**Canvas** is a data visualization and presentation tool that allows you to pull live data from Elasticsearch, then combine the data with colors, images, text, and your imagination to create dynamic, multi-page, pixel-perfect displays.

With **Canvas**, you can:

* Create and personalize your work space with backgrounds, borders, colors, fonts, and more.
* Customize your workpad with your own visualizations, such as images and text.
* Pull your data directly from Elasticsearch, then show it off with charts, graphs, progress monitors, and more.
* Focus the data you want to display with filters.

**Presentations:**

Present Elasticsearch data in a way that tells your company's story — whether it's presenting flight statistics at an airport kiosk or reviewing auth logs in a conference room. Keep your team engaged with beautiful, real-time displays featuring social analytics, user engagement, operational analytics, KPIs, or any data really. Plus, take your visualizations outside of Kibana — Canvas shareables enable you to embed static workpads directly into HTML sites.

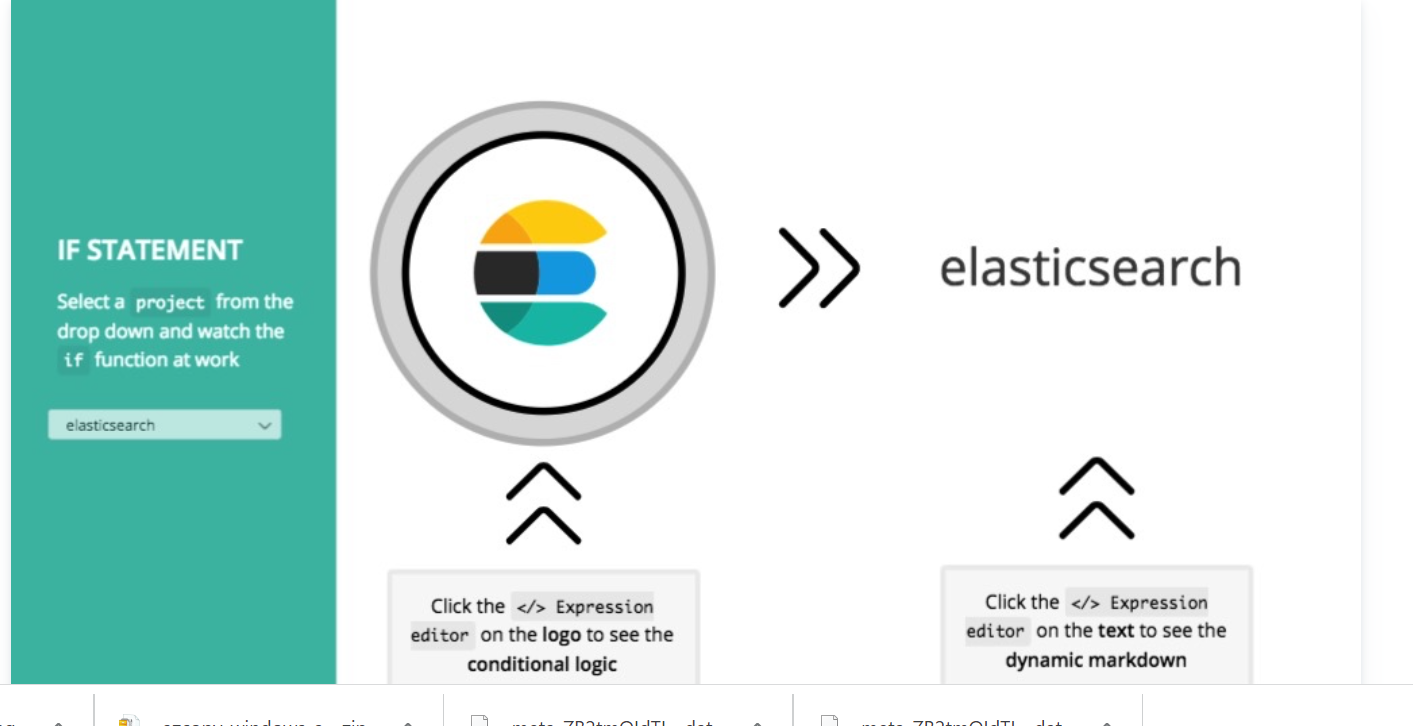


**Implementation**

Canvas is flexible so you can break from the rigidity of a grid and bring to life the delightful things that matter. Think yellow submarines, purple rain, blackbirds — or whatever speaks to you. Everything is a data point.

Options:

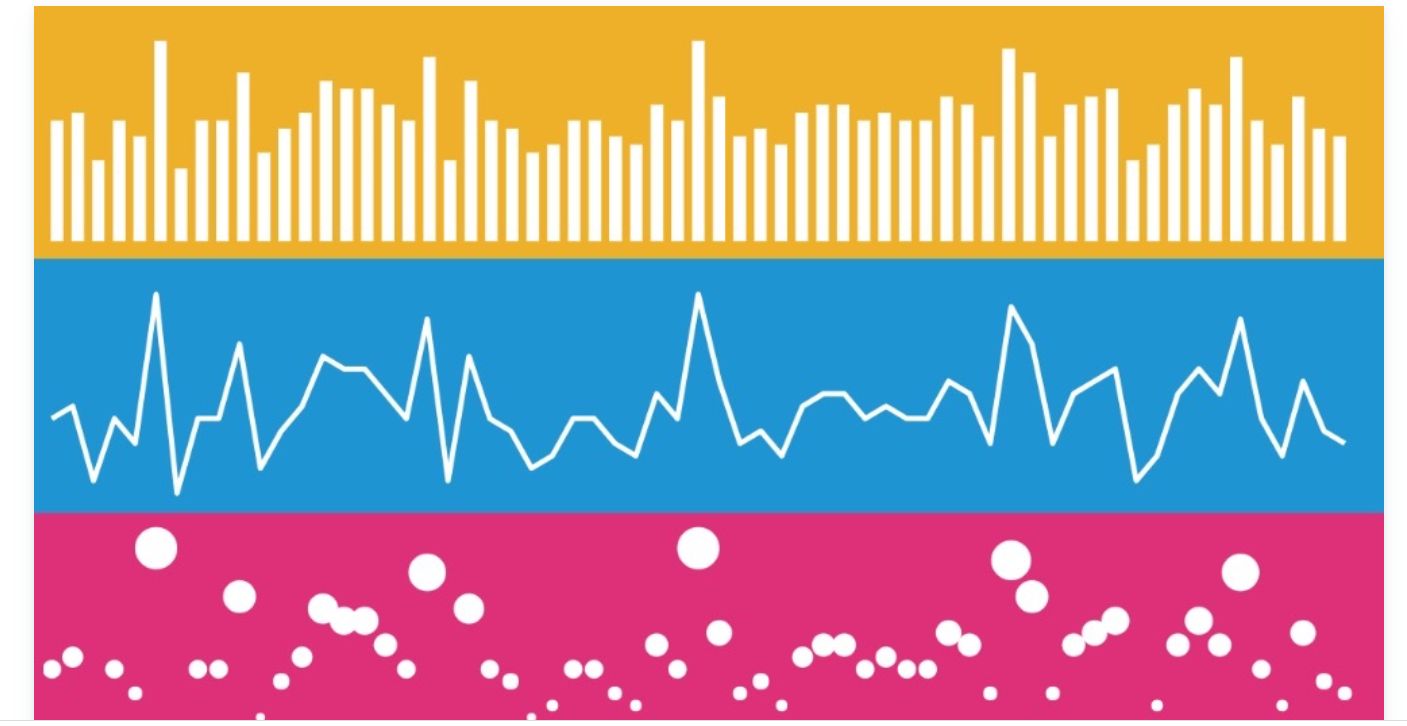
Tutorials:



Design



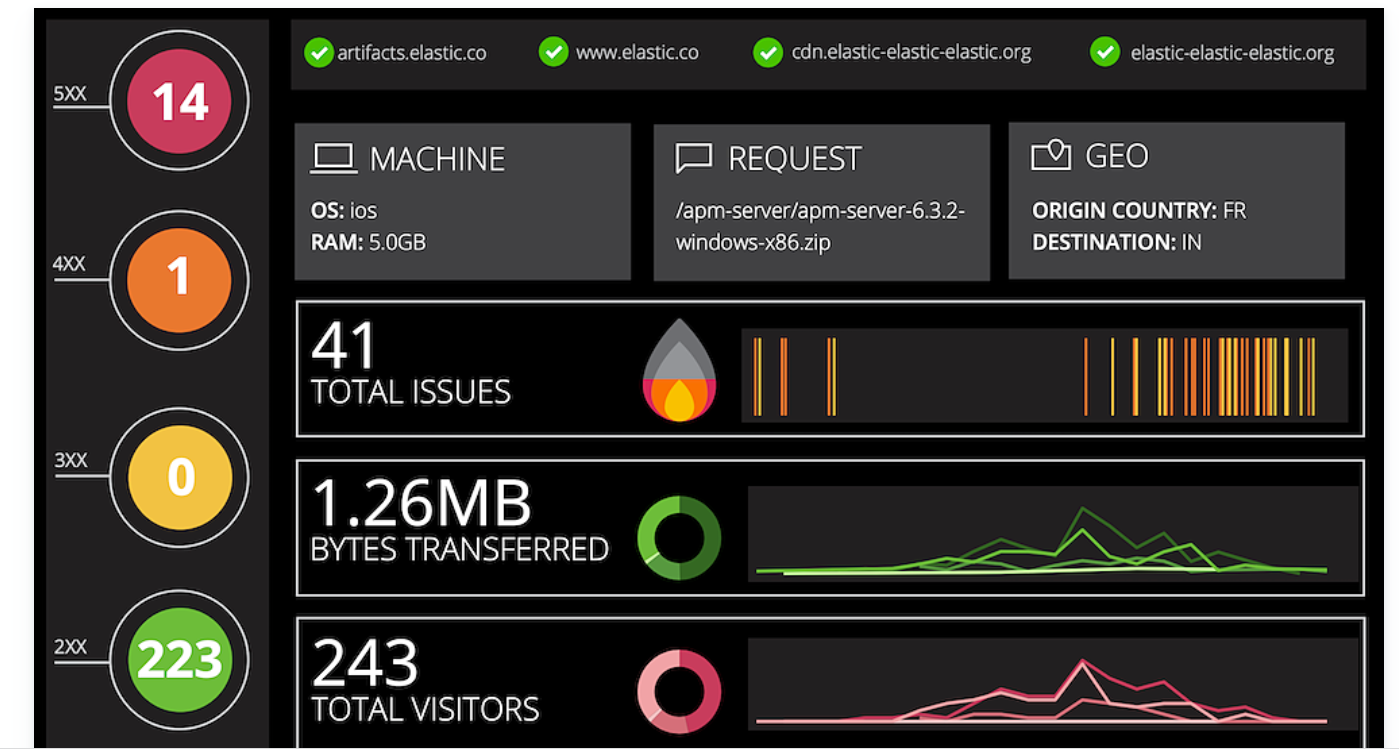
Analysis



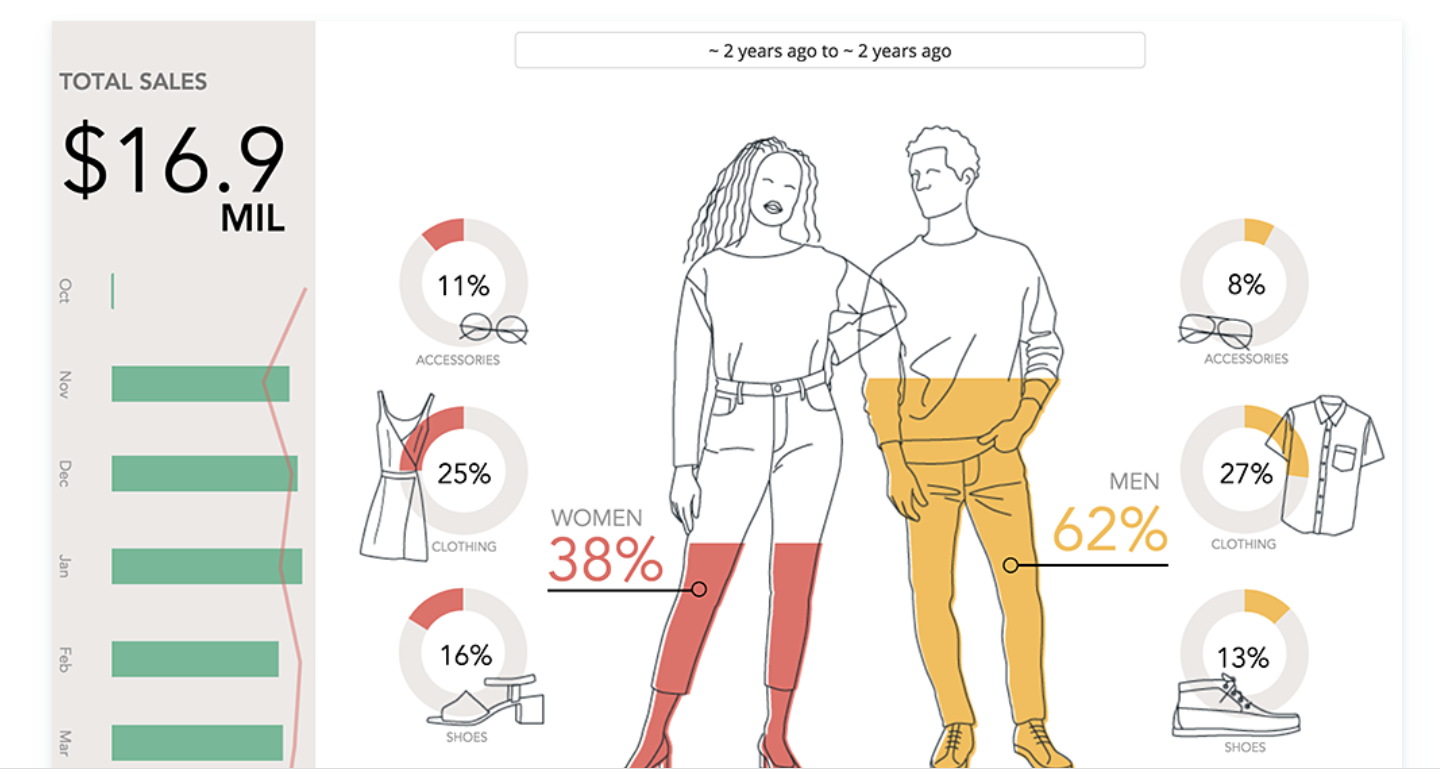
Log Analysis



Infrastructure Monitoring



Business analytics



**First step:**

**Create workpads**

A workpad provides you with a space where you can build presentations of your live data. You can create a workpad from scratch, start with a preconfigured workpad, import an existing workpad, or use a sample data workpad.

**Start with a blank workpad**

To use the background colors, images, and data of your choice, start with a blank workpad.

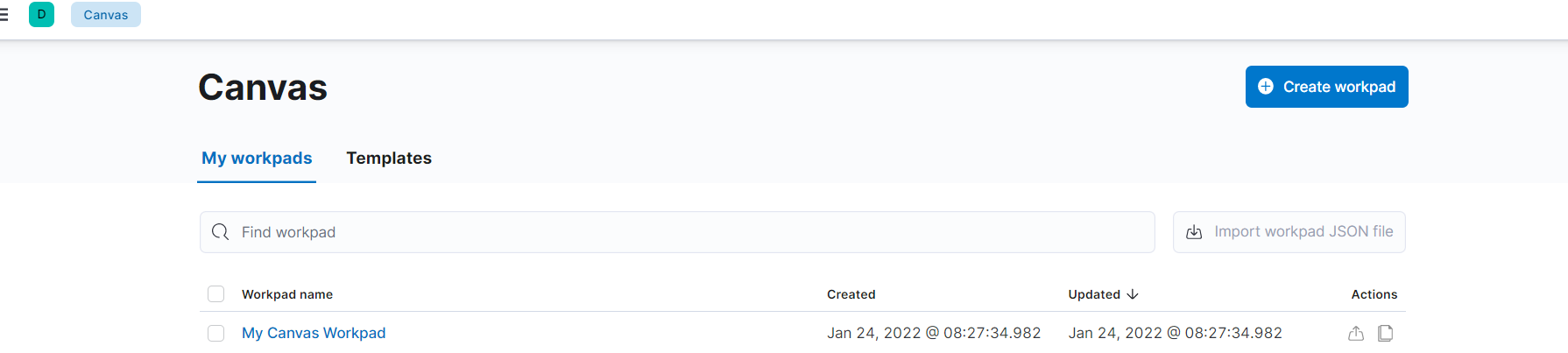
On the Canvas workpads page, click Create workpad.

