



GRADO EN INGENIERÍA DE SISTEMAS AUDIOVISUALES Y
MULTIMEDIA

Curso Académico 2019/2020

Trabajo Fin de Grado

IMPLEMENTACIÓN DE UN PLUGIN DE VISUALIZACIÓN EN
REALIDAD VIRTUAL EN KIBANA

Autor: Andrea Villaverde Hernández

Tutor: Dr. Jesús María Gonzalez Barahona

Trabajo Fin de Grado

IMPLEMENTACIÓN DE UN PLUGIN DE VISUALIZACIÓN EN REALIDAD VIRTUAL EN KIBANA

Autor : Andrea Villaverde Hernández

Tutor : Dr. Jesús Mará Gonzalez Barahona

La defensa del presente Proyecto Fin de Grado se realizó el día de
de 2020, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 2020

*Dedicado a
todos aquéllos que
nunca se rindieron.*

Agradecimientos

Gracias a...

«If you can dream it, you can do it.» - Walter Elias Disney

Resumen

Este proyecto...

Summary

This...

Índice general

1. Introducción	1
1.1. El problema	1
2. Contexto y tecnologías utilizadas	3
2.1. HTML5	3
2.1.1. Definición	3
2.1.2. En este proyecto	3
2.2. JavaScript	3
2.2.1. Definición	3
2.2.2. En este proyecto	3
2.3. A-Frame	3
2.3.1. Definición	3
2.3.2. THREE.JS	5
2.3.3. En este proyecto	5
2.4. ELK	5
2.4.1. Definición	5
2.4.2. Logstash	6
2.4.3. Elasticsearch	6
2.4.4. En este proyecto	6
2.5. Kibana	7
2.5.1. Definición	7
2.5.2. En este proyecto	7
2.6. NodeJS y NPM	8
2.6.1. Definición	8

2.6.2. En este proyecto	8
3. Desarrollo	9
3.1. Metodología SCRUM	9
3.2. Sprint 0: Investigación e Instalación	9
3.3. Sprint 1: Plugins Simples	9
3.4. Sprint 2: Visualización con datos Elasticsearch	9
3.4.1. Editor y Schemas	9
3.4.2. Plugin Simple con datos	12
3.4.3. Integración de A-frame con datos	15

Índice de figuras

3.1. Resultado del Editor	12
3.2. Captura de visData en la consola	13
3.3. Resultado Visualización con datos	14
3.4. Esfera A-Frame con Javascript	16
3.5. Resultado de integración de A-Frame con datos	18

Capítulo 1

Introducción

1.1. El problema

Capítulo 2

Contexto y tecnologías utilizadas

2.1. HTML5

2.1.1. Definición

2.1.2. En este proyecto

2.2. JavaScript

2.2.1. Definición

2.2.2. En este proyecto

2.3. A-Frame

2.3.1. Definición

A-frame es un framework web que permite la creación de experiencias VR (Realidad Virtual). Desarrollado por Mozilla pensado para implementar contenido VR a nuestra web de manera sencilla, sin la necesidad de instalar nada. Se trata de un proyecto de Código Abierto, por lo que ha sido muy bien recibido en las comunidades VR.

A primera vista, A-frame parece de fácil manejo; pues permite la creación de escenarios 3D utilizando simple etiquetas HTML. Pero no todo esto se queda aquí, pues es un poderoso

framework de entity-component que viene dado por su fichero three.js.

A-Frame soporta dispositivos de VR como Vive, Rift, Daydream, Gear VR o Cardboard. Además, gracias a dispositivos de tracking y controladores de posición, permite sumergirse en experiencias VR en escenarios a 360°.

Características

- Crea VR de forma sencilla: simplemente utilizando las etiquetas «script» y «a-scene» podrás crear un escenario 3D con toda la configuración para VR, de forma predeterminada, sin la necesidad de instalar nada.
- Declaraciones en HTML: al estar basado en HTML es muy fácil de usar ya seas desarrollador web, amante del VR, artista, diseñador, educador o ni no.
- VR Multiplataforma: permite usar los diferentes dispositivos con sus respectivos controladores. En caso de no disponer de ninguno, también se puede usar en portátiles, tablets o teléfonos móviles.
- Arquitectura Entity-Component: A-frame es un poderoso framework donde se provee de una poderosa estructura entity-component. Al tratarse de HTML, los desarrolladores tienen acceso ilimitado a javascript, DOM API, three.js, WebVR y WebGL.
- Rendimiento: A-Frame está optimizado desde cero para WebVR. Como A-Frame usa el DOM, sus elementos no tocan el motor del navegador. Los objetos 3D se actualizan en la memoria con una sola llamada “requestAnimationFrame”.
- Tool Agnostic: como la web se crea en HTML, A-frame es compatible con la mayoría de las bibliotecas y herramientas web tales como React, Preact, Vue.js, d3.js, Ember.js y jQuery.
- Inspector Visual: A-frame proporciona un visor 3D incorporado. En la que permite abrir la escena 3D y modificar algunos de sus elementos.
- Registro: al igual que Unity Assets Store, a-Frame recopila componentes para que los desarrolladores puedan publicar y buscarlos de forma sencilla.

