**INTEGRANTES:**

* Cahui Parillo Yanet Yovana
* Campos Perez Monica Milagros
* Liberato Conde Jessica

|  |
| --- |
| Github  <https://github.com/yanetyovana208/xctic/tree/master/ProyectoBigData>  Medium  <https://medium.com/@moni090180/proyecto-de-big-data-d08ec55c6064>  Tablero Scrum  <https://trello.com/b/3bV1RIq9/proyecto-bigdata>  <https://trello.com/invite/b/3bV1RIq9/1e9f9bcef1e093fb512d9d5b6609ebc1/proyecto-bigdata>  Slide  <https://slides.com/jessicaadrianaliberatoconde/deck> |

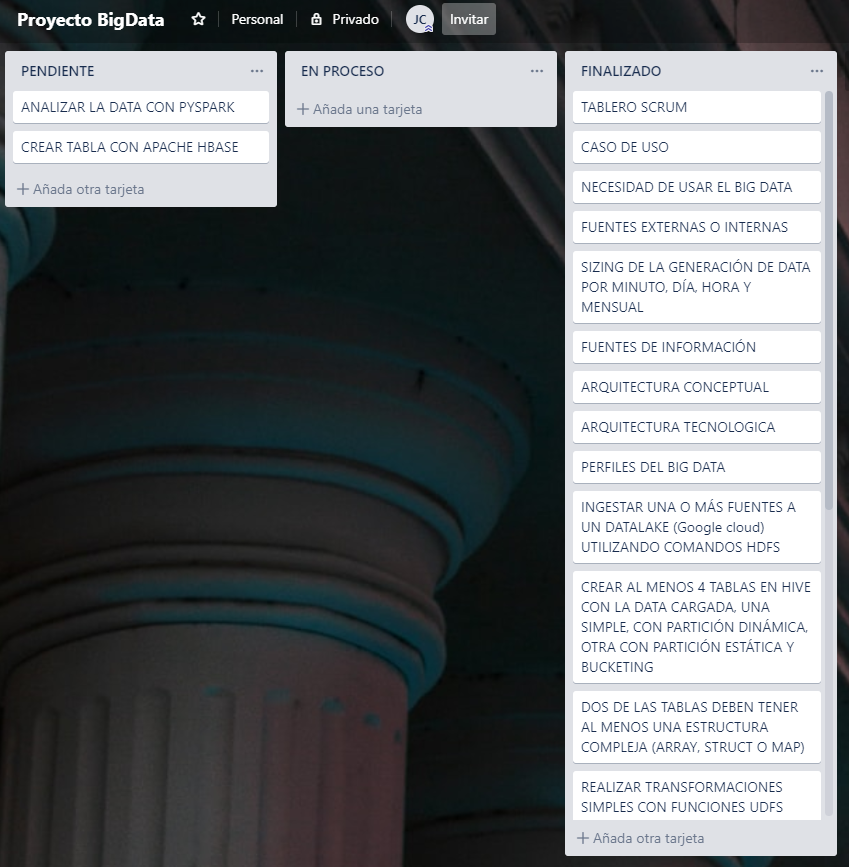
**PROYECTO BIG DATA**

1. **TABLERO SCRUM** (Detalle de actividades)

**Nombre Historia:** Compra por internet

**Descripción de la funcionalidad:** Caso que consiste en que uno o más clientes desde un pc, laptop u otro dispositivo realice una compra por Internet en la página Web de la Empresa Sodimac.

|  |
| --- |
| **Actividades:**   1. Solicitud de Registro por correo electrónico      1. Elegir opciones de despacho – Despacho a Domicilio y Despacho en Tienda          1. De la lista mostrada se selecciona una tienda para el despacho.      1. Persona que retira el Producto en Tienda      1. Pago del Producto – Elegir medio de pago |



1. **CASO DE USO**

Los clientes buscan realizar sus compras desde la comodidad de su hogar, por ello la tienda Sodimac (<https://www.sodimac.com.pe/sodimac-pe/>) mediante su plataforma nos permite visualizar los diferentes productos que ofrecen, brindándonos información de la descripción, características y su calificación que le dieron otros clientes cuando adquirieron al producto.

