

HowTo TP Redes de Información

Tema 9 - Grupo 22: Serverless (GraphQL)

Integrantes:

- Milagros Cornidez - 61432
- Santino Augusto Ranucci - 62092
- Agustín Zakalik - 62068

Consigna

- Diseñar una API utilizando GraphQL (serverless) con 3 endpoints, donde al menos debe haber 1 método de inserción (Mutation) y dos consultas (Query)
- Los requests deben manejar e interpretar todo tipo de errores.
- Configure funciones (AWS Lambda, Azure Functions, GCP Functions) que atiendan a las solicitudes de la API y procese los datos de entrada si aplica.
- Todos los datos que se reciban en la inserción (Mutation) deberán ser almacenados en una base de datos serverless.
- El body de la función que utilizaron para la Mutation deberá tener un valor "name". Caso contrario debe devolver el error.
- Cuando se hace una consulta (Query), deberá traer al menos un campo de la base de datos (dato que se almacenó allí por haber hecho previamente un POST).
- Mostrar gráficos con métricas en CloudWatch (o similar) de todos los componentes serverless.
- Explicar cómo escala el sistema anterior.
- Explique cuáles son las principales ventajas de usar GraphQL por sobre una API Rest
- Explicar cómo funciona y las ventajas de usar un servicio Serverless frente a un servidor común.
- Explique cómo utilizaría el servicio de Lambda (o similar) en otros dos escenarios que no sea con el fin de una API.
- ¿Cómo se podría configurar un proveedor de identidad como Cognito (o similar) para atender requests autenticados mediante token JWT? ¿Qué debe cambiar o qué debe tener en cuenta a la hora de implementar el cambio?

Idea principal

Para poder dar un ejemplo práctico de los aspectos de la consigna, decidimos hacer un modelado de una empresa de alquiler de autos.

Para ello, hicimos lo siguiente:

- Funcion serverless que expone API GraphQL
- Función serverless que hace web-scraping de Mercado Libre
- Función serverless que aplica transformaciones sobre CosmosDB

Herramientas utilizadas

Para el desarrollo del TP, se utilizó la cuenta educativa del cloud de Azure.

GraphQL

Herramientas:

- Node Js + Typescript como runtime
- Apollo Server como proveedor de servicios GraphQL
- Azure functions core framework
- Firebase como proveedor de autenticación.
- Node js (version 20.11.1) o superior
- Azure Cli

Como lo hicimos

Primero instalamos azure cli, la instalación varía según el sistema operativo

<https://learn.microsoft.com/en-us/cli/azure/install-azure-cli>

Despues instalamos el framework de azure functions local. Este se encuentra en el siguiente link de documentación de microsoft, la instalación varia según el sistema operativo usado:

<https://learn.microsoft.com/en-us/azure/azure-functions/functions-run-local?tabs=linux%2Cisolated-process%2Cnode-v4%2Cpython-v2%2Chttp-trigger%2Ccontainer-apps&pivots=programming-language-javascript>

NOTA: A partir de ahora, las directivas que se corrieron fueron sobre linux.

despues, a la hora de inicializar el proyecto, usamos el siguiente comando

```
santino@santino-HP-Pavilion-Laptop-15-eh1xxx:~/Documents/GitHub/TP-Redes$ func init graphql-cosmosdb-example --worker-runtime typescript --model v3
```

> func init graphql-cosmosdb-example --worker-runtime typescript --model v3

Una vez creado este proyecto, creamos 4 funciones dentro del mismo con el siguiente tipo de directivas

```
santino@santino-HP-Pavilion-Laptop-15-eh1xxx:~/Documents/GitHub/TP-Redes$ func new --template "Http Trigger" --name authUsers
```

Corremos este comando para generar 4 funciones distintas:

```
>func new --template "Http Trigger" --name authUsers
```

```
>func new --template "Http Trigger" --name graphql
```

```
>func new --template "Http Trigger" --name graphqlMutation
```

```
>func new --template "Http Trigger" --name scrapperHandler
```

Modificamos los archivos function.json de la siguiente manera

```
{
  "bindings": [
    {
      "authLevel": "function",
      "type": "httpTrigger",
      "direction": "in",
      "name": "req",
      "route": "authadmin",
      "methods": [
        "post"
      ]
    },
    {
      "type": "http",
      "direction": "out",
      "name": "$return"
    }
  ],
  "scriptFile": "../dist/graphqlmutation/index.js"
}
```

Lo importante es que tengan el **campo route definido**, para llegar a /api/<nombre ruta>

Por ejemplo, aca esta función se dispara cuando llega un pedido http a la ruta /api/authadmin, Además agregamos el **authLevel function**. Para que sea accesible por todos, la validación la manejaremos a nivel JWT. Adicionalmente, agregamos en **name: \$return** para poder introspectar las operaciones GraphQL con postman.

Tras esto copiamos el siguiente package json

```
{
  "name": "",
  "version": "1.0.0",
  "description": "",
```

```

"scripts": {
  "build": "tsc",
  "watch": "tsc -w",
  "prestart": "npm run build",
  "start": "func start --typescript",
  "test": "echo \"No tests yet...\""
},
"dependencies": {
  "@azure/cosmos": "^3.17.3",
  "@azure/functions": "^3.0.0",
  "@types/axios": "^0.14.0",
  "apollo-datasource-cosmosdb": "^1.1.0",
  "apollo-server-azure-functions": "^3.13.0",
  "axios": "^1.7.2",
  "firebase": "^10.12.2",
  "firebase-admin": "^12.1.1",
  "graphql": "^16.8.1",
  "typescript": "^4.0.0"
},
"devDependencies": {
  "@azure/functions": "^3.0.0",
  "@types/node": "18.x",
  "azure-functions-core-tools": "^4.x",
  "typescript": "^4.0.0"
}
}

```

Y corremos

> npm i

Para descargar las dependencias. Esto ya permite levantar el proyecto. Sin tener codeadas las funciones deberíamos poder ver una pantalla similar a esta según el nombre que le hayamos asignado a las funciones en cuestión.

```

Functions:

  authUsers: [GET,POST] http://localhost:7071/api/authusers
  blobTriggerFunction: [POST] http://localhost:7071/api/processBlob
  graphql: [GET,POST,OPTIONS] http://localhost:7071/api/users
  graphqlmutation: [POST] http://localhost:7071/api/authadmin
  scraperHandler: [GET,POST] http://localhost:7071/api/scraperHandler

For detailed output, run func with --verbose flag.

```

Una vez que esto funciona, creamos cada función. Escribimos cada función en el archivo index.ts

Y por cada azure function definimos un schema

```
▼ authUsers
  {} function.json
  TS index.ts
  TS schema.ts
```

El schema define los modelos GraphQL.

El código se encuentra en el repositorio. Pero lo más importante es lo siguiente

1 - Exportamos como módulo default el apollo server, con los datasources correspondientes.

```
// Create our server.
const server = new ApolloServer({
  typeDefs,
  resolvers,
  dataSources: () => ({
    car: buildCosmosDataSource<Car>("cars"),
    user: buildCosmosDataSource<User>("users"),
  }),
});

export default server.createHandler();
```

```
const buildCosmosDataSource = <TData extends { id: string }>(
  containerId: string
) => {
  const client = new CosmosClient(process.env.COSMOS_CONNECTION_STRING);
  const container = client
    .database(process.env.COSMOS_DATABASE_NAME)
    .container(containerId);
  return new CosmosDataSource<TData, unknown>(container);
};
```

Y después, escribimos nuestras queries

```

const resolvers = {
  //DONE
  Query: {
    listFreeCars: async (_, params, context) => {

      const limit = params.limit ?? 10;
      const offset = params.offset ?? 10;
      const token = params.token;

      const auth = getAuth(app);

      const carClient = context.dataSources.car as CosmosDataSource<Car, unknown>

      try {
        //exception if token is invalid
        await auth.verifyIdToken(token);

        //
        const { resources } = await carClient.findManyByQuery({
          query: "SELECT * FROM c WHERE c.assignedToUserWithId = null OFFSET @offset LIMIT @limit",
          parameters: [{ name: "@offset", value: offset }, { name: "@limit", value: limit }]
        })

        return { status: 200, items: resources, itemCount: resources.length };
      } catch (err) {
        return { status: 401 };
      }
    }
  }
};

```

Y nuestras mutations

```

Mutation: {
  signUp: async (_, params, context) => {
    const { name, lastName, age, email, password } = params.user;

    const auth = getAuth(app);
    const userDbClient = context.dataSources.user as CosmosDataSource<User, unknown>
    try {
      const userRecord = await auth.createUser({ email, password })

      const newUser: User = { name, lastName, age, id: email, carId: null };

      try {
        await userDbClient.createOne(newUser);
      } catch (err) {
        await auth.deleteUser(userRecord.uid);
        return { success: false, message: "Error creating user", status: 500 };
      }

      return { success: true, message: "Successfully signed up", status: 201 };
      //store in database
    } catch (err) {
      return { success: false, message: err.errorInfo.message, status: 409 };
    }
  },
};