Specify the Workpad settings.

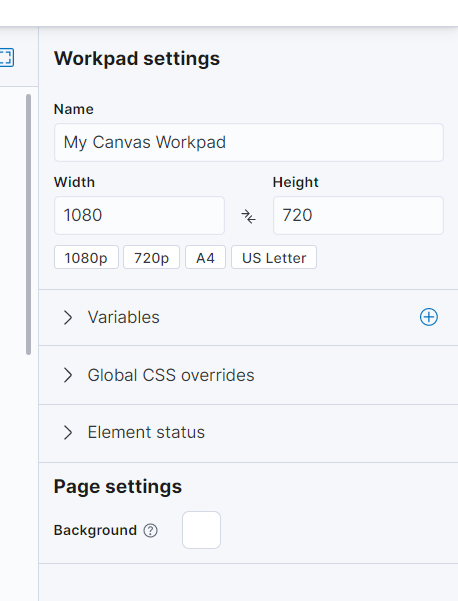
Add a Name to your workpad.

In the Width and Height fields, specify the size, or select one of default layouts.

Click the Background color picker, then select the color for your workpad.



Below you can find your workpad settings:



You can also create workpads from templates or import them.

**Create workpads from templates**

If you’re unsure about where to start, you can use one of the preconfigured templates that come with **Canvas**.

1. On the **Canvas workpads** page, select **Templates**.
2. Click the preconfigured template that you want to use.
3. Add your own **Name** to the workpad.

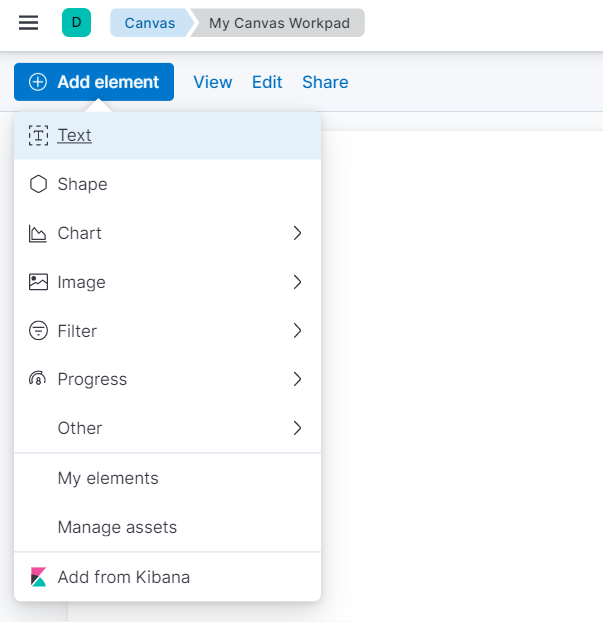
**Import existing workpads**

When you want to use a workpad that someone else has already started, import the JSON file.

To begin, drag the file to the **Import workpad JSON file** field on the **Canvas workpads** page.

ADD Elements:

Create a story about your data by adding elements to your workpad that include images, text, charts, and more.



**Create Elements**

Choose the type of element you want to use, then use the preconfigured demo data to familiarize yourself with the element. When you’re ready, connect the element to your own data. By default, most of the elements you create use the demo data until you change the data source. The demo data includes a small data set that you can use to experiment with your element.

1. Click **Add element**, then select the element you want to use.
2. To connect the element to your data, select **Data**, then select one of the following data sources:
   * **Elasticsearch SQL** — Access your data in Elasticsearch using [SQL syntax](https://www.elastic.co/guide/en/elasticsearch/reference/7.16/sql-spec.html).
   * **Elasticsearch documents** — Access your data in Elasticsearch without using aggregations. To use, select an index and fields, and optionally enter a query using the [Lucene Query Syntax](https://www.elastic.co/guide/en/kibana/current/lucene-query.html). Use the **Elasticsearch documents** data source when you have low volume datasets, to view raw documents, or to plot exact, non-aggregated values on a chart.
   * **Timelion** — Access your time series data using [**Timelion**](https://www.elastic.co/guide/en/kibana/current/timelion.html) queries. To use **Timelion** queries, you can enter a query using the [Lucene Query Syntax](https://www.elastic.co/guide/en/kibana/current/lucene-query.html).

Each element can display a different data source, and pages and workpads often contain multiple data sources.

As you can see in the image , you can also add visualisations and dashboards from Kibana directly or enhance your presentation with Images and Text.

**ADD Kibana objects:**

Add a panel that you saved in **Visualize Library** to your workpad.

1. Click **Add element > Add from Kibana**.
2. Select the panel you want to add.
3. To use the customization options, open the panel menu, then select one of the following options:
   * **Edit map** — Opens [Maps](https://www.elastic.co/guide/en/kibana/current/maps.html) so that you can edit the panel.
   * **Edit visualization** — Opens the visualization editor so that you can edit the panel.
   * **Edit panel title** — Allows you to change the panel title.
   * **Customize time range** — Allows you to change the time filter dedicated to the panel.
   * **Inspect** — Allows you to drill down into the panel data.