**Caso de Uso:** Compra por Internet

1. **Solicitud de Registro:** Para poder acceder a la compra de algún producto es necesario que el cliente registre su dirección de correo electrónico.
2. **Elegir opciones de despacho:** El cliente puede elegir de qué forma le entregaran su producto que compre, ya sea por despacho a domicilio o recogerlo en la misma tienda. Por ejemplo en el caso, de elegir retirar en tienda te indica las tiendas más cercanas
3. **Elegir tienda:** Después que el cliente eligió la tienda donde va a retirar su producto el aplicativo muestra la dirección de la tienda y el mapa de cómo llegar a ella.
4. **Persona que retira el producto:** En el caso que el Cliente no pueda retirar el producto personalmente también se tiene la opción de indicar el nombre de la persona quien recogerá el producto.

Asimismo, según el tipo de producto que se desea comprar muestra el peso y volumen a fin que el cliente use un medio de transporte adecuado

1. **Pago del Producto:** Por último se debe elegir el medio de pago ya sea por tarjeta de crédito de la tienda (CMR VISA FALABELLA), otra tarjeta (crédito o débito) o algún otro medio de pago.

El cliente deberá llenar el formulario de validación y realizar el pago.



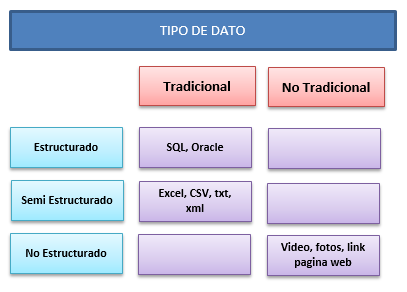
1. **NECESIDAD DE USAR EL BIG DATA**

El uso del Big Data en el caso de uso en estudio justifica mediante la definición de las 3V’s del Big Data.

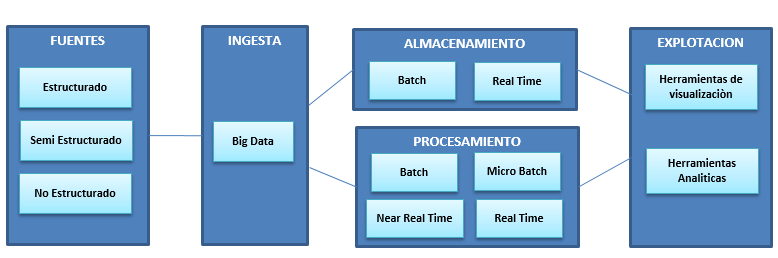
* 1. **Volumen:**
  + SODIMAC maneja gran volumen de información de transacciones que realizan sus clientes al momento de realizar sus compras online.
  + SODIMAC maneja gran volumen de productos, características y/o especificaciones técnicas
  1. **Velocidad:**
  + Los datos se generan en línea (web) dado que las transacciones se realizan de manera muy rápida, ya sea a nivel de stock de productos, tanto para la compra de productos como el pago respectivo.
  + Validación de datos del cliente en línea (usuario y crédito disponible) del medio de pago a usar
  1. **Variedad:**
  + SODIMAC maneja diversas fuentes de datos estructurada (base de datos relacionales SQL, Oracle), semi estructurado (archivos excel, csv, txt y xml) y no estructurada (videos y fotos)



1. **FUENTES EXTERNAS O INTERNAS**
   1. Fuente Externa: Medios de pago a través de tarjetas de crédito o débito.
   2. Fuente Interna: Base de Datos de productos, Clientes, Tiendas
2. **SIZING DE LA GENERACIÓN DE DATA POR MINUTO, DÍA, HORA Y MENSUAL.**
3. **FUENTES DE INFORMACIÓN**



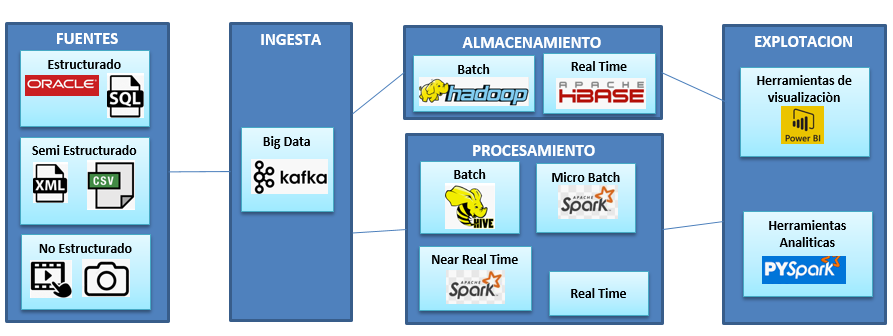
1. **ARQUITECTURA CONCEPTUAL**



1. **ARQUITECTURA TECNOLOGICA**

Respecto a la Arquitectura utilizada en el proyecto se ha tomado en cuenta la Arquitectura Lambda que realiza procesamiento en modo batch y en modo Stream.

