1. Introducción
   1. Contexto
   2. Actores
   3. Definiciones
   4. El problema
2. Justificación
3. Alcance
   1. Objetivos
   2. Requerimientos
   3. Riesgos
4. Metodología
5. Planificación temporal
   1. Planificación Inical
   2. Planificación Actual
6. Base tecnologica
7. Anàlisi de technologies
   1. Pipelines
   2. Azure
      1. Function App
      2. Web App
      3. Azure kubernetes service
      4. Azure sql
8. Sistema de ficheros
9. Ejemplo
10. Gestión económica
    1. Presupuesto
    2. Control de gestión
11. Sostenibilidad
12. Bibliografía

**Trabajo hecho:**

En este apartado vamos a detallar en las tecnologías que se han usado para este trabajo, y como se han usado para poder terminar el proyecto.

**Bicep**

En este trabajo usamos bicep para definir toda la infraestructura de los diferentes recusos.

Principalmente tenemos un fichero principal que se llama main.bicep que es el fichero que se va a ejecutar y después tenemos otros ficheros dentro de la carpeta bicep-templates, que allá están definidos las plantillas de todos los recursos que vamos a usar, y de este forma si un recuso lo vamos a usar más de una vez, entonces no hace falta definirlo más de una vez, solamente con llamarlo las veces que hace falta con pasar los parámetros ya lo tendremos

Los nombres de los ficheros como los recusos siguen las abreviaturas que definen Microsoft para azure :

App Service Plan: plan

Function app: func

Web app: app

Log Analytics Workspace: log

Application Insights: appi

Azure SQL database: sqldb

Azure SQL Database server: sql

Azure Kubernetes Service: aks

Azure Container Registry: acr

Storage Account: st

**Function App**

El Function App en este proyecto sirve como un entorno de ejecución para que el usuario pone sus códigos que quiere que se ejecuten en Python versión 3.9, el usuario solamente pone su código y también las librarías y las versiones que use, y Azure se encarga del resto.

La manera que el usuario tiene para poder lanzar su código dentro del Function App, es usando el URL del Function, y el código se ejecutará y responderá con el resultado esperado

El mismo Function App lo podemos usar para poner más de un código y que todos se ejecuten de forma separada (cada uno se ejecuta con su URL).

Para que el usuario pueda monetizar sus códigos, el Function App se le crea previamente un Application Insights y este Application Insights se conecta con un Log Analytics para que el usuario puede ver todos los logs (excepciones, errores, warning, etc) y las métricas(la memoria disponible, los procesos del CPU, las requestas, etc), en dos modos, en modo directo y también se puede ver las dadas de fechas anteriores que están guardadas.

storage account para poder crear el Azure App, previamente necesitamos crear un Storage Account, como un lugar para guardar los códigos que se tiene que ejecutar.

El Function app tiene que estar creado encima de un service app plan, que se considera como el servidor donde se corre el código y el usuario puede elegir las características físicas del este servidor

**Azure Kubernetes Service:**

Todas las aplicaciones que vienen con el proyecto están empaquetados usando Helm, para qué pueden tener la máxima portabilidad posible.

Las aplicaciones que vienen predefinidos con el aks son los siguientes:

* Ingress-Ngnix: Esta aplicación tiene todos los componentes necesarios, para hacer funcionar el controlado Ingress de Nginx, porque las aplicaciones en el clúster cuando quieren comunica con el exterior lo hacen utilizando este controlado, de esta manera ahorramos el dinero, ya que, si no para cada aplicación que se quiere comunicar con el exterior tendremos que crear un servicio de tipo loadbalancer que en el aks, este tipo de servicio es un servicio de pago, porque Azure te crea un loadbalancer por cada servicio de este tipo
* Grafana: Esta aplicación tiene todos los componentes de Grafana y Prometheus para que Prometheus coge las métricas desde el clúster de Kubernetes y también tiene una aplicación que se llama azure-exporter creada dentro de la aplicación de Grafana, esta aplicación lo que hace es usar unos permisos que se asignan al clúster de Kubernetes para que pueda leer los datos de monitorización en Azure y exporta las métricas de Azure en un puerto que posteriormente el Prometheus lea las métricas de Azure de este puerto, y exporta estas métricas a Grafana para que los usuarios pueden ver las métricas en la interfaz de Grafana. Para Prometheus y Grafana hay una regla en el controlador de Ingress para que los usuarios externos pueden entrar a la interfaz gráfica de Grafana como Prometheus
* La última aplicación es la aplicación creada para que el usuario pueda usarla para poner su proyecto aquí, y también tiene una regla en el controlador de Ingress para que los usuarios externos también pueden conectar a la aplicación

El aks dentro de un pipeline se le asigna los permisos necesarios para que el azure-exporte puede leer las métricas desde Azure.

