

En este tutorial, vamos a instalar un cluster de Druid que podrá funcionar en modo multi-nodo. El cluster tendrá las siguientes dependencias:

- Deep Storage: S3 AWS
- Metadata Storage: PostgreSQL
- Sincronización: ZooKeeper
- Ingesta de eventos desde Kafka.

## Instalación ZooKeeper y Kafka

---

[Explicación Kafka](#)

## Instalación metadata storage (PostgreSQL)

---

En primer lugar vamos a instalar PostgreSQL.

Ubuntu/Debian

```
apt-get install -y postgresql
```

OSX

```
brew install postgresql
```

Una vez instalado nos cambiamos al usuario del sistema `postgres`, para gestionar la base de datos.

```
su postgres
```

Creamos un nuevo usuario en postgresql de nombre `druid` y le ponemos una contraseña en este caso `diurd`

```
createuser druid -P
```

Creamos una base de datos de nombre `druid` que pertenece al usuario que acabamos de crear.

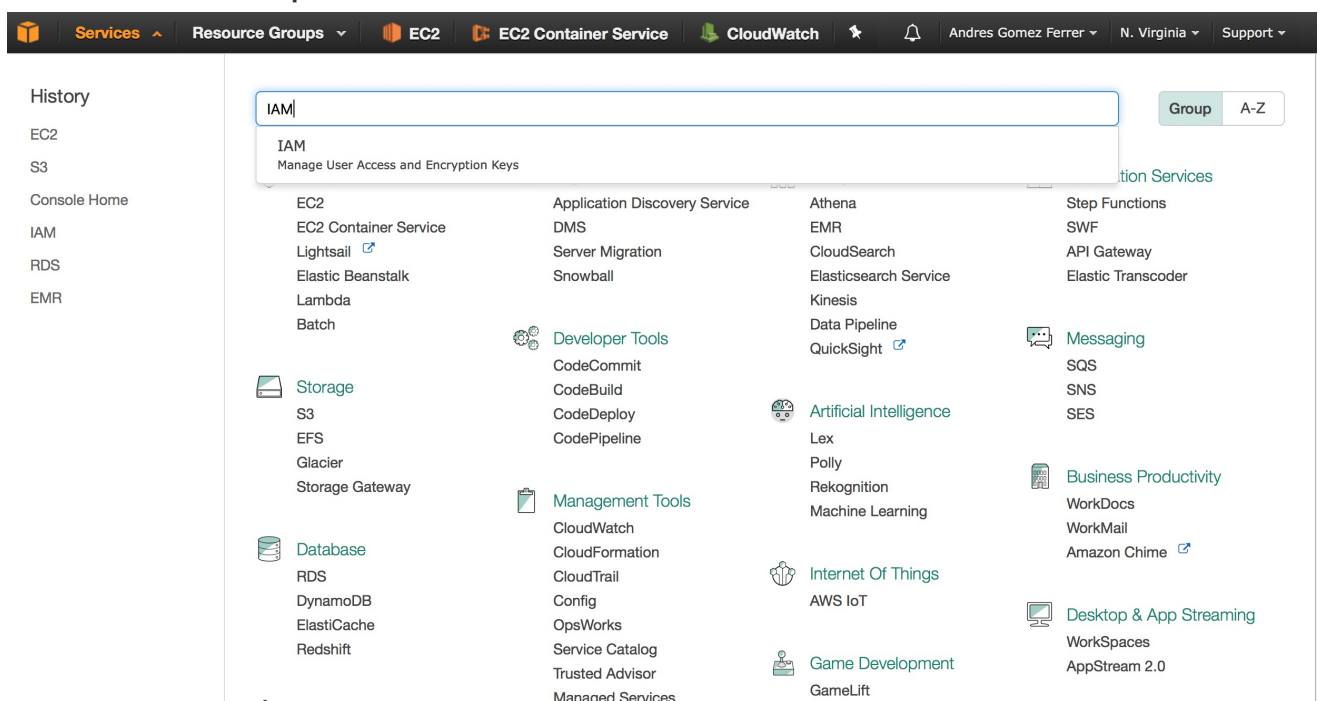
```
createdb druid -O druid
```

Ya tenemos configurado postgresql y podemos salir del usuario `postgres` .