La siguiente figura muestra la Arquitectura Tecnológica que se ha diseñado para el proyecto.



1. **PERFILES DEL BIG DATA**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PERFIL** | **TECNICO** | **NEGOCIO** | **ETAPA** | **FASE** |
| **DATA ENGINEER** | Alto: Programación | Alto: Encargado de la data de visualización | Captura | Producción |
| Bajo: Conocimiento de arquitectura y cloud | Bajo: Conocimiento del negocio | Almacenamiento |  |
|  | Apoyo con análisis de información | Análisis |  |
|  |  | Visualización |  |
| **DATA SCIENTIST** | Alta: Capacidad análisis programación,  estadística | Medio: conocimiento del negocio | Análisis | Exploratorio |
|  | Debe saber contar la historia sin términos técnicos. | Visualización | Desarrollo |
|  | Ve el modelo como un servicio | Realiza, transformaciones, visualizaciones,  análisis con el objetivo exploratorio y de desarrollo |  |
| **DATA QUALITY** | Bajo: Programación (verificación de datos de origen y destino) | Bajo: Conocimiento negocio | Esta afuera y dentro con tecnología antes del almacenamiento |  |
| Control (Algoritmos de control) |  |  |  |
| **DATA ARCHITECT** | Bajo: Programación | Bajo: Conocimiento del negocio | Por fuera de las fases |  |
| Alto: Tecnología | Diversidad de áreas |  |  |
| **DATA EXPERT** | Nada técnico | Alto: Conocimiento de negocio | Conoce los datos y variables que tiene el negocio |  |
| **DATA ANALYST** | Bajo: Programación y reportes | Medio: Conocimiento de negocio | Análisis |  |
|  | Medio: Presentación de la historia (story telling) | Visualización |  |
| **CHIEF DATA OFFICER** | Medio: Tecnología | Alto: Conocimiento del negocio |  |  |
| Nada: Programa | Alto: Presentación de la historia (story telling) |  |  |
|  | Rol de gerente de data |  |  |
| **HADOOP ADMIN** |  | DBA de Big Data | Almacenamiento |  |
| **CLOUD ARCHITECT** | Trabaja de la mano con Data Architect |  |  |  |

1. **INGESTAR UNA O MÁS FUENTES A UN DATALAKE (Google cloud) UTILIZANDO COMANDOS HDFS**

|  |
| --- |
| 1. **CREAMOS EL DIRECTORIO HDFS:**   hdfs dfs -mkdir /proyecto  hdfs dfs -mkdir /proyecto/producto  hdfs dfs -mkdir /proyecto/cliente  hdfs dfs -mkdir /proyecto/tienda  hdfs dfs -mkdir /proyecto/distrito  hdfs dfs -mkdir /proyecto/comprobante  hdfs dfs -mkdir /proyecto/transaccion  **Visualizar el directorio creado**  hdfs dfs -ls /proyecto     1. **COPIAMOS LOS ARCHIVOS PLANOS AL DIRECTORIO HDFS**   hdfs dfs -put /home/dataproc/proyecto/producto.txt /proyecto/producto  hdfs dfs -put /home/dataproc/proyecto/cliente.txt /proyecto/cliente  hdfs dfs -put /home/dataproc/proyecto/tienda.txt /proyecto/tienda  hdfs dfs -put /home/dataproc/proyecto/distrito.txt /proyecto/distrito  hdfs dfs -put /home/dataproc/proyecto/comprobante.txt /proyecto/comprobante  **Visualizar archivos copiados en el directorio Proyecto**  hdfs dfs -ls /proyecto/             1. **CREAMOS LAS TABLAS:**   beeline -u jdbc:hive2://localhost:10000/proyecto -f /home/dataproc/proyecto/producto.hql  beeline -u jdbc:hive2://localhost:10000/proyecto -f /home/dataproc/proyecto/cliente.hql  beeline -u jdbc:hive2://localhost:10000/proyecto -f /home/dataproc/proyecto/tienda.hql  beeline -u jdbc:hive2://localhost:10000/proyecto -f /home/dataproc/proyecto/distrito.hql  beeline -u jdbc:hive2://localhost:10000/proyecto -f /home/dataproc/proyecto/comprobante.hql   * **Creación de tabla Producto**      * **Creación de tabla Cliente**      * **Creación de tabla Tienda**      * **Creación de tabla Distrito**      * **Creación de tabla Comprobante** |

