**Notas del Curso de Computo en la nube**

**CRUD**

CRUD es el acrónimo de "Crear, Leer, Actualizar y Borrar" (del original en inglés: Create, Read, Update and Delete), que se usa para referirse a las funciones básicas en bases de datos.

**Create** 🡪 Método para ingresar un nuevo registro.

**Read** 🡪 Método de búsqueda para ver uno o más registros.

**Update** 🡪 Método para actualizar un registro ya existente.

**Delete** 🡪 Método para eliminar un registro ya existente.

**Servicios REST (Recours Based Web Services)**

Interfaz para conectar varios sistemas basados en el protocolo HTTP y nos sirve para obtener y generar datos y operaciones, devolviendo esos datos en formatos muy específicos, como XML y JSON.

* Request 🡪 Petición que hace el cliente al servidor.
* Response 🡪 Respuesta que regresa el servidor al cliente.

**HTTP Request**

* Metodos
* Headers
* Parametros

**HTTP Response**

* Status Code
* Headers
* Body

**Métodos REST**

**POST = ADD = INSERT = Create**

Método para ingresar un nuevo registro.

**GET = INQ = SELECT = Read**

Método para hacer una consulta, puede tener o no parámetros requeridos.

**PUT = MOD = UPDATE = Update**

Método para actualizar un registro existente, requiere parámetros.

**DELETE = DEL = DELETE = Delete**

Método para borrar un registro existente, requiere parámetros.

**Parámetros REST**

* Path 🡪 Parámetros requeridos que viajan en la URL con {}
* Body 🡪 Resultado del response.
* Query 🡪 Parámetros opcionales que ayudan en el filtrado y búsqueda.
* Header 🡪 Contexto de la transacción para el request y el response.

**Status Code**

* 2xx 🡪 Códigos satisfactorios.
* 3xx 🡪Códigos de redirección.
* 4xx 🡪 Códigos de error por parte del cliente.
* 5xx 🡪 Códigos de error por parte del servidor.

**URL y URI**

* URL 🡪 Path completo que especifica como actuar sobre la representación de un recurso

**https://www.youtube.com/watch?v=FqkLjGS1IXE**

* URI 🡪 Path que identifica un recurso especifico.

[**https://www.youtube.com/watch?v=FqkLjGS1IXE**](https://www.youtube.com/watch?v=FqkLjGS1IXE)

**Mongo DB**

MongoDB es un sistema de base de datos NoSQL orientado a documentos de código abierto. Haciendo una comparación con las bases de datos SQL, en mango se podría tomar la siguiente interpretación.

|  |  |
| --- | --- |
| NoSQL | SQL |
| DB | DB |
| Colecciones | Tablas |
| Tuplas | Documentos |

**Comandos Básicos**

**show dbs** 🡪 Muestra las bases de datos.

**use [nombre db]** 🡪 selecciona la base de datos que deseamos utilizar.

**show collections** 🡪 Muestra las colecciones que existan dentro de una base de datos.

**db.[nombre colección].help()** 🡪 Muestra los comandos disponibles dentro de una colección.

**db.[nombre colección].count()** 🡪 Muestra el total de registros que existen en una colección

**CRUD en Mongo DB**

**Insert = POST**

Db.[nombre coleción]insert({

“clave”: 12,

“nombre”:”Ismael”,

“edad”: 21

})

**Update = PUT**

Db.[nombre coleción].update({

“clave”:12

},{$set:{ “nombre”:”Ismael Ortiz”,“edad”: 21}})

**Delete = DELETE**

Db.[nombre coleción].remove({“clave”: 12})

**Select= GET**

Db.[nombre coleción].find()

Para que este mejor ordenado se agregaría el método pretty() después del método find().

**Partes de un query**

Un query se compone de dos partes el query que es la condición a cumplir (=, <, >, <>) y la proyection que es el filtro para saber que campos serán visibles y cuales no (0 🡪 no mostrar, 1 🡪 mostrar), enseguida un ejemplo:

**db.[nombre colección].metodo.({query},{proyection})**

() 🡪 Obtiene todos los documentos con sus campos.