```

Para resolver tipos escribimos nuestros resolvers

```
User: userResolver,
```

Que podemos definirlos ahí mismo o importarlos de otro archivo como hicimos nosotros

```
export const userResolver = {
  car: async (parent, __, context) => {
    if (!parent.carId) {
      return null;
    }
    return context.dataSources.car.findOneById(parent.carId);
  },
}
```

Y finalmente definimos nuestros tipos graphql en schema.ts

```
export const typeDefs = gql`
  type Query {
    listFreeCars(limit: Int, offset: Int, token: String!): CarPage
    myData(token: String!): User
  }
  type Mutation {
    signUp(user: UserInp! ): MutationResponse!
    takeCar(id: String!, token: String!): MutationResponse!
    releaseCar(token: String!): MutationResponse!
  }
  type User {
    id: String
    name: String
    lastName: String
    age: Int
    car: Car # Return a Car object instead of ID
  }
  input SignInUserInp{
    email: String!
    password: String!
  }
  input UserInp {
    name: String!
    lastName: String!
    age: Int!
    email: String!
    password: String!
  }
`
```

Y esto es lo que le pasamos a nuestra instancia de apollo server

```
// Create our server.
const server = new ApolloServer({
  typeDefs,
  resolvers,
  dataSources: () => ({
    car: buildCosmosDataSource<Car>("cars"),
    user: buildCosmosDataSource<User>("users"),
  }),
});
```

Los modelos a persistir en la base de datos los definimos aparte, Los modelos a guardar en la base de datos son distintos de los modelos que se definen en el schema de ts, por lo tanto lo definimos como un archivo typescript aparte, en nuestro caso, está definido en la carpeta models, acá tenemos un ejemplo del modelo User que se persiste en la base de datos

```
els > TS user.ts > User
export interface User {
  id: string;
  name: string;
  lastName: string;
  age: number;
  carId: string;
}
```

NOTA: Todo este código de los schemas, resolvers y demás se encuentra en el repositorio. Aca se destacaron las cosas importantes de la estructura.

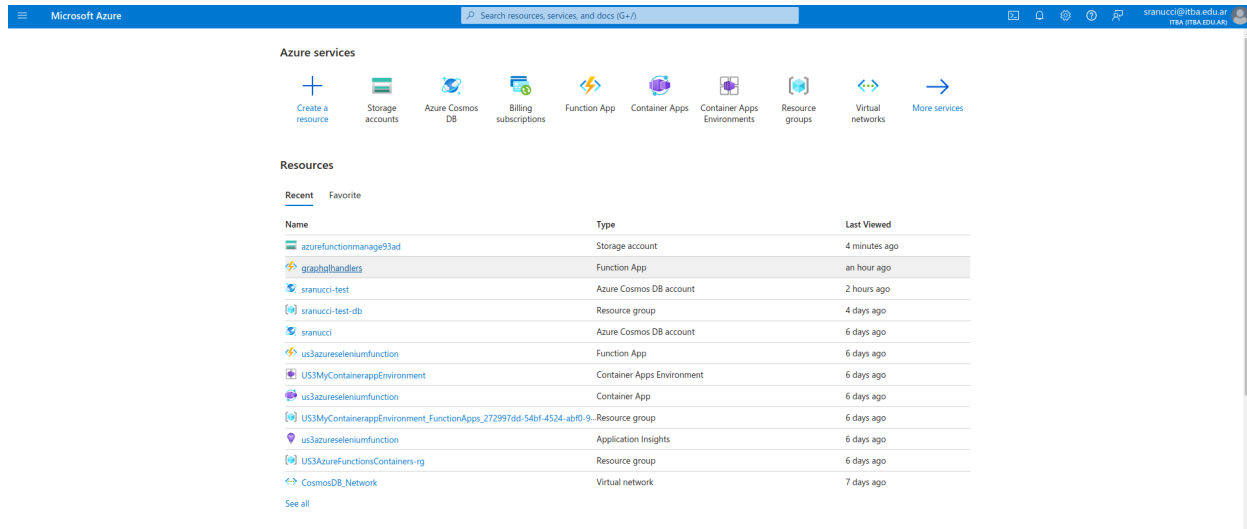
Ahora bien, para configurar GraphQL hay que primero crear una base de datos cosmos.

Para eso seguimos los siguientes pasos.

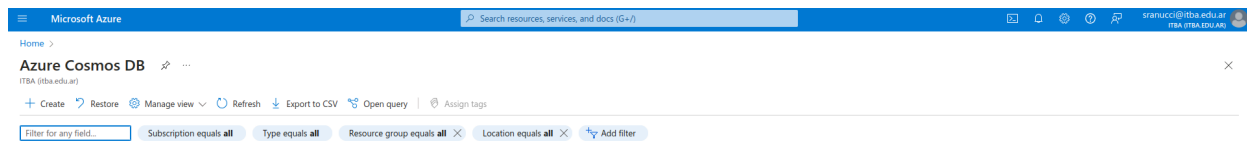
1- Nos registramos en azure

<https://azure.microsoft.com/en-us/get-started/azure-portal>

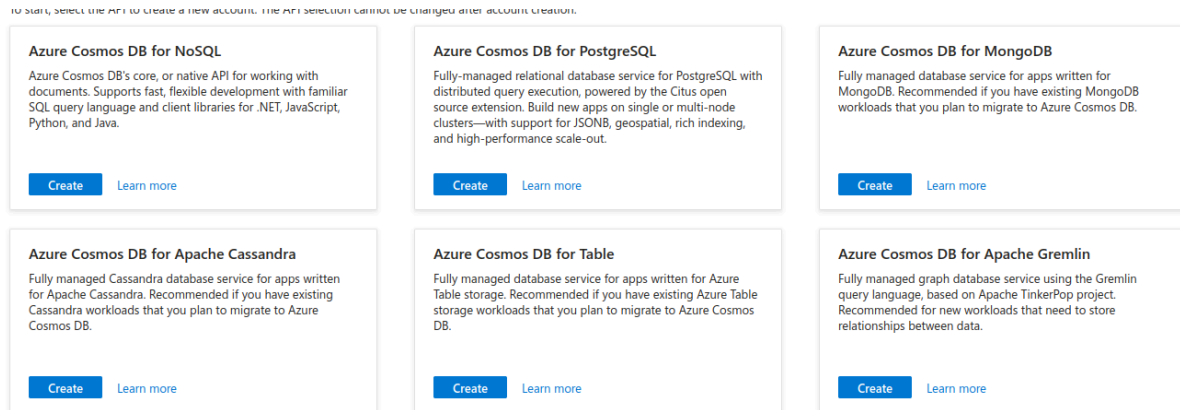
2- Entramos al dashboard, deberíamos ver una consola similar a esta (sin necesariamente todos los recursos que ya tengo creados).



Buscamos azure CosmosDB en el buscador y seleccionamos la modalidad NoSQL. Tocamos Create



Seleccionamos la primera opción, la que dice NoSQL



Después creamos un resource group y un account name, podemos elegir los nombres arbitrariamente. **Lo fundamental es poner la base de datos en modo serverless. Adicionalmente elegimos la region**

Microsoft Azure Search resources, services, and docs (G+)

Home > Azure Cosmos DB >

Create Azure Cosmos DB Account - Azure Cosmos DB for NoSQL

Basics Global distribution Networking Backup Policy Encryption Tags Review + create

Azure Cosmos DB is a fully managed NoSQL and relational database service for building scalable, high performance applications. [Try it for free](#), for 30 days with unlimited renewals. Go to production starting at \$24/month per database, multiple containers included. [Learn more](#)

Project Details
Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * Azure for Students
Resource Group * (New) rg1
[Create new](#)

Instance Details
Account Name * srnucuidatabase

Configure availability zone settings for your account. You cannot change these settings once the account is created.

Availability Zones ☐ Enable ☒ Disable

Location * (US) East US 2
Available locations are determined by your subscription's access and availability zone support (if that is enabled). If you don't see or cannot select your desired location, please open a support request for region access. [Click here for more details on how to create a region access request](#)

Capacity mode ☐ Provisioned throughput ☒ Serverless
[Learn more about capacity mode](#)

El siguiente paso lo dejamos como esta...

Microsoft Azure Search resources, services, and docs (G+)

Home > Azure Cosmos DB >

Create Azure Cosmos DB Account - Azure Cosmos DB for NoSQL

Basics Global distribution Networking Backup Policy Encryption Tags Review + create

Multiple regions are not supported with serverless capacity mode.