## Configuración AWS S3

En primer lugar para poder acceder al servicio desde S3 desde Druid se utilizaran un AccessKey y un SecretKey, entonces vamos a generarlos.

### 1. Accedemos al panel de IAM.



## 2. Seleccionamos Manage Security Credentials .

Search IAM

Dashboard

Groups

Users

Roles

Policies

Identity providers

Account settings

Credential report

Encryption keys

Welcome to Identity and Access Management

IAM users sign-in link:  
<https://400775011942.signin.aws.amazon.com/console>

Customize | Copy Link

IAM Resources

Users: 0Roles: 2Groups: 0Identity Providers: 0Customer Managed Policies: 0

Security Status

1 out of 5 complete.

✓ Delete your root access keys

Delete your AWS root account access keys, because they provide unrestricted access to your AWS resources. Instead, use IAM user access keys or temporary security credentials. [Learn More](#)

Manage Security Credentials

⚠ Activate MFA on your root account

⚠ Create individual IAM users

⚠ Use groups to assign permissions

⚠ Apply an IAM password policy

Feature Spotlight

Introduction to AWS IAM

0:00 / 2:16

Additional Information

[IAM documentation](#)[Web Identity Federation Playground](#)[Policy Simulator](#)[Videos, IAM release history and additional resources](#)

## 3. Creamos una nueva Access Key.

Search IAM

Dashboard

Groups

Users

Roles

Policies

Identity providers

Account settings

Credential report

Encryption keys

Your Security Credentials

Use this page to manage the credentials for your AWS account. To manage credentials for AWS Identity and Access Management (IAM) users, use the [IAM Console](#). To learn more about the types of AWS credentials and how they're used, see [AWS Security Credentials](#) in AWS General Reference.

+ Password

+ Multi-Factor Authentication (MFA)

- Access Keys (Access Key ID and Secret Access Key)

You use access keys to sign programmatic requests to AWS services. To learn how to sign requests using your access keys, see the [signing documentation](#). For your protection, store your access keys securely and do not share them. In addition, AWS recommends that you rotate your access keys every 90 days.

Note: You can have a maximum of two access keys (active or inactive) at a time.

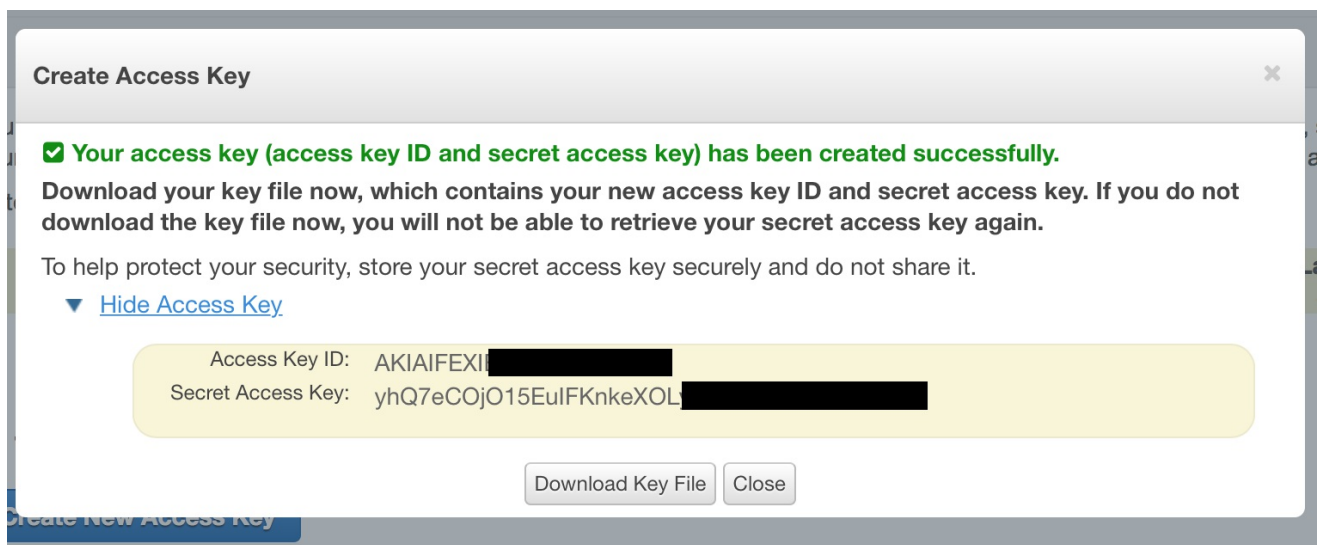
Created	Deleted	Access Key ID	Last Used	Last Used Region	Last Used Service	Status	Actions
Mar 2nd 2017	Mar 2nd 2017	AKIAJPKCFAEYWVKZE4TQ	N/A	N/A	N/A	Deleted	
Mar 1st 2017	Mar 2nd 2017	AKIAJ7QN5LYWFFH4KCCQ	N/A	N/A	N/A	Deleted	
Feb 5th 2016	Mar 2nd 2017	AKIAISZIAIT7NFBNA3YA	N/A	N/A	N/A	Deleted	
Jan 30th 2016	Feb 5th 2016	AKIAIWVS7P6P53O5XYQA	N/A	N/A	N/A	Deleted	

