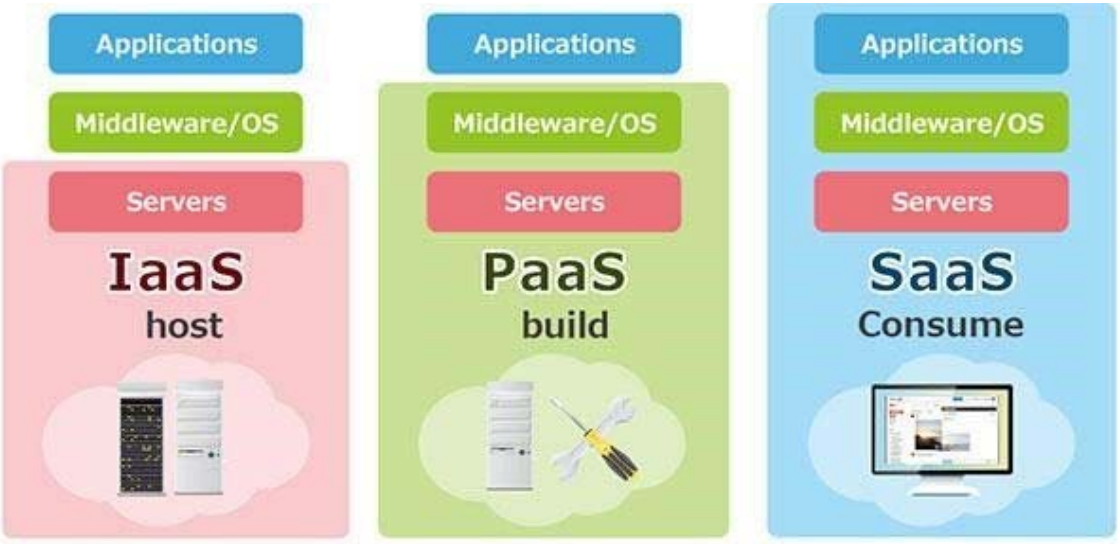

Figura 13: Esquema de modelos de negocio de cloud computing (Bitec Business Intelligence Technology S.L)

3.3. Modelos de despliegue

En cuanto a los modelos de despliegue, existen multitud de formas de implementar una nube, aunque el NIST reconoce los siguientes:

- **Nube privada**: la infraestructura de la nube es desplegada para uso exclusivo de una única organización. La propiedad, gestión y operación de la infraestructura puede ser de un proveedor externo, de la propia organización o una combinación de ellos.
- **Nube de comunidad**: en este caso, la nube es desplegada para uso de una comunidad de organizaciones que comparten intereses y puntos comunes entre ellas. Al igual que la nube privada, la propiedad, gestión y operación de la infraestructura puede ser de un proveedor externo, de la propia organización o una combinación de ellos.
- **Nube pública**: en las nubes públicas, la infraestructura es desplegada para el uso abierto del público general. En estos casos, la propiedad, gestión y operación puede pertenecer a empresas, entidades académicas y gubernamentales, o algún proveedor externo.
- **Nube híbrida**: representa aquellos despliegues en que se hace uniendo dos o más de los modelos anteriores.

3.4. Principales proveedores

Actualmente, existen numerosos proveedores de servicios de *cloud computing*. Algunos de los más importantes son:

3.4.1. Amazon Web Services (AWS)

Se trata de un conjunto de servicios de *cloud computing* ofrecidos por Amazon.com. Actualmente, AWS cuenta, además de los servicios que ofrece a usuarios finales, con dos modelos de servicios para desarrolladores distintos. El primero es Amazon Elastic Compute Cloud (EC2). Fue el primer servicio *cloud* lanzado por Amazon. Este servicio implementa el modelo IaaS ofreciendo distintas unidades de procesamiento, mediante el uso tecnologías de virtualización, permitiendo la ejecución de todo tipo de sistemas operativos y de software.

Figura 14: Logo de Amazon Web Services

En 2014, apareció un nuevo servicio ofrecido por Amazon que añadía una capa más de abstracción, situándose en el modelo PaaS, Amazon Lambda. Este servicio ofrece la posibilidad de ejecutar código de lenguajes como Node.js, Python, Java, C#, entre otros.

3.4.2. Microsoft Azure

Se trata del conjunto de servicios en la nube que ofrece la empresa Microsoft. Al igual que Amazon Web Services, en Microsoft Azure se encuentran productos pertenecientes tanto al modelo IaaS como al modelo PaaS.

Figura 15: Logo de Microsoft Azure

Entre los productos que ofrecen se encuentran: máquinas virtuales de todo tipo, funciones para ejecución de distintos lenguajes de programación, espacio de almacenamiento, bases de datos como infraestructura o como servicio, etc.

3.4.3. Google Cloud Platform

Google Cloud Platform engloba todos los servicios en la nube ofrecidos por la empresa Google. Al igual que los ejemplos anteriores, Google Cloud Platform se compone de servicios pertenecientes tanto al modelo IaaS como PaaS.

Figura 16: Logo de Google Cloud Platform

Su catálogo se divide en tres servicios principales: Google Compute Engine, para la ejecución de sistemas operativos sobre máquinas virtuales; Google Kubernetes Engine, para la ejecución de aplicaciones en contenedores basados en Kubernetes; y Google App Engine, para ejecución de código en su plataforma.

CAPÍTULO 4: IMPLEMENTACIÓN TRADICIONAL

Tras el desarrollo de la aplicación, y como paso previo al despliegue de esta en Azure, se llevó a cabo un despliegue siguiendo el modelo tradicional. El primer paso de este proceso hubiese sido realizar una predicción acerca de la capacidad que iba a ser necesaria para ejecutar la aplicación, seguido de un diseño del sistema y finalizado con la compra de los componentes y su configuración. En cambio, para el desarrollo de este trabajo, en el que se pretende hacer un ejercicio teórico acerca de la implementación y, además, se trata de una aplicación sencilla, el despliegue se ha ceñido a la configuración del servidor en un ordenador personal.

4.1. Módulo de datos

Para el proceso de implementación del módulo de datos, se ha escogido un paquete de software que incluye un sistema de base de datos MySQL y un servidor web, llamado XAMPP.

4.1.1. XAMPP

XAMPP es un paquete de software libre formado por Apache, como servidor web, MariaDB, como sistema de gestión de bases de datos MySQL, e intérpretes de PHP y PERL (X, Apache, MariaDB, PHP, Perl; XAMPP). Se ha elegido este paquete concreto por diversos motivos. El primero de ellos es el hecho de que se trata de *software* libre y

por tanto no requiere de pago de ninguna licencia. El segundo motivo es que contiene las herramientas necesarias para nuestra aplicación: un servidor web, una base de datos y un intérprete de PHP. Por último, ha sido determinante el hecho de que el proceso de instalación y configuración es sencillo y rápido.

Como se ha comentado, el proceso de instalación del paquete es simple. El primer paso es descargar el instalador del paquete de la página oficial de XAMPP (https://www.apachefriends.org). Después, ejecutamos el instalador. Durante el proceso, el asistente solicita que se seleccione los componentes a instalar. Estrictamente hablando, para el uso que se le da en este trabajo, solo sería necesario seleccionar Apache, MySQL y PHP, aunque, para facilitar el proceso de configuración inicial, también se seleccionó el complemento PHPMyAdmin. Tras el proceso de instalación, ya están disponibles los componentes del servidor web y se puede acceder a ellos a través del panel de XAMPP representado en la siguiente figura:

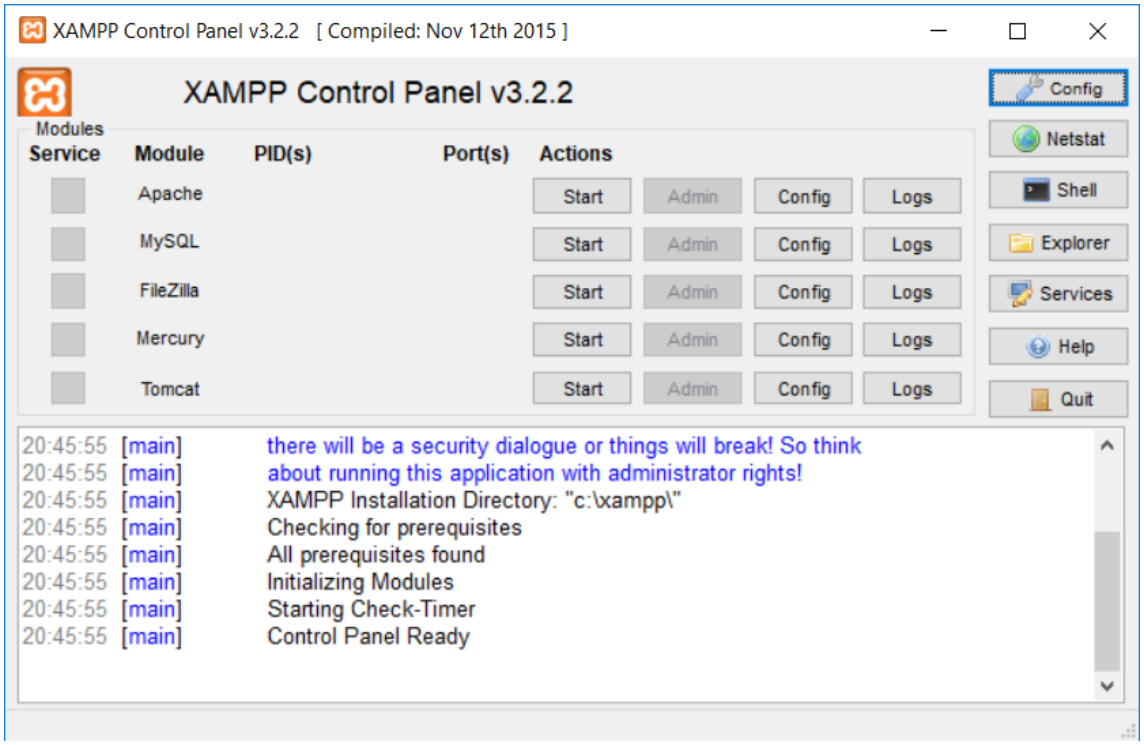

Figura 17: Panel de control de XAMPP

Desde este panel se puede iniciar y detener los distintos módulos del paquete, así como acceder a sus configuraciones.

Tras iniciar ambos módulos, Apache y MySQL, se procede a realizar las configuraciones iniciales.

4.1.2. Servidor MySQL

En primer lugar, se generaron las dos tablas que van a ser necesarias para el correcto funcionamiento del sistema. Para hacerlo, se ejecutaron las siguientes líneas a través de línea de comandos en el equipo:

```
mysql -u root -p
********

CREATE DATABASE dashboardtfm;

USE dashboardtfm;

CREATE TABLE jsondata (fecha DATETIME NOT NULL
                                    DEFAULT CURRENT_TIMESTAMP,
                       json LONGTEXT);

CREATE TABLE data (fecha DATETIME NOT NULL
                                  DEFAULT CURRENT_TIMESTAMP,
                   json LONGTEXT);

INSERT INTO jsondata (fecha, json)
     VALUES ('2018-10-20 12:00', '{"Datos":{...}}');
```

Figura 18: Comandos para la creación y población inicial de la base de datos

Con la última sentencia mostrada en la figura anterior, se introdujeron los primeros datos sin tratar requeridos para poder realizar pruebas de funcionamiento. Con esto, ya tendríamos todo lo necesario por la parte de la base de datos MySQL.

4.1.3. Servidor Apache

En cuanto al código PHP, el único requisito es que el servidor Apache esté encendido y que el script se aloje en la carpeta htdocs del directorio de instalación de XAMPP. Para ejecutar el script no haría falta más que introducir la URL http://127.0.0.1/lib/postrequest.php en un navegador o a través de curl en línea de comandos. Esta última opción es especialmente interesante ya que permitiría, a través del Programador de tareas de Windows o cron en Linux, programar una tarea para que, de forma periódica, actualizase los datos útiles de la aplicación.

```
curl http://127.0.0.1/lib/postrequest.php
```

Figura 19: Llamada a script postrequest.php de actualización de datos

4.2. Módulo de tratamiento de datos

A la hora de llevar a cabo la implementación del módulo de tratamiento de datos se ha optado por el uso de OpenCPU.