ADD Images:

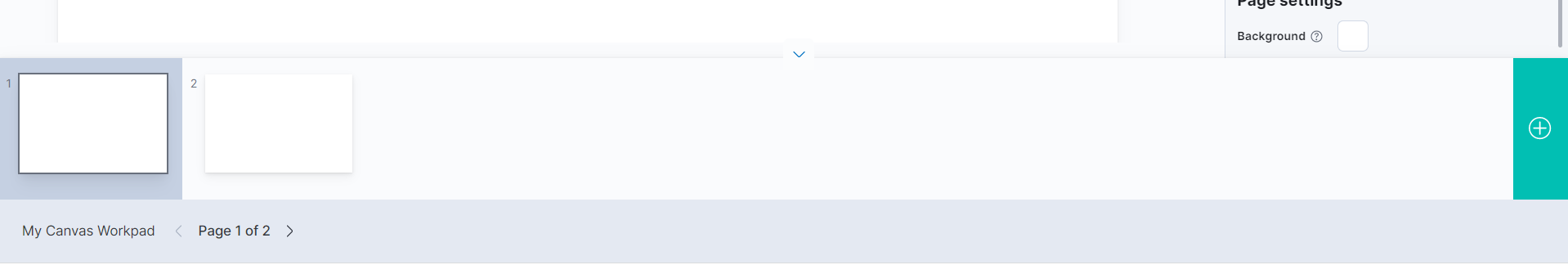
To personalize your workpad, add your own logos and graphics.

1. Click **Add element > Manage assets**.
2. On the **Manage workpad assets** window, drag and drop your images.
3. To add the image to the workpad, click the **Create image element** icon.

ADD Pages:

Organize and separate your ideas by adding more pages.

1. Click **Page 1**, then click **+**.
2. On the **Page** editor panel, select the page transition from the **Transition** dropdown.



**Canvas expression lifecycle**

Elements in Canvas are all created using an **expression language** that defines how to retrieve, manipulate, and ultimately visualize data. The goal is to allow you to do most of what you need without understanding the **expression language**, but learning how it works unlocks a lot of Canvas’s power.

Expressions simply execute [functions](https://www.elastic.co/guide/en/kibana/current/canvas-function-reference.html) in a specific order, which produce some output value. That output can then be inserted into another function, and another after that, until it produces the output you need.

To use demo dataset available in Canvas to produce a table, run the following expression:

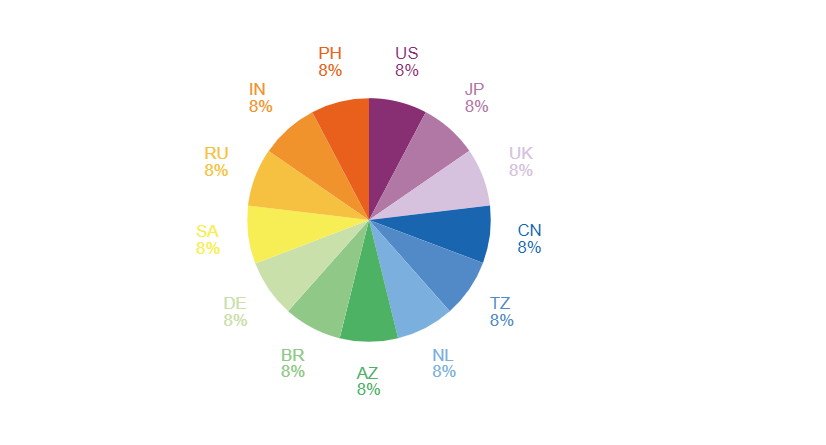


This expression starts out with the [filters](https://www.elastic.co/guide/en/kibana/current/canvas-function-reference.html#filters_fn) function, which provides the value of any time filters or dropdown filters in the workpad. This is then inserted into [demodata](https://www.elastic.co/guide/en/kibana/current/canvas-function-reference.html#demodata_fn), a function that returns exactly what you expect, demo data. Because the [demodata](https://www.elastic.co/guide/en/kibana/current/canvas-function-reference.html#demodata_fn) function receives the filter information from the [filters](https://www.elastic.co/guide/en/kibana/current/canvas-function-reference.html#filters_fn) function before it, it applies those filters to reduce the set of data it returns. We call the output from the previous function context.

The filtered [demo data](https://www.elastic.co/guide/en/kibana/current/canvas-function-reference.html#demodata_fn) becomes the context of the next function, [table](https://www.elastic.co/guide/en/kibana/current/canvas-function-reference.html#table_fn), which creates a table visualization from this data set. The [table](https://www.elastic.co/guide/en/kibana/current/canvas-function-reference.html#table_fn) function isn’t strictly required, but by being explicit, you have the option of providing arguments to control things like the font used in the table. The output of the [table](https://www.elastic.co/guide/en/kibana/current/canvas-function-reference.html#table_fn) function becomes the context of the [render](https://www.elastic.co/guide/en/kibana/current/canvas-function-reference.html#render_fn) function. Like the [table](https://www.elastic.co/guide/en/kibana/current/canvas-function-reference.html#table_fn), the [render](https://www.elastic.co/guide/en/kibana/current/canvas-function-reference.html#render_fn) function isn’t required either, but it allows access to other arguments, such as styling the border of the element or injecting custom CSS.

Function Arguments:

Let’s look at another expression, which uses the same [demodata](https://www.elastic.co/guide/en/kibana/current/canvas-function-reference.html#demodata_fn) function, but instead produces a pie chart.



Expression editor



To produce a filtered set of random data, the expression uses the [filters](https://www.elastic.co/guide/en/kibana/current/canvas-function-reference.html#filters_fn) and [demodata](https://www.elastic.co/guide/en/kibana/current/canvas-function-reference.html#demodata_fn) functions. This time, however, the output becomes the context for the [pointseries](https://www.elastic.co/guide/en/kibana/current/canvas-function-reference.html#pointseries_fn) function, which is a way to aggregate your data, similar to how Elasticsearch works, but more generalized. In this case, the data is split up using the color and size dimensions, using arguments on the [pointseries](https://www.elastic.co/guide/en/kibana/current/canvas-function-reference.html#pointseries_fn) function. Each unique value in the state column will have an associated size value, which in this case, will be the maximum value of the price column.

