WHITE PAPER

Embedded Analytics

Solutions for Secure Embedding

Phil Ballai

Enterprise Architect, Sigma

Table of Contents

Introduction

In the age of digital transformation, data's importance has grown exponentially. Enterprises, regardless of their size, face increasing demands from customers seeking access to their data. Historically, businesses have responded by establishing "customer portals" facilitating secure access to a self-service analytics environment. In today's dynamic market, providing such a service is no longer just an added benefit—it's a necessity.

Embedding analytics seamlessly integrates your customer's experience. If a customer spends significant time in your application, embedding ensures they don't need to leave your platform to obtain insights elsewhere. Essentially, embedding amplifies user engagement, encouraging customers to spend more time exploring data insights within your ecosystem.

At its core, embedding involves integrating one software application with another. Everyday examples include watching an embedded YouTube video on social media or a blog post. Sigma can securely embed your dashboards within any application, enhancing user accessibility and experience.

Although some entities still rely on "data dumps" or emailing data sets, these methods are increasingly perceived as archaic, insecure, and inefficient.

This white paper aims to provide an in-depth overview of the various aspects to consider when integrating a customer-centric analytics solution into your online platforms and a demonstration of how embedding works in Sigma.

Offering embedded analytics to customers provides a myriad of benefits, such as:

- Monetizing Your Data: Transform data into a valuable asset.
- Creating New Revenue Streams: Innovate business models by leveraging data.
- Differentiating Your Business: Stand out in a competitive market.
- Enhancing Service Engagement: Promote regular interactions with your services.
- Boosting Customer Satisfaction & Retention: Offer more value, keeping customers loyal.
- Elevating Brand Value: Reinforce your brand's importance in the digital age.

While the advantages are evident, many organizations either don't offer analytics portals or have limited, static versions that lack interactive depth. When businesses consider introducing or enhancing a customer portal with analytics, they face the critical decision of building in-house versus purchasing a ready solution.

Build vs Buy

The decision to either build an in-house analytics solution or purchase an existing one often arises when a business identifies the need for advanced data analytics for its customers. Whether you already have an in-house solution that's proving challenging to maintain, or you're starting fresh and seeking the most efficient route, it's vital to continually assess both the time and monetary investments involved.

The build vs buy decision isn't novel; it has been a topic of discussion for years, with businesses weighing the pros and cons of each option. The crux of the decision hinges on the following considerations:

Opportunity Cost

Pros of Buying: Over time, investing in a pre-built Business Intelligence (BI) tool may prove more economical than dedicating resources to build one from scratch. Choosing the right BI tool can also liberate your developers, allowing them to concentrate on other pressing projects.

Pros of Building: Creating a bespoke solution can cater to your unique business needs, ensuring that every feature aligns with your specific requirements. However, this can often come at a higher upfront cost and longer development timeline.

Maintenance

Pros of Buying: Purchasing a solution, especially a cloud-based Software as a Service (SaaS) offering, can alleviate much of the maintenance burden. Such platforms generally offer regular updates, security patches, and support, ensuring a seamless experience without extensive input from your team.

Pros of Building: On the flip side, building your solution allows for tailor-made maintenance schedules and patches specific to your business's needs. But remember, everything you build requires dedicated support, which might strain your resources.

Constant Demand for Change

Pros of Buying: Established analytics platforms often evolve in response to industry trends and feedback from a broad user base, ensuring they remain relevant and up-to-date. Adopting such a platform can ease the process of scaling and implementing new features.

Pros of Building: Building your analytics solution grants flexibility. As your business grows and needs change, you have direct control over adjustments and improvements. However, swift adaptation requires constant vigilance and can be resource-intensive.

Summary

The decision to build or buy hinges on a business's specific needs, resources, and long-term vision. Both paths offer advantages and challenges. It's crucial to make an informed decision, considering not only the immediate requirements but also the future evolution of your analytics needs and the broader business landscape.

Decision Criteria

In the burgeoning world of Business Intelligence (BI), numerous tools offer embedding capabilities. As a decision-maker looking to seamlessly integrate dashboards into your application, the variety of options can be daunting. However, insights from Sigma customers reveal some critical themes that underscore why Sigma stands out for embedding:

Ease of Implementation

Sigma's setup process is refreshingly straightforward. By employing an iFrame and constructing a server-side API, integration is accomplished effortlessly. Sigma even offers API samples across popular programming languages, with tasks like customizing the user

experience integration managed by the API itself. On average, this setup process can be completed within a day.

Learn how to implement Sigma's embedded API

Quick Time to Value

Agility is paramount in today's fast-paced business environment. When introducing a new dashboard element or entirely novel visualization, the transition from conceptualization to customer access should be quick. Laborious coding or intricate data modeling can hinder this process, making your embedding solution less effective.

Live Data Querying

Customers expect real-time or near-real-time data insights. To fulfill this expectation, embedded solutions should fetch the most current data whenever accessed. This dynamic updating ensures that users always access the most relevant, up-to-date insights, regardless of the size of the dataset.

Learn how Sigma queries data

Performance

The user experience hinges on performance. Rapid response times during data drilling or filtering processes are essential. Traditional BI performance remedies, such as in-memory or on-premise solutions, might offer solutions but often at the expense of higher costs and complexity. The more advanced approach? The Cloud Data Warehouse (or Lakehouse). It's pivotal to select a BI tool specifically designed for CDWs rather than one merely adapted to them.

Scalability

The ideal embedded solution should grant users access to comprehensive, fine-grain data without restrictions like data aggregation or limited timeframes. This ensures that users can engage in intricate, ad-hoc queries without hindrance, even if there are billions of rows involved.

Security

Ensuring data exclusivity for each customer is critical. Through mechanisms like row-level security (RLS), BI tools can ensure data exclusivity. Effective embedding solutions also offer single-use URLs, enhancing security without demanding user authentication against the BI tool.

Learn how to implement RLS.

Authentication and Authorization

Proper authentication ensures that users only access data relevant to them. Whereas "internal embedding" often employs Single Sign-On (SSO) via SAML, customer portals benefit from signed embedding schemes, which use a confidential, unique key. The takeaway? Opt for BI tools that don't mandate explicit, cumbersome authentication processes.

Learn how to implement Sigma with SSO using Okta

Cloud-Native Orientation

Modern BI tools should be cloud-native, emphasizing aspects like scalability, comprehensive functionality, and a modern feel. It's essential to avoid tools that may be cloud-hosted but aren't cloud-native by design.

Interactivity

A high degree of user interactivity ensures better engagement. This includes features like drilling down, filtering, sorting, and overall data exploration. For optimal interactivity, a BI tool should possess a refined query execution model, supporting varied data exploration techniques.

Summary

The desired characteristics of an embedding solution are multifaceted: it should be fast, offer real-time data, guarantee top-tier performance, scale as required, maintain stringent security protocols, ensure proper user authentication, operate natively in the cloud, and

provide a rich interactive experience. A holistic embedded solution, like the one offered by Sigma, encapsulates all of these attributes.

Ways to Embed

The embedding method often varies based on the target audience or consumers of the data. Typically, embedding needs can be categorized into three primary types:

Public Embedding

Purpose: Used when the data is intended for the public domain or isn't specific to a particular company or individual.

Security: No stringent security measures are required since the data is not confidential.

Example: A government agency might want to showcase a census dashboard.

How Sigma Supports: Embed the public link within an iFrame and the dashboard is ready for public viewing.

Internal Embedding

Purpose: Designed for an organization's internal stakeholders, such as employees, and offers specific access based on authentication.

Usage: Organizations use this to monitor various internal metrics and KPIs. For instance, Sigma embeds dashboards within Salesforce to track business performance metrics like sales pipelines, forecasts, and software adoption rates.

Learn how to embed Sigma in Salesforce

Authentication: Either a local account within the BI tool is created, or Single Sign-On (SSO) via Security Assertion Markup Language (SAML) is employed.