4.2.1. OpenCPU

OpenCPU es un *framework* que permite la ejecución de código R para tratamiento de datos y computación científica. El principal atractivo de este *framework* es que proporciona una API HTTP que habilita el intercambio de información con el código de R. OpenCPU proporciona las herramientas necesarias para desplegar todo tipo de servicios de tratamiento de datos explotando la potencia que ofrece R y posibilitando el acceso desde cualquier otro servicio o aplicación.

Para adecuar el código desarrollado inicialmente a OpenCPU, ha hecho falta aplicar algunas directrices de formato y estructura. La primera de ellas ha sido el encapsulado en un paquete de R, ya que esta es la estructura básica con la que trabaja el *framework*. El código de tratamiento de los datos ha tenido que ubicarse en una función llamada *input* que posteriormente es exportada para que sea accesible al instalar el paquete. Por último, se han rellenado los ficheros DESCRIPTION y NAMESPACE.

```
Package: dashboardTFM
Type: Package
Title: Procesado de datos para DashboardTFM
Version: 1.0.0
Author: Víctor Huéscar López
Maintainer: Víctor Huéscar López <victor.huescar@gmail.com>
Description: Herramienta para procesamiento de datos de
     rendimiento del alumnado
License:
Encoding: UTF-8
LazyData: true
Imports:
    dplyr,
    jsonlite
```

Figura 20: Fichero DESCRIPTION del paquete de procesado de datos

El fichero DESCRIPTION contiene los metadatos del paquete, así como una breve descripción de su utilidad y sus dependencias.

```
export(input)
import("dplyr")
import("jsonlite")
```

Figura 21: Fichero NAMESPACE del paquete de procesado de datos

En el fichero NAMESPACE se definen las dependencias requeridas por el paquete para su funcionamiento, así como las funciones que van a estar disponibles para uso público. Tras esto, solo es necesario compilar el paquete para que pueda estar disponible.

Por último, para que la aplicación permanezca abierta y a la espera de peticiones HTTP, es necesario ejecutarla desde una consola de R a través del paquete opencpu. Para ello, primero debemos instalar el paquete. Después, iniciamos la aplicación con la siguiente sucesión de comandos:

```
library("opencpu")
library("dashboardTFM")

ocpu_start_app("dashboardTFM")
```

Figura 22: Secuencia de comandos para iniciar la app en OpenCPU

Finalmente, en el script de PHP se usaría la url http://127.0.0.1:5656/ocpu/library/dashboardTFM/R/input para realizar las peticiones al script.

Se ha optado por hacer uso de este método para el servidor OpenCPU por su sencillez y porque permite experimentar las bondades que ofrece la librería. Para un entorno de producción más complejo sería necesario ejecutar OpenCPU en un servidor a través de las imágenes de Docker que la misma plataforma ofrece.

4.3. Módulo de representación

A la hora de llevar a cabo la implementación del módulo de representación de datos se ha optado por Shiny.

4.3.1 Shiny

Shiny es un paquete libre de R que permite construir aplicaciones web de forma sencilla e intuitiva a partir de R. Al tratarse de un paquete desarrollado por RStudio, Shiny está perfectamente integrado con el entorno de programación, por lo que su uso es muy fácil.

El primer paso para su uso es la instalación del paquete a través de la consola de R. Una vez instalado, al intentar crear un nuevo proyecto en RStudio aparecerá la opción

de generar un paquete de *Shiny Web Application* que nos generará los ficheros básicos necesarios para una aplicación Shiny.

Una vez está creado el proyecto, se incorporan los ficheros que contienen la estructura y el comportamiento de la aplicación al directorio. Por último, solo quedaría iniciar la aplicación a través de RStudio para que permanezca a la espera de cualquier petición de servicio.

```
library("shiny")
runApp()
```

Figura 23: Comandos para iniciar la aplicación Shiny

4.4. Conclusiones de la implementación

Cabe mencionar que, aunque no se ha dicho de forma explícita en los apartados de los módulos de tratamiento y representación de datos, es necesario tener instalado R y RStudio en el equipo que ejecuta las aplicaciones como paso previo a su despliegue.

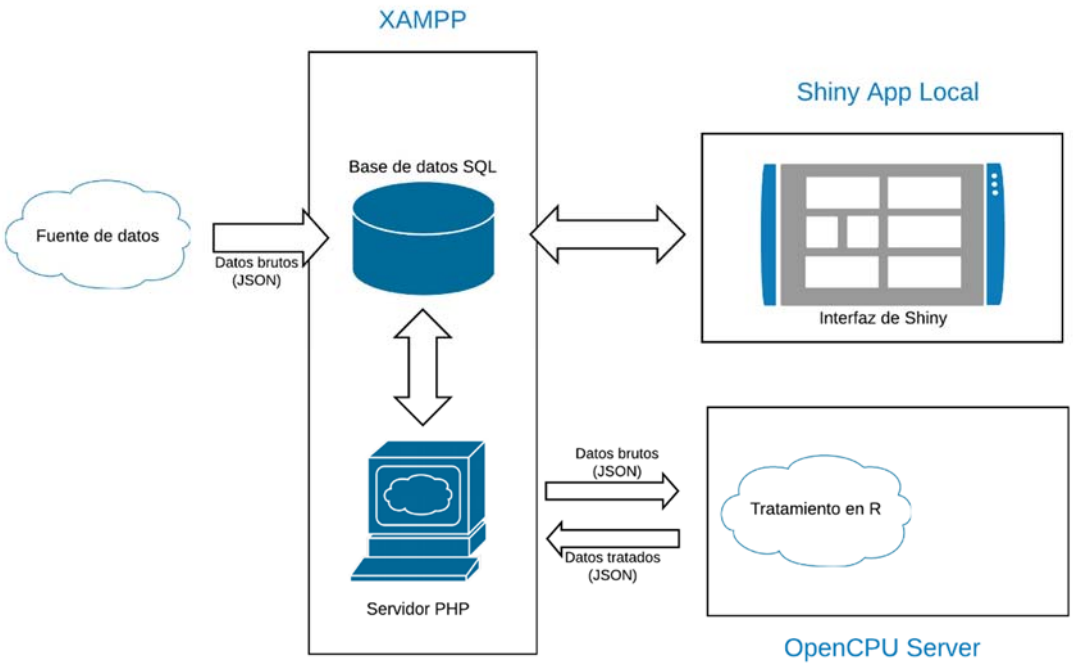

Figura 24: Esquema general del sistema implementado con el modelo tradicional

En este caso, debido al paquete escogido para implementar el módulo de datos y a que R, que es la base de los módulos de tratamiento y representación de datos, dispone

de un gestor de paquetes, el despliegue y las configuraciones iniciales necesarias han sido sencillas. En cuanto a las tareas de gestión requeridas por este sistema, se restringirían a mantener los paquetes y las herramientas que se emplean actualizadas y supervisar que los servicios que gestionan las peticiones permanecen activos.

CAPÍTULO 5: IMPLEMENTACIÓN “SERVERLESS”

La implementación siguiendo el modelo *serverless*, ha desembocado en el uso de tres servicios *cloud* independientes, los cuales están ligados a cada uno de los módulos en que se ha dividido el sistema. Aprovechando el esquema de la implementación tradicional, ubicado al final del capítulo anterior, y el paralelismo entre ellas, se ha generado un nuevo esquema que refleja esta nueva disposición:

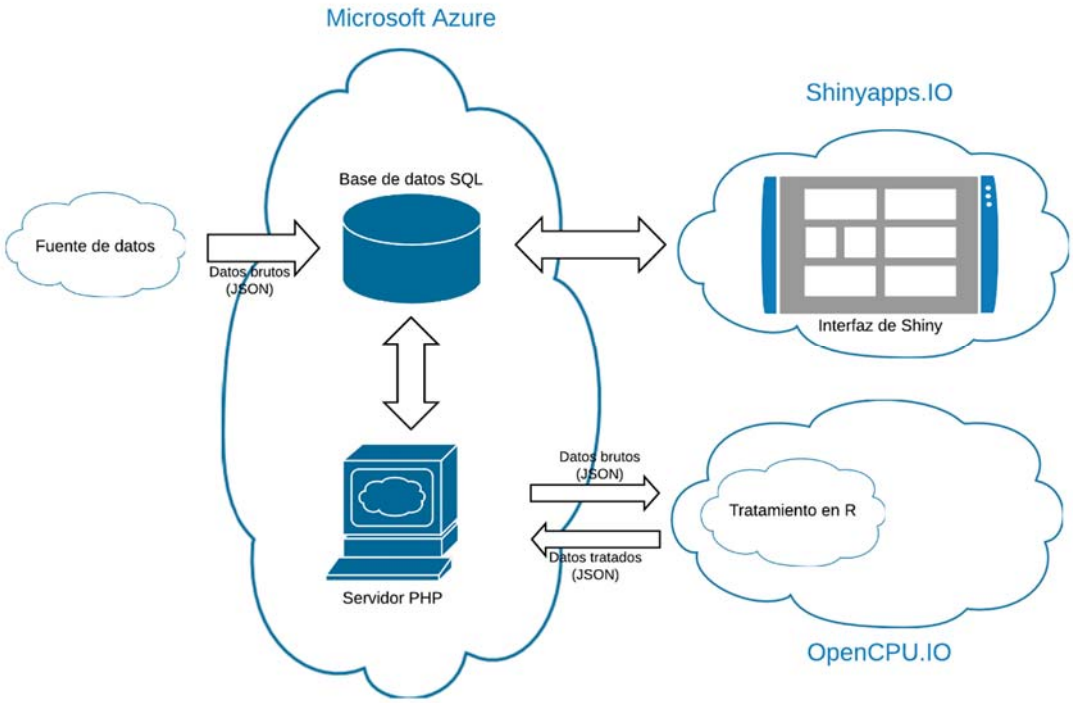

Figura 25:Esquema general del sistema implementado con el modelo "serverless" usando OpenCPU.io

5.1. Módulo de datos

5.1.1. Microsoft Azure

El primer paso a la hora de realizar el despliegue usando tecnologías *cloud* consistió en hacer la migración del módulo de datos en local a Azure. Para ello, fue necesario crear una cuenta en la web de Microsoft Azure. Dado que, para este trabajo, el número de recursos requeridos iba a ser pequeño, bastó con hacer uso del crédito disponible durante la prueba de treinta días que ofrece la plataforma para poder llevar a cabo el despliegue y las pruebas pertinentes.

Tras el proceso de registro, se hizo una predicción acerca de los servicios que iban a ser necesarios dentro del amplio catálogo de la plataforma. En primer lugar, haría falta un servicio de aplicación (*App* Service) que permitiese la ejecución del código PHP y que, por tanto, cumpliese con las funciones que llevaba a cabo el servidor Apache en la implementación tradicional. Tras esto, se requería una base de datos que albergase toda la información, tanto procesada como sin procesar, y que realizase las funciones de MariaDB y MySQL. Con este fin, se escogió el servicio de *SQL Database*. Tras escoger los dos servicios principales, se procedió a llevar a cabo la configuración.