- Componentes: con a-Frame se puede correr geometrías, luces, materiales, animaciones, modelos, sombras, audios, texto, etc (además de los controladores para los dispositivos). Además de, gracias a su comunidad, sistemas de partículas, físicas, multijugador, aguas, montañas, reconocimiento de voz y un gran etcétera.

2.3.2. THREE.JS

Es una biblioteca escrita en Javascript que permite la creación y visualización de objetos 3D en entornos web. Es muy conveniente pues permite utilizarse en conjunto con Canvas (HTML5) SVG y WebGL. Por lo que podemos decir que es compatible con cualquier navegador que soporte WebGL.

Además, permite importar modelos 3D, en formato JSON, creados en Maya, Blender o Max3D.

2.3.3. En este proyecto

A-Frame supone una parte importante de este proyecto, pues el objetivo de este proyecto es crear un plugin que permita dar una experiencia VR a la hora de representar la visualización de los datos mostrados en Kibana.

2.4. ELK

2.4.1. Definición

Es un conjunto de herramientas de gran potencial que ayuda con la administración de registros, permitiendo monitorizar, consolidar y analizar logs (no siempre son logs) generados en distintos servidores.

Estas herramientas son: Elasticsearch, Logstash y Kibana. Las tres se complementan entre sí pero, se pueden utilizar de forma independiente.

2.4.2. Logstash

Es la herramienta encargada de recolectar los logs de una aplicación, parsearlos; traducirlos y pasarlos a formato JSON para luego poder almacenarla en elasticsearch.

Esta parte no la utilizaremos en este proyecto en ningún momento.

2.4.3. Elasticsearch

Se trata de un motor de búsqueda y análisis fácilmente escalable. Permite almacenar, buscar y analizar grandes volúmenes de datos casi en tiempo real. Se puede acceder de forma sencilla gracias a su elaborada API.

Está escrito en Java, de código abierto y generalmente utilizado con fines empresariales o de investigación.

Características

- Documentos: está orientado a documentos que se insertan en formato JSON, son esquemas sin indexar. Lo que permite una búsqueda mucho mas rápida.
- API: cuenta con una potente API muy fácil de usar. Ésta permite hacer peticiones de tipo HTTP.
- Rapidez: Gracias a su distribución de escalado dinámico; elasticsearch encuentra rápidamente cualquier consulta que se le haga, incluso cuando tenemos grandes cantidades de datos. Ya sean búsquedas simples, como complejas.
- Gran componente: Elasticsearch junto con Kibana y Logstash forman un conjunto de herramientas perfecta para la recopilación, análisis y visualización de datos.
- En tiempo real: Las actualizaciones de los índices de elasticsearch se realizan de manera tan rápida que prácticamente se puede consultar en tiempo real.

2.4.4. En este proyecto

Para este proyecto, no es una parte importante; pues solo la utilizaremos para indexar los datos de prueba que vamos a visualizar posteriormente en Kibana.

2.5. Kibana

2.5.1. Definición

Es una plataforma que permite visualizar los datos almacenados en elasticsearch, para su posterior monitorización y análisis de estos desde el propio navegador web.

Al tratarse de código abierto, la propia empresa invita a que desarrolladores puedan contribuir con su mejoría o a la creación, como en nuestro caso, de plugins para personalizarlos al gusto del usuario.

Características

- Visualizaciones: podemos encontrar representaciones con histogramas, gráficas de tiempo, roscos o tablas que nos permite visualizar e interactuar con los datos almacenados en elasticsearch.
- Datos en tiempo real: la buena conectividad entre ellos, permite visualizar y buscar la información unos pocos segundos después de ser introducida en elasticsearch.
- Dashboards: que recogen las visualizaciones en paneles para poder tener una vista global y así poder entender mejor grandes cantidades de datos.
- Geolocalización: en caso de tener datos de ubicaciones; ésta te muestras las distintas coordenadas en mapas.
- Extras: también incluyen extras como timeseries, graphs o machine learning.