Working Principle: When users access a dashboard, they can only view it if they are authenticated against the BI tool. SSO is commonly used: Once users log into a portal

through SAML, the BI tool checks for authentication. If the Identity Provider (IDP) confirms, users can access the dashboard.

Suitability: It is ideal for businesses that rely heavily on metrics to drive their strategies and operations, especially when the users typically undergo SAML authentication.

External Embedding

Purpose: Designed for customers who log into custom-built portals, specifically crafted to provide them with tailored data.

Appearance & Branding: These portals often bear the brand of the provider and are seamlessly integrated into the services offered by them.

Authentication: With Sigma, the embed inherits the security from the parent application at runtime, using whatever security mechanism the parent application used to grant the user access.

User Experience: Ideally, customers shouldn't even discern that the dashboards are embedded. The transition should be seamless, and the interface should be congruent with the rest of the application.

Summary

Choosing the right embedding type hinges on your data's intended audience and the level of access and security needed. A well-integrated and appropriately chosen embedding type can offer a seamless and efficient experience for the end-users, enhancing their interaction and trust in your application.

External Embedding

External embedding has emerged as the favored method for businesses that have bespoke customer portals and wish to introduce or elevate their analytics offerings. For organizations contemplating the incorporation of external embedding within their customer portals, Sigma stands out as a leading choice.

Sigma is the first analytics and business intelligence solution tailored for cloud data warehouses. It seamlessly combines a contemporary spreadsheet interface directly linked to your cloud data warehouse, ensuring a real-time, immersive experience. Its platform aligns with the critical requirements highlighted previously in the "Decision Criteria" section.

With Sigma, businesses unlock the following capabilities:

Preserved Application Authentication

Sigma adopts your existing security at runtime, eliminating the need to create local users or navigate a new authentication structure. This simplified process helps to significantly reduce implementation time. While SAML is optional and can be integrated if desired, Sigma's design allows you to lean on the authentication system you've already put in place. More on this in the next section.

Federated Access Control

Sigma supports Teams, Workspaces, and Role-Based Access Control so that you have fine-grained control over what customers, groups, and users can see and do in the embedded platform.

Learn about Federated Access Control with Sigma

Row-Level Security (RLS)

Design a universal set of dashboards tailored to your customers. Each customer, upon accessing a dashboard, is presented with data specific to them. This approach streamlines maintenance, eliminating the need for individual customer views. Notably, RLS operates even without direct user authentication against Sigma.

Learn how to implement RLS

Versatility in Embedding

Choose between embedding entire dashboards or singular visualizations based on your needs.

Customization

Sigma supports tailor-made layouts and themes to ensure visual cohesion with your brand and portal.

Interactivity

Users can actively engage with the data through functionalities like drilling down, filtering, and utilizing charts as controls, provided they have the appropriate role assignment.

Responsive Design

Sigma's embedded dashboards are optimized for various devices, including mobile phones, tablets, and laptops. The dashboards adjust dynamically based on screen size.

Learn how to create responsiveness in Sigma

Filter Integration via URL

This feature enables default filter configurations once users access a dashboard. For instance, if a user focuses on the "West" region within your application and subsequently accesses an embedded dashboard, the information will already be filtered to highlight the "West" region, courtesy of the URL parameters.

Learn how to leverage parameters in Sigma

JavaScript Event Listener

Sigma activates a JavaScript event whenever a dashboard filter is modified. This capability facilitates the transfer of context from one dashboard to another. While similar to the URL filter integration, in this scenario, filters transition between Sigma-embedded dashboards.

An illustrative example might be a tab-based interface where every tab displays a distinct embedded dashboard. If a user sets a date range filter on one tab (e.g., “last 30 days”), that specific date range can be carried over to the next accessed tab.

Numerous Sigma embeds clients utilize this functionality to transition filter values between their portal modules and Sigma embeds. This integrated approach ensures a fluid user experience, making the embedded BI component appear innate to the platform.

Learn about using Actions / Events in Sigma

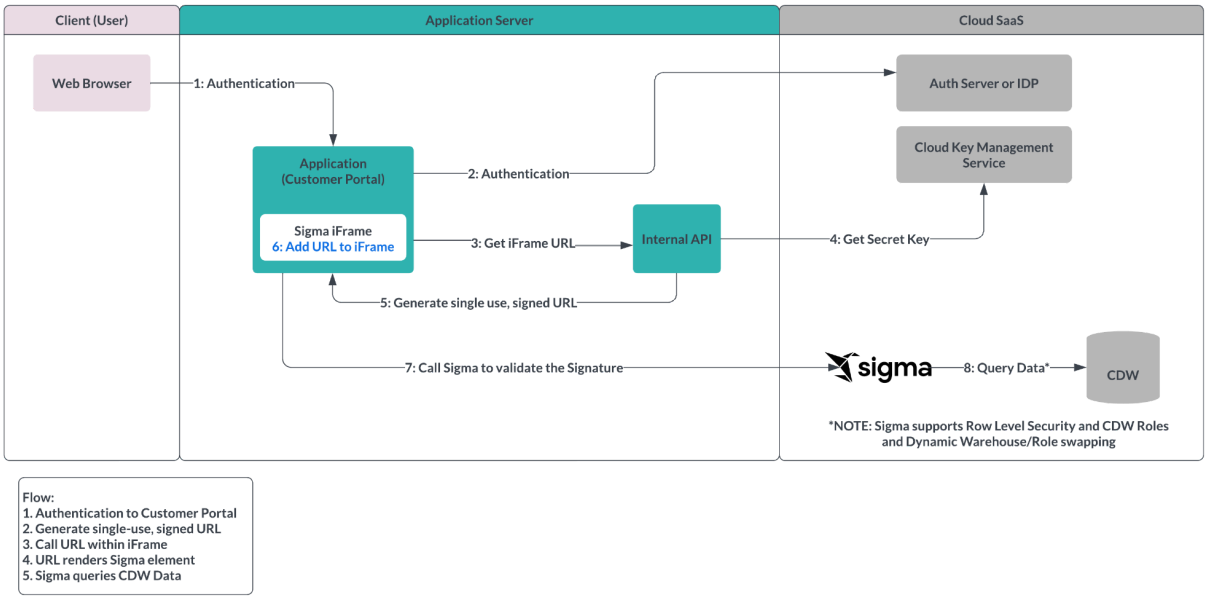
Summary

External embedding with Sigma offers a harmonized, interactive, and responsive analytics experience. Integrating seamlessly with custom-built portals ensures that users derive insights efficiently, enhancing overall user engagement and satisfaction.

How Embedding Works in Sigma

External embedding is made possible by the creation of a unique, encrypted, one-time-use embed URL pointing to the workbook, workbook page, or element you wish to display, along with any optional parameters you wish to include.

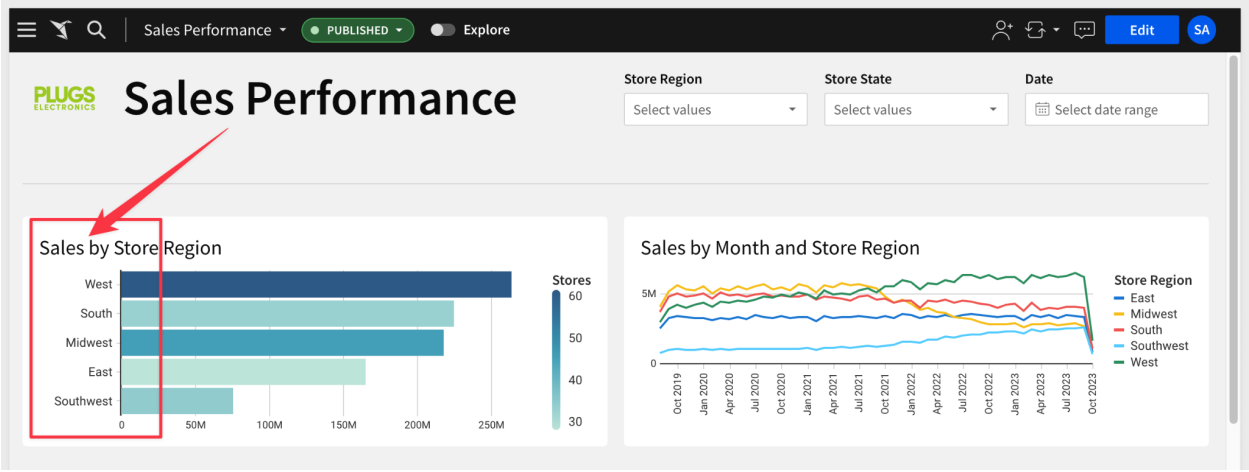
This URL is generated on the server side of your parent application through an API you set up and is rendered client side in an HTML iFrame element.