**db.[nombre de la colección].find()**

({query}) 🡪 Obtienen todos los documentos que sus campos cumplan la condición.

**db.[nombre de la colección].find({“clave”:”sad1”})**

({query},{proyection}) 🡪 Obtiene todos los documentos con campos definidos y que cumplan la condición.

**db.[nombre de la colección].find({“edad”:{$gt:18}},{clave:0,nombre:1,edad:1})**

**Operadores de comparación**

$eq 🡪 =

$gt 🡪 >

$gte 🡪 >=

$in[x,y,z…] 🡪 Busca un valor entre los valores dados

$it 🡪 <

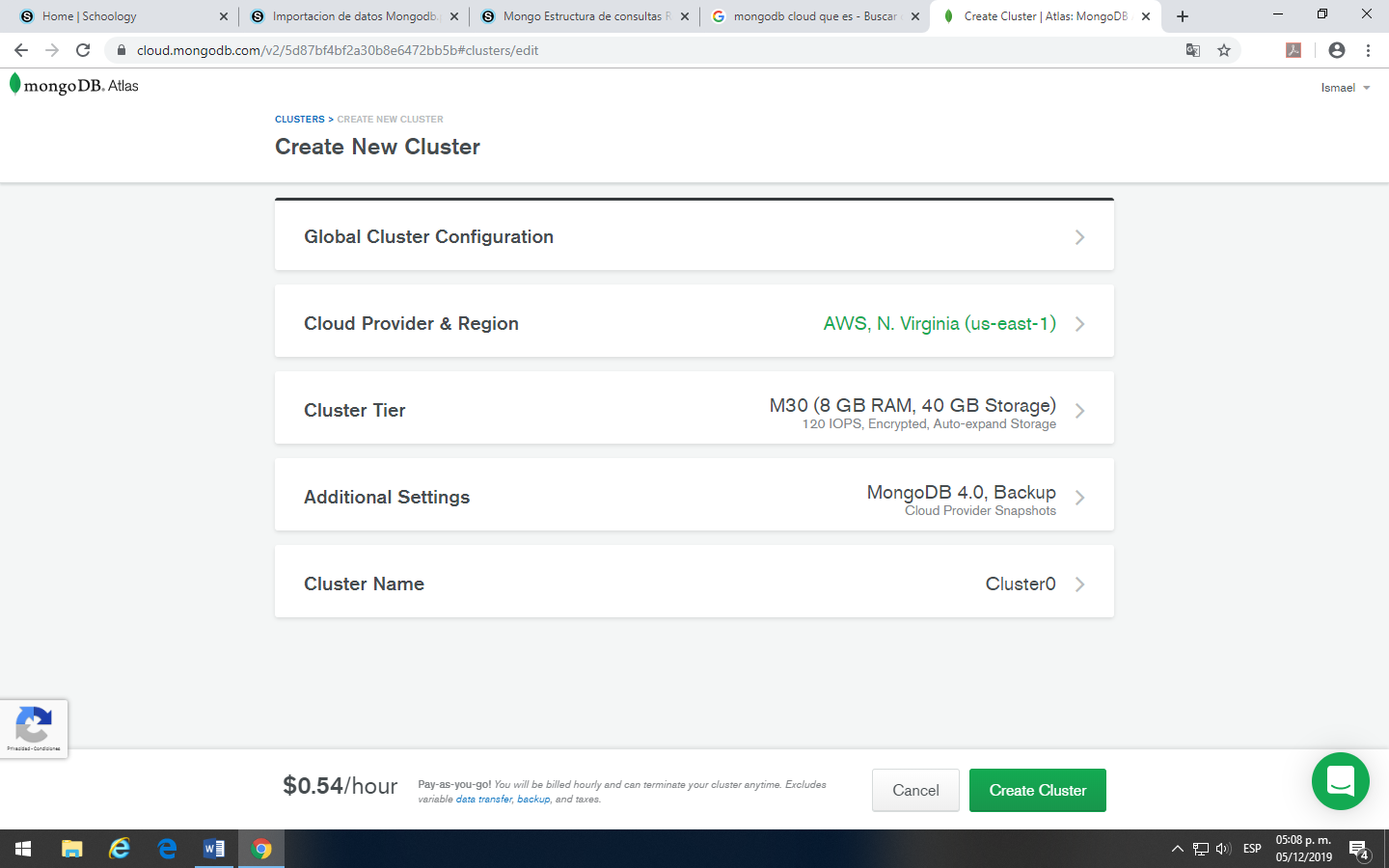
$ite 🡪 <=

$ne 🡪 =!