Create New Access Key

⚠ Important Change - Managing Your AWS Secret Access Keys

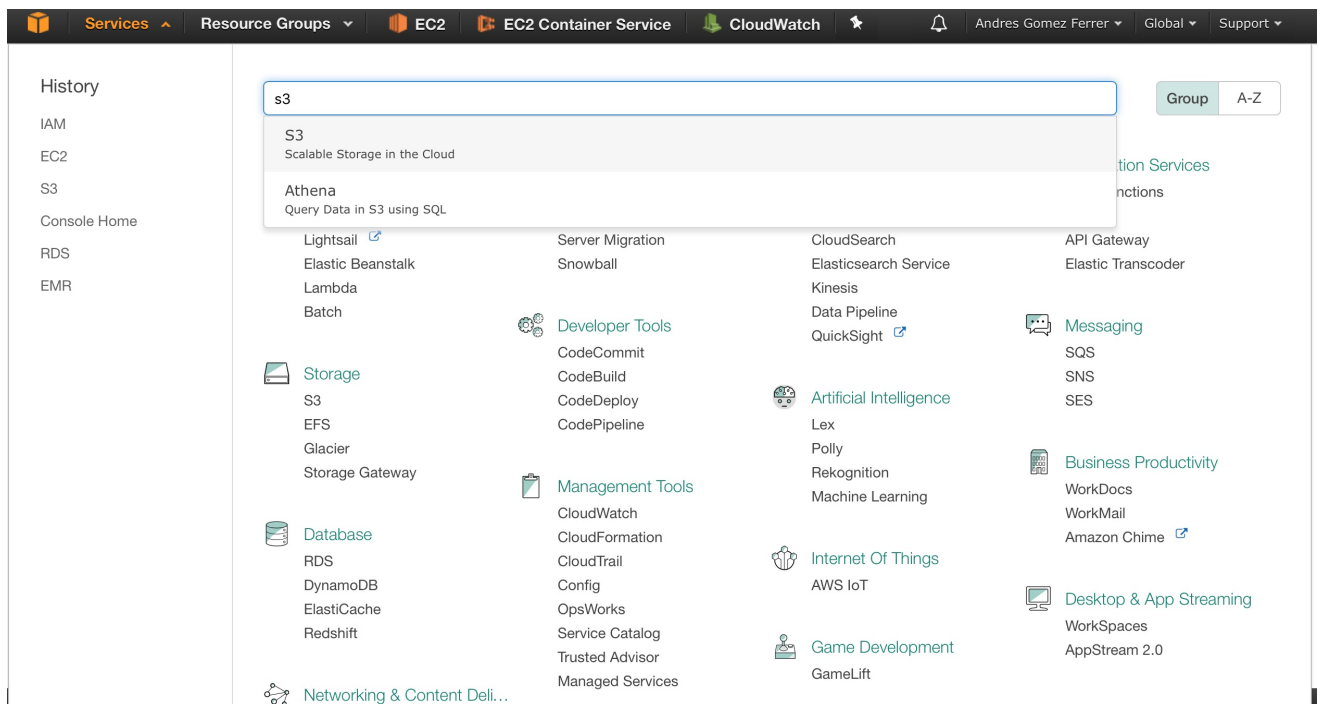
As described in a [previous announcement](#), you cannot retrieve the existing secret access keys for your AWS root account, though you can still create a new root access key at any time. As a [best practice](#), we recommend [creating an IAM user](#) that has access keys rather than relying on root