The image above is a workflow that provides a high-level overview of the process. It is very important to note that each API-generated URL can only be used once and if modified externally and resent, it will generate an error message in the browser.

Step 1: Identify the content to embed

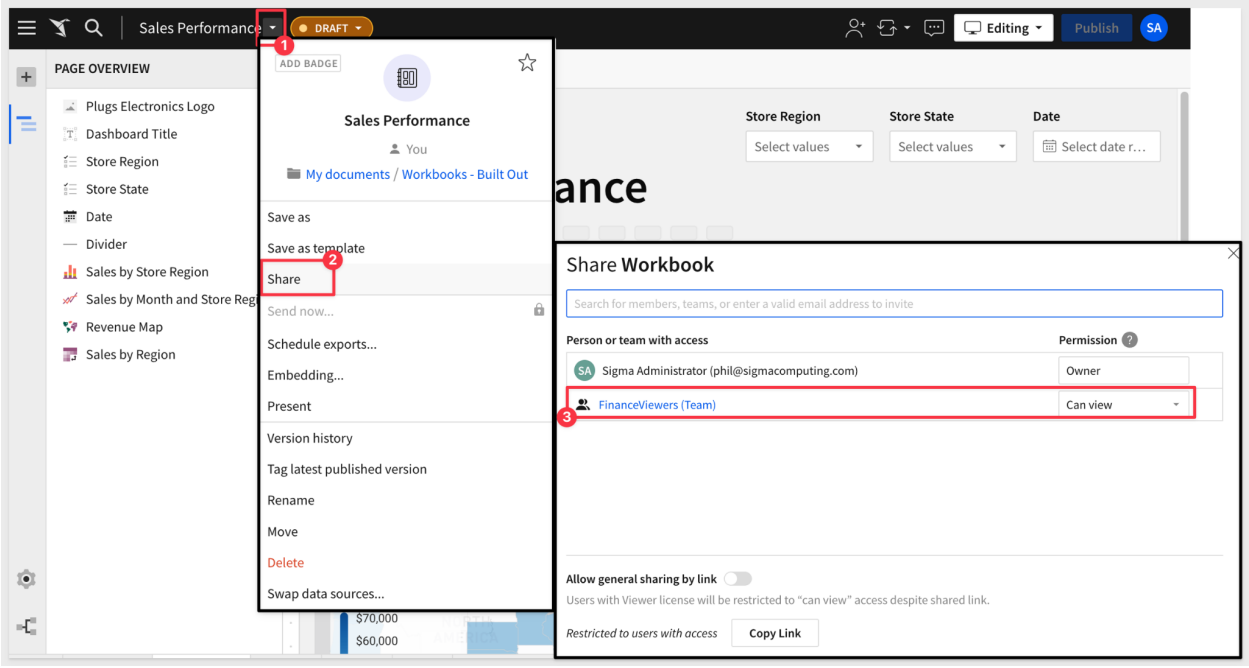
In Sigma, we will select a dashboard to embed. It does not matter which you select, but the below dashboard is provided in the Sigma trial environment so that we will use it for simplicity. Note that the red arrow indicates that the base dashboard shows all sales regions. Later, we will use RLS to show only the east and west regions automatically.



Sample dashboard from Sigma trial environment.

Step 2: Share the dashboard

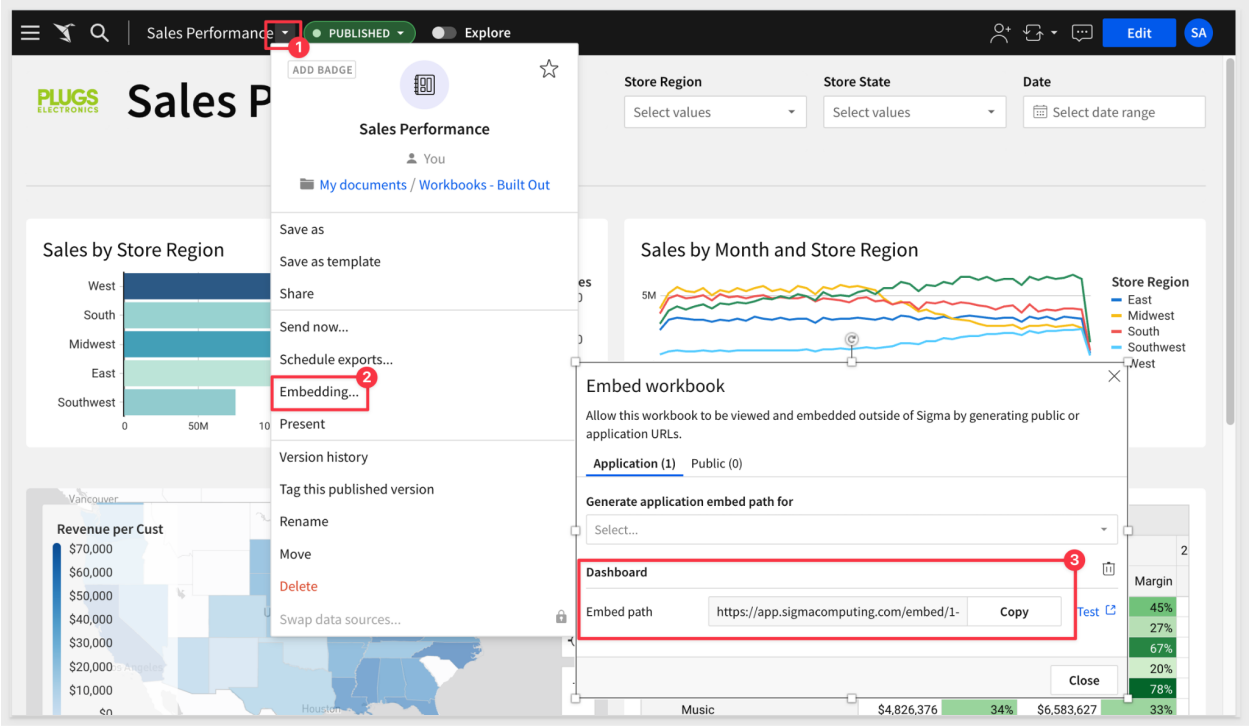
For this example, we will share the dashboard with the “FinanceViewers” team. To do this, click on the caret next to “Sales Performance” (the title of the workbook), then click “Share.” From there, select the person or team to grant access to. In this example, select the “FinanceViewers” team, then click “Share.”



How to share a dashboard with a person or team.