Siguiendo algunos de los tutoriales que la plataforma pone a disposición de los usuarios, se llevaron a cabo los siguientes pasos para poner en marcha el sistema. La primera parte de la configuración se realizó a través de Azure Cloud Shell. Se trata de un *shell* interactivo, accesible desde el propio portal de Azure, que permite crear y gestionar recursos de Azure a través de comandos. Haciendo uso de esta herramienta, se comenzó creando un usuario de implementación que posteriormente se usaría para subir los scripts a Azure:

```
Initializing your account for Cloud Shell...\
Requesting a Cloud Shell.Succeeded.
Connecting terminal...

Welcome to Azure Cloud Shell

Type "az" to use Azure CLI 2.0
Type "help" to learn about Cloud Shell

victor_huescar@Azure:~$ az webapp deployment user set --user-name vic-hl --password [redacted]
{
  "id": null,
  "kind": null,
  "name": "web",
  "publishingPassword": null,
  "publishingPasswordHash": null,
  "publishingPasswordHashSalt": null,
  "publishingUserName": "vic-hl",
  "scmUri": null,
  "type": "Microsoft.Web/publishingUsers/web"
}
```

Figura 26: Creación de usuario de implementación en Azure

Tras esto, se genera un grupo de recursos que contendría los servicios asignados a nuestra aplicación:

```
victor_huescar@Azure:~$ az group create --name dashboardTFM --location "West Europe"
{
  "id": "/subscriptions/87ab25e4-1a15-46a5-9442-1d6ea4369c42/resourceGroups/dashboardTFM",
  "location": "westeurope",
  "managedBy": null,
  "name": "dashboardTFM",
  "properties": {
    "provisioningState": "Succeeded"
  },
  "tags": null
}
```

Figura 27: Creación de un grupo de recursos en Azure

Después, se genera un plan de servicio y se asigna el grupo de recursos anteriormente creado a él:

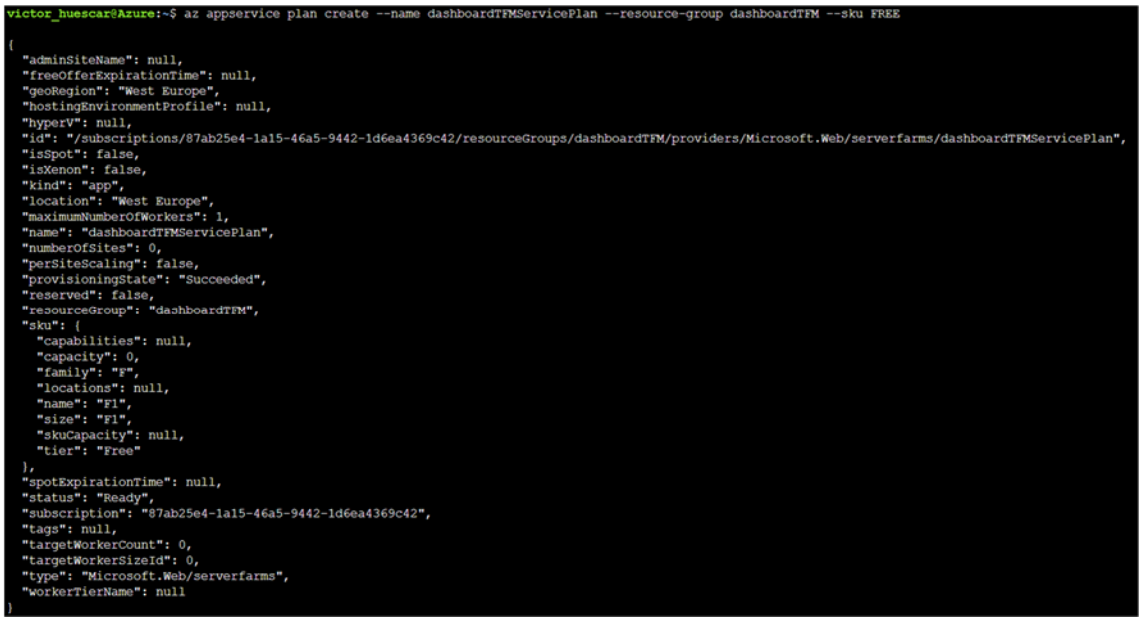

Figura 28: Creación de plan de servicio en Azure

Por último, se crea el *App Service* con el siguiente comando:

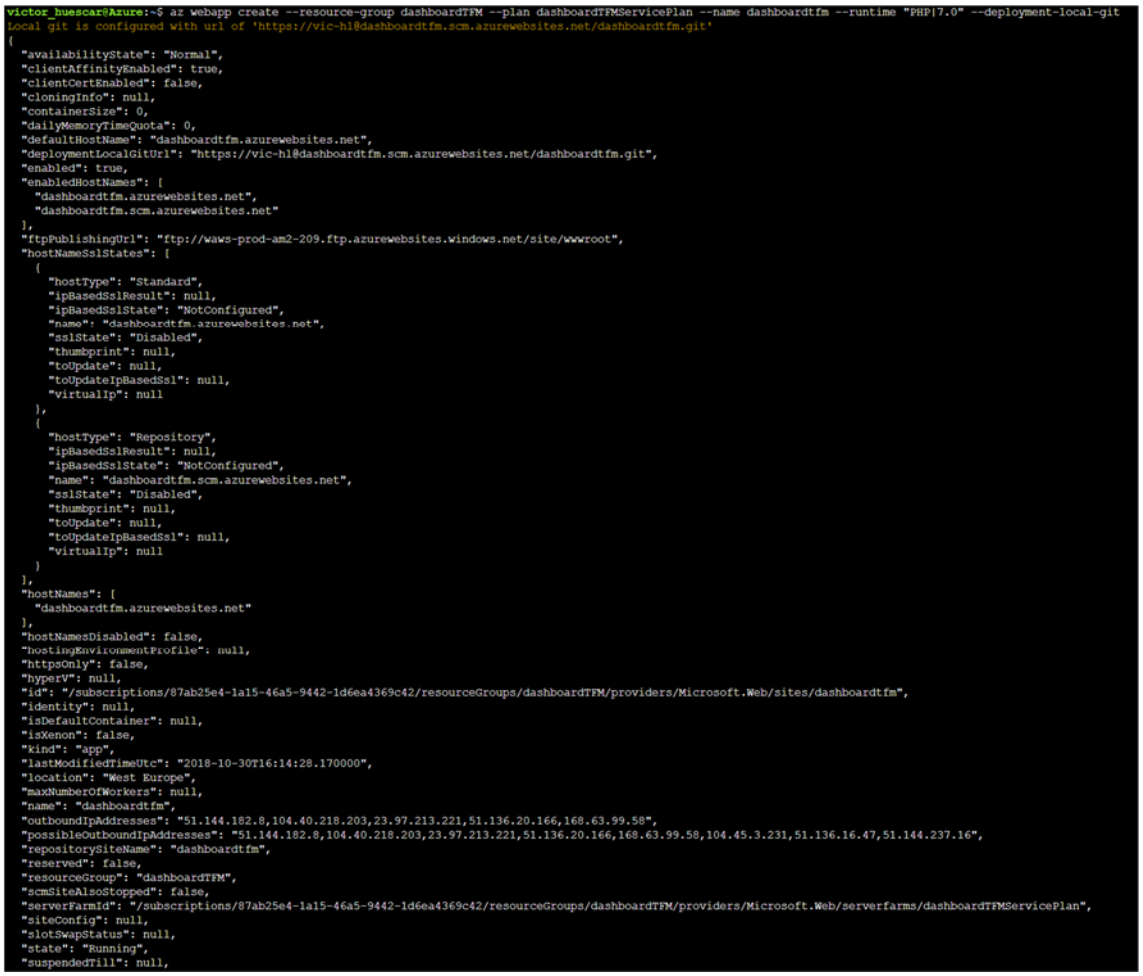

Figura 29: Creación del App Service en Azure

Tras esta secuencia de comandos, ya estaba disponible en el portal de Azure de *App Service* preparado para subir el código. El acceso a la aplicación, tal y como se indica en el nombre devuelto en el JSON tras el último comando, se hace a través de la url http://dashboardtfm.azurewebsites.net/. Además, también viene reflejado el acceso a través de FTP/FTPS al directorio principal de la web (ftps://waws-prod-am2-209.ftp.azurewebsites.windows.net). A través de esa dirección y las credenciales generadas en el primer comando del proceso, posteriormente se hizo la carga de los scripts PHP en la aplicación.

Para la creación del recurso de bases de datos se hizo uso de la interfaz de Azure a través del siguiente proceso. Primero, se selección "Crear un nuevo recurso" en el panel principal de Azure y se eligió la opción "SQL Database".

Figura 30: Creación de una base de datos SQL en Azure

Tras esto, aparece un menú solicitando algunas de las configuraciones básicas del servicio:

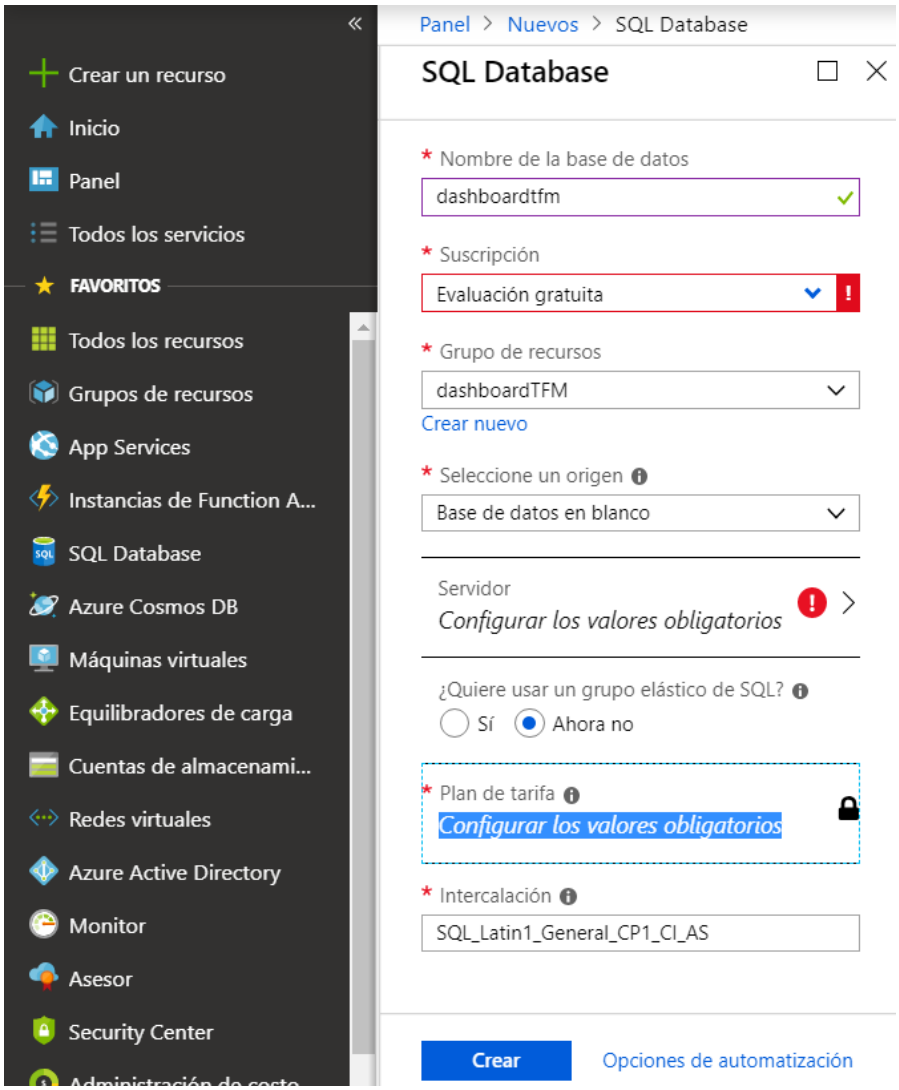

Figura 31: Definición de parámetros de base de datos SQL en Azure

En este menú, entre los parámetros que se solicitan para la creación de la base de datos SQL, se encuentra la selección de un servidor. Esta entrada hace referencia al servidor lógico que va a gestionar la base de datos y que constituye otro servicio independiente dentro del catálogo de Azure. Dado que en ese momento no había ninguno previamente creado, se generó uno dentro del propio menú de configuración de la base de datos. Los parámetros finalmente quedaron de la siguiente manera:

- Nombre de la base de datos: dashboardtfm
- Suscripción: Evaluación gratuita
- Grupo de recursos: dashboardTFM
- Origen: Base de datos en blanco
- Servidor:
    - Nombre del servidor: dashboardtfm
    - Administrador del servidor: vic-hl
    - Contraseña: *********
    - Ubicación: Oeste de Europa
- Grupo elástico: No
- Plan de tarifa: Básico
- Intercalación: SQL_Latin1_General_CP1_CI_AS

Tras esto, el panel de inicio del portal de Azure recoge todos los elementos necesarios para nuestro sistema:

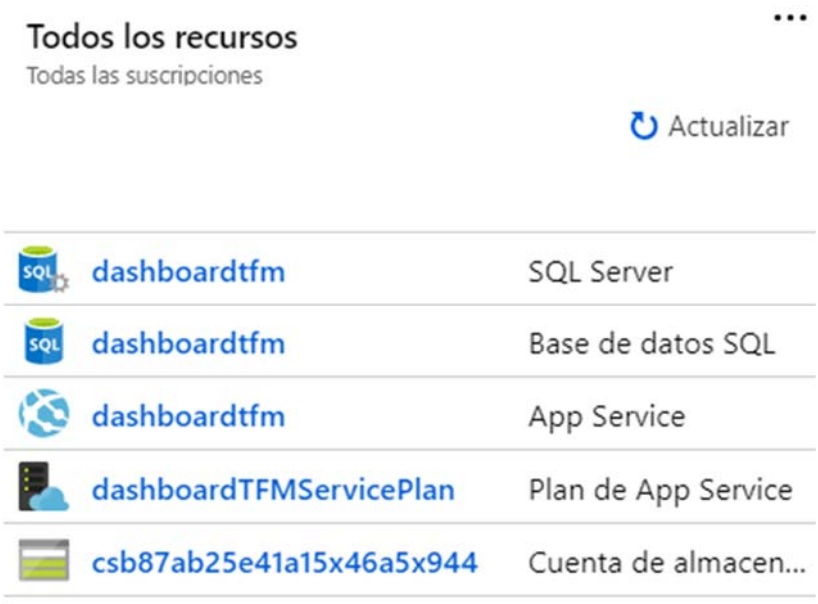

Figura 32: Panel de recursos en Azure

El siguiente paso consistió en la creación de la estructura de la base de datos y la población de las tablas con la información. Para ello, a través del recurso de base de datos SQL, se accedió a un Editor de consultas. Tras hacer *login* con las credenciales definidas en el proceso de creación del servidor que contiene la base de datos SQL, se procedió a generar las tablas y a introducir los datos con las siguientes sentencias:

```
CREATE TABLE jsondata (fecha DATETIME
                           NOT NULL DEFAULT CURRENT_TIMESTAMP,
                       json TEXT);

CREATE TABLE data (fecha DATETIME
                       NOT NULL DEFAULT CURRENT_TIMESTAMP,
                   json TEXT);

INSERT INTO data
VALUES ('2018-10-30 12:00', '{"Datos":{...}}');
```

Figura 33: Secuencia de comandos para crear tablas en Azure

Después de terminar de configurar la base de datos, se llevó a cabo la preparación de los scripts PHP para su subida a Azure. Antes de realizar la carga a través de FTPS, hubo que cambiar las cadenas de conexión que se habían usado originalmente en la implementación tradicional por las nuevas que permitirían el acceso a la base de datos alojada en Azure. Se modificó el script dbconnection.php para que quedase de la siguiente manera:

```
<?php
      $servername = "dashboardtfm.database.windows.net";
      $username = "vic-hl";
      $password = "Dashpss10";
      $dbname = "dashboardtfm";
?>
```

Figura 34: Cadena de conexión con base de datos SQL en Azure

Los datos contenidos en esta cadena de conexión deberían ser utilizados también por la aplicación o el servicio que proporciona los datos sin procesar y los carga en la base de datos.

Por último, se subieron los archivos a Azure usando los datos de conexión FTPS obtenidos durante la creación de los recursos. Con este último paso, el módulo de datos quedó completamente configurado y operativo. Para la ejecución del script, y por tanto la actualización de los datos, simplemente sería necesario hacer una llamada a través de curl a la url http://dashboardtfm.azurewebsites.net/lib/postrequest.php.

5.2. Módulo de tratamiento de datos

5.2.1. OpenCPU.io

Como extensión natural de las herramientas empleadas en la implementación tradicional, se decidió alojar el módulo de tratamiento de datos en la nube que ofrece OpenCPU.io. Entre las ventajas que esto supone, encontramos que el alojamiento no tiene coste, el procesado se hace de forma paralela e independiente a la aplicación web y/o a cualquier otro proceso que ejecutemos en el resto de módulos, y que el acceso se hace a través de peticiones HTTP.

Para poder hacer el despliegue del paquete, previamente preparado para la implementación tradicional, en OpenCPU.io, será necesario conectar y subir el código a GitHub. Para esto, en primer lugar, se instaló Git en Windows. Tras la instalación, se definió el usuario y el email, que posteriormente deben coincidir con los empleados en GitHub:

```
git config --global user.name "vic-hl"
git config --global user.email "victor.huescar@gmail.com"
```

Figura 35: Configuración de usuario en Git

Después, se creó una cuenta gratuita en GitHub utilizando los mismos nombre de usuario e email que los introducidos en la configuración local de Git. Además, se añadió la clave SSH de RStudio a GitHub para hacer seguras las posteriores actualizaciones de repositorio. Para ello, se comprobó la clave pública en las opciones globales de RStudio, dentro de la sección de Git/SVN y se añadió en las opciones de seguridad de GitHub:

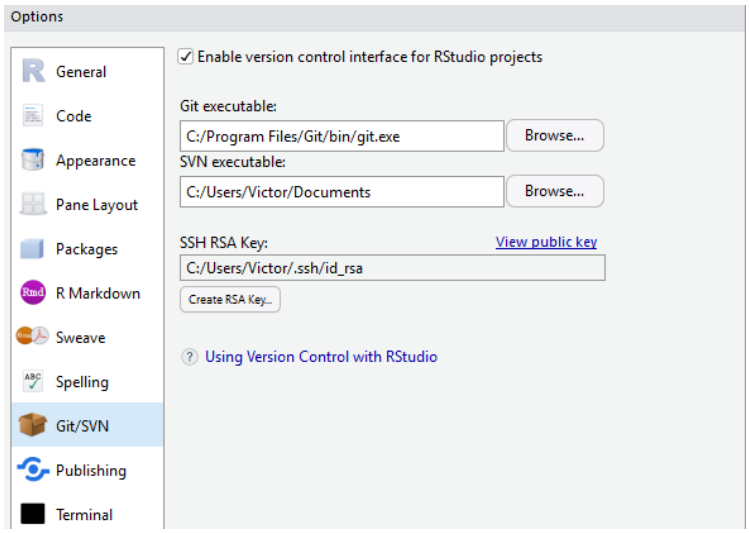

Figura 36: Comprobación de la clave pública en RStudio

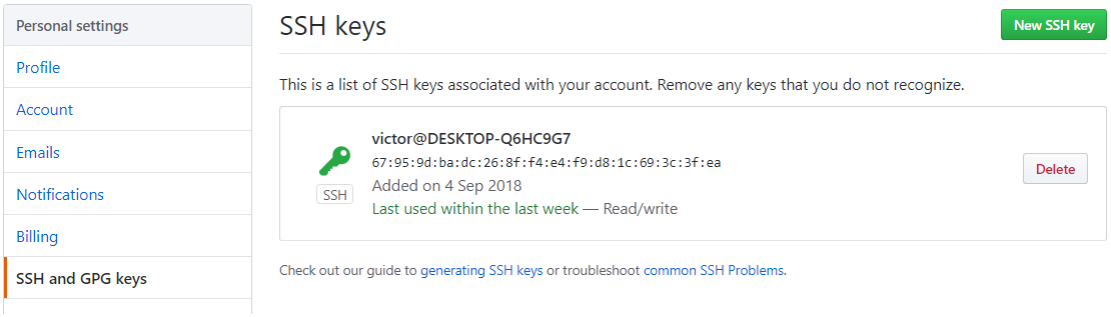

Figura 37: Adición de la clave pública en GitHub

Después, se añadió Git como herramienta de control de versiones en RStudio a través de las opciones de proyecto:

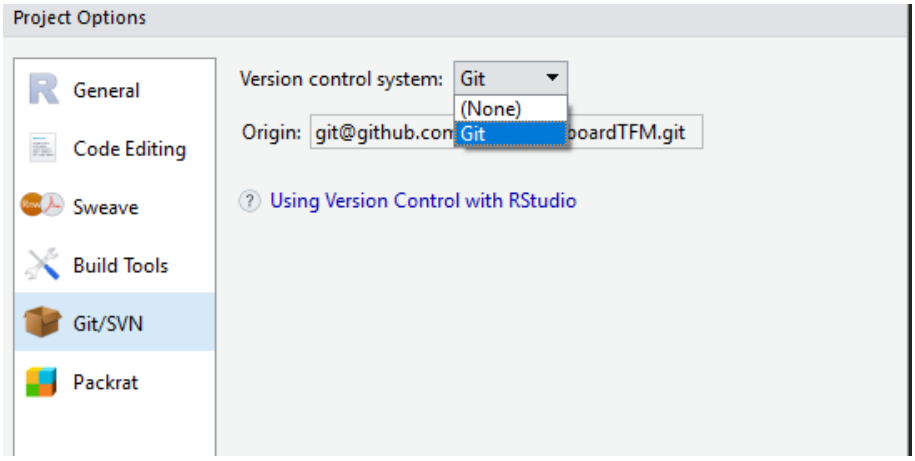

Figura 38: Incorporación de Git como control de versiones al proyecto

Tras esto, se reinició RStudio y se hizo el primer *backup* de código en Git. Habiendo guardado ya el paquete por primera vez en Git de forma local, se procedió a la creación de un repositorio en GitHub con el mismo nombre que el paquete para, inmediatamente después, realizar la carga con los siguientes comandos:

```
git remote add origin git@github.com:vic-hl/dashboardTFM
git push -u origin master
```

Figura 39: Carga inicial del repositorio en GitHub

Con esto, ya teníamos el paquete de tratamiento de datos disponible en GitHub. El siguiente pasó realizado fue conectar el repositorio con OpenCPU.io. Para ello, se añadió el *webhook* ofrecido por OpenCPU.io en las opciones del repositorio.