## 4. Visualizamos y descargamos las credenciales. Hay que tener en cuenta que no podremos volver a ver nuestro SecretKey desde la consola de AWS, por lo que hay que guardarla.

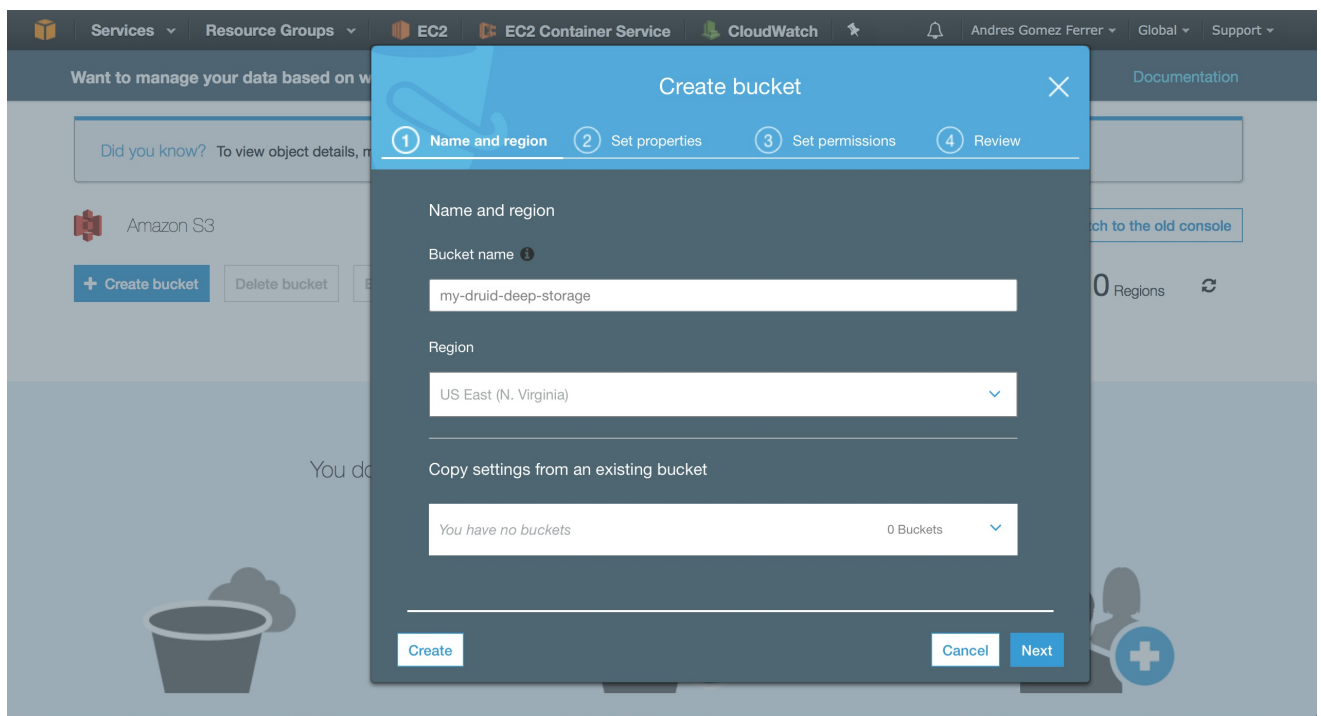


Una vez ya tenemos nuestras credenciales vamos a crear un Bucket de S3, que utilizaremos como Deep Storage.