$nin[x,y,z…]! 🡪 Busca un valor que no se encuentre entre los valores dados

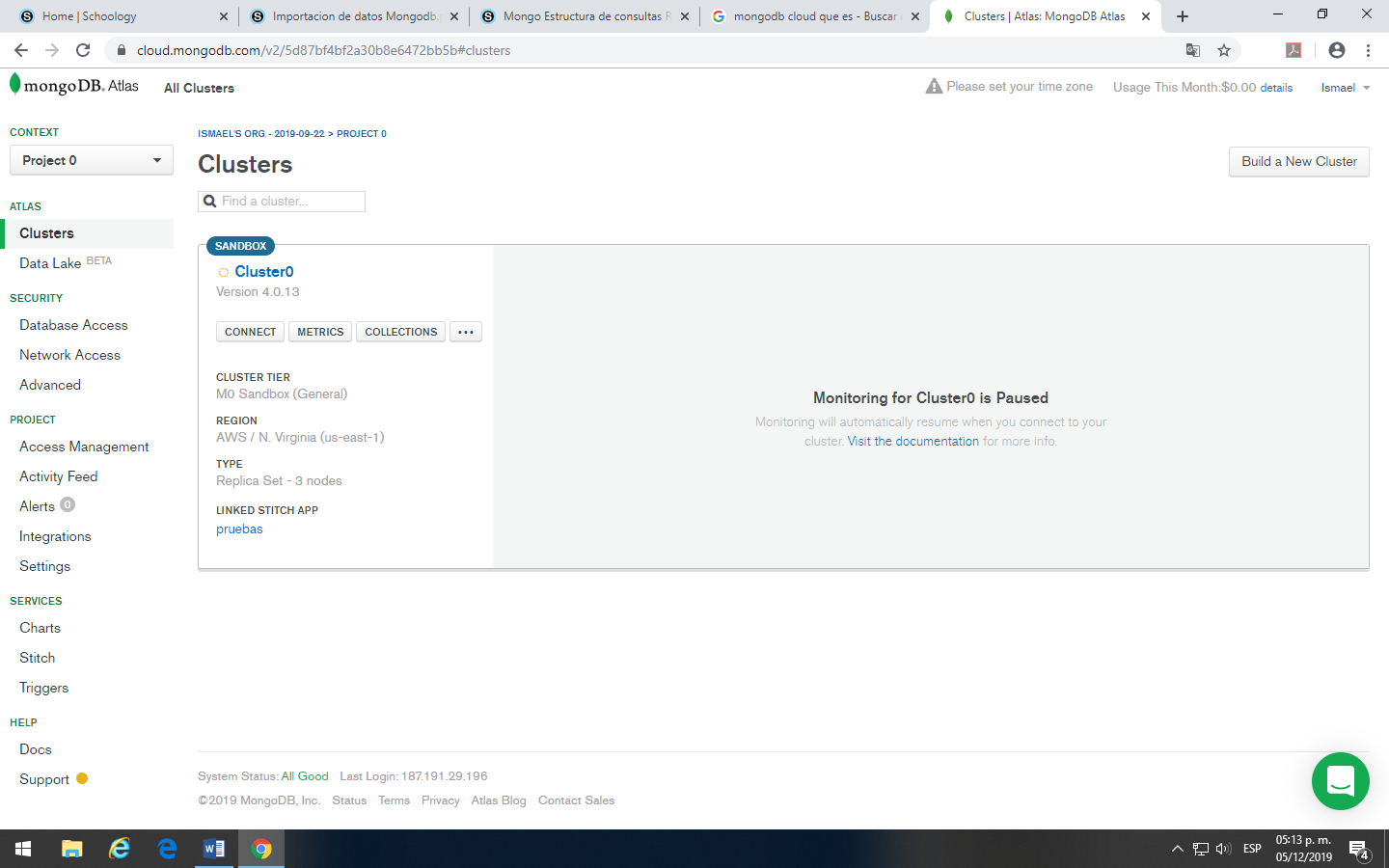
**Conexión con mongodb cloud**

Lo primero que tenemos que hacer es crear una cuenta en mongodb, una vez ahí generamos un nuevo cluster.