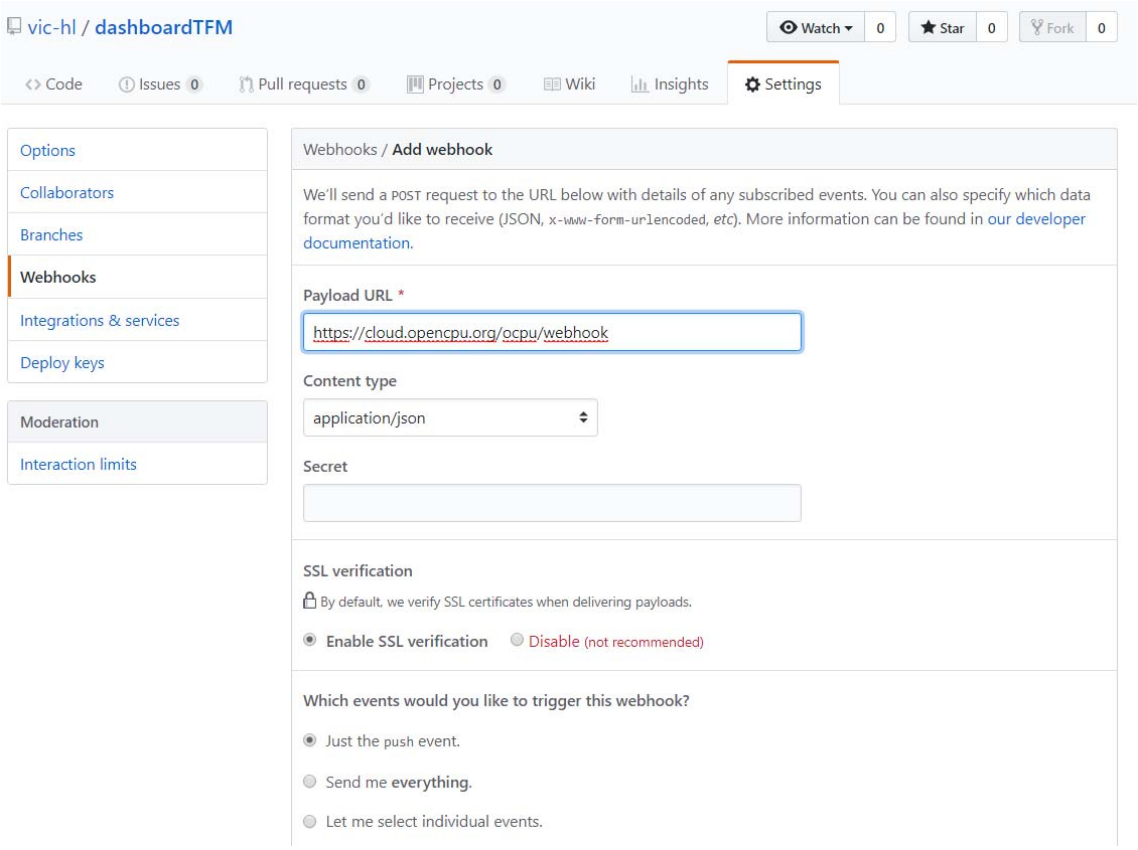

Figura 40: Configuración del webhook de OpenCPU.io

Con este enlace, el código del repositorio es cargado de forma automática en OpenCPU.io cada vez que se lleva a cabo alguna actualización. Cuando se confirmó esta configuración, se realizó la primera carga del código, momento a partir del cual, el servicio comenzó a estar disponible. El acceso a él se llevaba a cabo a través de la url https://vic-hl.ocpu.io/dashboardTFM/R/input/json, por tanto, tuvo que introducirse ese cambio en el código PHP para que la llamada se realizase de forma correcta.

Es digno de mención el hecho de que OpenCPU dispone de una página de test en la que se pueden realizar llamadas HTTP a los servicios alojados en la nube a través del enlace https://public.opencpu.org/ocpu/test/:

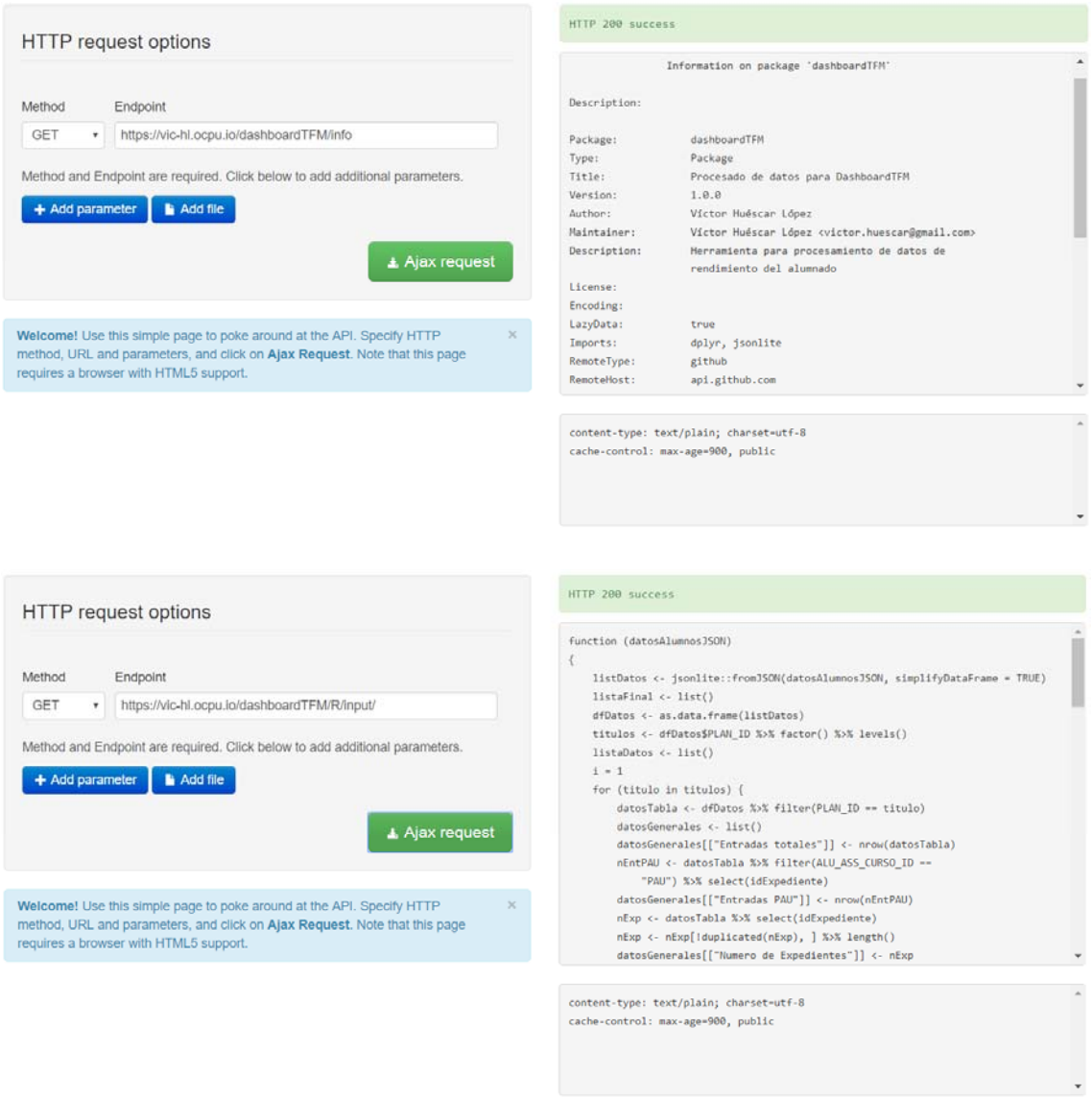

Figura 41: Pruebas de llamadas al módulo de tratamiento de datos

5.2.2. Azure Functions

Además de la implementación a través del servicio OpenCPU.io, se exploró la posibilidad de alojar el tratamiento de los datos dentro de Azure, mediante el uso de Azure Functions. Esta alternativa, a pesar de tener el inconveniente de que sí tiene un coste, a diferencia de OpenCPU.io, ofrece la gran ventaja de aunar en la misma plataforma tanto la base de datos como todos los scripts necesarios para el tratamiento. Esta circunstancia facilita enormemente el mantenimiento de estos dos módulos de la aplicación al estar ambos localizados dentro de un mismo entorno.

A pesar de que en el momento de la presentación de la propuesta de este trabajo Azure Functions no ofrecía soporte nativo para R, sí que disponía de una extensión que permitía la ejecución de scripts en este lenguaje. Sin embargo, Azure Functions eliminó dicha extensión en octubre de 2018. Para solventar esta circunstancia, se exploraron distintas opciones hasta conseguirlo a través de la ejecución del script usando CMD desde un script de C#. Los detalles de esta solución vienen recogidos a continuación en el proceso de despliegue del código en Azure Functions.

En primer lugar, dentro del portal de Azure, se seleccionó la opción de crear un nuevo recurso de tipo *Funtion App*.

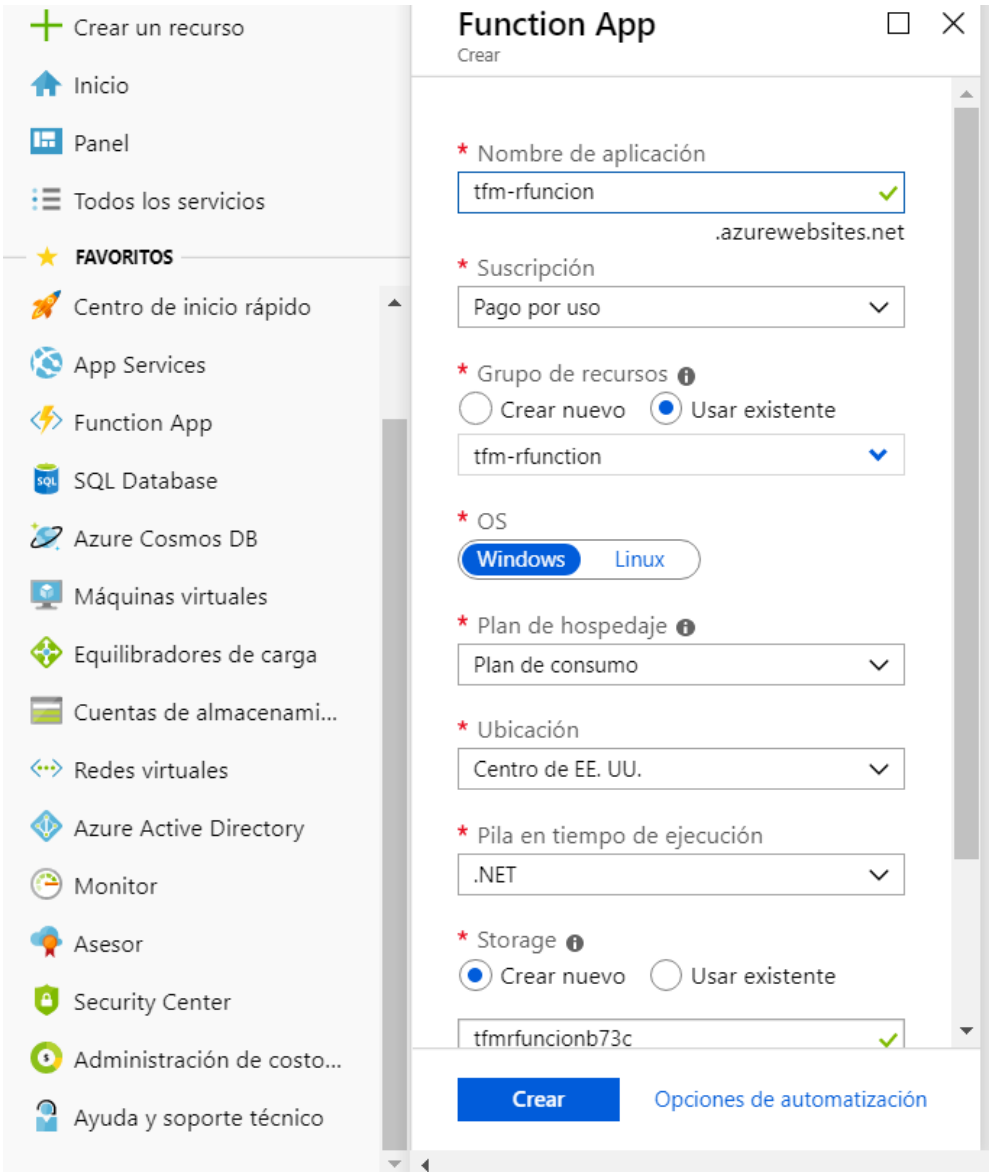

Figura 42: Creación del recurso Function App

Tras esto, se generó una función con activación por petición HTTP (*HTTP Trigger*). Se seleccionó esta opción para que la función fuese capaz de ejecutarse gracias a la petición hecha con curl desde el módulo de datos, de forma equivalente al funcionamiento en su implementación en OpenCPU.io.