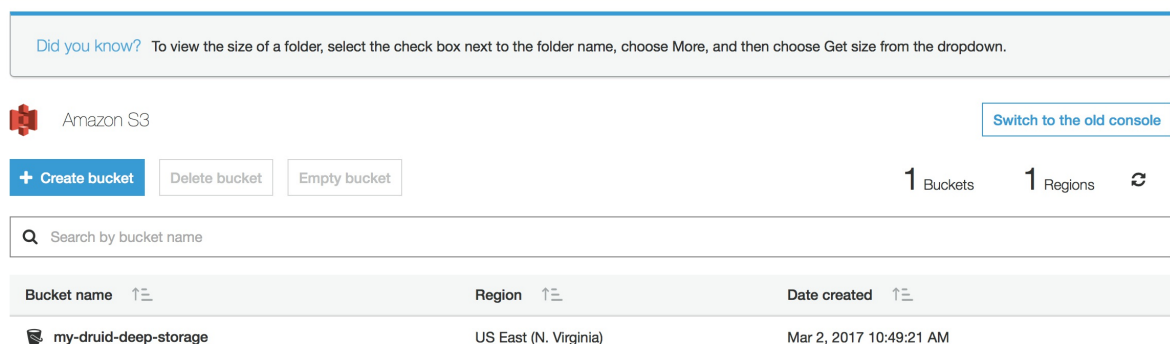
### 1. Accedemos al panel de S3.



### 2. Seleccionamos crear Bucket, introducimos el nombre del bucket y nuestra region y seleccionamos siguiente, siguiente, siguiente y crear bucket.



### 3. Finalmente podremos ver nuestro Bucket ya creado.



Con esto ya tenemos configurado nuestro Deep Storage para utilizarlo desde Druid.

## Instalación Druid

En primer lugar descargamos la última versión estable de Druid y la descomprimos.

```
wget http://static.druid.io/artifacts/releases/druid-0.9.2-b
```

```
tar -xvf druid-0.9.2-bin.tar.gz
```

Una vez ya tenemos la distribución, lo primero que vamos a hacer es configurar Druid para que use nuestro ZooKeeper y nuestro Deep Storage y Metadata Storage. Para ello editamos el fichero:

```
druid-dist/conf/druid/_common/common.runtime.properties
```

1. Cambiamos las extensiones `druid.extensions.loadList` que utilizaremos en nuestro cluster de Druid. Dejamos las extensiones por defecto, cambiando mysql por postgresql y añadimos la extension de kafka-indexing que usaremos más adelante.

```
druid.extensions.loadList=["druid-kafka-eight", "druid-s3"]
```

2. Configuramos nuestro servidor de ZooKeeper.

```
druid.zk.service.host=localhost:2181
```

3. Comentamos las propiedades de metadata.storage pertenecientes a derby y descomentamos la de postgresql. Comentamos -->

```
druid.metadata.storage.type=derby
druid.metadata.storage.connector.connectURI=jdbc:derby://
druid.metadata.storage.connector.host=metadata.store.ip
druid.metadata.storage.connector.port=1527
```

Configuramos -->

```
druid.metadata.storage.type=postgresql
druid.metadata.storage.connector.connectURI=jdbc:postgresql:
druid.metadata.storage.connector.user=druid
druid.metadata.storage.connector.password=diurd
```

4. Ahora vamos a configurar nuestro Deep Storage, para eso comentamos la parte al deep storage local y descomentamos la parte de S3. Comentamos -->

```
druid.storage.type=local
druid.storage.storageDirectory=var/druid/segments
```

Configuramos -->

```
druid.storage.type=s3
druid.storage.bucket=my-druid-deep-storage
druid.storage.baseKey=druid/segments
druid.s3.accessKey=AKIAIFEXIBU2R7BA----
druid.s3.secretKey=yhQ7eC0j015EuIFKnkeX0Lyf-----
```

5. Configuramos S3 como almacenamiento de los logs de las tareas de indexación: Comentamos -->

```
#druid.indexer.logs.type=file
#druid.indexer.logs.directory=var/druid/indexing-logs
```

Configuramos -->

```
druid.indexer.logs.type=s3
druid.indexer.logs.s3Bucket=my-druid-deep-storage
druid.indexer.logs.s3Prefix=druid/indexing-logs
```

Una vez configurado el fichero de `common.runtime.properties` podemos iniciar los servicios y dejar el resto de ficheros con la configuración por defecto.

```
bin/init
```

```
bin/coordinator.sh start
```

```
bin/broker.sh start
```