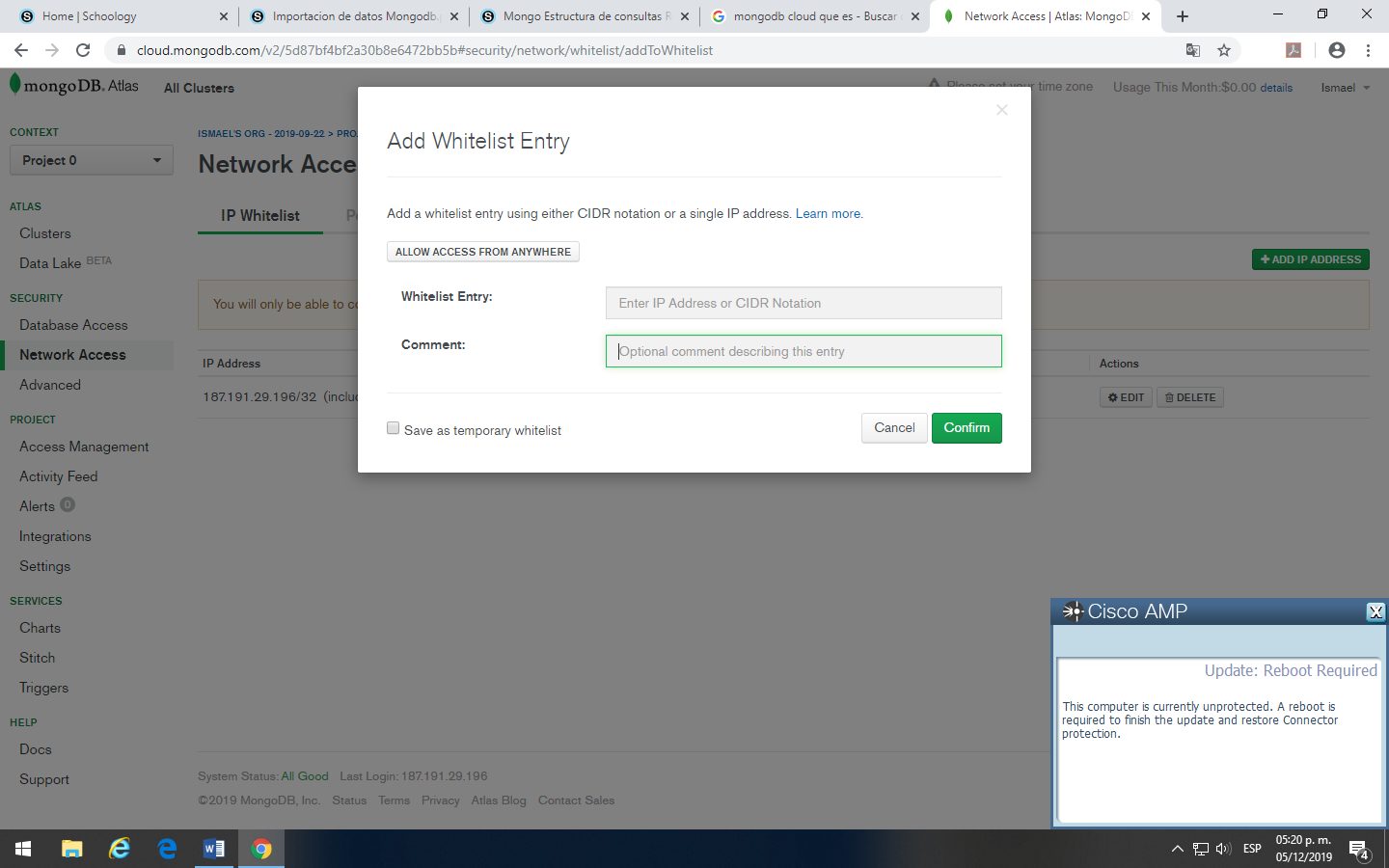


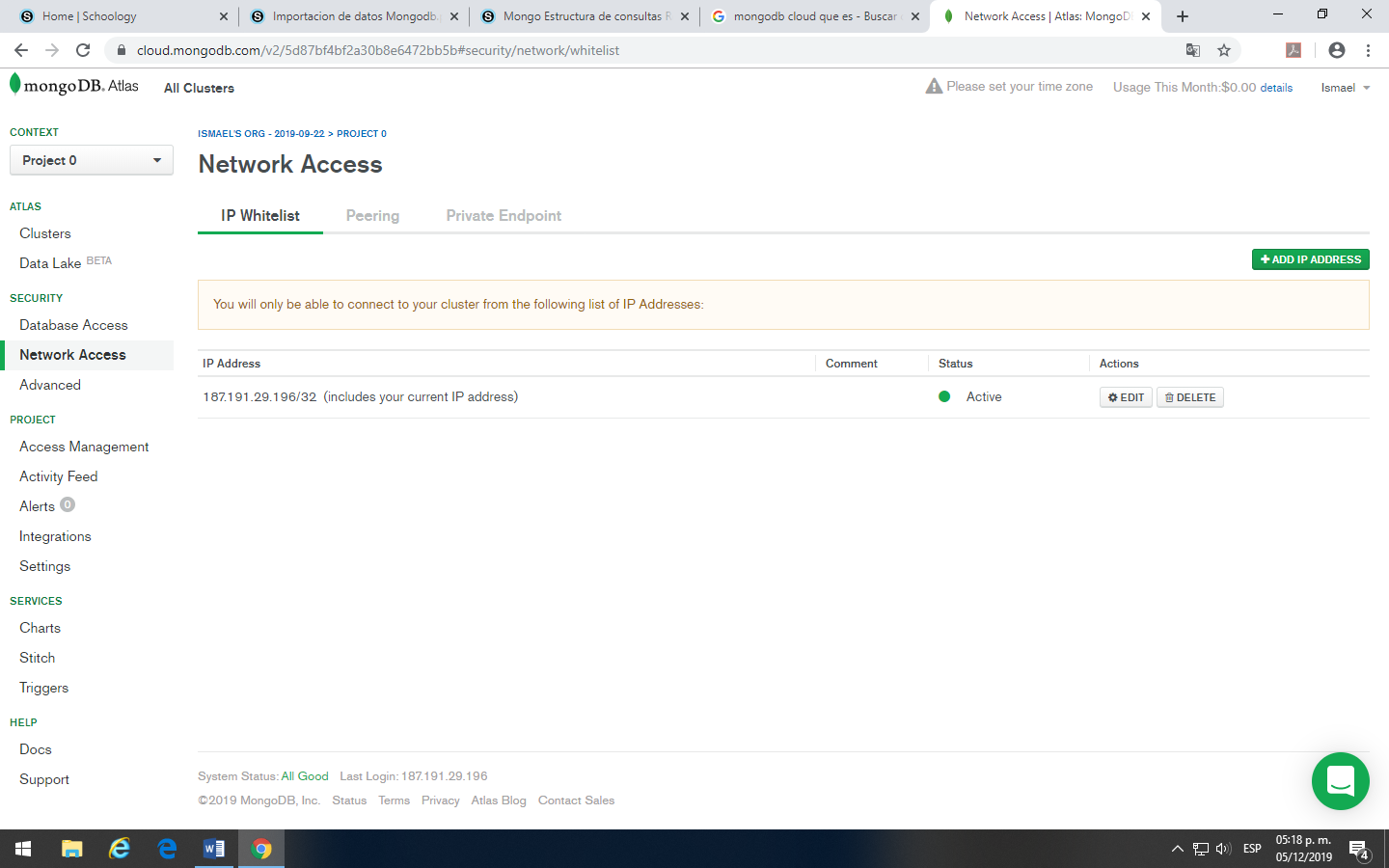
**“El primer cluster será gratis, pero si deseamos crear otro, entonces nos cobraran un determinado precio por hora”**