Geo-Redundancy ☐ Enable ☒ Disable

Multi-region Writes ☐ Enable ☒ Disable

En network por ahora lo dejamos publico, mas adelante creamos la Virtual Network

Microsoft Azure Search resources, services, and docs (G+)

Home > Azure Cosmos DB >

Create Azure Cosmos DB Account - Azure Cosmos DB for NoSQL

Basics Global distribution Networking Backup Policy Encryption Tags Review + create

Network connectivity
You can connect to your Azure Cosmos DB account either publically, via public IP addresses or service endpoints, or privately, using a private endpoint.

Connectivity method * ☒ All networks
☐ Public endpoint (selected networks)
☐ Private endpoint

All networks will be able to access this CosmosDB account. [Learn More](#)

Connection Security Settings
Minimum Transport Layer Security Protocol TLS 1.2

La backup policy la dejamos en las opciones default

Microsoft Azure Search resources, services, and docs (G+/)

Home > Azure Cosmos DB >

Create Azure Cosmos DB Account - Azure Cosmos DB for NoSQL

Basics Global distribution Networking **Backup Policy** Encryption Tags Review + create

Azure Cosmos DB provides three different backup policies. You will not be able to switch to Periodic mode once you adopt Continuous mode. [Learn more](#) about the differences of the backup policies and pricing details.

Backup policy ☐ Periodic
Backup is taken at periodic interval based on your configuration

☐ Continuous (7 days)
Provides backup window of 7 days / 168 hours and you can restore to any point of time within the window. This mode is available for free.

☐ Continuous (30 days)
Provides backup window of 30 days / 720 hours and you can restore to any point of time within the window. This mode has cost impact.

Backup interval 60-1440

Backup retention 8-720

Copies of data retained 2

Y lo mismo para la sección de encryption

Microsoft Azure Search resources, services, and docs (G+/)

Home > Azure Cosmos DB >

Create Azure Cosmos DB Account - Azure Cosmos DB for NoSQL

Basics Global distribution Networking Backup Policy **Encryption** Tags Review + create

Data Encryption

Azure Cosmos DB encryption protects your data at rest by seamlessly encrypting your data as it's written in our datacenters, and automatically decrypting it for you as you access it.

By default your Azure Cosmos DB account is encrypted at rest using service-managed keys. At the moment, you will not be able to switch back to service-managed key after opting into using custom-managed key while creating your account. [Learn More](#)

Data Encryption * ☒ Service-managed key
☐ Customer-managed key (CMK)

No creamos ningun tag

Microsoft Azure Search resources, services, and docs (G+/)

Home > Azure Cosmos DB >

Create Azure Cosmos DB Account - Azure Cosmos DB for NoSQL

Basics Global distribution Networking Backup Policy Encryption **Tags** Review + create

Tags are name/value pairs that enable you to categorize resources and view consolidated billing by applying the same tag to multiple resources and resource groups. [learn more](#)

Note that if you create tags and then change resource settings on other tabs, your tags will be automatically updated.

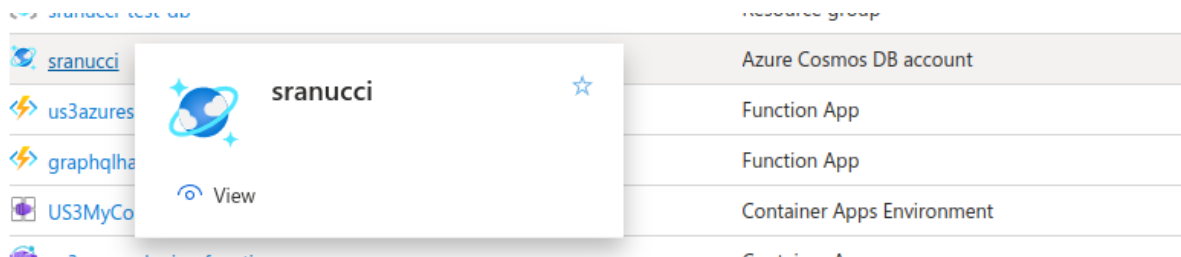
<input type="checkbox"/> Key	Value	Resource Type
<input type="text"/>	<input type="text"/>	Azure Cosmos DB account

Y ya tocamos la opción de create

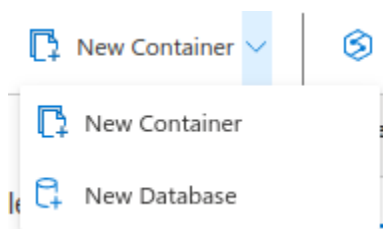
Esto levantará la base de datos de azure cosmos.

Creación de base de datos

Vamos al recurso recién creado



Seleccionamos en el tab de new database



Y la llamamos **exampleapp**

Una vez creada la base de datos. Debemos crear 2 containers para alojar los datos de la base de datos.

Primero creamos container de autos

* Database id ⓘ

☐ Create new ☒ Use existing

exampleapp ▼

* Container id ⓘ

cars

* Partition key ⓘ

/location

Add hierarchical partition key

Unique keys ⓘ

+ Add unique key

Analytical store ⓘ

☐ On ☒ Off

Azure Synapse Link is required for creating an analytical store container. Enable Synapse Link for this Cosmos DB account. [Learn more](#)

Enable

> Advanced

OK

Después creamos container de usuarios

New Container



* Database id ⓘ

☐ Create new ☒ Use existing

exampleapp



* Container id ⓘ

users

* Partition key ⓘ

/id

Add hierarchical partition key

Unique keys ⓘ

+ Add unique key

Analytical store ⓘ

☐ On ☒ Off

Azure Synapse Link is required for creating an analytical store container. Enable Synapse Link for this Cosmos DB account. [Learn more](#)

Enable

> Advanced

OK

Despues creamos el container de leases

New Container



* Database id ⓘ

☐ Create new ☒ Use existing

exampleapp



* Container id ⓘ

leases

* Partition key ⓘ

/id

Add hierarchical partition key

Unique keys ⓘ

+ Add unique key

Analytical store ⓘ

☐ On ☒ Off

Azure Synapse Link is required for creating an analytical store container. Enable Synapse Link for this Cosmos DB account. [Learn more](#)

Enable

> Advanced

OK

Y finalmente creamos el container maxPrices

New Container



* Database id ⓘ

☐ Create new ☒ Use existing

exampleapp



* Container id ⓘ

maxPrices

* Partition key ⓘ

/id

Add hierarchical partition key

Unique keys ⓘ

+ Add unique key

Analytical store ⓘ

☐ On ☒ Off

Azure Synapse Link is required for creating an analytical store container. Enable Synapse Link for this Cosmos DB account. [Learn more](#)

Enable

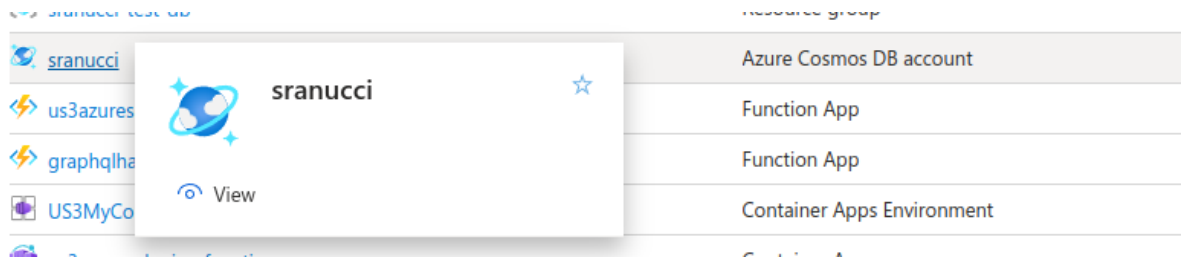
> Advanced

OK

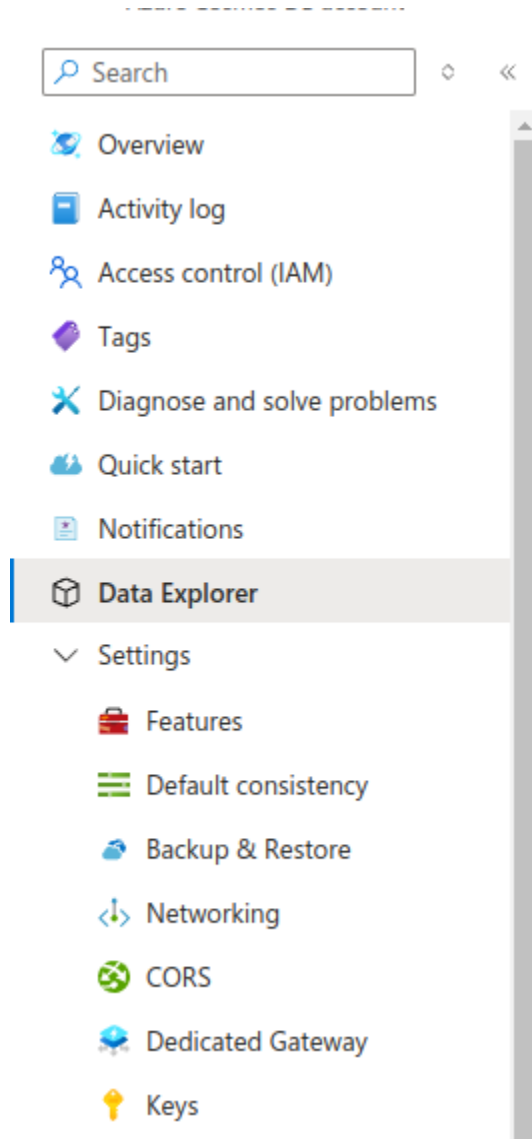
Teniendo esto creado, pasamos a conectarnos a la base de datos