El aks se le asigna un log analytics y así el usuario no le hace falta conectarse dentro del clúster para qué pueda ver los estados y los logs de los diferentes proyectos dentro del clúster, también desde el portal de Azure, Azure crear gráficas y reportes muy útiles y explicativos para todos los componentes del aks.

Con el aks también se crea un acr y se conecta con el aks, esto es para que después de crear la imagen del proyecto del cliente y la imagen del proyecto del azure- exporter, estas dos imágenes se suben al acr, porque posteriormente el clúster las puede usar para levantar los proyectos

**Azure SQL database**

El Sql Database es el servicio de Azure que utilizamos para manejar la base de datos del cliente

La base de datos se le asigna un log analytics para guardar los logs y las métricas de la base de datos

Con este servicio básicamente creamos dos recursos más:

* Un servidor de sql, básicamente es para mantener la base de datos (es la infraestructura donde corre la base de datos)
* Creamos una regla en el firewall del servidor de Sql para permitir que los servicios de Azure pueden interactuar con el servidor (creamos este regla básicamente para permitir que los pipelines pueden trabajar con el servidor)

**Web App**

La funcionalidad del web app es muy similar al Function App, la única diferencia es que uno es solamente es una función y la otra es una aplicación web

En la aplicación web el usuario tiene que poner todos los ficheros necesarios para la aplicación web y también las dependencias que usa la aplicación web y también sus versiones.

El web app también se conecta a un log analytics para guardar los logs y las métricas como los otros recursos, también como el Function App, tiene el modo de ver los logs en directo activado, pero esta vez sin la necesidad de un Application Insights

Para correr el web app solamente hace falta un Service App Plan, no hace falta un Storage Account

**Log analytics Worksapce**

Los cuatro recursos que hemos mencionado anteriormente usen Log Analytics Worksapce, pero los cuatro usen el mismo Log Analytics y esto es para ahorrar en los gastos

**Ejemplo:**

El ejemplo creado para este proyecto es un ejemplo muy sencillo que usa todos los recursos creados en este proyecto (App Service Plan, Function app, Web app, Log Analytics Workspace, Application Insights, Azure SQL Database, Azure SQL Database Server, Azure Kubernetes Service, Azure Container Registry, Storage Account) y también enseña como conectar entre ellos.

El ejemplo consiste de un pequeño calculador online que calcular el cuadrado y la raíz cuadrada de un número y en función de si se tiene que calcular el cuadrado o la raíz cuadrada, se llama al Azure Function o al Azure Kubernetes Service.

La parte frontal de esta aplicación esta subida al Web App y consiste del código del la web app hecho con Python usando Flask y las tres plantillas que se usen (la plantilla principal, la plantilla del cuadrado y la plantilla de la raíz cuadrada)

El funcionamiento de la web app es el siguiente el usuario entra a la plantilla principal y allá hay dos botones uno es el botón de calcular el cuadrado y el otro es el botón de la raíz cuadrada, en función si el usuario clica a un botón o a otro se mandara a una plantilla o a otra.

Si el usuario clica al botón de calcular el cuadrado, el usuario se mandara a la plantilla de exponential.html, que consiste de un formulario de un campo de input para que el usuario escribe el número que quiere conseguir su cuadrado y un botón, cuando el usuario clica al botón el número se mandara al Azure Kubernetes Servicey y el Azure Kubernetes Service lo que hará es buscar si este número está en la tabla de exponentiation y en caso de que sí. Lo devolverá al usuario y en caso de que no esta, el Azure Kubernetes Service se lo volverá a calcular y se lo mandara para guardar a la tabla de base de datos de exponentiation, y también se lo volverá al web app para mostrarlo al usuario.

El mismo proceso pasa con la raíz cuadrada, pero en lugar de mandar el número al Azure Kubernetes Service, se mandara al Function App, y la plantilla que se llamara esta vez es la squad.html

Para la web app las plantillas están dentro de la carpeta /content/webapp- content/templates y el código de la aplicación web que está hecho con Python estará en el /content/webapp-content/app.py y finalmente las dependencias que usa el web app está dentro del /content/webapp-content/requirements.txt

Para la base de datos Sql, la base de datos está definida dentro del fichero content\sql- content\database.sql

Para el Function App, el código está dentro del fichero content\function- content\HttpTrigger1\\_\_init\_\_.py y las dependencias están dentro de content\function- content\requirements.txt

Para el aks la definición de la imagen de Docker que usa la Aplicación está dentro del fichero content\aks-content\Dockerfile, la Aplicación que corre dentro de la imagen de Docker está dentro del fichero content\aks-content\app\app.py, y las dependencias que usa esta imagen está en el fichero content\aks-content\app\requirements.txt