2.5.2. En este proyecto

Esta es la base de dicho proyecto, pues lo que queremos es crear un plugin que permita modificar los distintos tipos de visualizaciones en formato 3D, aportando esa experiencia en VR que tanto queremos conseguir.

2.6. NodeJS y NPM

2.6.1. Definición

2.6.2. En este proyecto

Capítulo 3

Desarrollo

3.1. Metodología SCRUM

3.2. Sprint 0: Investigación e Instalación

3.3. Sprint 1: Plugins Simples

3.4. Sprint 2: Visualización con datos Elasticsearch

3.4.1. Editor y Schemas

Hasta este punto hemos aprendido a crear un plugin simple para kibana e integrarlo con aframe. El siguiente paso será aprender a sacar datos de elasticsearch y poder dibujar un figura de aframe, en este caso un cubo, con las dimensiones que le obtenemos de dichos datos. Para ello, seguiremos los siguiente pasos:

1. Añadir Editor y Schemas.
2. Construir Visualización que muestre datos.
3. Integración de Aframe con datos.

Antes de ponernos a crear una nueva visualización donde nos muestre datos de elasticsearch, necesitamos un editor que nos permita seleccionar los datos que queremos mostrar.

Comenzaremos creando el diseño de nuestro editor. Kibana nos permite seleccionar dos tipo de datos: metrics y buckets.

- metrics: hace referencia a datos que se pueden calcular, por ejemplo, media, máximo, sumatorio, etc.
- buckets: refiere a un conjunto de datos que no se pueden calcular, por ejemplo, fechas, sucursales, tipos, etc.

Como el objetivo final de este sprint es crear una visualización que nos muestre un cubo, determinamos que necesitaremos 3 datos metrics que serán los datos que usaremos para crear las dimensiones del cubo que dibujaremos más adelante.

Estos datos vienen estructurados dentro de Schemas que el propio Kibana permite declarar a la hora de crear la visualización para luego registrarla. Para ello, añadiremos las siguiente líneas en “kbn_awesome.js”:

Antes de nada, importamos los correspondientes paquetes que nos permite declarar los schemas e indicar el tipo de dato que vamos a usar, en este caso metrics.

```
import { Schemas } from 'ui/vis/editors/default/schemas';  
import { AggGroupNames } from 'ui/vis/editors/default';
```

Una vez importado, añadiremos nuevos parámetros dentro de la función createBaseVisualization():

```
schemas: new Schemas([  
  {  
    group: AggGroupNames.Metrics ,  
    name: 'x-axis',  
    title: 'X-axis',  
    min: 1,  
    max: 1,  
    aggFilter: ['count', 'avg', 'sum', 'min', 'max',  
      'cardinality', 'std_dev']  
  },  
  {
```

```
        group: AggGroupNames.Metrics ,
        name: 'y-axis ',
        title: 'Y-axis ',
        min: 1,
        max: 1,
        aggFilter: [ 'count ', 'avg ', 'sum ', 'min ', 'max ',
        'cardinality ', 'std_dev ' ]
    },
    {
        group: AggGroupNames.Metrics ,
        name: 'z-axis ',
        title: 'Z-axis ',
        min: 1,
        max: 1,
        aggFilter: [ 'count ', 'avg ', 'sum ', 'min ', 'max ',
        'cardinality ', 'std_dev ' ]
    }
  ],
```

Parámetros utilizados:

- group: indica el tipo de dato: metric o bucket.
- name: nombre que recibirá el dato.
- title: el título que mostrará en el editor.
- min: indica el mínimo de datos que tenemos usar.
- max: indica el máximo de datos que podemos usar.
- aggFilter: aquí podemos indicarle qué tipos de dato, en este caso tipos de métricas, queremos usar.