Conexión a la base de datos

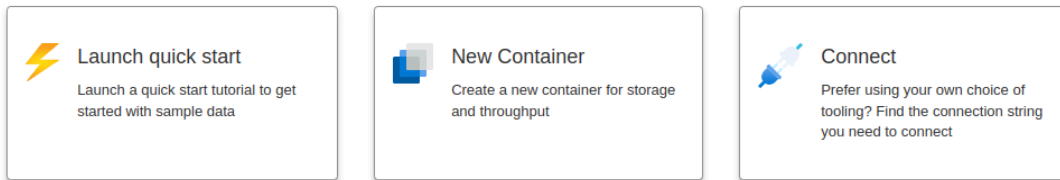
Para conectarnos necesitamos un connectionString. Vamos al recurso recién creado desde nuestro dashboard



Navegamos a data explorer



Y tocamos connect



Alli aparecera una pestaña y copiamos el primary connection string

Ese connectionString lo copiamos a nuestro archivo **localsettings.json** con ese nombre de variable de entorno

```
{
  "IsEncrypted": false,
  "Values": {
    "FUNCTIONS_WORKER_RUNTIME": "node",
    "AzureWebJobsStorage": "",
    "COSMOS_CONNECTION_STRING": "AccountE",
    "COSMOS_DATABASE_NAME": "..."
  }
}
```

Creacion de proveedor de auth: Firebase

Primero nos registramos en la consola de firebase

<https://firebase.google.com/>

Seleccionamos Authentication desde el menu



Descripción gen...



IA generativa



Build with Gem...

NUEVO

Novedades



App Hosting

NUEVO



Data Connect

NUEVO

Categorías de producto

Compilación

App Check



App Hosting

NUEVO



Authentication



Data Connect

NUEVO



Extensions



Firestore Database



Functions



Hosting



Machine Learning



Realtime Database



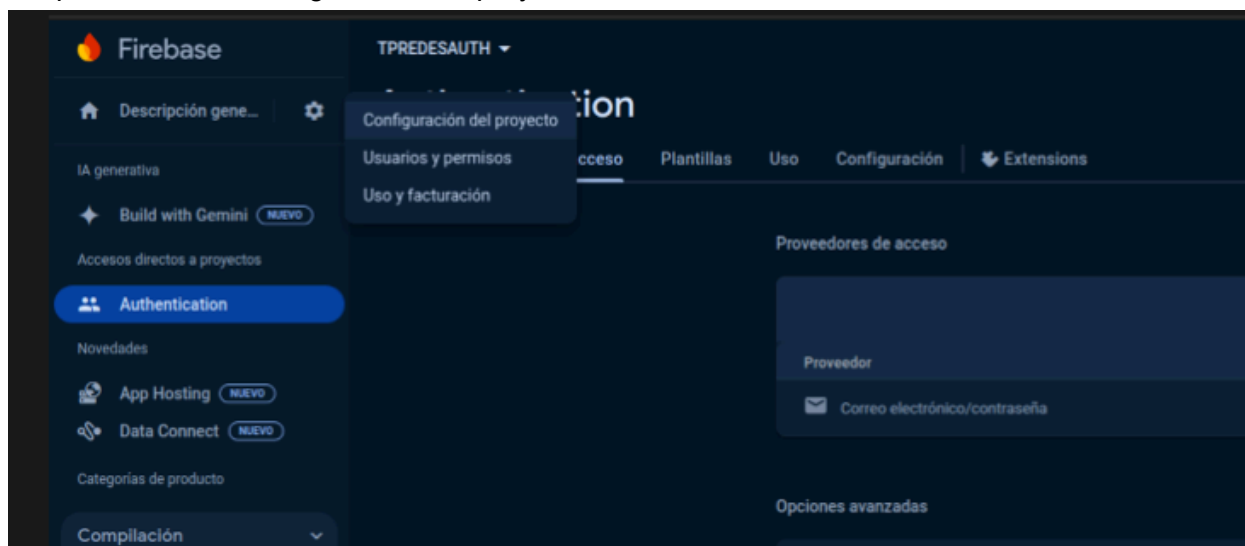
Storage

Ejecución

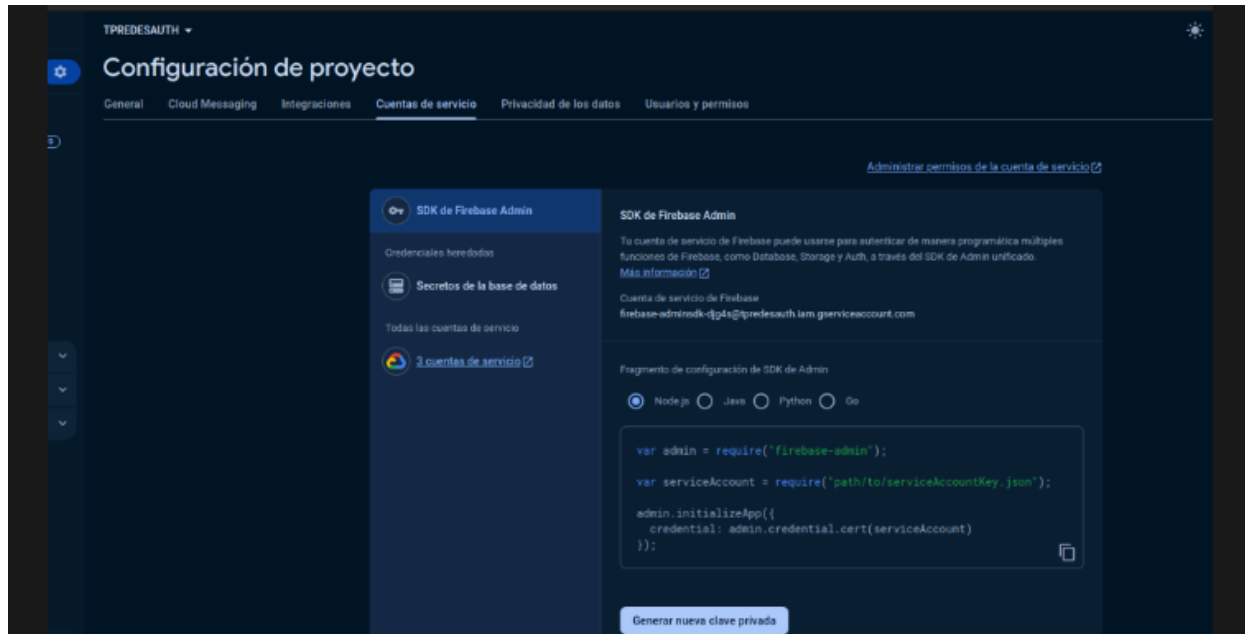
Agregamos correo electrónico + contraseña



Despues vamos a configuración del proyecto

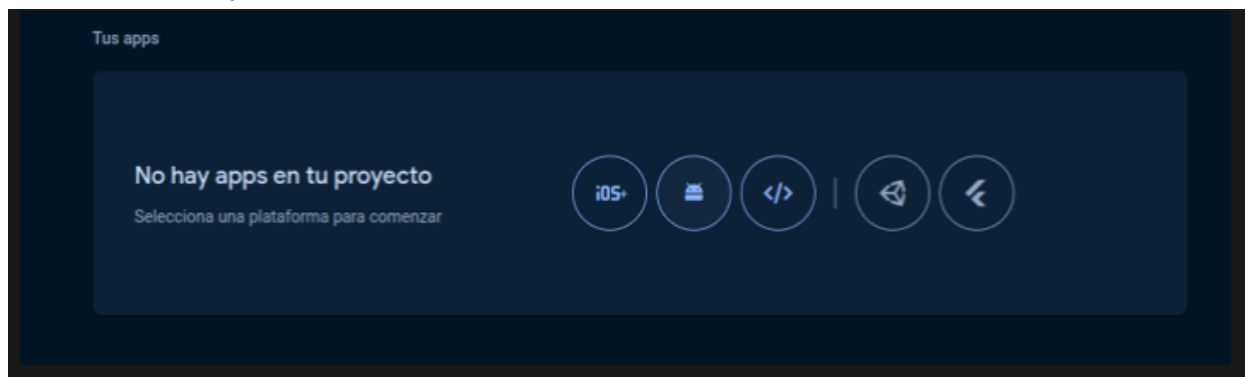


Elegimos la opcion de node js y tocamos generar clave privada



Esta clave privada es fundamental para la autenticación de firebase. La vamos a colocar como variable de entorno del proyecto.