If the expression stopped there, it would produce a pointseries data type as the output of this expression like below.

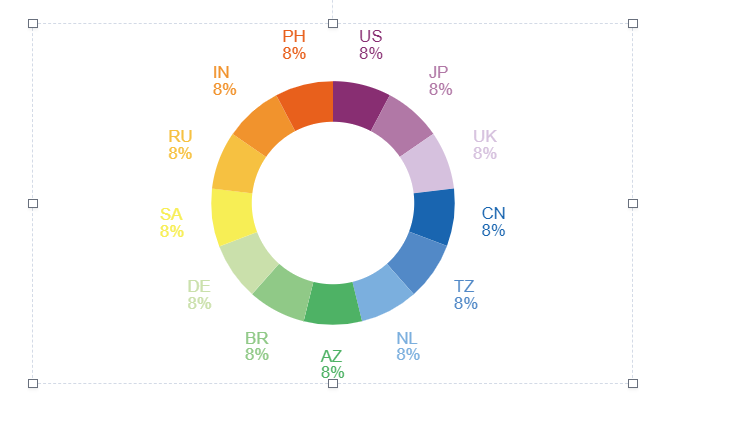


But instead of looking at the raw values, the result is inserted into the [pie](https://www.elastic.co/guide/en/kibana/current/canvas-function-reference.html#pie_fn) function, which will produce an output that will render a pie visualization. And just like before, this is inserted into the [render](https://www.elastic.co/guide/en/kibana/current/canvas-function-reference.html#render_fn) function, which is useful for its arguments.

The end result is a simple pie chart that uses the default color palette, but the [pie](https://www.elastic.co/guide/en/kibana/current/canvas-function-reference.html#pie_fn) function can take additional arguments that control how it gets rendered. For example, you can provide a hole argument to turn your pie chart into a donut chart by changing the expression to:



And the output would be as follows:

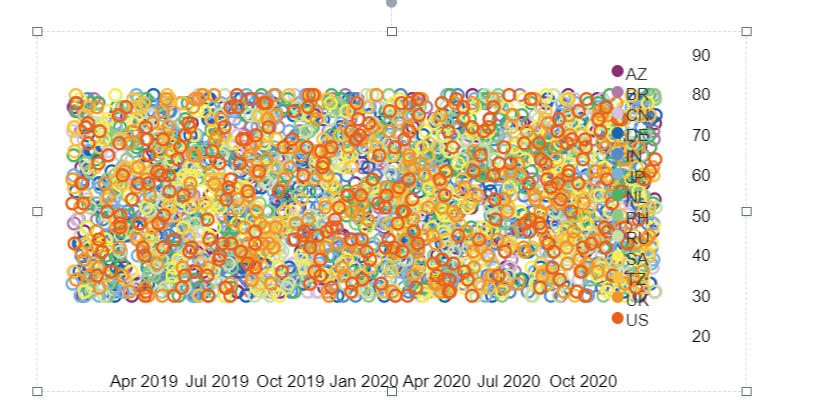


**Changing the output by changing the expression**

You can substitute one function for another to change the output. For example, you could change the visualization by swapping out the pie function for another renderer, a function that returns a render data type.

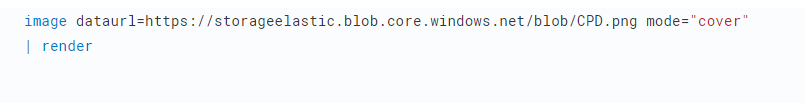
Let’s change that last pie chart into a bubble chart by replacing the pie function with the plot function. This is possible because both functions can accept a pointseries data type as their context. Switching the functions will work, but it won’t produce a useful visualization on its own since you don’t have the x-axis and y-axis defined. You will also need to modify the pointseries function to change its output. In this case, you can change the size argument to y, so the maximum price values are plotted on the y-axis, and add an x argument using the @timestamp field in the data to plot those values over time .Also the plot can be enhanced by controlling the axis.



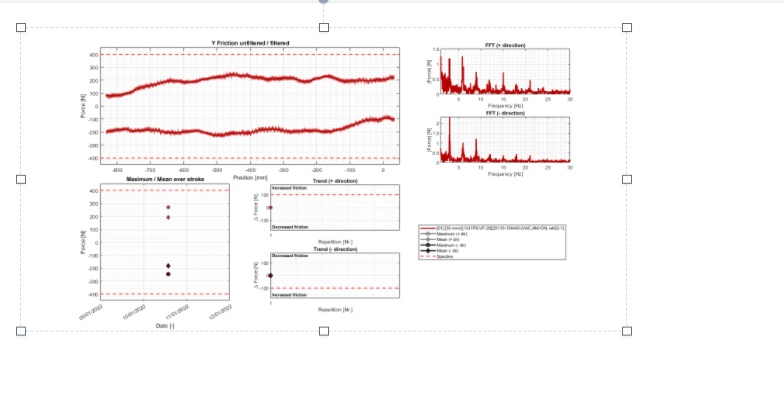


**Manipulate datasets**

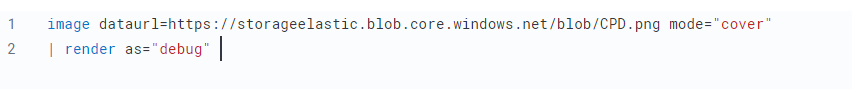
So far, you have only seen expressions as a way to produce visualizations, but that’s not really what’s happening. Expressions only produce data, which is then used to create something, which in the case of Canvas, means rendering an element. An element can be a visualization, driven by data, but it can also be something much simpler, like a static image. Either way, an expression is used to produce an output that is used to render the desired result. For example, here’s an expression that shows an image:



Here is the output:



But as mentioned, this doesn’t actually render that image, but instead it produces some output that can be used to render that image. That’s an important distinction, and you can see the actual output by adding in the render function and telling it to produce debug output. For example:



This will display the following JSON output:



Canvas uses this output’s data type to map to a specific renderer and passes the entire output into it. It’s up to the image render function to produce an image on the workpad’s page. In this case, the expression produces some JSON output, but expressions can also produce other, simpler data, like a string or a number. Typically, useful results use JSON.