Solamente le damos un nombre al custer y le damos en crear, después aparecerá la siguiente página:

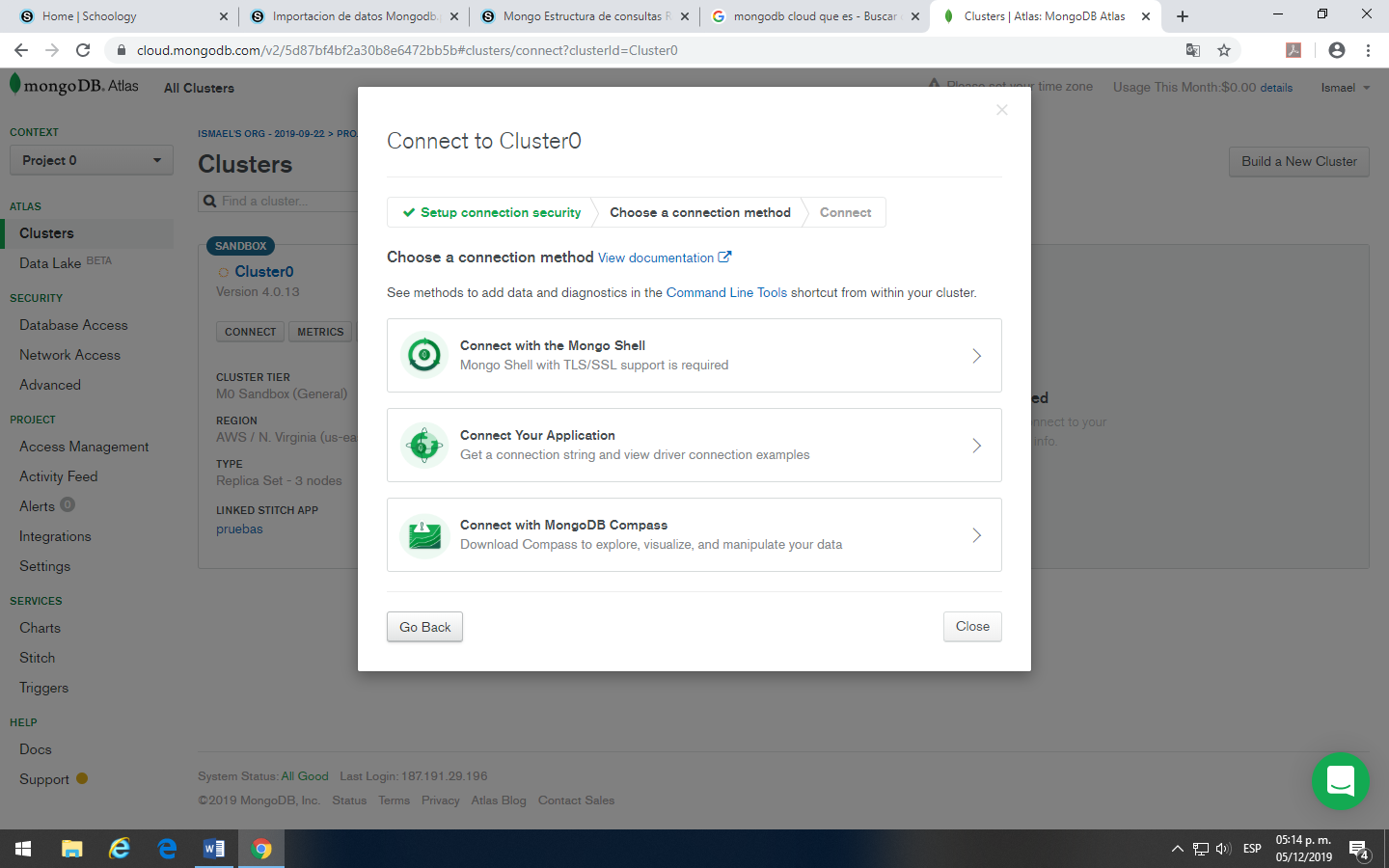


Despues tenemos que configurar nuestra conexión de red, para ello damos click en Network Access y agregamos la dirección ip de nuestro equipo