Si abrimos el archivo que se descargó, vemos que es un json. Después navegamos a la pestaña de apps y creamos una app con el nombre que queramos



Volviendo al archivo descargado, tenemos que pasarlo por alguna herramienta para que lo transforme a string, un ejemplo podría ser la siguiente

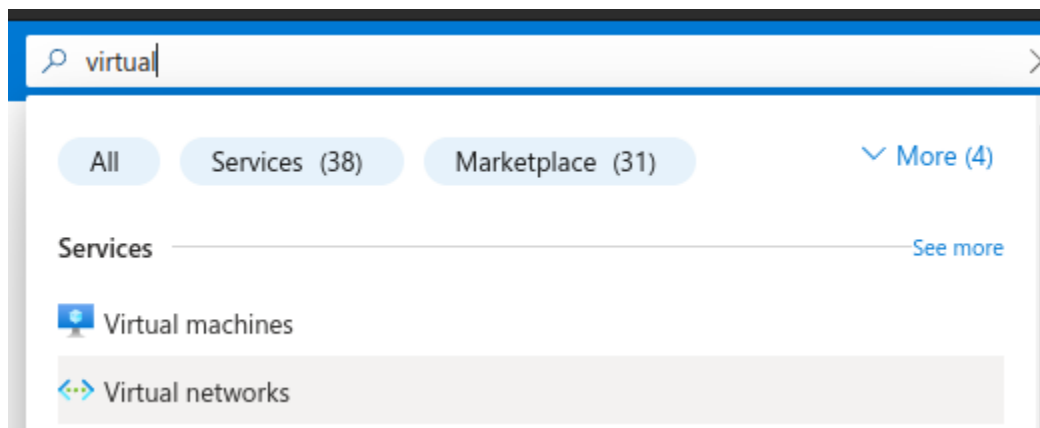
<https://jsonformatter.org/json-stringify-online>

Tras hacer eso, pegamos el string resultante como variable de entorno

```
1  {
2    "IsEncrypted": false,
3    "Values": {
4      "FUNCTIONS_WORKER_RUNTIME": "node",
5      "AzureWebJobsStorage": "",
6      "COSMOS_CONNECTION_STRING": "AccountEnd
7      "COSMOS_DATABASE_NAME": "exampleapp",
8      "SCRAPPER_SECRET": "SCRAPPERSECRET",
9      "FIREBASE_SECRET": "{\n  \"type\": \"se
10 }
```

Creación de virtual network

Buscamos virtual network en el buscador de azure



Seleccionamos un resource group y nombre arbitrario

Create virtual network ...

Basics Security IP addresses Tags Review + create

Azure Virtual Network (VNet) is the fundamental building block for your private network in Azure. VNet enables many types of Azure resources, such as Azure Virtual Machines (VM), to securely communicate with each other, the internet, and on-premises networks. VNet is similar to a traditional network that you'd operate in your own data center, but brings with it additional benefits of Azure's infrastructure such as scale, availability, and isolation.

[Learn more.](#) 

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription *	<div>Azure for Students</div>
Resource group *	<div>AzureFunctionManager</div>

[Create new](#)

Instance details

Virtual network name *	<div>CosmosDb_Network</div>
Region * ⓘ	<div>(US) East US 2</div>

[Deploy to an Azure Extended Zone](#)

IMPORTANTE: La region debe matchear con donde pusimos la base de datos cosmos

No tocamos nada de security

Basics **Security** IP addresses Tags Review + create

Enable Virtual network encryption to encrypt traffic traveling within the virtual network. Virtual machines must have accelerated networking enabled. Traffic to public IP addresses is not encrypted. [Learn more.](#)

Virtual network encryption ☐

Azure Bastion

Azure Bastion is a paid service that provides secure RDP/SSH connectivity to your virtual machines over TLS. When you connect via Azure Bastion, your virtual machines do not need a public IP address. [Learn more.](#)

Enable Azure Bastion ☐

Azure Firewall

Azure Firewall is a managed cloud-based network security service that protects your Azure Virtual Network resources. [Learn more.](#)

Enable Azure Firewall ☐

Azure DDoS Network Protection

Azure DDoS Network Protection is a paid service that offers enhanced DDoS mitigation capabilities via adaptive tuning, attack notification, and telemetry to protect against the impacts of a DDoS attack for all protected resources within this virtual network. [Learn more.](#)

Aca para las subnets aparecen las siguientes opciones

Basics **Security** **IP addresses** Tags Review + create

Configure your virtual network address space with the IPv4 and IPv6 addresses and subnets you need. [Learn more.](#)

Define the address space of your virtual network with one or more IPv4 or IPv6 address ranges. Create subnets to segment the virtual network address space into smaller ranges for use by your applications. When you deploy resources into a subnet, Azure assigns the resource an IP address from the subnet. [Learn more.](#)

Add IPv4 address space | ▾

10.0.0.0/16

10.0.0.0

/16 ▾

10.0.0.0 - 10.0.255.255 65,536 addresses

+ Add a subnet

Subnets	IP address range	Size	NAT gateway
default	10.0.0.0 - 10.0.0.255	/24 (256 addresses)	-

Debemos crear una subnet privada para cosmos. Lo hacemos tocando add subnet

Select an address space and configure your subnet. You can customize a default subnet or select from subnet templates if you plan to add select services later. [Learn more](#)

Subnet purpose ⓘ	Default
Name * ⓘ	CosmosDB_INTERNAL
IPv4	
Include an IPv4 address space	<input checked="" type="checkbox"/>
IPv4 address range * ⓘ	10.0.0.0/16 10.0.0.0 - 10.0.255.255
Starting address * ⓘ	10.0.1.0
Size ⓘ	/24 (256 addresses)
Subnet address range ⓘ	10.0.1.0 - 10.0.1.255
IPv6	
Include an IPv6 address space	<input type="checkbox"/> This virtual network has no IPv6 address ranges.
Private subnet PREVIEW	
Private subnets enhance security by not providing default outbound access. To enable outbound connectivity for virtual machines to access the internet, it is necessary to explicitly grant outbound access. A NAT gateway is the recommended way to provide outbound connectivity for virtual machines in the subnet. Learn more	
Enable private subnet (no default outbound access)	<input checked="" type="checkbox"/>

ⓘ This setting can't be changed after the subnet is created

Es importante que la subnet este marcada como privada. De lo contrario estaríamos exponiendo la base de datos para acceso publico.

Tras crear esa subnet pasamos al siguiente paso.

En tags lo dejamos como esta

[Home](#) > [Virtual networks](#) >

Create virtual network ...

Basics Security IP addresses **Tags** Review + create

Tags are name/value pairs that enable you to categorize resources and view consolidated billing by applying the same tag to multiple resources and resource groups. [Learn more about tags](#)

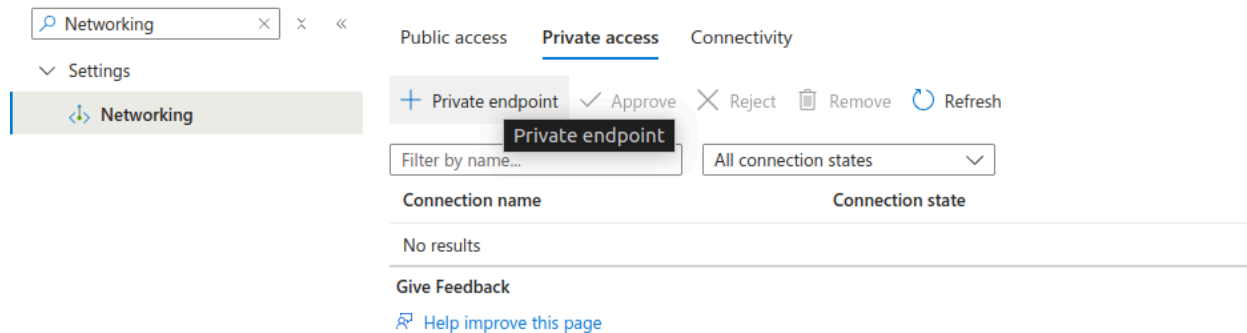
Note that if you create tags and then change resource settings on other tabs, your tags will be automatically updated.