Al instante de añadir esto, obtenemos como resultado un editor como el que vemos a continuación:

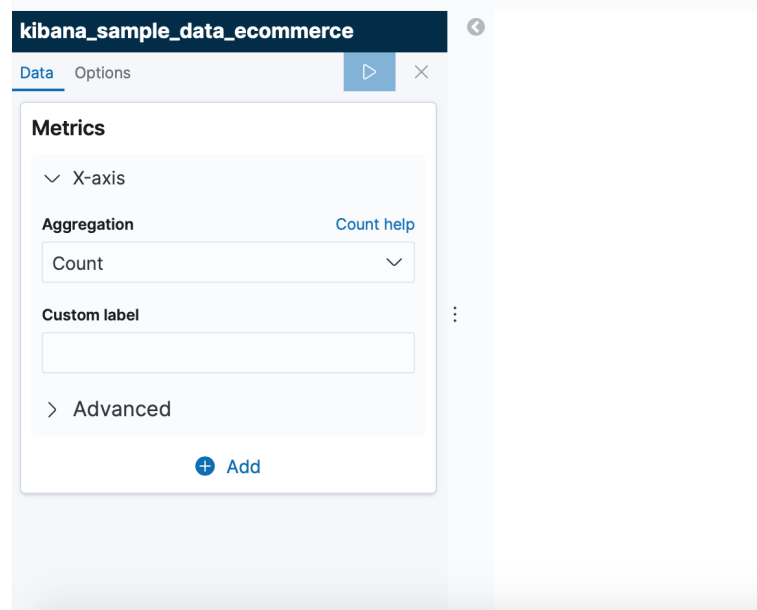


Figura 3.1: Resultado del Editor

3.4.2. Plugin Simple con datos

Ahora que ya podemos elegir los datos en el editor de Kibana, el siguiente paso es conseguir que la visualización muestre dichos datos de forma sencilla.

Para esto debemos haber aprendido, como se vió en el sprint anterior, el funcionamiento de creación y renderizado para entender dónde Kibana recibe los datos de elasticsearch. Así que lo primero que tenemos que hacer es encontrar dónde recibimos los datos que pedimos con el editor.

Si revisamos el esquema del proceso de creación de una visualización (ver capítulo), podemos suponer que los datos que recibimos vendrán en la variable `visData`. Lo comprobaremos de la siguiente forma:

```
render(visData , status) {  
  if (visData) {  
    console.log(visData);  
  }  
}
```

Con esto podremos ver que datos recibimos en la consola del navegador.

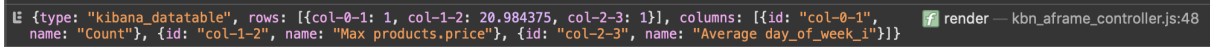


Figura 3.2: Captura de visData en la consola

Esto nos sirve para analizar en cómo es el formato de los datos recibidos de elasticsearch, para luego extraer los datos que nos interesa mostrar.

En primer lugar, vemos que se divide en 3 registros:

- type: tipo del archivo. En este caso, kibana_datatable.
- rows: contiene únicamente los datos obtenidos en formato registro en la que la clave con la forma col-X-X que indica la posición; y el valor con el dato obtenido.
- columns: contiene las etiquetas que hacen referencia a los datos obtenidos. Vienen en forma de array de registros, donde el id indica la posición del dato; y el name indica el nombre del valor.

Analizando todo esto sacaremos los datos, con sus nombres, para guardarlo en la variable metrics.

```
const table = visData;
const metrics = [];

// get metrics
table.columns.forEach((column, i) => {
  var value;
  const name = column.name;
  const id = column.id;
  table.rows.forEach(row => {
    if (row[id]){
      value = Math.round(row[id]*100)/100;
    }
  });

  metrics.push({
```

```

        name: name,
        value: value
    });
});

```

Tras esto, lo añadimos a la visualización, de forma simple, de la siguiente forma:

```

//render metric in vis
metrics.forEach((metric, i) => {
    const metricDiv = document.createElement('div');
    metricDiv.className = 'myvis-metric-div';
    metricDiv.innerHTML = '<b>${metric.name}</b>
    ${metric.value}';
    metricDiv.setAttribute('style', 'font-size:
    ${this.vis.params.fontSize}pt');
    this.container.appendChild(metricDiv);
});

```

Como resultado obtenemos las figuras, que agregamos anteriormente en el capítulo anterior, seguido de los 3 valores que pedimos desde el editor.