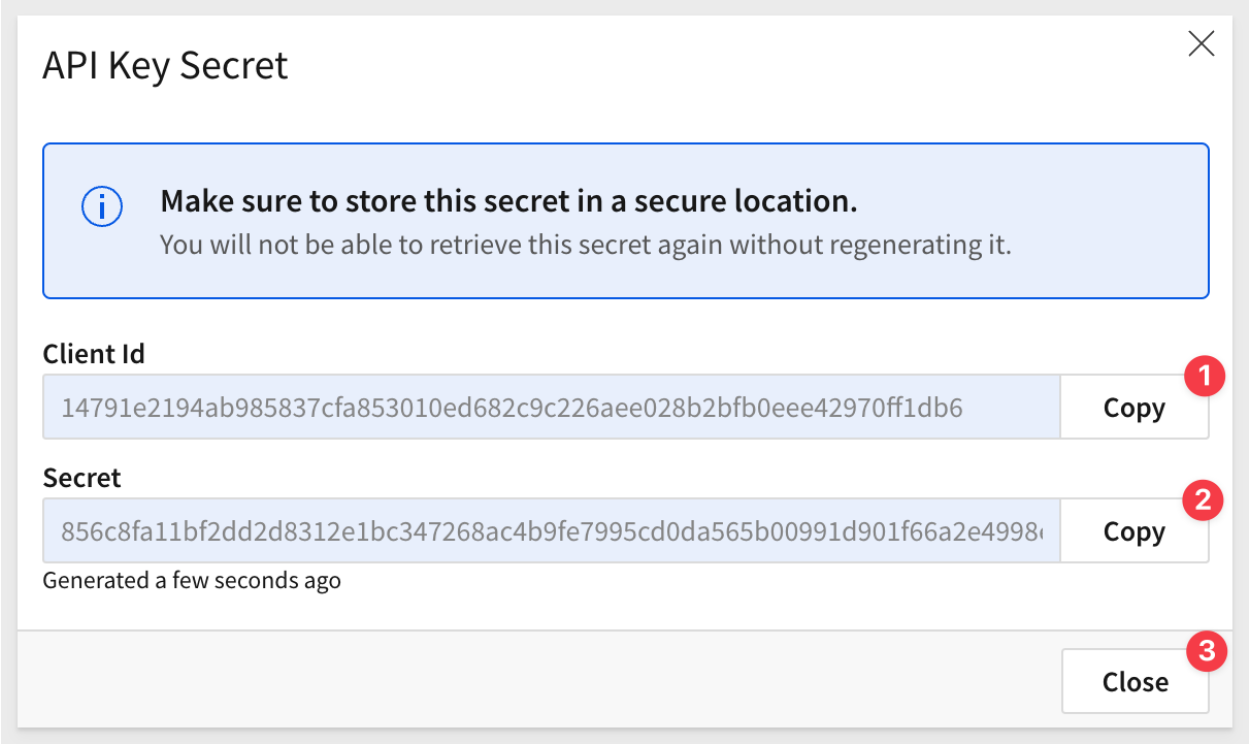
Step 3: Copy the embedding path

Next, we need to get the embed path (URL) from the dashboard menu. Again, go to “Sales Performance” in the header, click the caret, then select “Embedding…”. A pop-up will appear. Under the dashboard section, click “Copy” to the right of the embed path.



How to copy the embed path (URL).

From there, we must use Sigma’s administrative panel to generate keys. These keys work to establish a trust relationship between your application and your Sigma instance. Be sure to copy the provided Client Id and Secret and store them, then click “Close.”



How to copy the Client Id and Secret.

Now that we have copied our embed URL and secrets, we can place and configure our server-side API.

For this example, we will create a simple node.js project. This project will use JavaScript for the API and embed it into a simple HTML page.

NOTE: The code and process details are available here if you'd like to explore this on your own.

Step 4: Set Up the Node.js Express Server

Initialize the Server: Begin by creating the **`server.js`** file and initializing the Express application. Import the required **`express`** and **`crypto`** modules at the start of the file.

```
const express = require('express');
const crypto = require('crypto');
const app = express();
```

Configure Embedding Parameters: Define constants for EMBED_PATH and EMBED_SECRET. These should be replaced with the actual values you obtain from your Sigma embedding setup.

```
const EMBED_PATH = 'YOUR EMBED_PATH HERE'; // Replace with embed path
const EMBED_SECRET = 'YOUR API SECRET HERE'; // Replace with API secret
```

Serve the Main HTML Page: Set up a route handler to serve your main HTML page when the root URL is accessed.

```
app.get('/', (req, res) => {
    res.sendFile(__dirname + '/index.html');
});
```

Generate the Embed URL: Create an API endpoint that will handle the generation of the embed URL, which includes security measures like nonce generation and signature creation.

Insert the following code block at this point in your server.js file to define the **`/api/foo`** endpoint:

```
// API endpoint to generate the embed URL
app.get('/api/foo', (req, res) => {
    const nonce = crypto.randomUUID();
    let searchParams = new URLSearchParams({
        ':nonce': nonce,
        ':client_id': 'YOUR CLIENTID HERE', // Replace with client ID
        ':email': 'embed_viewer@sigmacomputing.com',
        ':external_user_id': 'embed_viewer@sigmacomputing.com',
        ':external_user_team': 'FinanceViewers',
        ':account_type': 'Viewer',
        ':mode': 'userbacked',
        ':session_length': '600',
        ':time': Math.floor(new Date().getTime() / 1000).toString()
    });

    const URL_WITH_SEARCH_PARAMS = EMBED_PATH + searchParams.toString();
    const signature = crypto
        .createHmac('sha256', Buffer.from(EMBED_SECRET, 'utf8'))
        .update(Buffer.from(URL_WITH_SEARCH_PARAMS, 'utf8'))
        .digest('hex');

    const URL_TO_SEND = `${URL_WITH_SEARCH_PARAMS}&:signature=${signature}`;
    res.status(200).send({ url: URL_TO_SEND });
});
```

Start the Server: Write the command to start the server, listening for requests on port 3000. You can confirm the server is running by checking for a console message.

```
app.listen(3000, () => {
    console.log('Server listening on port 3000');
});
```

The **`server.js`** file should contain all the steps above, with the cleaned-up code provided in the appropriate section to handle the **`/api/foo`** endpoint. This endpoint is crucial as it constructs the secure URL needed to embed Sigma visualizations with the proper parameters and security features in place.

Summary: You can find the entire code below:

```
const express = require('express');
const crypto = require('crypto');

const app = express();
const EMBED_PATH = 'YOUR EMBED_PATH HERE'; // Replace with your actual embed
path
const EMBED_SECRET = 'YOUR API SECRET HERE'; // Replace with your actual API
secret

// Serve the main HTML page
app.get('/', (req, res) => {
    res.sendFile(__dirname + '/index.html');
});

// API endpoint to generate the embed URL
app.get('/api/foo', (req, res) => {
    const nonce = crypto.randomUUID();
    let searchParams = new URLSearchParams({
        ':nonce': nonce,
        ':client_id': 'YOUR CLIENTID HERE', // Replace with your actual
client ID
        ':email': 'embed_viewer@sigmacomputing.com',
        ':external_user_id': 'embed_viewer@sigmacomputing.com',
        ':external_user_team': 'FinanceViewers',
        ':account_type': 'Viewer',
        ':mode': 'userbacked',
        ':session_length': '600',
        ':time': Math.floor(new Date().getTime() / 1000).toString()
    });

    const URL_WITH_SEARCH_PARAMS = EMBED_PATH + searchParams.toString();
    const signature = crypto
        .createHmac('sha256', Buffer.from(EMBED_SECRET, 'utf8'))
        .update(Buffer.from(URL_WITH_SEARCH_PARAMS, 'utf8'))
        .digest('hex');

    const URL_TO_SEND = `${URL_WITH_SEARCH_PARAMS}&:signature=${signature}`;
    res.status(200).send({ url: URL_TO_SEND });
});

// Start the server
app.listen(3000, () => {
    console.log('Server listening on port 3000');
});
```

Step 5: Match the values for your embed

In this step, you will update your **`server.js`** file to include the specific values for your Sigma embed configuration. This ensures that the server generates a correctly formatted URL that aligns with your Sigma setup and user requirements.