Name	Value	Resource
<input type="text"/>	:	<input type="text"/> All resources selected

Y después le damos a create.

Colocación de base de datos cosmos en subred privada

En nuestra base de datos cosmos, buscamos networking y vamos a la sección de private y tocamos para crear un nuevo private endpoint



La primera sección le ponemos los nombres que deseemos, eligiendo el resource group y nombre de private endpoint adecuado

1 Basics 2 Resource 3 Virtual Network 4 DNS 5 Tags 6 Review + create

Use private endpoints to privately connect to a service or resource. Your private endpoint must be in the same region as your virtual network, but can be in a different region from the private link resource that you are connecting to. [Learn more](#)

Project details

Subscription * ⓘ Azure for Students

Resource group * ⓘ sranucci-test-db
[Create new](#)

Instance details

Name * some ✓

Network Interface Name * some-nic ✓

Region * East US 2

Seleccionamos DatabaseAccounts en el siguiente menú y vemos que nos aparecen dropdowns con nuestras bases de datos, en nuestro caso tenemos 2 y elegimos la que queremos poner, hagámoslo con test

✓ Basics **2 Resource** 3 Virtual Network 4 DNS 5 Tags 6 Review + create

Private Link offers options to create private endpoints for different Azure resources, like your private link service, a SQL server, or an Azure storage account. Select which resource you would like to connect to using this private endpoint. [Learn more](#)

Connection method ①

☒ Connect to an Azure resource in my directory.

☐ Connect to an Azure resource by resource ID or alias.

Subscription * ①

Azure for Students

Resource type * ①

Microsoft.AzureCosmosDB/databaseAccounts

Resource * ①

Select Azure private link resource

Target sub-resource ①

TP-REDES

sranucci

sranucci-test-db

sranucci-test

La siguiente opción la dejamos como esta

✓ Basics ✓ Resource **3 Virtual Network** 4 DNS 5 Tags 6 Review + create

Networking

To deploy the private endpoint, select a virtual network subnet. [Learn more](#)

Virtual network ①

CosmosDB_Network (NetworkUtils)

Subnet * ①

cosmosDB_INTERNAL

Network policy for private endpoints Disabled ([edit](#))

Private IP configuration

☒ Dynamically allocate IP address

☐ Statically allocate IP address

Application security group

Configure network security as a natural extension of an application's structure. ASG allows you to group virtual machines and define network security policies based on those groups. You can specify an application security group as the source or destination in an NSG security rule. [Learn more](#)

+ Create

Application security group

Y vamos a DNS, esto también lo dejamos como esta

✓ Basics ✓ Resource ✓ Virtual Network **4 DNS** 5 Tags 6 Review + create

Private DNS integration

To connect privately with your private endpoint, you need a DNS record. We recommend that you integrate your private endpoint with a private DNS zone. You can also utilize your own DNS servers or create DNS records using the host files on your virtual machines. [Learn more](#)

Integrate with private DNS zone ☒ Yes ☐ No

Configuration name	Subscription	Resource group	Private DNS zone
privatelink-documents-az...	Azure for Students	NetworkUtils	privatelink.documents.az...

i Existing Private DNS Zones tied to a single service should not be associated with two different Private Endpoints as it will not be possible to properly resolve two different A-Records that point to the same service. However, Private DNS Zones tied to multiple services would not face this resolution constraint.

La seccion de tags también la dejamos como esta

✓ Basics ✓ Resource ✓ Virtual Network ✓ DNS **5 Tags** 6 Review + create

Tags are name/value pairs that enable you to categorize resources and view consolidated billing by applying the same tag to multiple resources and resource groups. [Learn more about tags](#)

Note that if you create tags and then change resource settings on other tabs, your tags will be automatically updated.

Name	Value	Resource
	:	2 selected

Y ya esta! Le damos a create en la siguiente sección.

Creacion de azure functions

Para la creación de las azure functions vamos al dashboard del portal de azure y buscamos functionapp.

NOTA: Es necesario elegir el plan premium, sino no se tendrá acceso a las funcionalidades de redes virtuales. El plan flex tambien lo permite pero se encuentra en beta y puede traer problemas. Como no deployar correctamente, etc...

Después tocamos las siguientes opciones

Create Function App (Functions Premium) ...

Project Details

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ

Azure for Students

Resource Group * ⓘ

(New) finalfunctionapp

Create new

Instance Details

Function App name *

graphqlhandlerstpredes

azurewebsites.net

Do you want to deploy code or container image? *

☒ Code ☐ Container Image

Runtime stack *

Node.js

Version *

20 LTS

Region *

East US 2

ⓘ

 Not finding your App Service Plan? Try a different region or select your App Service Environment.

Operating System *

☒ Linux ☐ Windows

Environment details

Linux Plan (East US 2) * ⓘ

(New) ASP-finalfunctionapp-86e4

Create new

Pricing plan


Basic B1 (100 total ACU, 1.75 GB memory, 1 vCPU)

[Home](#) > [Function App](#) > [Create Function App](#) >


Create Function App (Functions Premium) ...

Basics **Storage** Networking Monitoring Deployment Tags Review + create

Storage


When creating a function app, you must create or link to a general-purpose Azure Storage account that supports Blobs, Queue, and Table storage. [Learn more](#) 

Storage account *

(New) finalfunctionappa35b 

[Create new](#)

Diagnostic Settings


The storage account associated with the function app stores important app data. You may wish to enable monitoring for the account. You can quickly configure basic diagnostic settings as you create the function app, or you can fully customize the diagnostics settings on the storage account resource after creation. [Learn more](#) 

Blob service diagnostic settings ☒ **Don't configure diagnostic settings now** You can configure diagnostic settings later from the storage account resource. Choose this if you want full control over log destinations, retention policies, and which logs and metrics are configured.

☐ **Configure basic diagnostic settings now** Configure Azure Log Analytics with StorageWrite logs and Transaction metrics for the blob service. You can modify your diagnostic settings later from the storage account resource.

[Home](#) > [Function App](#) > [Create Function App](#) >

Create Function App (Functions Premium) ...

Subnet must be in the chosen virtual network. Subnets are isolated by virtual network, and you cannot reach into a virtual network. These aspects can also be changed after the app is provisioned. [Learn more](#) 


Enable public access * ☐ On ☒ Off

Enable network injection * ☒ On ☐ Off

Virtual Network

Select or create a virtual network that is in the same region as your new app.

Virtual Network *

CosmosDB_Network (NetworkUtils) 

[Create new](#)

Inbound access

Select a subnet in the chosen virtual network to place a private endpoint in. Private endpoints enable secure inbound access from only the chosen virtual network. When enabled, your app will not be accessible from the internet.

 Private endpoints are disabled if public access is enabled.

Enable private endpoints ☐ On ☒ Off

Outbound access

Select a subnet in the chosen virtual network to be used for integration. VNet Integration enables your app to make calls into the chosen virtual network. You can also put network security groups or route tables on this subnet to control all outbound traffic from your function app.

Enable VNet integration * ☒ On ☐ Off

Outbound subnet *

(New) azurefnspublic 

[Create new](#)

Aca creamos una subred! Llamada azurefnspublic.
Y aca dejamos todo como esta

[Home](#) > [Function App](#) > [Create Function App](#) >

Create Function App (Functions Premium) ...

Basics Storage Networking **Monitoring** Deployment Tags Review + create

Azure Monitor application insights is an Application Performance Management (APM) service for developers and DevOps professionals. Enable it below to automatically monitor your application. It will detect performance anomalies, and includes powerful analytics tools to help you diagnose issues and to understand what users actually do with your app. Your bill is based on amount of data used by Application Insights and your data retention settings. [Learn more](#)

[App Insights pricing](#)

Application Insights

Enable Application Insights *

☐ No ☒ Yes

Application Insights *

(New) graphqlhandlerstpredes (East US 2) ▾

[Create new](#)

Region

East US 2

Aca tambien dejamos todo como esta

[Home](#) > [Function App](#) > [Create Function App](#) >

Create Function App (Functions Premium) ...

Continuous deployment settings

Set up continuous deployment to easily deploy code from your GitHub repository via GitHub Actions. [Learn more](#)

Continuous deployment

☒ Disable ☐ Enable

GitHub settings

Set up GitHub Actions to push content to your app whenever there are code changes made to your repository. Note: Your GitHub account must have write access to the selected repository in order to add a workflow file which manages deployments to your app.