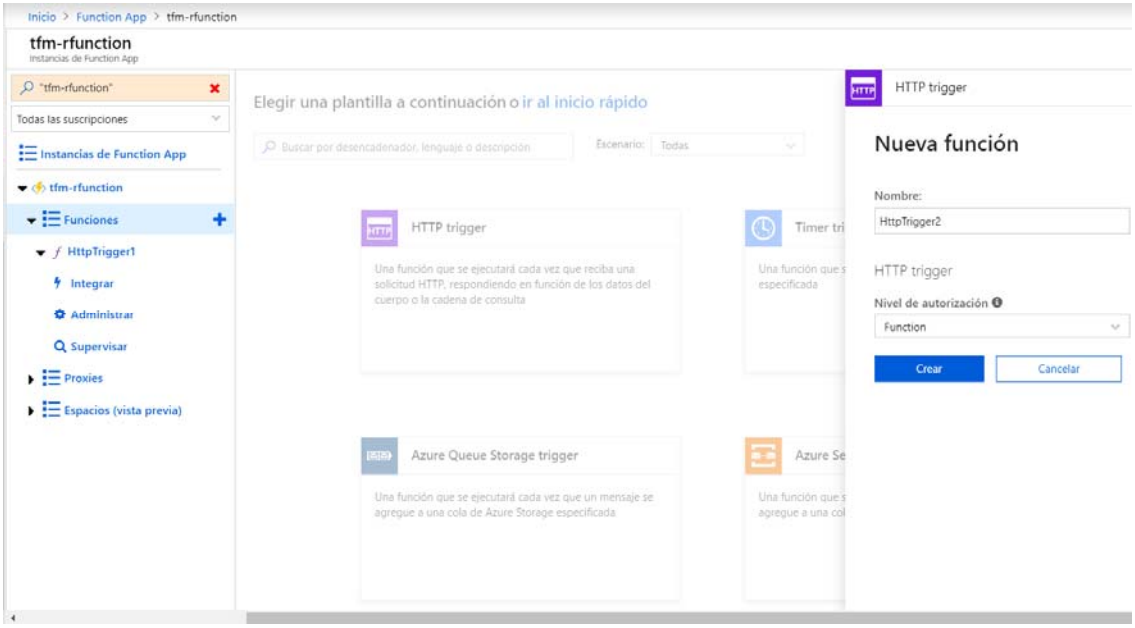

Figura 43: Creación de la función con HttpTrigger

Después de generar la función, se crea un directorio en la raíz de la *Function App* con el nombre de nuestra función que contiene un script en C# llamado run.csx. Este script es el que contiene el código que se ejecuta para responder a las peticiones HTTP que se envían a la función.

Debido a que ya no se disponía de la extensión para facilitar la integración de R, fue necesario subir el directorio de instalación al completo de R a Azure para que así pudiésemos disponer del intérprete. Dicha carpeta se subió a través de FTPS a la raíz D:\home\site\R-3.5.1. A la vez, se subió el script input.R a la carpeta donde se encontraba el script de C#.

Como se ha adelantado antes, se optó por la ejecución del script a través de CMD. Con este fin, se incorporó en el script de procesado de peticiones una clase capaz de llevar a cabo la ejecución a través de CMD y recoger el resultado, quedando el script en C# de la siguiente manera:

```
#r "Newtonsoft.Json"

using System.Net;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Web;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Primitives;
using Newtonsoft.Json;


public static async Task<IActionResult> Run(HttpRequest req,
ILogger log) {
    string datosAlumnosJSON = req.Query["datosAlumnosJSON"];
    string requestBody = await new
StreamReader(req.Body).ReadToEndAsync();
    dynamic data = JsonConvert.DeserializeObject(requestBody);
    datosAlumnosJSON = datosAlumnosJSON ?? data?.datosAlumnosJSON;

    string result =
RScriptRunner.RunFromCmd(@"D:\home\site\wwwroot\HttpTrigger1\input
.R", @"D:\home\site\R-3.5.1\bin\i386\RScript.exe",
datosAlumnosJSON);
    return datosAlumnosJSON != null
        ? (ActionResult)new OkObjectResult($"{result}")
        : new BadRequestObjectResult("Error, argumentos
inválidos");
}


public class RScriptRunner {
    public static string RunFromCmd(string rCodeFilePath, string
rScriptExecutablePath, string args) {
        string file = rCodeFilePath;
        string result = string.Empty;

        try {
            var info = new ProcessStartInfo();
            info.FileName = rScriptExecutablePath;
            info.WorkingDirectory =
Path.GetDirectoryName(rScriptExecutablePath);
            info.Arguments = rCodeFilePath + " " + args;

            info.RedirectStandardInput = false;
            info.RedirectStandardOutput = true;
            info.UseShellExecute = false;
            info.CreateNoWindow = true;
```

```
using (var proc = new Process()) {
                proc.StartInfo = info;
                proc.Start();
                result = proc.StandardOutput.ReadToEnd();
            }

            return result;
        }
        catch (Exception ex) {
            throw new Exception("R Script failed: " + result,
ex);
        }
    }
}
```

Figura 44: Script de procesado de peticiones HTTP en C#

Por último, fue necesario incorporar dos líneas extras de código al script input.R para que fuese capaz de leer los argumentos de entrada y de ejecutarse correctamente desde CMD.

```
args = commandArgs(trailingOnly = TRUE)
input(args[1])
```

Figura 45: Código añadido a input.R para adaptarlo a Azure Functions

Con esto, la función quedó operativa. Únicamente fue necesario cambiar el enlace de destino del script del módulo de datos para que dirigiese las peticiones a Azure Functions, en lugar de a OpenCPU.io.

5.3. Módulo de representación

5.3.1. ShinyApps.IO

De la misma manera que con el módulo de datos, se decidió emplear la nube ofrecida por RStudio para alojar la aplicación Shiny. Esto, nos ofrecía entre otras cosas, alojamiento sin coste, además de una integración y despliegue sencillos.

Para la configuración del enlace entre el proyecto local y la nube de Shiny, en primer lugar, se creó una cuenta en la plataforma asociándola con la cuenta de GitHub. Después se hizo login en la plataforma y se procedió a obtener la información de conexión con RStudio. Con este propósito, se accedió a la sección de Tokens, para poder generar uno que habilitase la conexión segura con el proyecto.

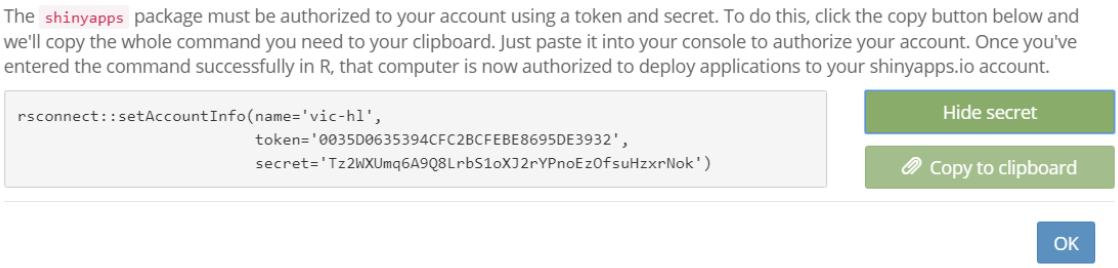

Figura 46: Generación de un token de conexión en ShinyApps.IO

Para añadir el token en el proyecto solo fue necesario introducir los siguientes comandos en la consola de R:

```
install.packages("rsconnect")

rsconnect::setAccountInfo(name='vic-hl',
      token='0035D0635394CFC2BCFEBE8695DE3932',
      secret='Tz2WXUmq6A9Q8LrbS1oXJ2rYPnoEzOfsuHzxrNok')
```

Figura 47: Configuración de la conexión con ShinyApps.IO en RStudio

Con esto, la conexión ya quedó configurada. Tras esto, la primera carga se llevó a cabo con los siguientes comandos:

```
library(rsconnect)

deployApp()
```

Figura 48: Comandos para realizar el primer despliegue en ShinyApps.IO

Tras la primera carga, la aplicación ya quedó operativa y accesible a través de la url https://vic-hl.shinyapps.io/shinyAppTFM/.

Desde la web de ShinyApps.IO se puede acceder a un dashboard para ver estadísticas de uso de la aplicación o realizar algunas configuraciones de uso.

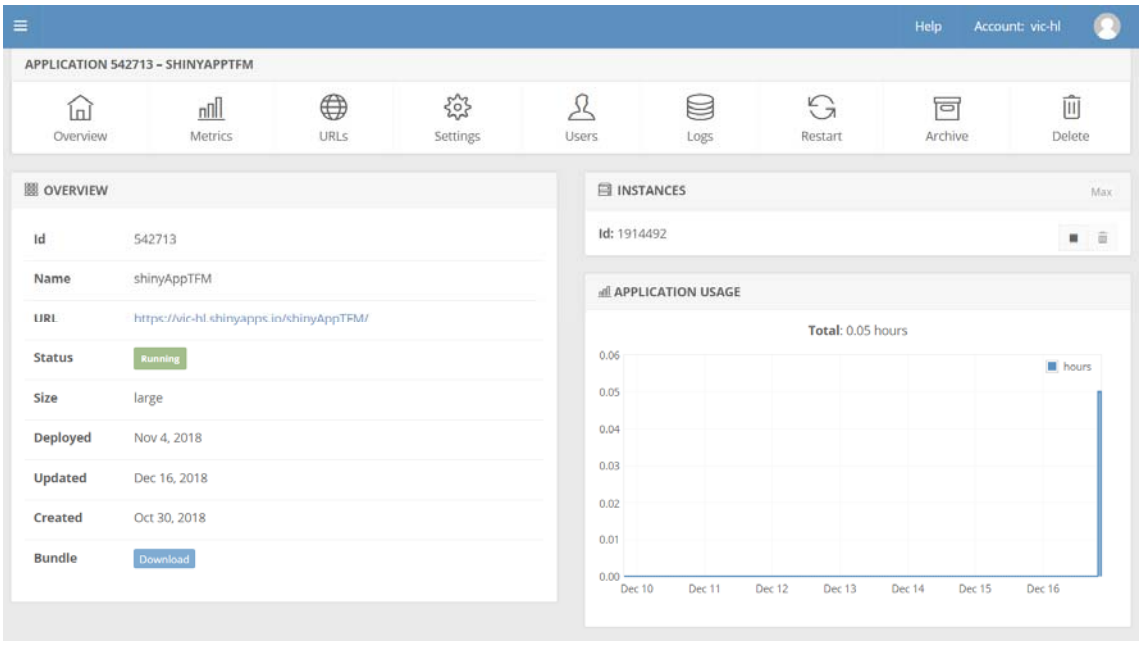

Figura 49: Dashboard de control y monitorización de la aplicación en ShinyApps.IO

5.4. Conclusiones de la implementación

El despliegue de la aplicación aplicando la filosofía *serverless* ha sido sencilla en general. La implementación del módulo de datos en Azure ha sido algo más costosa que en la implementación tradicional, debido básicamente a la curva de aprendizaje inicial de la plataforma y a la selección de los servicios a usar. Por su parte, el despliegue de los otros dos módulos ha sido muy sencillo y rápido.

En cuanto al coste, el módulo de datos en Azure ha supuesto un gasto de 40,17 euros del crédito gratuito ofrecido por la evaluación. Aunque no es una cifra muy representativa, debido a que el uso ha sido muy reducido y Azure factura los servicios utilizados por uso, sí que permite hacer una comparación a grandes rasgos de la diferencia que podría suponer con respecto a la implementación tradicional si se computa la compra del equipo que aloja el sistema. Por otro lado, el alojamiento en OpenCPU.io y ShinyApps.IO no han supuesto coste añadido.

CAPÍTULO 6: CONCLUSIONES

6.1. Conclusiones

A pesar de que la tesis principal de este trabajo no era el desarrollo de la aplicación, para su consecución, se ha diseñado y desarrollado una aplicación en distintos módulos.

Para el módulo de datos se ha diseñado una base de datos, en MySQL primero y SQL Server después, con una estructura dividida en dos tablas que contenían la información necesaria para la aplicación. Además, dentro del mismo módulo, se ha desarrollado un conjunto de *scripts* en PHP capaces de recuperar la información sin procesar de la base de datos, realizar solicitudes al servicio que alberga el procesado de datos y de almacenar de nuevo en la base de datos la información recibida como resultado de la petición.