1. **CREAR AL MENOS 4 TABLAS EN HIVE CON LA DATA CARGADA, UNA SIMPLE, CON PARTICIÓN DINÁMICA, OTRA CON PARTICIÓN ESTÁTICA Y BUCKETING**

|  |
| --- |
| 1. **TABLA SIMPLE**   **#Tabla Cliente**  CREATE EXTERNAL TABLE IF NOT EXISTS proyecto.cliente(  id\_cliente string COMMENT 'id\_cliente',  cliente string COMMENT 'cliente',  ubigeo string COMMENT 'ubigeo',  direccion string COMMENT 'direccion',  correo string COMMENT 'correo',  telefono string COMMENT 'telefono')  COMMENT 'Tabla cliente'  ROW FORMAT DELIMITED  FIELDS TERMINATED BY '|'  LINES TERMINATED BY '\n'  STORED AS TEXTFILE  LOCATION '/proyecto/cliente'  tblproperties("skip.header.line.count" = "1");  **#Tabla Comprobante**  CREATE EXTERNAL TABLE IF NOT EXISTS proyecto.comprobante(  IdComprobante string,  TipoComprobante string  )  COMMENT 'Tabla comprobante'  ROW FORMAT DELIMITED  FIELDS TERMINATED BY '|'  COLLECTION ITEMS TERMINATED BY ','  LOCATION '/proyecto/comprobante'  tblproperties("skip.header.line.count" = "1");   1. **TABLA CON PARTICIÓN DINÁMICA**   set hive.exec.dynamic.partition=true;  set hive.exec.dynamic.partition.mode=nonstrict;  CREATE TABLE proyecto.transaccion\_particion\_dinamica  (  cliente String,  tipotransaccion String  )  PARTITIONED BY (tipocliente STRING)  STORED AS PARQUET  LOCATION '/proyecto/transaccion\_particion\_dinamica';    **Carga de datos a la tabla particionada dinámica:**  set hive.exec.dynamic.partition.mode=nonstrict;  insert into table proyecto.transaccion\_particion\_dinamica partition (tipocliente)  SELECT distinct UPPER(cliente) AS Cliente, Case when tipocomprobante = 'BOLETA DE VENTA' then 'Boleta' when tipocomprobante = 'NOTA DE CREDITO' then 'Nota\_credito' else 'Factura' end as tipotransaccion, (CASE WHEN cast(venta\_mn as decimal(19, 2)) < '100' THEN 'BAJA' WHEN cast(venta\_mn as decimal(19, 2)) BETWEEN '100' AND '2000' THEN 'MEDIA' ELSE 'ALTA' END) AS tipocliente FROM proyecto.transaccion;    **Comprobando la partición dinámica:**     1. **TABLA CON PARTICIÓN ESTÁTICA**   CREATE TABLE proyecto.transaccion\_particion\_estatica  (  Fecha String,  Tipo\_cliente String,  Tipo\_Transaccion String,  Monto String  )  PARTITIONED BY (Mensual string)  STORED AS PARQUET  LOCATION '/proyecto/transaccion\_particion\_estatica';    **Carga de datos a la tabla particionada estatica;**  insert into proyecto.transaccion\_particion\_estatica partition(mensual ='201804')  select  from\_unixtime (unix\_timestamp(Concat(substring(fecha,1,4),SUBSTRING(Fecha,5,2), SUBSTRING(Fecha,7,2)), 'yyyyMMdd'), 'yyyy-MM-dd') as Fecha,  (CASE WHEN cast(venta\_mn as decimal(19, 2)) < '100' THEN 'BAJA' WHEN cast(venta\_mn as decimal(19, 2)) BETWEEN '100' AND '2000' THEN 'MEDIA' ELSE 'ALTA' END) AS tipocliente,  Case when tipocomprobante = 'BOLETA DE VENTA' then 'Boleta' when tipocomprobante = 'NOTA DE CREDITO' then 'Nota\_credito' else 'Factura' end as Tipo\_Transaccion,  cast(Venta\_mn as decimal(19, 2)) as Monto from proyecto.transaccion;    **Comprobando la partición estática:**     1. **TABLA CON BUCKETING**   CREATE EXTERNAL TABLE IF NOT EXISTS proyecto.tienda\_buckets (   id\_tienda string,   tienda string,   distrito string   )   CLUSTERED BY (tienda) INTO 2 BUCKETS   ROW FORMAT DELIMITED   FIELDS TERMINATED BY '|'   COLLECTION ITEMS TERMINATED BY ','   MAP KEYS TERMINATED BY ':'  LOCATION '/proyecto/tienda\_bucketing';  set map.reduce.tasks = 2;  set hive.enforce.bucketing = true;  INSERT OVERWRITE TABLE proyecto.tienda\_buckets SELECT id\_tienda, tienda,distrito FROM proyecto.tienda; |