Set Embed Path and Secret: Replace the placeholder values for **`EMBED_PATH`** and **`EMBED_SECRET`** with the actual values you have saved from your Sigma account setup.

```
const EMBED_PATH = 'INSERT YOUR EMBEDDED PATH HERE';
const EMBED_SECRET = 'INSERT YOUR SECRET HERE';
```

Build the Search Parameters: Construct the search parameters by incrementally adding the required details. This includes the client ID, user email, user ID, team, and account type. The parameters must match the values expected by Sigma for successful embedding.

```
// Start with the base parameters including the nonce
let searchParams = new URLSearchParams({
    ':nonce': crypto.randomUUID(),
    // Add more parameters below
});

// Add the client ID
searchParams.append(':client_id', 'YOUR CLIENTID HERE');

// Add the email address of the authenticated user
// Use dynamically retrieved email in production
searchParams.append(':email', 'finance_viewer@testco.com');


// Add the user ID of the authenticated user
// Use dynamically retrieved user ID in production
searchParams.append(':external_user_id', '12345');


// Add the team of the user
// Replace with the actual team name
searchParams.append(':external_user_team', 'FinanceViewers');

// Add the account type of the user
```

```
// Replace with the actual account type
searchParams.append(':account_type', 'Viewer');
```

Remember to replace the placeholder text with actual data specific to your environment. In a production environment, the email and user ID should be programmatically retrieved rather than hardcoded.

Complete the URL Construction: After appending all the necessary parameters, combine them with the embed path to form the entire URL string. Then, create a secure signature and append it to the final URL.

```
const URL_WITH_SEARCH_PARAMS = EMBED_PATH + searchParams.toString();

const signature = crypto
    .createHmac('sha256', Buffer.from(EMBED_SECRET, 'utf8'))
    .update(Buffer.from(URL_WITH_SEARCH_PARAMS, 'utf8'))
    .digest('hex');

const URL_TO_SEND = `${URL_WITH_SEARCH_PARAMS}&:signature=${signature}`;
```

Send the URL in Response: Within the **`/api/foo`** endpoint, ensure that the final URL is sent back in the response to the client.

```
app.get('/api/foo', (req, res) => {
    // ... previous code to generate the URL ...
    res.status(200).send({ url: URL_TO_SEND });
});
```

Once you have updated these values, save the changes to your server.js file. Your server is now configured to generate URLs that are matched to your Sigma embed configuration and can be used to embed Sigma visualizations into your web application securely.

Step 6: Start the Node.js Express Server

With the configurations set, it's time to start the Node.js server and verify that the Sigma embed is operational within your HTML page.

Launch the Server: Use the command line interface to navigate to the directory where your server.js file is located. Start the server by running the following command:

```
node server.js
```

Server Initialization: Upon running the command, the server will initialize and start listening for requests on the specified port (in this case, port 3000). You should see a console output confirming the action:

```
console.log('Server listening on port 3000');
```

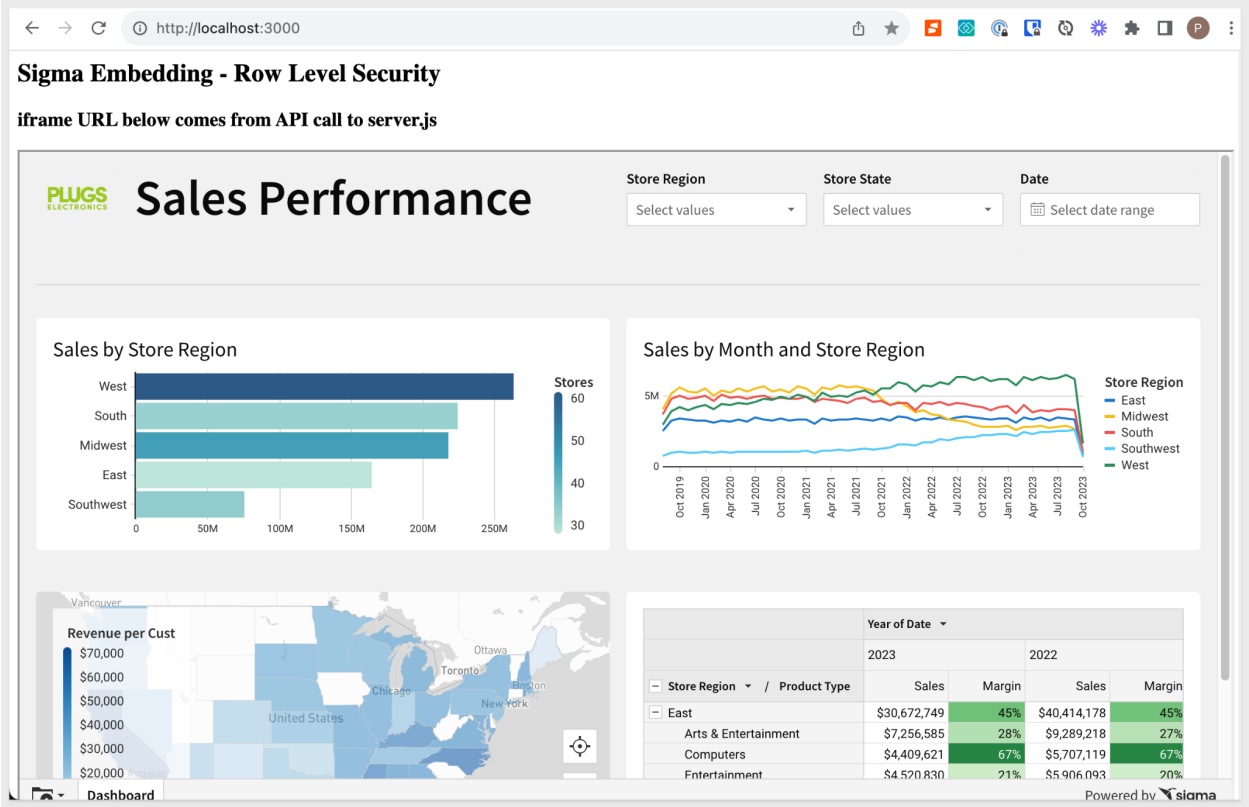
Access the Main Page: Open a web browser and navigate to **`http://localhost:3000/`**. You should see your main HTML page being served by the server.

Verify the Embed URL: The main HTML page should make a request to the **`/api/foo`** endpoint to retrieve the embed URL. This will be used to set the src attribute of an iframe that will display the Sigma visualization.

For reference, the HTML page code is as follows, with the iFrame code shown in red:

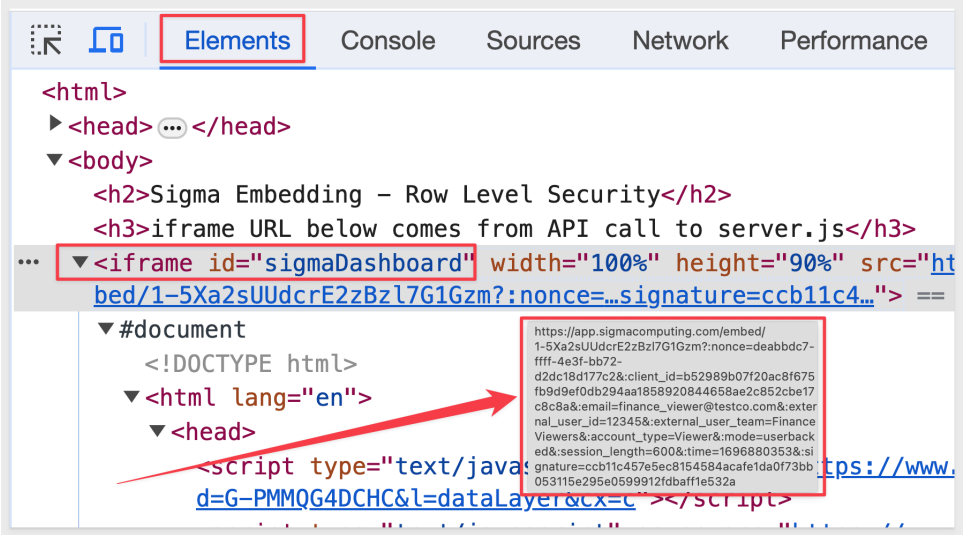
```
<html>
  <head>
    <title>Sigma Embedding - RLS</title>
  </head>
  <body>
    <h2>Sigma Embedding - Row Level Security</h2>
    <h3>iframe URL below comes from API call to server</h3>
    <iframe id="sigmaDashboard" width="100%" height="90%"></iframe>
    <script>
      const URL = "http://localhost:3000/api/foo";
      fetch(URL)
      .then(data => {return data.json()} )
      .then(res => {
         document.getElementById("sigmaDashboard).src = res.url
      })
      .catch(e => console.log(e));
    </script>
  </body>
</html>
```

Embed Rendering: With the URL set, the Sigma dashboard should render within the iframe. You'll see the embedded Sigma visualization on the page.