Para el módulo de tratamiento de datos se llevaron a cabo dos grandes tareas: una exploración previa de los datos y el desarrollo del procesado para la aplicación. Para la primera, se llevó a cabo un examen exhaustivo de los datos proporcionados por la UPCT con el fin de adquirir un conocimiento profundo acerca de sus características, estructura y posibilidades de análisis. Además, se elaboraron algunos ejemplos de representación que posteriormente sirvieron de base para las representaciones introducidas en el tercer módulo. Por otro lado, se desarrolló un paquete con una función de tratamiento de datos capaz de extraer estadísticas generales del conjunto, así como de extraer la información

necesaria para la representación de las tasas de presentados de las asignaturas de las titulaciones de grado ofertadas por la UPCT. Como parte de la funcionalidad del paquete, se implementó la posibilidad de la recepción y la devolución de los datos en formato JSON para facilitar el intercambio entre módulos en las implementaciones posteriores.

Para el enfoque del módulo de representación se optó por usar Shiny por ser una herramienta sencilla e intuitiva para la creación de interfaces dentro del propio entorno de R. A través del uso de este *framework* se ha desarrollado una interfaz, sencilla pero funcional, capaz de representar los datos extraídos por el módulo de tratamiento en tablas o gráficas y que permite la navegación entre ellos a través de una serie de controles.

En cuanto a las implementaciones llevadas a cabo. En primer lugar, se ha realizado un despliegue de la aplicación siguiendo el modelo tradicional que implica el alojamiento de la aplicación en equipos físicos propios. Esto ha supuesto el despliegue y configuración de un servidor web con base de datos MySQL a través del uso del paquete XAMPP. También ha conllevado la implementación de un servicio web para el módulo de tratamiento de datos. Y, por último, otro servicio web, en este caso con interfaz y más orientado al consumo final, a través del uso de Shiny. Estos dos últimos servicios ejecutados desde el entorno de RStudio.

Para la implementación *serverless*, se hizo una migración hacia las plataformas *cloud* correspondientes teniendo como referencia las herramientas y utilizadas en la implementación tradicional. Para el módulo de datos se ha empleado distintos servicios de la plataforma Azure de Microsoft. Esto, en primer lugar, ha supuesto un examen en detalle acerca de los servicios ofrecidos por la plataforma para seleccionar el más adecuado, así como de un proceso de aprendizaje para el manejo de la interfaz y sus distintas configuraciones. Al final se optó por el uso del *App Service* y el servicio *SQL Database* de azure que implementa el modelo DBaaS. Se llevó a cabo un proceso de creación y población de la base de datos similar al ejecutado en la implementación tradicional, así como una adaptación que posibilitara la conexión con esta nueva base de datos del código de PHP.

El primer despliegue del módulo de tratamiento de datos fue algo más sencillo debido a que en la implementación previa el paquete ya se preparó con el formato requerido por OpenCPU. El proceso consistió básicamente en dos pasos principales, instalación de Git y subida del código a GitHub; y conexión del repositorio en GitHub

con OpenCPU.io a través de un *webhook*. En el momento en que se hizo la primera conexión, la plataforma de OpenCPU.io recuperó el paquete de GitHub y desplegó el servicio automáticamente, estando disponible y funcionando desde ese instante. En cuanto a la implementación a través de Azure Functions, fue algo más tediosa debido a que R no está integrado de forma nativa en la plataforma. Para resolver esto, se ideó una solución a través de la subida a la plataforma del intérprete de R y la ejecución de los scripts desde CMD. El resultado fue un servicio completamente funcional capaz de recibir los datos de peticiones HTTP y de devolverlos en formato JSON, al igual que en el despliegue en OpenCPU.io.

Por último, la adaptación del módulo de representación a la filosofía *serverless* se llevó a cabo a través de ShinyApps.IO. El proceso de despliegue de la interfaz en la plataforma se limitó, en esencia, al proceso de registro en la web y a conectar el entorno de RStudio a ShinyApps.IO, habilitando la carga directa desde el proyecto de Shiny local a la nube.

A lo largo del trabajo, se ha podido evaluar el proceso de desarrollo y despliegue a través de dos modelos distintos: modelo tradicional y *serverless*. Esta doble implementación ha permitido el poder examinar las características de cada uno de los dos modelos, así como poder comparar las bondades y faltas de ambos. En cuanto a la velocidad y facilidad de despliegue, se ha encontrado que, a pesar de haber escogido herramientas de fácil instalación y de configuración rápida, el modelo tradicional ha sido ligeramente más costoso de desplegar. Esto se hubiese acrecentado si hubiésemos tenido en cuenta la instalación y configuración previa del sistema operativo en el equipo, así como si hubiésemos contado con cualquier otra infraestructura de red o equipo aparte que hubiese podido ser necesario para el sistema. La solución *serverless*, en cambio, por definición no requiere de ningún equipo físico, por lo que estos pasos previos se ahorran. Además, las plataformas elegidas ofrecen, en mayor o menor medida, soluciones de despliegue muy rápido que reducen el proceso de despliegue a tareas casi triviales. En lo relativo a las diferencias de coste, a pesar de que no se ha profundizado en exceso en este aspecto de la comparación, encontramos que la diferencia entre ambas implementaciones en cuanto a coste inicial sería enorme. Esta diferencia se fundamentaría principalmente en los costes de adquisición necesario para alojar los servicios en la implementación tradicional, ya que el software y las herramientas escogidas en ambos casos son gratuitas a excepción de los servicios de Azure. En cuanto a los costes de mantenimiento y

generados a lo largo del tiempo, en función del volumen de peticiones y que alcanzase la aplicación, el modelo *serverless* podría llegar a superar en costes mensuales a la implementación tradicional. A pesar de ello, la inversión inicial requerida en el modelo tradicional haría que a corto y, al menos, medio plazo la solución *serverless* fuese más rentable. Esto confirmaría las ideas recogidas en el apartado tres acerca de las diferencias de costes entre estas dos implementaciones.

Recapitulando, se concluye que las plataformas *cloud* ofrecen herramientas potentes que posibilitan el despliegue de aplicaciones de forma sencilla y con inversiones iniciales bajas, lo que reduce el factor de riesgo inicial y las hace idóneas para pequeños y medianas empresas que quieren implementar sus servicios. También, se prevé que va a ser un modelo en alza que va a continuar expandiéndose y restringiendo el modelo de implementación tradicional a grandes empresas o servicios que disponen de la capacidad de obtener beneficio de la propiedad de la infraestructura.

6.2. Líneas futuras

Como posibles líneas futuras, se presentan tres grandes ramas como posibilidades para profundizar y continuar con el tema de este trabajo.

En primer lugar, la posibilidad de ampliar las funcionalidades de la aplicación. A pesar de que la aplicación se ha desarrollado con funcionalidades orientadas a la representación gráfica de la información, los datos poseen un gran potencial como para desarrollar y profundizar en el análisis y la posible predicción de información a partir de los datos del conjunto.

En segundo lugar, se podría sustituir alguno de los módulos por otra implementación. El sistema se ha diseñado modularmente para que cada uno de los módulos que lo componen pueda ampliarse o sustituirse de forma sencilla siempre que se respete el formato y las interfaces de intercambios de datos. Por ello, una posible línea de futuro podría ser implementar el módulo de representación con tecnologías web (HTML, CSS y Javascript) junto con algún *framework* de representación. O sustituir el módulo de tratamiento de datos por alguna implementación realizada en otro lenguaje como Python. También podría plantearse el dar un enfoque centrado en el desarrollo de tratamientos costosos computacionalmente que podrían ejecutarse en segundo plano sin impedimento

gracias a la independencia de los módulos del sistema. Esta opción pasaría por el empleo de tratamientos de tipo incremental

Por último, podría ser interesante el probar el despliegue de este mismo sistema en otras plataformas como Amazon Web Services o Google Cloud Platform para comprobar las similitudes y diferencias que se encuentran con respecto al uso de Azure, además de contrastar si las ventajas de que, en principio disponen sobre la implementación tradicional, también se mantienen.

BIBLIOGRAFÍA

Amazon. (2018). *Amazon Web Services*. Obtenido de https://aws.amazon.com/es/

Google. (2018). *Google Cloud*. Obtenido de https://cloud.google.com/

Mell, P., & Grance, T. (2011). The NIST Definition of Cloud Computing. *National Institute of Standards and Technology Special Publication 800-145, U.S. Department of Commerce*.

Microsoft Azure. (2018). *Portal de Microsoft Azure*. Obtenido de https://azure.microsoft.com

RStudio Inc. (2018). *Shiny from RStudio*. Obtenido de https://shiny.rstudio.com/

The R Foundation. (2018). *The Comprehensive R Archive Network (CRAN)*. Obtenido de https://cran.r-project.org/

Zhang, Q., Cheng, L., & Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications, Springer London*.

Todo el código desarrollado para este proyecto se encuentra en un repositorio público en la plataforma GitHub al que se puede acceder a través del siguiente enlace: https://github.com/vic-hl/TFMDashboard

ANEXO 1: CÓDIGO PHP

Anexo 1.1. Script dbconnection.php

```php
<?php
	$servername = "localhost";
	$username = "root";
	$password = "dashpss10";
	$dbname = "dashboardtfm";
?>
```

Anexo 1.2. Script postrequest.php

```php
<?php
	header("Content-Type:text/plain; charset=utf-8");

// Esto le dice a PHP que generaremos cadenas UTF-8
	require "dbconnection.php";
// Establecemos el tiempo máximo de ejecución en 15 mins debido a que el procesamiento de los ficheros es lento por la cantidad potencial de datos
	ini_set('max_execution_time', 900);

// Generamos la cadena de JSON que se va a enviar en la petición POST a OpenCPU
	$sql = "SELECT * FROM jsondata ORDER BY fecha LIMIT 20;";
// Creamos la conexión con la BBDD
	$conn = new mysqli($servername, $username, $password, $dbname);
// Comprobamos que la conexión es correcta
	if ($conn->connect_error) {
	    die("Connection failed: " . $conn->connect_error);
	}
```

```
$jsondata = "";

      $flag = false;
      // Realizamos la petición SQL
      $result = $conn->query($sql);
      if ($result->num_rows > 0) {
          while($row = $result->fetch_assoc()) {
            // if para añadir una coma antes de cada paquete de
datos, excepto en el primero
            if($flag){
                 $jsondata = $jsondata.",";
            }
              $jsondata = $jsondata.substr($row["json"], 1, -3);
              $flag = true;
          }

      }
      else {
           echo "0 results";
      }
      $conn->close();
      $jsondata = utf8_encode($jsondata);
      $jsondata = str_replace('"', '\"', $jsondata);
      $jsondata = "[{\"datosAlumnosJSON\":\"[".$jsondata."]\"}]";

      // URL de la función a la que hacemos la llamada. NOTA:
especificamos al final de la URL que los datos que debe devolver
deben estar en formato JSON
      $url = "https://vic-hl.ocpu.io/dashboardTFM/R/input/json";
      // $url =
"http://127.0.0.1:5656/ocpu/library/dashboardTFM/R/input";

      $curl = curl_init();
      // Configuramos las opciones de la petición CURL
      curl_setopt_array($curl, array(
        CURLOPT_URL => $url,
        CURLOPT_RETURNTRANSFER => 1,
        CURLOPT_MAXREDIRS => 10,
        CURLOPT_TIMEOUT => 1500,
        CURLOPT_CUSTOMREQUEST => "POST",
        CURLOPT_POST => 1,
        CURLOPT_POSTFIELDS => $jsondata,
        CURLOPT_HTTPHEADER => array(
          "Content-Type: application/json",
          "Charset:utf-8",
          "Content-Length: " . strlen($jsondata)
        ),
      ));
      // Ejecutamos la petición
      $response = curl_exec($curl);
      $info = curl_getinfo($curl);

      curl_close($curl);


      // INTRODUCCIÓN DE LOS DATOS RECIBIDOS EN LA BBDD
      $today = date("Y-m-d H:i", time());
      $sql = "INSERT INTO data (fecha, json)".
              " VALUES ('".$today."', '". $response ."')";
```

```php
// Creamos la conexión con la BBDD
      $conn = new mysqli($servername, $username, $password,
$dbname);
      // Comprobamos que la conexión es correcta
      if ($conn->connect_error) {
          die("Connection failed: " . $conn->connect_error);
      }
      // Realizamos la inserción de datos
      if ($conn->query($sql) === TRUE) {
          echo "Se han introducido los datos correctamente.";
      } else {
          echo "Error: " . $sql . "<br>" . $conn->error;
      }
      $conn->close();

    exit();
 ?>
```