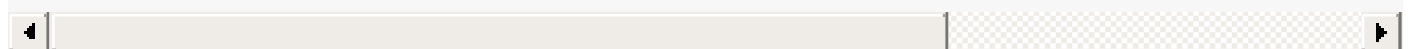
```
bin/historical.sh start
```

```
bin/overlord.sh start
```

```
bin/middleManager.sh start
```

Todos los logs se encuentran dentro de la carpeta `log`, en el inicio de los servicios debemos indentificar esta linea que significa que el servicio ha iniciado correctamente.

```
2017-03-02T10:53:44,593 INFO [main] org.eclipse.jetty.server
```



Una vez todos los servicios se han iniciado deberiamos poder ver sus procesos utilizando:



```
root@ip-172-31-59-196:~/druid# ps aux | grep druid | grep ja
root      10736   1.7   1.7 8939096 570460 pts/1    Sl   10:49   (
root      10905   2.8   2.1 31731828 693608 pts/1    Sl   10:53   (
root      11017   6.3   1.6 13598160 542940 pts/1    Sl   10:57   (
root      11090  11.2   1.7 8924712 573360 pts/1    Sl   10:58   (
root      11197  17.4   0.5 4968964 180244 pts/1    Sl   10:59   (
```

Ahora que ya tenemos todo el sistema funcionando, vamos a crear una tarea de indexación que consuma de un topic de kafka llamado `metrics` y guarde los datos en un `dataSource`.

Creamos un fichero con este contenido llamado `kafka-index.json`.

```
{
  "type": "kafka",
  "dataSchema": {
    "dataSource": "metrics-kafka",
    "parser": {
      "type": "string",
      "parseSpec": {
        "format": "json",
        "timestampSpec": {
          "column": "timestamp",
          "format": "ruby"
        },
        "dimensionsSpec": {
          "dimensions": [],
          "dimensionExclusions": [
            "timestamp",
            "value"
          ]
        }
      }
    },
    "metricsSpec": [
```

```
{
  "name": "count",
  "type": "count"
},
{
  "name": "value_sum",
  "fieldName": "value",
  "type": "doubleSum"
},
{
  "name": "value_min",
  "fieldName": "value",
  "type": "doubleMin"
},
{
  "name": "value_max",
  "fieldName": "value",
  "type": "doubleMax"
}
],
"granularitySpec": {
  "type": "uniform",
  "segmentGranularity": "HOUR",
  "queryGranularity": "NONE"
}
},
"tuningConfig": {
  "type": "kafka",
  "maxRowsPerSegment": 5000000
},
"ioConfig": {
  "topic": "metrics",
  "consumerProperties": {
    "bootstrap.servers": "localhost:9092"
  },
  "taskCount": 1,
  "replicas": 1,
  "taskDuration": "PT1H"
}
```

```
}
```

Para ejecutar la tarea de indexación debemos subir el fichero mediante una petición POST al Overlord.

```
curl -X POST -H 'Content-Type: application/json' -d @kafka-i
```

Ahora si la tarea se ha levantado podemos verificar que esta funcionando si miramos en la interfaz web del overlod

```
http://${OVERLORD_IP}:8090/console.html
```

**Coordinator Console**

### Running Tasks

Show 10 entries

id	id	createdTime	queueInsertionTime	location host	location port	more
index_kafka_metrics-kafka-pt15_b0faf5c28d8778d_pdmoaahk		2017-03-02T11:13:48.543Z	2017-03-02T11:13:48.546Z	ip-172-31-59-196.ec2.internal	8101	<a href="#">payload</a> <a href="#">status</a> <a href="#">log (all)</a> <a href="#">log (last 8kb)</a>
index_kafka_metrics-kafka_bc4421be5fe2de3_eebajfep		2017-03-02T11:06:49.195Z	2017-03-02T11:06:49.206Z	ip-172-31-59-196.ec2.internal	8100	<a href="#">payload</a> <a href="#">status</a> <a href="#">log (all)</a> <a href="#">log (last 8kb)</a>