Canvas uses the output to render an element, but other applications can use expressions to do pretty much anything. As stated previously, expressions simply execute functions, and the functions are all written in Javascript. That means if you can do something in Javascript, you can do it with an expression.

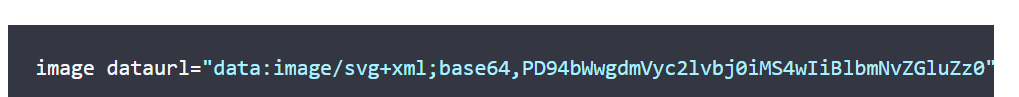
**Expressions and subexpressions**

Expressions inside of curly brackets are called sub-expressions, and they can be used to provide a calculated value to another expression, instead of just a static one.

A simple example of this is when you upload your own images to a Canvas workpad. That upload becomes an asset, and that asset can be retrieved using the asset function. Usually you’ll just do this from the UI, adding an image element to the page and uploading your image from the control in the sidebar, or picking an existing asset from there as well. In both cases, the system will consume that asset via the asset function, and you’ll end up with an expression similar to this:



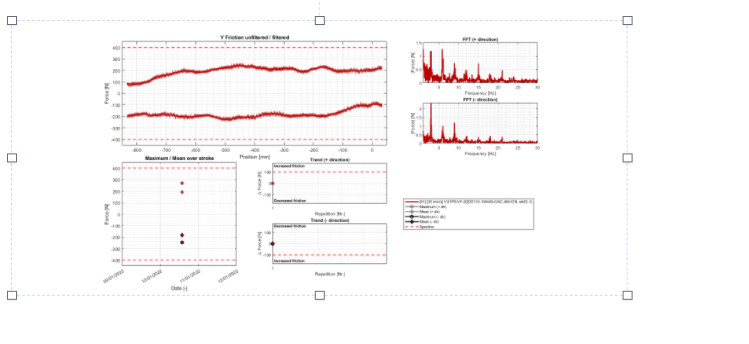
Sub-expressions are executed before the function that uses them is executed. In this case, asset will be run first, it will produce a value, the base64-encoded value of the image and that value will be used as the value for the dataurl argument in the image function. After the asset function executes, you will get the following output:



Using conditionals:

Since all of the sub-expressions are now resolved into actual values, the [image](https://www.elastic.co/guide/en/kibana/current/canvas-function-reference.html#image_fn) function can be executed to produce its JSON output, just as it’s explained previously. In the case of images, the ability to nest sub-expressions is particularly useful to show one of several images conditionally. For example, you could swap between two images based on some calculated value by mixing in the [if](https://www.elastic.co/guide/en/kibana/current/canvas-function-reference.html#if_fn) function, like in this example expression:

Here is the output , related to the 2 images:



And the code which led to the above:

demodata

| image dataurl={

  if condition={getCell price | gte 100}

    then={asset "asset-bcc89217-98a0-4e46-ba97-b1ba5cbf6fd8"}

    else={asset "asset-ca5254b1-e731-47b4-8f6b-9877e0c4adfc"}

}

Here, the expression to use for the value of the condition argument, getCell price | gte 100, runs first since it is nested deeper.

The expression does the following:

Retrieves the value from the price column in the first row of the demodata data table

Inputs the value to the gte function

Compares the value to 100

Returns true if the value is 100 or greater, and false if the value is 100 or less

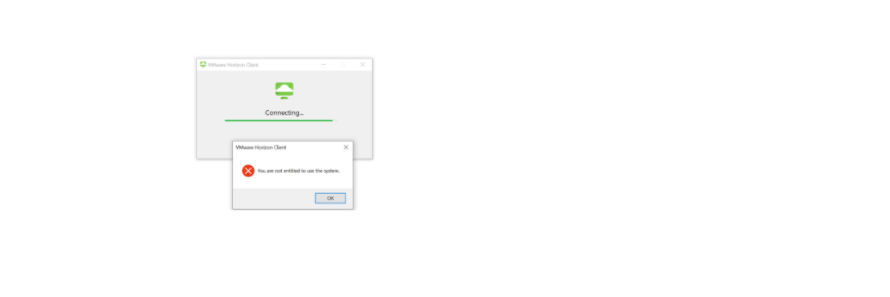
That boolean value becomes the value for the condition argument. The output from the then expression is used as the output when condition is true. The output from the else expression is used when condition is false. In both cases, a base64-encoded image will be returned, and one of the two images will be displayed.

You might be wondering how the getCell function in the sub-expression accessed the data from the demoData function, even though demoData was not being directly inserted into getCell. The answer is simple, but important to understand. When nested sub-expressions are executed, they automatically receive the same context, or output of the previous function that its parent function receives. In this specific expression, demodata’s data table is automatically provided to the nested expression’s getCell function, which allows that expression to pull out a value and compare it to another value.