ANEXO 2: CÓDIGO R

Anexo 2.1. Script input.R

```
input <- function(datosAlumnosJSON){

  listDatos <- jsonlite::fromJSON(datosAlumnosJSON, simplifyDataFrame =
TRUE)
  listaFinal <- list()
  dfDatos <- as.data.frame(listDatos)

  titulos <- dfDatos$PLAN_ID %>% factor() %>% levels()

  # Cálculo de datos generales
  listaDatos <- list()
  i = 1
  for(titulo in titulos){
    datosTabla <- dfDatos %>% filter(PLAN_ID == titulo)
    datosGenerales <- list()
    datosGenerales[["Entradas totales"]] <- nrow(datosTabla)

    nEntPAU <- datosTabla %>% filter(ALU_ASS_CURSO_ID == "PAU") %>%
select(idExpediente)
    datosGenerales[["Entradas PAU"]] <- nrow(nEntPAU)

    nExp <- datosTabla %>% select(idExpediente)
    nExp <- nExp[!duplicated(nExp),] %>% length()
    datosGenerales[["Numero de Expedientes"]] <- nExp

    nExpPAU <- nEntPAU[!duplicated(nEntPAU),] %>% length()
    datosGenerales[["Numero de Expedientes PAU"]] <- nExpPAU

    anios <- datosTabla$ANYACA_MAT_ID %>% factor() %>% levels()
    datosGenerales[["Intervalo de cursos comprendidos"]] <-
c(paste(min(anios), max(anios), sep="/"))


    listaDatos[[titulo]] <- datosGenerales
  }
  listaFinal[["Datos Generales"]] <- listaDatos
```

```
# Cálculo de presentados por curso
  listaDatos <- list()
  for(titulo in titulos){
    listaPorcentajes <- list()
    for(curso in c("1", "2", "3", "4")){
      datosGrafica <- dfDatos %>% filter(PLAN_ID == titulo)
      temp2 <- datosGrafica %>% filter(descValor == "CALIFICACION_DESC" &
ALU_ASS_CURSO_ID == curso) %>%
        select(idExpediente, ASS_ASS_DESC, valor) %>% na.omit()

      temp2 <- temp2 %>% mutate(PRESENTADO = ifelse(temp2$valor != "NO
PRESENTADO", "PRESENTADO", "NO PRESENTADO"))

      temp2 <- temp2[!duplicated(temp2), ]

      tabla <- as.data.frame(table(temp2 %>% select(ASS_ASS_DESC,
PRESENTADO)))
      porcentajes <- data.frame("ASS_ASS_DESC" = character(),
"PRESENTADO" = character(), "Freq" = numeric(), stringsAsFactors = FALSE)


for(asig in temp2$ASS_ASS_DESC %>% factor() %>% levels()){
        t <- tabla %>% filter(ASS_ASS_DESC == asig) %>% select(Freq) %>%
sum()
        if(length(as.numeric(tabla[tabla$ASS_ASS_DESC == asig &
tabla$PRESENTADO == "PRESENTADO", "Freq"]/t)) != 0){
          porcentajes[length(porcentajes$ASS_ASS_DESC)+1, ] <- c(asig,
"PRESENTADO", as.numeric(tabla[tabla$ASS_ASS_DESC == asig &
tabla$PRESENTADO == "PRESENTADO", "Freq"]/t))
        }
        else{
          porcentajes[length(porcentajes$ASS_ASS_DESC)+1, ] <- c(asig,
"PRESENTADO", 0)
        }
        if(length(as.numeric(tabla[tabla$ASS_ASS_DESC == asig &
tabla$PRESENTADO == "NO PRESENTADO", "Freq"]/t)) != 0){
          porcentajes[length(porcentajes$ASS_ASS_DESC)+1, ] <- c(asig,
"NO PRESENTADO", as.numeric(tabla[tabla$ASS_ASS_DESC == asig &
tabla$PRESENTADO == "NO PRESENTADO", "Freq"]/t))
        }
        else{
          porcentajes[length(porcentajes$ASS_ASS_DESC)+1, ] <- c(asig,
"NO PRESENTADO", 0)
        }
      }
      listaPorcentajes[[curso]] <- porcentajes
    }
    listaDatos[[titulo]] <- listaPorcentajes
  }
  listaFinal[["Datos Presentados"]] <- listaDatos
  return(listaFinal)
}
```

ANEXO 3: CÓDIGO SHINY

Anexo 3.1. Script ui.R

```
library(shiny)
library(DBI)
library(odbc)
library(dplyr)
# library(dbplyr)
library(pool)
library(ggplot2)
library(scales)
library(knitr)
library(kableExtra)
library(jsonlite)


source("R/functions.R")

# Define UI for app that draws a histogram ----

fluidPage(

  # App title ----
  titlePanel("Panel de Control"),

  # Sidebar layout with input and output definitions ----
  sidebarLayout(
    sidebarPanel(
      selectInput("planEstudios", h4("Selecciona un plan de
estudios:"),
```

```
choices = list("5011 - Grado en Arquitectura" = "5011",
 "5021 - Grado en Ingeniería de Edificación" = "5021",
 "5031 - Grado en Turismo" = "5031",
 "5041 - Grado en Ingeniería de Sistemas de Telecomunicaciones"
= "5041",
 "5051 - Grado en Ingeniería Telemática" = "5051",
 "5061 - Grado en Ingeniería Eléctrica" = "5061",
 "5071 - Grado en Ingeniería Electrónica Industrial y
Automática" = "5071",
 "5081 - Grado en Ingeniería Mecánica" = "5081",
 "5091 - Grado en Ingeniería Química Industrial" = "5091",
 "5101 - Grado en Administración y Dirección de Empresas" =
"5101",
 5111 - Grado en Ingeniería de Organización Industrial" =
"5111",
 "5121 - Grado en Ingeniería de Tecnologías Industriales" =
"5121",
 "5131 - Grado en Arquitectura Naval e Ingeniería de Sistemas
Marinos" = "5131",
 "5141 - Grado en Ingeniería de la Hortofruticultura y
Jardinería" = "5141",
 "5151 - Grado en Ingeniería de las Industrias Agroalimentarias"
= "5151",
 "5161 - Grado en Ingeniería Civil" = "5161",
 "5171 - Grado en Ingeniería de Recursos Minerales y Energía" =
"5171",
 "5181 - Grado en Ingeniería Agroalimentaria y de Sistemas
Biológicos" = "5181",
 "5191 - Grado en Fundamentos de Arquitectura" = "5191"),
selected = 5041),
      h4("Opciones de visualización"),
      fluidRow(
        column(6, checkboxInput("showTablaDatosGenerales",
"Resumen de datos", value = TRUE)),
        column(6, checkboxInput("showGrafica", "Gráfica", value
= FALSE))
      ),
      conditionalPanel("input.showGrafica",
                                selectInput("cursoID",
h4("Curso"),
                                                    choices =
list("1er Curso" = "1",

"2do Curso" = "2",

"3er Curso" = "3",

"4to Curso" = "4")))

    ),
    mainPanel(
      # textOutput("texto"),
      conditionalPanel("input.showTablaDatosGenerales",
        tableOutput("tabla")),
      conditionalPanel("input.showGrafica",
        plotOutput("grafica", height = "600px"))
    )
  )
)
```

Anexo 3.2. Script server.R

```
function(input, output) {
  output$tabla <- renderTable({
    as.data.frame(
      jsonlite::fromJSON(datos_generales(input$planEstudios))) %>%
      rename("Entradas totales" = Entradas.totales,
        "Entradas PAU" = Entradas.PAU,
        "Nº de expedientes" = Nº.de.Expedientes,
        "Nº de expedientes PAU" = Nº.de.Expedientes.PAU,
        "Intervalo de cursos" = Intervalo.de.cursos.comprendidos)
    }, align = 'c')
  output$grafica <- renderPlot({
    porcentajes <- as.data.frame(
      jsonlite::fromJSON(
        datos_presentados(input$planEstudios, input$cursoID)))
    ggplot(data = porcentajes, aes(x = ASS_ASS_DESC)) +
      geom_col(aes(y = as.numeric(Freq),
                   fill = PRESENTADO),
                   position = position_dodge()) +
      scale_y_continuous(labels=percent) +
      labs(x = "Nombre de la asignatura",
           y = "Porcentaje de presentados/no presentados") +
      theme(axis.text.x = element_text(angle = 75, hjust = 1))
  })
}
```

Anexo 3.3. Script functions.R

```
  my_db <- dbPool(
  RMySQL::MySQL(),
  dbname = "dashboardtfm",
  host = "127.0.0.1",
  username = "root",
  password = "dashpss10"
)


datos_generales <- function(planEstudios){
  dfDatos <- dbGetQuery(my_db,
    "SELECT json FROM data ORDER BY fecha DESC LIMIT 1;")
  jsonTmp <- jsonlite::fromJSON(dfDatos$json)
  datos <- jsonTmp$`Datos Generales`
  json <- jsonlite::toJSON(datos[[planEstudios]])
  return(json)
}

datos_presentados <- function(planEstudios, curso){
  dfDatos <- dbGetQuery(my_db,
    "SELECT json FROM data ORDER BY fecha DESC LIMIT 1;")
  jsonTmp <- jsonlite::fromJSON(dfDatos$json)
  datos <- jsonTmp$`Datos Presentados`
  json <- jsonlite::toJSON(datos[[planEstudios]][[curso]])
  return(json)
}
```

ANEXO 4: CÓDIGO C#

Anexo 4.1. Script server.R

```
#r "Newtonsoft.Json"

using System.Net;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Web;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Primitives;
using Newtonsoft.Json;


public static async Task<IActionResult> Run(HttpRequest req,
ILogger log) {
    string datosAlumnosJSON = req.Query["datosAlumnosJSON"];
    string requestBody = await new
StreamReader(req.Body).ReadToEndAsync();
    dynamic data = JsonConvert.DeserializeObject(requestBody);
    datosAlumnosJSON = datosAlumnosJSON ?? data?.datosAlumnosJSON;

    string result =
RScriptRunner.RunFromCmd(@"D:\home\site\wwwroot\HttpTrigger1\input
.R", @"D:\home\site\R-3.5.1\bin\i386\RScript.exe",
datosAlumnosJSON);
    return datosAlumnosJSON != null
        ? (ActionResult)new OkObjectResult($"{result}")
        : new BadRequestObjectResult("Error, argumentos
inválidos");
}
```

```
public class RScriptRunner {
    public static string RunFromCmd(string rCodeFilePath, string
rScriptExecutablePath, string args) {
        string file = rCodeFilePath;
        string result = string.Empty;

        try {
            var info = new ProcessStartInfo();
            info.FileName = rScriptExecutablePath;
            info.WorkingDirectory =
Path.GetDirectoryName(rScriptExecutablePath);
            info.Arguments = rCodeFilePath + " " + args;

            info.RedirectStandardInput = false;
            info.RedirectStandardOutput = true;
            info.UseShellExecute = false;
            info.CreateNoWindow = true;

            using (var proc = new Process()) {
                proc.StartInfo = info;
                proc.Start();
                result = proc.StandardOutput.ReadToEnd();
            }

            return result;
        }
        catch (Exception ex) {
            throw new Exception("R Script failed: " + result, ex);
        }
    }
}
```