How an embed will look when running localhost:3000.

Note: The signature is a SHA256 hash of the secret and the final URL, with all of the parameters.



Screenshot of the one-time URL.

Accessing the “inspect” mode of the browser, you will see a one-time URL.

If we inspect that URL, here is what we will find:

```
<iframe id="sigmaDashboard" width="100%" height="90%" src="

  THIS IS THE EMBED PATH:
  https://app.sigmacomputing.com/embed/1-5Xa2sUUdcrE2zBzl7G1Gzm

  THIS IS THE NONCE:
  ?:nonce=deabbdc7-ffff-4e3f-bb72-d2dc18d177c2

  THIS IS THE CLIENTID:
  &amp;:client_id=b52989b07f20ac8f675fb9d9ef0db294aa144658ae2c852cbe17c8c8a

  THIS IS THE USER'S EMAIL:
  &amp;:email=finance_viewer@testco.com

  THIS IS THE USER'S UNIQUE ID:
  &amp;:external_user_id=12345

  THIS IS THE USER'S TEAM:
  &amp;:external_user_team=FinanceViewers

  THIS IS THE USER'S ACCOUNT TYPE:
  &amp;:account_type=Viewer

  THIS IS THE TYPE OF SIGMA EMBED (External):
  &amp;:mode=userbacked

  THIS IS THE TIME THE EMBED MAY LIVE BEFORE A NEW URL IS GENERATED:
  &amp;:session_length=600

  THIS IS THE UTC TIME THE URL WAS LAST GENERATED:
  &amp;:time=1696880353

  THIS IS THE UNIQUE SIGNATURE:
  &amp;:signature=ccb11c457e5ec8154584acafe1a0f73b5e295e0599912fdbaff1e532a

  ">
</iframe>
```

Troubleshooting

- If the Sigma visualization does not appear, check the browser's developer console for any errors.
- Ensure that CORS policies are appropriately handled if the request is made to a different domain.
- Verify that the **`EMBED_PATH`** and **`EMBED_SECRET`** are correctly set in your **`server.js`** file.

Enforcing Row Level Security

Next, we must govern the data individual users can access. We'll accomplish this through Row Level Security (RLS) in external embedding.

Let's assume we want our "FinanceViewers" to only access stores in the East and West regions. Our data has a column (Store Region) that allows us to support this with RLS.

In Sigma, our example workbook has multiple tabs, with one tab (Data) acting as the "source." We are using this data to build dashboards and visualizations, querying the warehouse once. There are a few different methods to apply a user attribute to Sigma data:

1. Use the user attribute value that is passed in a custom SQL query as a condition in a where clause. Since this involves writing some SQL, we probably want to avoid that and let Sigma do the SQL work for us in the background.

2. Add a Sigma filter control to the workbook that is viewable by the creator, but hidden from the average user.

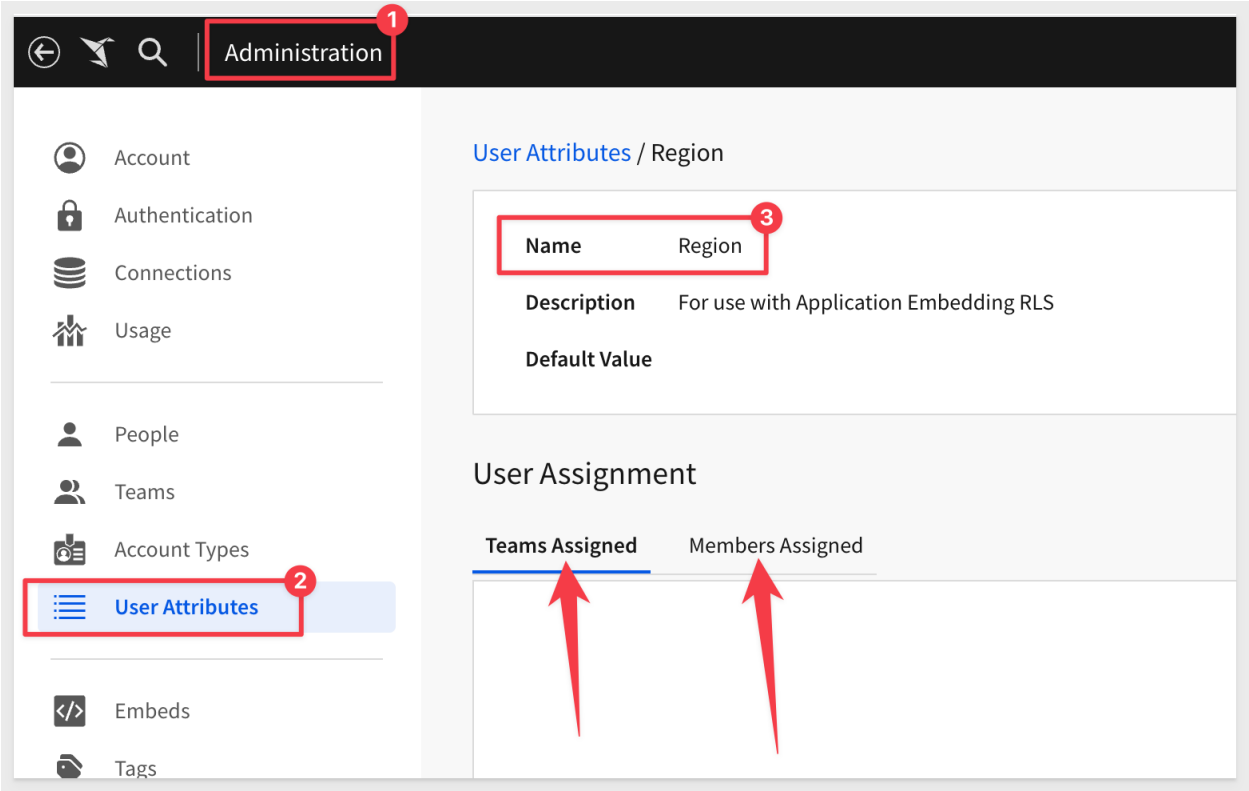
3. Apply the user attribute directly to the source data as a function that drives the column data, in this case, "Store Region." This is the preferred method and will be used in the following example.

Sigma's API supports any value passed to a custom "user attribute" at runtime. By simply adding a formula to the dataset, the API evaluates the new "user attribute" and filters the data according to the value(s) passed. First, we need to add a new user attribute in Sigma. We will call our user attribute "Region" and leave "Default Value" blank. This ensures that no data will be shown unless the API passes a value for ua_Region.

NOTE: you can also assign user attributes to Teams or individual members. In this example, we will use the user attribute directly.

Row-Level Security Example

To assign users to a user attribute, go to the Administration screen, then on the left select User Attributes. You should be able to add a Region for row-level security. In the example below, you will also be able to see Teams or Members assigned.