1. **DOS DE LAS TABLAS DEBEN TENER AL MENOS UNA ESTRUCTURA COMPLEJA (ARRAY, STRUCT O MAP)**

|  |
| --- |
| 1. **MAP 🡪 Tabla: Producto**   **#Tabla producto**  use proyecto;  CREATE EXTERNAL TABLE IF NOT EXISTS proyecto.producto(  id\_articulo string COMMENT 'Id\_articulo',  rubro string COMMENT 'rubro',  familia string COMMENT 'Familia',  marca string COMMENT 'marca',  articulo string COMMENT 'articulo',  familiamarca **map**<string,string> COMMENT '**Familiamarca**'  )  COMMENT 'Tabla producto'  ROW FORMAT DELIMITED  FIELDS TERMINATED BY '|'  COLLECTION ITEMS TERMINATED BY ','  MAP KEYS TERMINATED BY ':'  STORED AS TEXTFILE  LOCATION '/proyecto/producto'  tblproperties("skip.header.line.count" = "1");  **En spark-shell 🡪 Vemos el contenido de la tabla**  spark.sql("select \* from proyecto.producto").show()     1. **MAP 🡪 Tabla: Distrito**   **#Tabla distrito**  CREATE EXTERNAL TABLE IF NOT EXISTS proyecto.distrito(  Id\_Ubicacion string COMMENT 'Id\_Ubicacion',  Ubigeo string COMMENT 'Ubigeo',  Distrito string COMMENT 'Distrito',  Provincia string COMMENT 'Provincia',  Departamento string COMMENT 'Departamento',  Zona string COMMENT 'Zona',  departamentozona **map**<string,string> COMMENT '**departamentozona**'  )  COMMENT 'Tabla distrito'  ROW FORMAT DELIMITED  FIELDS TERMINATED BY '|'  COLLECTION ITEMS TERMINATED BY ','  MAP KEYS TERMINATED BY ':'  STORED AS TEXTFILE  LOCATION '/proyecto/distrito'  tblproperties("skip.header.line.count" = "1");  **En spark-shell 🡪 Vemos el contenido de la tabla**  spark.sql("select \* from proyecto.distrito").show()     1. **STRUCT 🡪 Tabla: Tienda**   CREATE EXTERNAL TABLE IF NOT EXISTS proyecto.tienda(  id\_tienda string,  tienda string,  figura string,  id\_ubicacion string,  direccion string,  distrito string,  distritotienda **struct**<distrito:STRING,tienda:string>  )  COMMENT 'Tabla tienda'  ROW FORMAT DELIMITED  FIELDS TERMINATED BY '|'  COLLECTION ITEMS TERMINATED BY ','  MAP KEYS TERMINATED BY ':'  LOCATION '/proyecto/tienda'  tblproperties("skip.header.line.count" = "1");  **En spark-shell 🡪 Vemos el contenido de la tabla**  spark.sql("select \* from proyecto.tienda").show() |