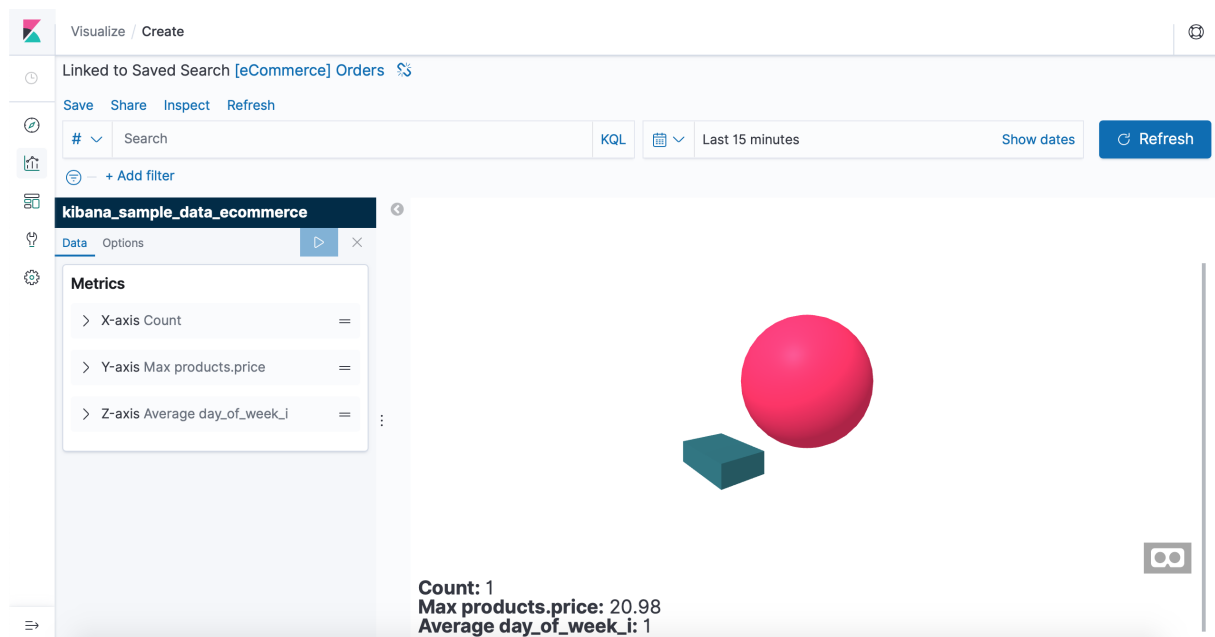


Figura 3.3: Resultado Visualización con datos

3.4.3. Integración de A-frame con datos

Ahora que ya hemos aprendido a obtener los datos que pedimos a elasticsearch para incluirlos en nuestra visualización, el siguiente paso es saber añadir estos datos para poder crear las figuras en base a estos datos.

Únicamente nos centraremos en crear el cubo tomando como valores las 3 métricas que pedimos desde el editor. Para ello, tenemos dos opciones:

1. Insertar los datos al html usando angularJS.
2. Crear la figura directamente desde el controller usando a-frame con javascript.

Analizando un poco como kibana crea sus visualizaciones predeterminadas, creo que la mejor opción es la segunda y así nos ahorramos tener que usar más bibliotecas.

Por eso, eliminaremos el archivo “index.html”, eliminamos la línea donde importamos dicho archivo y cambiamos la siguiente línea de render():

```
this.container.innerHTML = '';
```

Ahora crearemos el escenario justo de donde añadimos las métricas obtenidas de elasticsearch.

```
// Creando escenario  
var escena = document.createElement('a-scene');  
escena.setAttribute('embedded', true);
```

El siguiente paso será añadir las figuras. Empezaremos por dibujar una esfera usando las primitivas que tiene a-frame. Usaremos los mismos valores que teníamos en “index.html” pero variando algún dato al gusto. Por el momento solo le añadimos los datos de forma manual porque lo único que queremos es probar cómo podemos crear una figura de esta forma.

Lo creamos de la siguiente forma:

```
// entity primitive  
var entidad = document.createElement('a-entity');  
entidad.setAttribute('geometry', {  
  primitive: 'sphere',  
  radius: 1.25
```

```
});  
entidad.setAttribute('position', {  
  x: -2.5,  
  y: 1,  
  z: -5  
});  
entidad.setAttribute('material', {  
  color: '#EF2D5E'  
});  
entidad.setAttribute('shadow', true);  
  
escena.appendChild(entidad);
```

Esta última línea, añade la figura en el escenario anteriormente creado.

Ahora que está creado todo la escena con la figura, la añadimos el contenedor de la visualización para que nos la muestre en Kibana.

```
this.container.appendChild(escena);
```

Obteniendo como resultado la siguiente visualización.