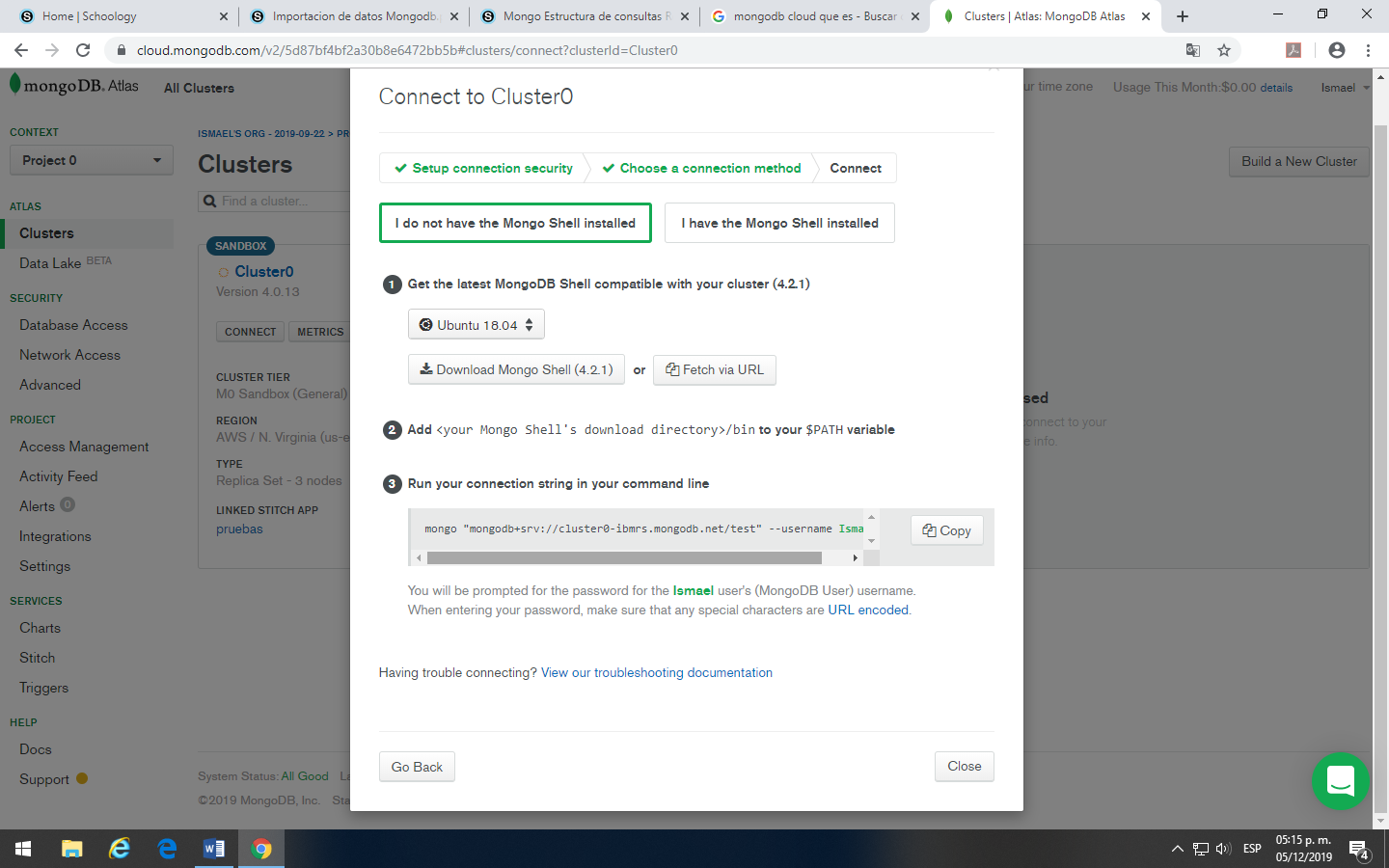




Para hacer una conexión a nuestro cluster por medio de nuestra terminal, basta con hacer click en CONNECT y seleccionar la opción de Connect with the Mongo Shell



Si no tenemos instalado mongodb en nuestra computadora, podemos descargar una CLI para hacer uso de una conexión



Para finalizar solo necesitamos copiar el código que se encuentra en la pantalla, en mi caso

mongo "mongodb+srv://cluster0-ibmrs.mongodb.net/test” --username Ismael

Después abrimos un CLI y nos dirigirnos a la carpeta donde esté instalado nuestro mongodb (si no lo tenemos instalado, que descargar el Shell de mongo e ir a la dirección donde se descargó) y pegamos el código que nos proporciona el cluster y eso es todo, se iniciara la sesión en el Shell de mongo y tendremos acceso a nuestras dbs y a nuestras colecciones

**JSON (Java Script Object Notation)**

**Tipos de JSON**

JSON – String 🡪 Siempre en comillas simples o dobles

JSON – Number 🡪 Entero decimal, ya sea positivo o negativo.

JSON – Boolean 🡪 TRUE o FALSE, sin comillas

JSON – Null 🡪 Valor vacio.

JSON – Collection 🡪 Array(lista) y Diccionarios (llave y valor)