1. **REALIZAR TRANSFORMACIONES SIMPLES CON FUNCIONES UDFS NATIVAS APACHE HIVE (USO CONCAT, CAST, SUBSTRING)**

|  |
| --- |
| Detalle de UDFs nativas en Hive;  SELECT distinct CASE WHEN cast(venta\_mn as decimal(19, 2)) < '100' THEN 'BAJA' WHEN cast(venta\_mn as decimal(19, 2)) BETWEEN '100' AND '2000' THEN 'MEDIA' ELSE 'ALTA' END AS tipocliente, UPPER(cliente) AS Cliente, Case when tipocomprobante = 'BOLETA DE VENTA' then 'Boleta' when tipocomprobante = 'NOTA DE CREDITO' then 'Nota\_credito' else 'Factura' end as tipotransaccion FROM proyecto.transaccion |

1. **REALIZAR TRANSFORMACIONES COMPLEJAS (JOINS, AGGREGATES, ETC) CON APACHE SPARK**

|  |
| --- |
| spark.sql("select a.id\_tienda,b.tienda, sum(cast(venta\_mn as decimal(19, 2))) total\_venta from proyecto.transaccion a inner join proyecto.tienda b on a.id\_tienda=b.id\_tienda group by a.id\_tienda, b.tienda").show(); |

1. **EXPLICAR TEÓRICAMENTE CÓMO FUNCIONARÍA EL FLUJO EN STREAMING USANDO APACHE KAFKA**

|  |
| --- |
| Kafka es el sistema de intermediación de mensajes basado en el modelo publicador/subscriptor que se encarga de dividir el flujo streaming en temas o topics (mensaje) en particiones. Cada partición es una secuencia ordenada de mensajes y cada partición es consumida por un único consumidor. A cada topic se le puede definir un número de particiones, en función del número de servidores y de conexiones que vayamos a tener. Esto aumenta considerablemente la disponibilidad.  Cada topic o tema tiene un offset para que cada consumidor indique que mensaje quiere que se le devuelva. A mayor número de particiones más tardará el productor (escribe mensaje) en guardar el mensaje, pero tardará menos el consumidor (lee mensajes) en recuperarlo. La idea está pensada para procesamiento en paralelo.  Cada mensaje publicado en un topic se entrega a una instancia de consumidor dentro de cada grupo de consumidores suscriptores.  Apache Kafka se distribuye junto con Zookeeper este está compuesto por brokers que actúan como líder de una o más particiones. |

1. **EXPLICAR TEÓRICAMENTE CÓMO FUNCIONARÍA EL FLUJO EN STREAMING USANDO APACHE FLUME**

|  |
| --- |
| Flume es un**servicio distribuido** que mueve de forma fiable y eficiente grandes cantidades de datos, especialmente **logs, es** sencilla y flexible; además funciona de tal forma que los flujos de datos en streaming permiten construir flujos múltiples por donde viajan los eventos a través de diferentes agentes hasta que alcanzan el destino final. |

1. **TODO EL CÓDIGO GENERADO DEBE ESTAR CORRECTAMENTE VERSIONADO EN GITHUB**

|  |
| --- |
| **Subimos el proyecto versionado en GITHUB:**  $ git remote add original https://github.com/yanetyovana208/xctic.git  $ git remote -v  origin https://github.com/yanetyovana208/xctic.git (fetch)  origin https://github.com/yanetyovana208/xctic.git (push)  $ git fetch original master  From https://github.com/yanetyovana208/xctic  \* branch master -> FETCH\_HEAD  \* [new branch] master -> original/master  $ git init  Reinitialized existing Git repository in D:/Curso Bigata/Proyecto BigData/xctic/Proyecto/.git/  $ git add .  $ git commit -m "Initial commit"  $ git push origin master          **LINK ACCESO A LOS ARCHIVOS DEL PROYECTO**  <https://github.com/yanetyovana208/xctic/tree/master/ProyectoBigData> |

1. **MOSTRAR GRÁFICAS Y/O INDICADORES**

|  |
| --- |
| 1. **Ventas por Tienda**      1. **Tipo de Documento emitidos:** |

1. **CREAR EL CLUSTER CON UN NOTEBOOK DE JUPYTER INSTALADO**

|  |
| --- |
| 1. Jupyter      1. Instalar ipyparallel      1. Habilitar el IPython Clusters tab in Jupyter Notebook:      1. Instalar los usuarios en el JupyterHub como root:          1. Se observa el clúster creado. |