Figura 3.4: Esfera A-Frame con Javascript

Una vez hemos conseguido que nos dibuje la esfera, haremos el mismo procedimiento para crear el cubo pero con un par de variaciones:

1. Usaremos el cubo que creamos por componente.
2. Añadiremos los datos obtenidos de elasticsearch.

Como usaremos el componente, revisaremos que “box.js” este importado en el controlador.

Ahora, declaramos un array con las dimensiones que vamos a usar para crear el cubo.

```
const axis = [];
```

Y añadimos los valores en la variable axis tras añadirlo también en metrics.


```
// add axis  
axis.push(value);
```

Por último, solo nos queda crear la figura de la misma forma y añadirlo en la escena:

```
// entity con box.js  
var caja = document.createElement('a-entity');  
caja.setAttribute('box', {  
  height: axis[0]/10, //0.5  
  width: axis[1]/10, //0.25  
  depth: axis[2]/10, //1  
});  
caja.setAttribute('position', {  
  x: 0,  
  y: 1,  
  z: -5  
});  
caja.setAttribute('rotation', {  
  x: 0,  
  y: -45,  
  z: 0  
});  
caja.setAttribute('material', 'color', '#4CC3D9');  
  
escena.appendChild(caja);
```

Como se ve, la diferencia con la esfera es que creamos un a-entity con el atributo 'box' y no 'geometry' y dándole los datos que declaramos en el constructor de box.js.

Con esto, obtenemos como resultado un cubo que varía en base a los datos que le pedimos a elasticsearch.

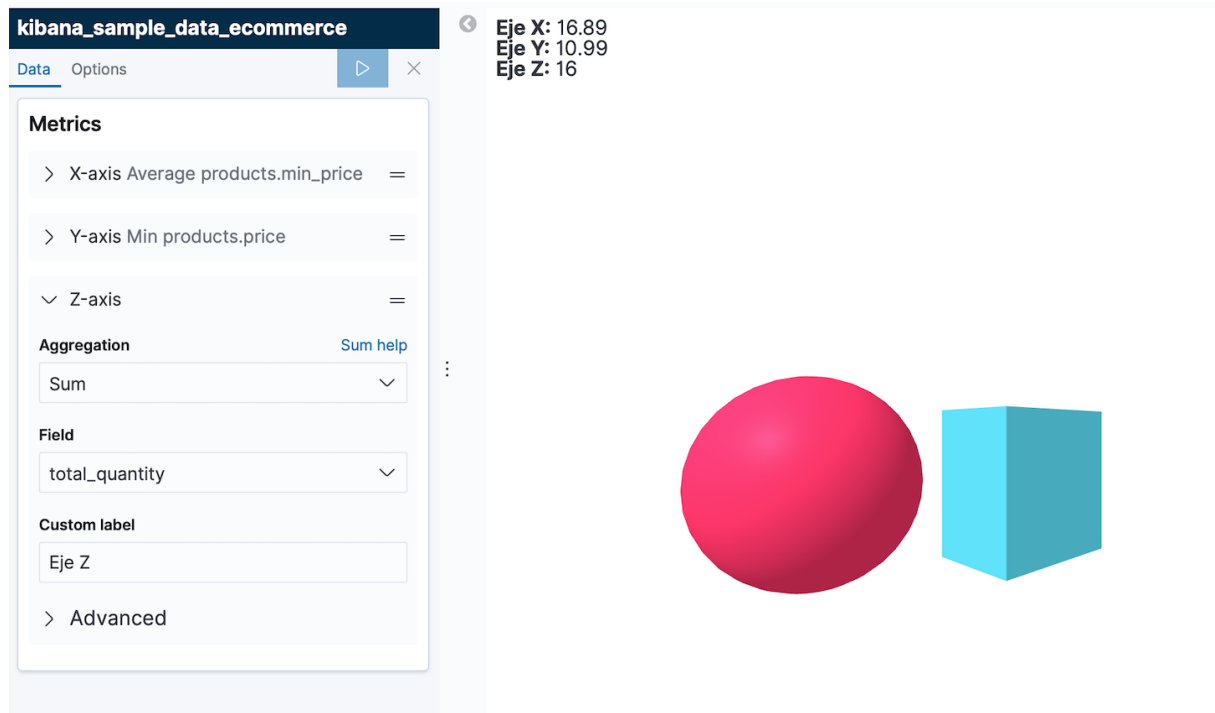


Figura 3.5: Resultado de integración de A-Frame con datos