By changing the condition we get the else statement displayed:

if condition={getCell price | lte 100}

And now the other image gets displayed:



USING ELASTICSEARCH SQL as DATABASE

You can fetch data from Elasticsearch using essql, which allows you to aggregate the data, provide a custom name for the value, and insert that data directly to another function that only accepts pointseries even though essql will output a datatable type. This makes the following example expression valid:

filters

| essql

  query="SELECT header.resource.id AS x, latency AS y

FROM \"ltds-\*-v002\"

WHERE header.profile= 'sd' and header.resource.id= '#3FM=B04++0+VU2+KS+0000-00000-BT511'

"

|table

| render

X and Y axis have been defined separately in the query and then rendered in the table.

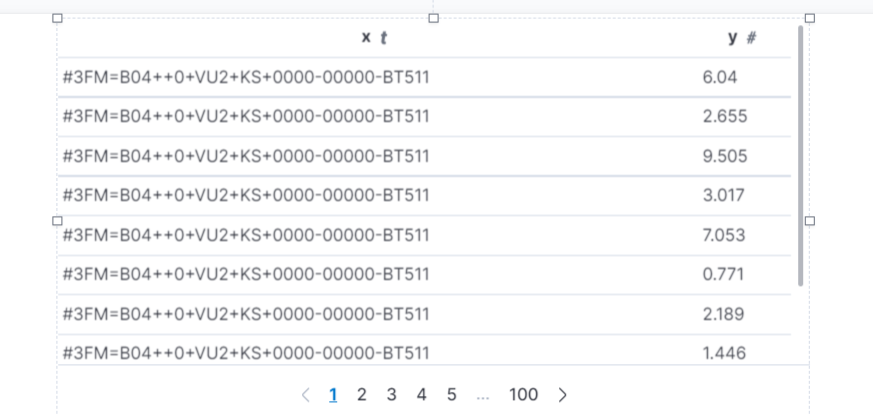
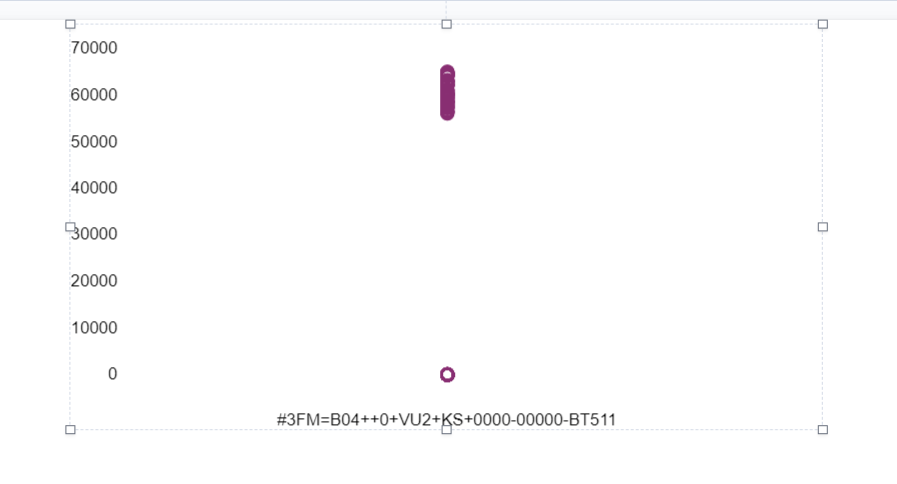
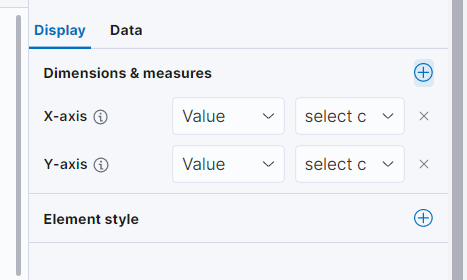


Table can be substituted by Plot to obtain the following :



To customize the query further , pointseries can also be used and the dimensions x and y can then be added from the UI:



The expression editor and resulting graph will look like this:

filters

| essql

  query="SELECT header.resource.id AS x, latency AS y

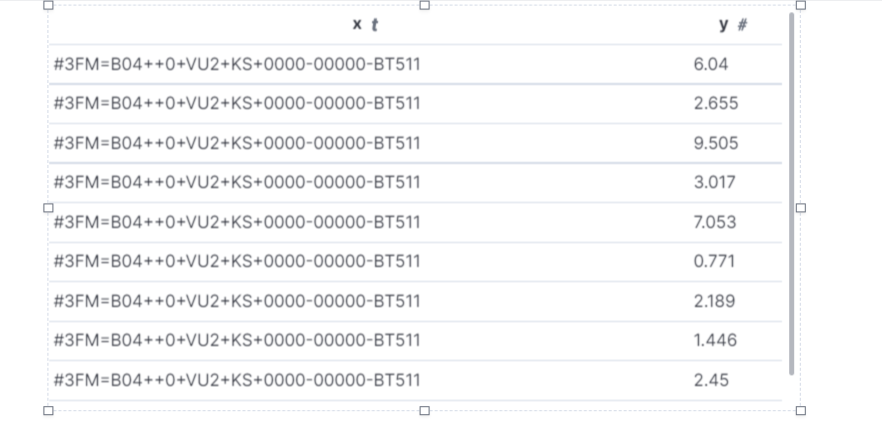
FROM \"ltds-\*-v002\"

WHERE header.profile= 'sd' and header.resource.id= '#3FM=B04++0+VU2+KS+0000-00000-BT511'

"

| pointseries x="x" y="y"

| render



Further functions can be implemented:

filters

| essql

  query="SELECT header.resource.id AS x, latency AS y

FROM \"ltds-\*-v002\"

WHERE header.resource.id= '#3FM=B04++0+VU2+KS+0000-00000-BT511'

"

| pointseries x="x" y="mean(y)"

| render