JSON – Dictionary 🡪 Colección de llaves y sus valores, está separado por :

{ “nombre”:”Ismael” }

JSON – Array 🡪 Siempre entre [] o (), sus valores están separados por comas

[{“nombre”:”Ismael”,”edad”:21}]

**Docker**

Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software.

**Comandos básicos**

* docker build –t [nombre imagen] 🡪 crea una imagen.
* docker images 🡪 lista todas las imágenes.
* docker pull [nombre imagen] 🡪 descargar una imagen ya creada.
* docker inspect [nombre imagen] 🡪 inspeccionar una imagen.
* docker rmi [nombre imagen o id] 🡪 elimina una imagen.
* docker run -it image\_name sh 🡪 ver el contenido de una imagen.
* docker ps 🡪 lista todos los contenedores activos.
* docker ps –a 🡪 lista todos los contenedores activos e inactivos.
* docker start [nombre contenedor] 🡪encender un contenedor.
* docker stop [nombre contenedor] 🡪 detener un contenedor.
* docker rm [nombre contenedor o id] 🡪 eliminar un contenedor (el contenedor debe estar detenido).
* docker-compose up –d 🡪 crea un contenedor a partir de un archivo docker-compose
* docker logs [nombre contenedor] 🡪 Obtener los registros de un contenedor.