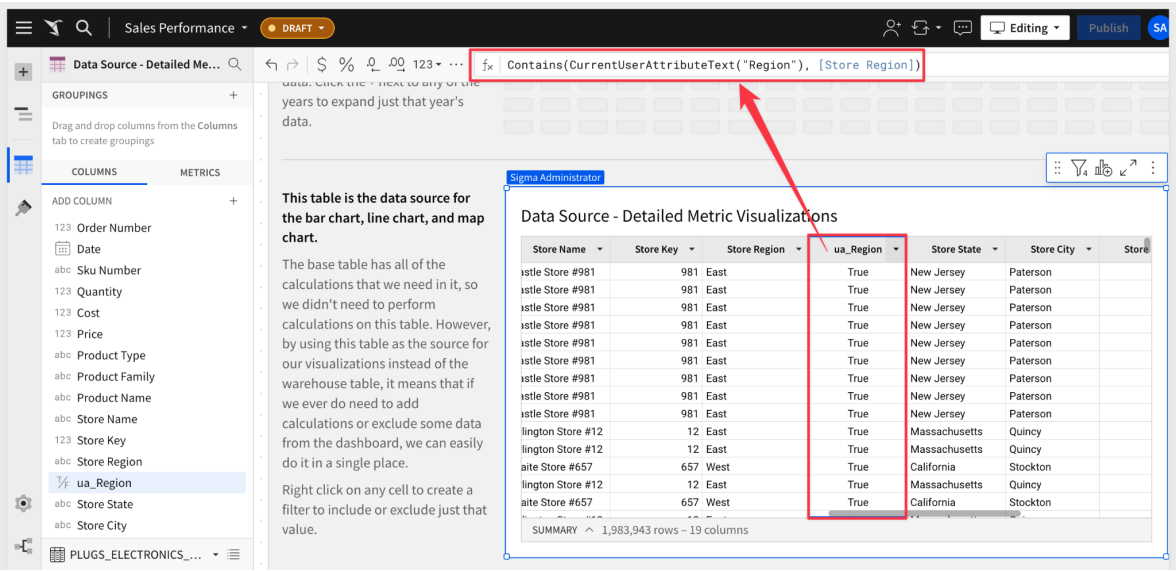


Screenshot of where to find and assign user attributes

Next, go back to your data source, and add a new column to the source data on the data tab called **`ua_Region`**.

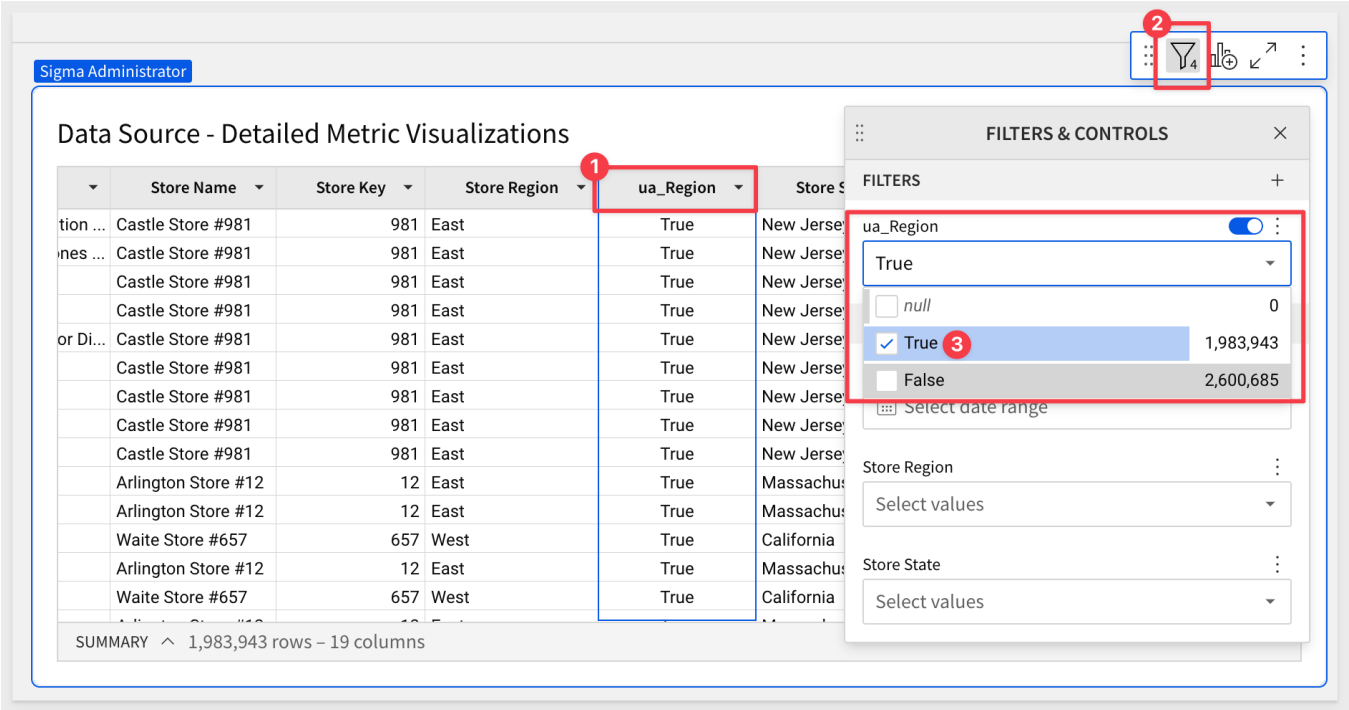
This column will use the following formula:

```
Contains(CurrentUserAttributeText("Region"), [Store Region])
```

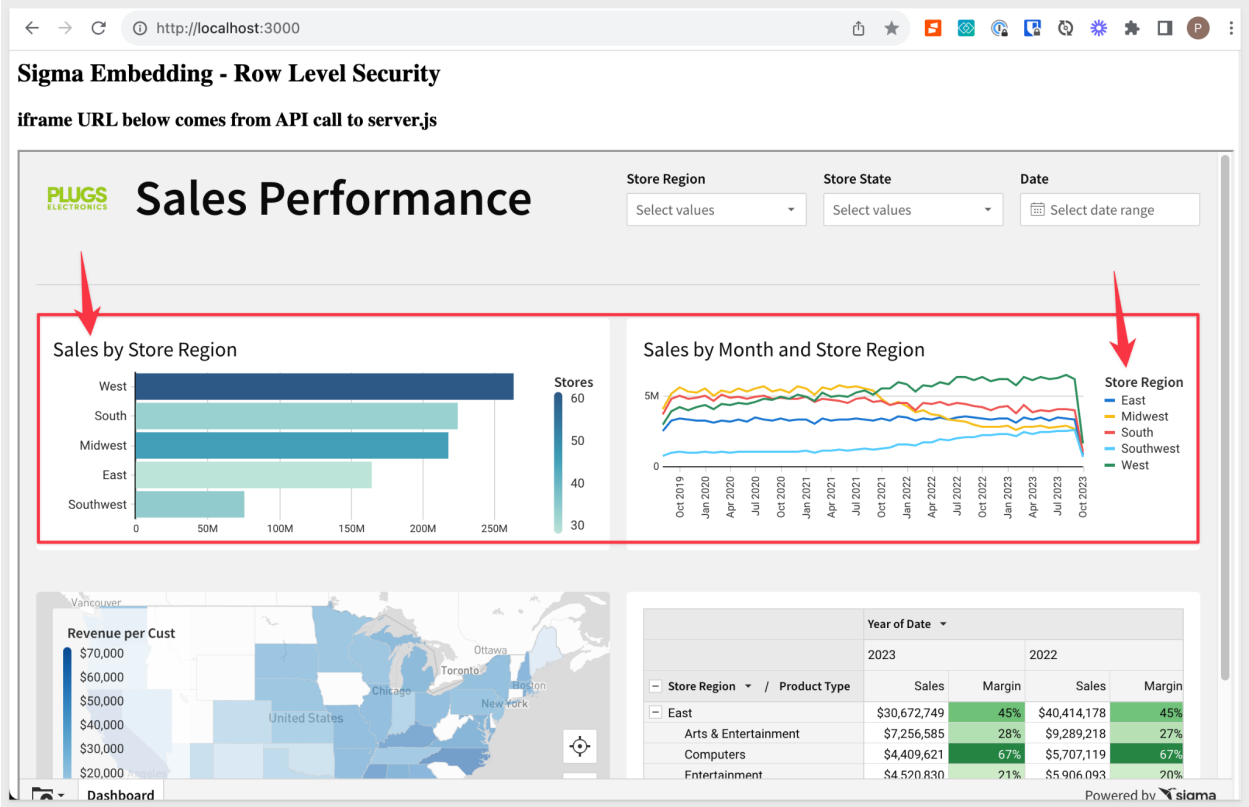


The contains function in the new ua_region column.