GitHub account

Authorize

Organization

Select organization ▾

Repository

Select repository ▾

Branch

Select branch ▾

Workflow configuration

Click the button below to preview what the GitHub Actions workflow file will look like before setting up continuous deployment.

i Complete the Basics tab and the form above to preview the GitHub Actions workflow file.

Preview file

Authentication settings

Choose if you would like to allow basic authentication to deploy code to your app. [Learn more](#)

Basic authentication

☒ Disable ☐ Enable

Aca tambien

Create Function App (Functions Premium) ...

Basics Storage Networking Monitoring Deployment **Tags** Review + create

Tags are name/value pairs that enable you to categorize resources and view consolidated billing by applying the same tag to multiple resources and resource groups.

Note that if you create tags and then change resource settings on other tabs, your tags will be automatically updated.

Name ⓘ	Value ⓘ	Resource
<input type="text"/>	: <input type="text"/>	5 selected

Y después le damos a create

Para subir las funciones a esta function app, desde el directorio raíz del proyecto ejecutamos este comando

```
To https://github.com/azakalik/TP-Redes.git
81f45aa..8c9e745 maxprices -> maxprices
santino@santino-HP-Pavilion-Laptop-15-e9hxxr:~/Documents/GitHub/TP-Redes$ func azure functionapp publish <nombre function app> --publish-local-settings --typescript
```

```
>func azure functionapp publish graphqlhandlertpredes --publish-local-settings --typescript
```

En nuestro caso.

Nota: En caso de usar nuestro código, se puede notar que en localsettings.json está la siguiente variable

Tras correr estos pasos suponiendo que se usó nuestro repositorio y no funciones propias estamos listos para ejecutar las azure functions.

```
5 "AzureWebJobsStorage": "UseDevelopmentStorage=true",
6 "COSMOS_CONNECTION_STRING": "azureEndpoint=https://sranucci.documents.azure.com:443/;AccountKey=j0SA3HiLcHSDlv5u00RzCqkiKoK65B3301JH714aUUQNZ4jH5874Bzgsfwj"
7 "COSMOS_DATABASE_NAME": "exampleapp",
8 "SCRAPPER_SECRET": "SCRAPPERSECRET",
9 "ETBREFSE_SECRET": "{\n  \"type\": \"service account\",\n  \"project id\": \"f0redesauth\",\n  \"private key id\": \"e2d1clcf6a5aeebf05813af2150de357f190fa7\"
```

El scrapper secret es un parámetro de header http que se pasa para aceptar el scraping, es una clave simétrica que se usa para imitar el comportamiento de azure function keys, es una funcionalidad que con la cuenta de estudiantes no podíamos acceder.

Uso de cliente para azure function autenticada (/api/authusers)

Los requests a este endpoint requieren del envío de un token JWT, para ello se brinda un cliente js encontrado en la carpeta client.

Para usarlo, primero nos movemos a la carpeta client

```
santino@santino-HP-Pavilion-Laptop-15-eh1xxx:~/Documents/GitHub/TP-Redes/clients$
```

Después modificamos el archivo .env con las credenciales que hayamos usado para registrarnos en el endpoint /api/authusers con el método GraphQL signup.

```
1 EMAIL=sranucci1@example.com
2 PASSWORD=sranucci1
```

Finalmente corremos el programa

```
santino@santino-HP-Pavilion-Laptop-15-eh1xxx:~/Documents/GitHub/TP-Redes/clients$ node --env-file=.env signin.js
```

```
>node --env-file=.env signin.js
```

Se habrá generado un archivo que posee el JWT en cuestión



Simplemente copiamos de ahí el JWT y listo.

Web-Scraper

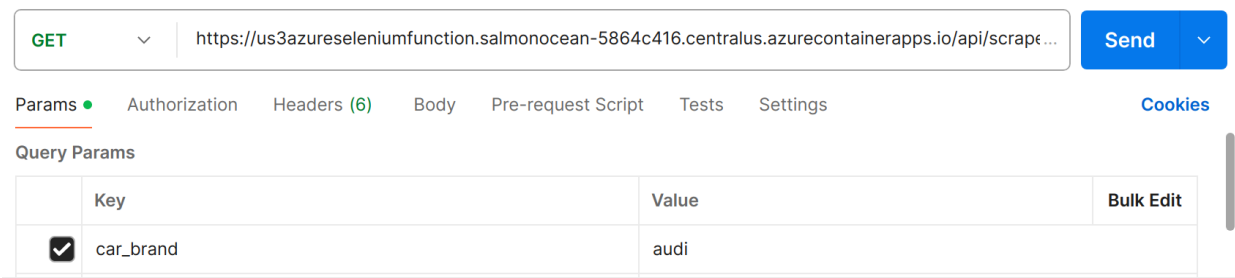
Idea

El web-scraper consiste de una función serverless que utiliza Selenium para extraer datos de publicaciones de autos en Mercado Libre.

Recibe un solo parámetro, car_brand, que puede ser por ejemplo “audi”. El valor de retorno es un JSON que contiene una serie de elementos clave-valor, en donde la clave es el ID único de publicación en Mercado Libre, y el valor es información útil de la publicación, como título, precio, link a la imagen, descripción, etc. La función también guarda en una base de datos CosmosDB los resultados obtenidos, que luego serán usados por GraphQL para responder a las queries.

Ejemplo de invocación

Acá se puede ver un ejemplo de llamado a la función en Postman:



Y esto es una parte de la respuesta obtenida:

```
{
  "1786039248": {
    "title": "Audi Q3 2.0 Tfsi Stronic Quattro 220cv",
    "url": "https://auto.mercadolibre.com.ar/MLA-1786039248-audi-qe-as-_JM#position=1&search_layout=grid&type=item&tracking_id=1d6ec15f-c26f-431a-9d4a-a1a37d90052e",
    "price": 31900000,
    "km": 83000,
    "year": 2018,
    "location": "Vicente López - Bs.As. G.B.A. Norte",
    "img_url": "https://http2.mlstatic.com/D_NQ_NP_745477-MLA76623267667_052024-W.webp"
  },
  "1785219312": {
    "title": "Audi Q5 Sportback 2.0 45tfsi Quattro S Tronic Advanced 245cv",
    "url": "https://auto.mercadolibre.com.ar/MLA-1785219312-audi-q5-sportback-advanced-45tfsi-245cv-quattro-q3-q7-q8-a5-_JM#position=2&search_layout=grid&type=item&tracking_id=1d6ec15f-c26f-431a-9d4a-a1a37d90052e",
  },
}
```

Como lo hicimos

Lo primero a tener en cuenta es que no se puede utilizar una función de Azure básica con algún runtime (de Python o Node por ejemplo). Esto se debe a que selenium necesita de dos binarios para poder funcionar, que son versiones específicas de Chrome y Chromedriver.

La única forma de asegurar la existencia de este binario en runtime es utilizando un tipo particular de función serverless, que utiliza una docker image creada por nosotros. Luego, es necesario subir el docker image a un registry público, y finalmente crear una función serverless en Azure y enlazarla a dicha imagen. A continuación, explicaremos en más detalle como hacer todas estas cosas.

Paso 1: Dockerfile

Como mencionamos previamente, nuestra función scraper hace uso de una docker image para asegurar la existencia de las dependencias necesarias. El Dockerfile utilizado es este:

```

# To enable ssh & remote debugging on app service change the base image to the one below
# FROM mcr.microsoft.com/azure-functions/python:4-python3.10-appservice
FROM mcr.microsoft.com/azure-functions/python:4-python3.10

ENV AzureWebJobsScriptRoot=/home/site/wwwroot \
    AzureFunctionsJobHost__Logging__Console__IsEnabled=true

# 1. Install essential packages
RUN apt-get update \
    && apt-get install -y \
        git \
        wget \
        curl \
        unzip \
        cmake \
        unixodbc-dev \
        build-essential \
    && rm -rf /var/lib/apt/lists/*
RUN apt-get update && apt-get upgrade && apt-get -y install libnss3-dev libgdk-pixbuf2.0-dev libgtk-3-dev libxss-dev

# 2. Install Chrome and Chromedriver
RUN \
    curl -Lo "/tmp/chromedriver-linux64.zip" "https://storage.googleapis.com/chrome-for-testing-public/123.0.6312.105/linux64/chromedriver-linux64.zip" && \
    curl -Lo "/tmp/chrome-linux64.zip" "https://storage.googleapis.com/chrome-for-testing-public/123.0.6312.105/linux64/chrome-linux64.zip" && \
    unzip /tmp/chromedriver-linux64.zip -d /opt/ && \
    unzip /tmp/chrome-linux64.zip -d /opt/

# 3. Install python requirements
COPY requirements.txt /
RUN pip install -r /requirements.txt

# 4. Copy the code to the image
COPY . /home/site/wwwroot

```

Lo primero que hacemos es extender la imagen oficial del runtime de python para funciones serverless de Azure, utilizando la directiva FROM y el nombre de dicha imagen.