Showing 1 to 2 of 2 entries

First Previous 1 Next Last

### Pending Tasks - Tasks waiting to be assigned to a worker

### Waiting Tasks - Tasks waiting on locks

### Complete Tasks - Tasks recently completed

### Remote Workers

Show 10 entries

worker host	worker ip	worker capacity	worker version	currCapacityUsed	availabilityGroups	more
ip-172-31-59-196.ec2.internal:8091	ip-172-31-59-196.ec2.internal	3	0	2	["index_kafka_metrics-kafka_bc4421be5fe2de3","index_kafka_metrics-kafka-pt15_b0faf5c28d8778d"]	<a href="#">payload</a> <a href="#">status</a> <a href="#">log (all)</a> <a href="#">log (last 8kb)</a>

Showing 1 to 1 of 1 entries

First Previous 1 Next Last

### Autoscaling Activity

Si tenemos la tarea funcionando correctamente podemos enviar datos utilizando el producer de consola de Kafka.

```
root@ip-172-31-59-196:~# date +%s
1488453335
```

```
root@ip-172-31-59-196:~# kafka/bin/kafka-console-producer.sh
{"timestamp":1488453335, "value":20.00, "sensor":"ABC"}
```

```
{"timestamp":1488453335, "value":20.00, "sensor":"ABC"}
{"timestamp":1488453335, "value":20.00, "sensor":"ABC"}
{"timestamp":1488453335, "value":20.00, "sensor":"ABC"}
{"timestamp":1488453335, "value":20.00, "sensor":"ABC"}
{"timestamp":1488453335, "value":20.00, "sensor":"ABC"}
{"timestamp":1488453335, "value":20.00, "sensor":"ABC"}
{"timestamp":1488453335, "value":20.00, "sensor":"ABC"}
```

Finalmente podemos realizar una query a los datos indexados para comprobar su funcionamiento.

Query:

```
{
  "queryType": "topN",
  "dataSource": "metrics-kafka",
  "granularity": "all",
  "dimension": "sensor",
  "threshold": 1000,
  "metric": "valueSum",
  "aggregations": [
    {
      "type": "longSum",
      "name": "count",
      "fieldName": "count"
    },
    {
      "name": "valueSum",
      "fieldName": "value_sum",
      "type": "doubleSum"
    }
  ],
  "intervals": [
    "2017-03-02T11:00:00/2017-03-02T11:30:00"
  ]
}
```

```
curl -sX POST http://${BROKER_IP}:8082/druid/v2/?pretty=true
```

Resultado:

```
[
  {
    "timestamp" : "2017-03-02T11:15:35.000Z",
    "result" :
    [
      {
        "count" : 8,
        "valueSum" : 160.0,
        "sensor" : "ABC"
      }
    ]
  }
]
```

Una vez ya tenemos datos vamos a forzar la creación de un segmento, para ello vamos a apagar la tarea de indexación:

```
curl -X POST -H 'Content-Type: application/json' http://${OVERTHROW_IP}:8082/druid/v2/?pretty=true
```

Cuando haya finalizado podemos consultar en la web del Coordinator como ya uno de los historicals tienen segmentos cargados:

```
http://${COORDINATOR_IP}:8081/
```

druid cluster

cluster

indexing

datasources

old coordinator console

loading...

old coordinator console

1 datasources

a

filter datasources

metrics-kafka

Key

fully available

> 99% available

≤ 99% available

Hover over the icon next to each datasource for its specific availability.

\_default\_tier

nothing to load

nothing to drop

n + 1.0

130 GB free

1 nodes, 130 GB each

130 GB total

6.56 kB (0.0%) used

show nodes

Indexing Service

y podemos verificar en S3 AWS, que tenemos el segmento y los logs del las tareas de indexación.

Amazon S3

my-druid-deep-storage

druid

Objects

Properties

Lifecycle

Permissions

Management

Upload

Create folder

More

All

Deleted objects

US East (N. Virginia)

Q

Type a prefix and press Enter to search. Press ESC to clear.

Viewing 1 to 2

<input type="checkbox"/>	Name	Last modified	Size	Storage class
<input type="checkbox"/>	indexing-logs	--	--	--
<input type="checkbox"/>	segments	--	--	--

Viewing 1 to 2