Then add a filter against the new column, **`ua_Region`**, and select **`True`**. The result of this will be a table with only the matching rows passed by the API for **`ua_Region`**.



Screenshot of the filtered table on the user attribute of Region.



Screenshot of the visualizations before leveraging row-level security.

Adjusting the API to Utilize User Attributes

To incorporate the newly defined user attributes into the API, follow these steps to modify your **`server.js`** file:

Specify Embed Path and Secret: Assign the **`EMBED_PATH`** and **`EMBED_SECRET`** variables with the values provided by your Sigma embedding setup.

```
const EMBED_PATH = 'INSERT_YOUR_EMBED_PATH_HERE';
const EMBED_SECRET = 'INSERT_YOUR_EMBED_SECRET_HERE';
```

Constructing the Embed URL: Begin building your embed URL by appending each required parameter to the searchParams object.

```
// Initialize searchParams with the nonce
let searchParams = new URLSearchParams({ ':nonce': crypto.randomUUID() });

// Append the client ID
// Replace with your client ID
searchParams.append(':client_id', 'INSERT_YOUR_CLIENT_ID_HERE');

// Append the email address of the user being authenticated
// Replace with the email of the authenticated user
searchParams.append(':email', 'embed_viewer@sigmacomputing.com');

// Append the external user ID, set to match the user's email for simplicity
// Replace with the external user ID
searchParams.append(':external_user_id', 'embed_viewer@sigmacomputing.com');

// Append the team associated with the user
// Replace with the appropriate team name
searchParams.append(':external_user_team', 'FinanceViewers');

// Append the account type for the user
// Ensure this account type is predefined in Sigma
searchParams.append(':account_type', 'Viewer');
```

Incorporate User Attributes for Row-Level Security (RLS): To apply RLS, add the user attribute ua_Region to the searchParams. The value for this should be the regions you wish to include, separated by commas.

```
// Append the user attribute for RLS. Case sensitivity matters.
searchParams.append(':ua_Region', 'West,Midwest,South,East,Southwest');
```

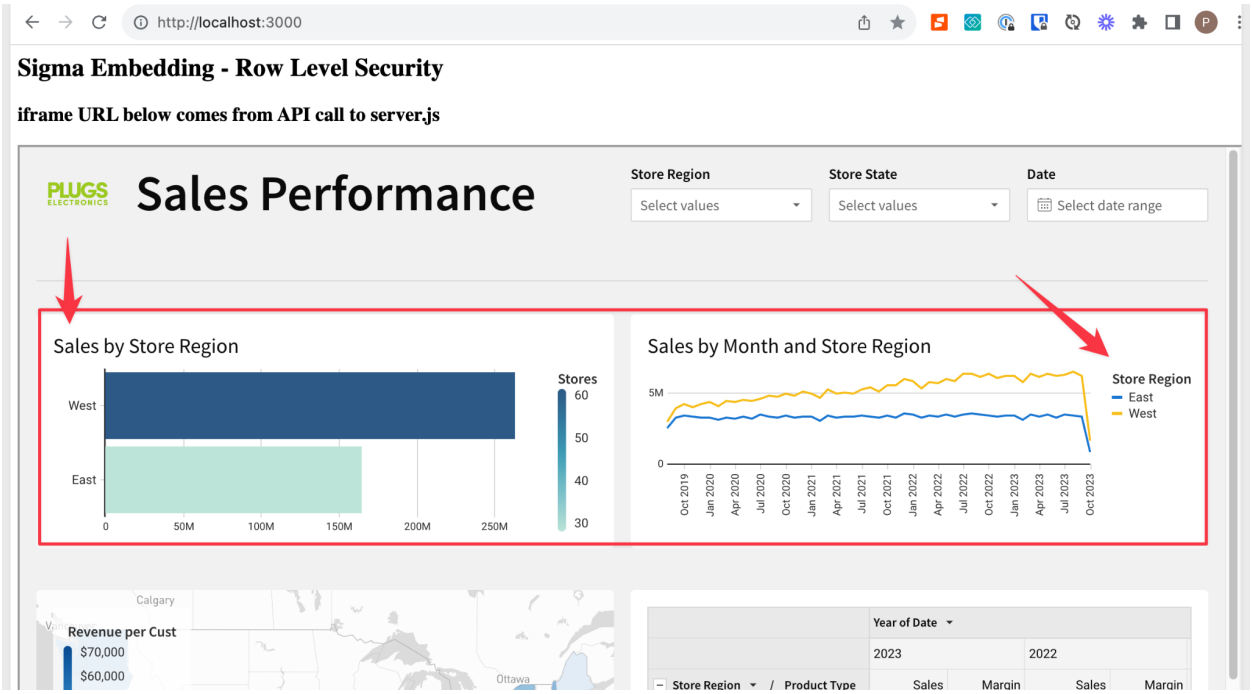
Finalize the URL: Combine the **`searchParams`** with the **`EMBED_PATH`** to form the complete embed URL. Then, generate the secure signature and append it to the URL.

```
// Combine the base embed path with the search parameters
const URL_WITH_SEARCH_PARAMS = `${EMBED_PATH}?${searchParams.toString()}`;

// Generate the secure signature
const signature = crypto
    .createHmac('sha256', EMBED_SECRET)  // Use the embed secret
    .update(URL_WITH_SEARCH_PARAMS)      // Apply the full URL
    .digest('hex');                      // Output as hexadecimal string

// Append the signature to the final URL to authenticate the request
const FINAL_URL = `${URL_WITH_SEARCH_PARAMS}&:signature=${signature}`;
```

Save and Test: After updating your code, save the **`server.js`** file and refresh your browser to test the changes. If implemented correctly, you should now see Sigma visualizations reflecting data for all specified store regions.



Screenshot of the visualizations after leveraging row-level security.

Sigma's Unique Encryption

We previously discussed how external embedding is made possible by creating a unique, encrypted, one-time-use embed URL. To test this, copy and paste the below URL that has been generated by the API and modified to include the South region into your browser. You'll notice the URL returns an error message as it can only be used once.

```
https://app.sigmacomputing.com/embed/1-5QFPkoPbkjUHD2wzC8fLgl?:nonce=a0fc25ed-7bc5-481b-b793-9272c9c48b2b&amp;:client_id=b52989b07f20ac8f675fb9d9ef0db294a1858920844658ae2c852cbe17c8c8a&amp;:email=finance_viewer@testco.com&amp;:external_user_id=12345&amp;:external_user_team=FinanceViewers&amp;:account_type=Viewer&amp;:ua_Region=East,West,South&amp;:mode=userbacked&amp;:session_length=600&amp;:time=1696885437&amp;:signature=1637ea9a037fe8f223bbaa0e0c1b09311c0d5e0b0715d691c934a4bf86b0f9f0
```

[Learn more about Sigma's API]

Conclusion

Throughout this white paper, we explored different ways to embed visualizations while navigating through various authentication mechanisms. The decision rests on configuring your existing environment and your organization's security goals.

Public Embedding

Offers an open, non-restricted view without security layers. It's akin to a public domain, accessible to all without authentication barriers.

Internal and External Embedding

Both present comparable security features, notably Row-Level Security (RLS). While internal embedding often relies on pre-existing authentication systems like SSO, external embedding relies on unique, secure URLs.

It's important to understand that embedding isn't a one-size-fits-all approach. Your organizational needs might utilize multiple embedding techniques. For instance, while internal embedding might resonate for your internal platforms, external embedding will better suit your external, customer-centric portals.

To try out Sigma's embedded platform capabilities today, sign up for a free trial.

You can quickly connect to your data and be on your way to prototyping your new embedding solution!