Luego, configuramos una variable de entorno de acuerdo a lo que dice la documentación de funciones serverless de Azure.

A continuación, utilizamos el comando RUN para descargar todas las dependencias de sistema operativo necesarias para que pueda correr nuestro programa de scraping. Algunas de estas son para que se pueda ejecutar correctamente el binario de Chrome.

Después, instalamos usando el comando RUN con curl la versión 123 de tanto Chrome como Chromedriver, que son los dos binarios esenciales para que Selenium pueda ejecutar un headless browser.

Luego, instalamos las dependencias de python necesarias, incluyendo el mismo Selenium entre otras. Estas dependencias existen en un archivo llamado requirements.txt

Finalmente, copiamos el código de nuestro scraper al directorio /home/site/wwwroot, que es el path en donde la documentación pide que lo situemos.

Paso 2: Probar la imagen localmente

En este paso ya tenemos una imagen de Docker que puede correr nuestro programa. Podemos hacerle build con el comando:

```
docker build --tag therealpackjarrow/azurefunctionsimage:v1.0.0 .
```

Y podemos probarla en local con el comando:

```
docker run -p 8080:80 -it therealpackjarrow/azurefunctionsimage:v1.0.0
```

En este punto ya podemos acceder a algún link como este en Postman:
`http://localhost:8080/api/scrape_ml?car_brand=ford`
y podemos verificar que todo funciona correctamente.

Paso 3: Subir la imagen a Docker Hub

Primero nos aseguramos de tener el último build de nuestra imagen según nuestro Dockerfile, con

```
docker build --tag therealpackjarrow/azurefunctionsimage:v1.0.0 .
```

Luego, subimos la imagen a DockerHub con el siguiente comando:

```
docker push therealpackjarrow/azurefunctionsimage:v1.0.0
```

Recordar reemplazar “therealpackjarrow” por el usuario de Docker Hub que corresponda. Si es que no inició sesión en Docker Hub, puede hacerlo con el comando “docker login”.

Paso 4: Deployar a Azure Functions

Primero es necesario que creamos los siguientes recursos:

1. Resource group
2. Storage account
3. Container apps environment

Para ello, ejecutamos los siguientes comandos:

```
az group create --name US3AzureFunctionsContainers-rg --location centralus
az containerapp env create --name US3MyContainerappEnvironment
--enable-workload-profiles --resource-group US3AzureFunctionsContainers-rg
--location centralus
az storage account create --name us3storage --location centralus
--resource-group US3AzureFunctionsContainers-rg --sku Standard_LRS
```

Podemos verificar que todo se haya creado correctamente con el siguiente comando:

```
az containerapp env show -n US3MyContainerappEnvironment -g
US3AzureFunctionsContainers-rg
```

Finalmente, deployamos nuestra función con este comando:

```
az functionapp create --name us3azureseleniumfunction --storage-account
us3storage --environment US3MyContainerappEnvironment
--workload-profile-name "Consumption" --resource-group
US3AzureFunctionsContainers-rg --functions-version 4 --runtime python
--image therealpackjarrow/azurefunctionsimage:v1.0.0
```

Podemos verificar que se deployó correctamente con este comando:

```
az functionapp function show --resource-group
US3AzureFunctionsContainers-rg --name us3azureseleniumfunction
--function-name HttpExample --query invokeUrlTemplate
```

Además, el comando anterior también nos deja ver el link desde el cual se puede llamar a la misma función. Para verificar que todo funcione, podemos hacer un llamado similar al que

hicimos localmente usando Postman, pero reemplazando la URL local por la URL obtenida en este último paso.

Listener

Utilizamos Azure Functions para implementar un listener para monitorear el container “cars” en CosmosDB y calcular el precio máximo de todos los autos almacenados. Esta función se activa cada vez que se inserta un nuevo “car” en el container y actualiza un contenedor llamado “maxPrices” con el precio más alto registrado, junto con el id del auto al que pertenece.

Implementación

Paso 1: Creación de la función

Ejecutamos el siguiente comando para crear una Azure Function llamada “cosmosDBTriggerFunction” en base al template de Azure “Cosmos DB Trigger”:

```
func new --template "Cosmos DB Trigger" --name cosmosDBTriggerFunction
```

Paso 2: Configurar el archivo “function.json”

Modificamos el archivo “function.json” de la función para definir el binding con CosmosDB:

```
{
  "bindings": [
    {
      "type": "cosmosDBTrigger",
      "direction": "in",
      "name": "inputDocument",
      "connection": "COSMOS_CONNECTION_STRING",
      "databaseName": "exampleapp",
      "containerName": "cars"
    }
  ],
  "scriptFile": "../dist/blobTriggerFunction/index.js"
}
```

- Bindings: definen cómo la función interactúa con otros servicios.
 - Direction - in: indica que los datos fluyen **hacia** cosmos
 - Name - inputDocument: es el nombre que luego se usará en el input.ts
 - Connection: especifica la cadena de conexión con CosmosDB

- Database Name: el nombre de la base de datos que la función va a monitorear
- Container Name: el nombre del contenedor en la base de datos que la función va a monitorear
- Script File: la ruta de index.ts

Paso 3: Implementar “index.ts”

La función “cosmosDBTriggerFunction” está diseñada para activarse cuando se producen cambios en el contenedor cars de CosmosDB.

Al activarse, la función registra el número de documentos que desencadenaron el evento. Si hay documentos, crea un cliente de CosmosDB. Luego, accede a “exampleapp” y selecciona los contenedores “cars” y “maxPrices”.

A continuación, la función obtiene todos los documentos del contenedor cars y utiliza el método reduce para determinar el “car” con el precio más alto. Extrae el precio máximo y el ID. Luego, llama a una función auxiliar saveMaxPriceToCosmos para guardar estos datos en el contenedor maxPrices.

```
const cosmosDBTriggerFunction: AzureFunction = async function (context:
Context, inputDocument: any[]): Promise<void> {
    context.log(`Triggered with ${inputDocument.length} changes.`);

    if (inputDocument.length > 0) {
        const cosmosClient = new
CosmosClient(process.env.COSMOS_CONNECTION_STRING);
        const database =
cosmosClient.database(process.env.COSMOS_DATABASE_NAME);
        const carsContainer = database.container("cars");
        const maxPricesContainer = database.container("maxPrices");

        const { resources: cars } = await
carsContainer.items.readAll().fetchAll();

        if (cars.length > 0) {
            const maxPriceCar = cars.reduce((prev, current) => (prev.price
> current.price) ? prev : current);
            const maxPrice = maxPriceCar.price;
            const maxPriceId = maxPriceCar.id;
        }
    }
}
```

```

        await saveMaxPriceToCosmos(maxPrice, maxPriceId,
maxPricesContainer);
        context.log(`Max price item inserted successfully with ID
${maxPriceId}`);
    } else {
        context.log("No cars found in the database.");
    }
}
};

```

La función `saveMaxPriceToCosmos` genera un identificador único para el nuevo documento utilizando `uuidv4`. Luego, crea un objeto que incluye el ID único, el precio máximo, el ID del precio máximo y la fecha actual en formato ISO. Este objeto se inserta en el container “maxPrices”.

```

const saveMaxPriceToCosmos = async (maxPrice: number, maxPriceId: string,
container: any) => {
    const uniqueId = uuidv4();
    const item = {
        id: uniqueId,
        max_price: maxPrice,
        max_price_id: maxPriceId,
        date: new Date().toISOString()
    };

    await container.items.create(item);
};

```

Invocación

La función se ejecuta automáticamente en respuesta a eventos. Para su correcto funcionamiento, se deben crear dos contenedores en CosmosDB adicionalmente a “cars”: “maxPrice” y “leases”. Ambos con partition key “/id”.

- MaxPrice: Se almacenan los maxPrices correspondientes con los cambios en “cars”.

```

{
    "id": "dacbf787-a1b8-4f83-9118-a57a24d60f74",
    "max_price": 69500000,
    "max_price_id": "1430546993",
    "date": "2024-06-05T17:27:57.417Z"
}

```

- id: el id del entry
 - max_price: el precio máximo al momento de la invocación
 - max_price_id: el id del auto al que corresponde el precio máximo
 - date: la fecha y horario de la invocación
-
- Leases: Guarda el estado de procesamiento de cambios para la función CosmosDB Trigger. Esto incluye información sobre los cambios que han sido procesados y los que están pendientes. Es un container obligatorio por parte de "Cosmos DB Trigger"