

salesforce

Lightning Web Components in CRM Analytics Dashboards

Salesforce, Spring '24

'24



Last updated: March 22, 2024

CONTENTS

LIGHTNING WEB COMPONENTS IN CRM ANALYTICS DASHBOARDS	1
Architecture	1
Setup	3
HELLO WORLD COMPONENT EXAMPLE	5
Component Attributes	5
CRM Analytics Configuration	6
Query Data	10
Other Widgets	10
REFERENCE	15
RELEASE NOTES	19

LIGHTNING WEB COMPONENTS IN CRM ANALYTICS DASHBOARDS

With Lightning Web Components in dashboards, you can now extend CRM Analytics functionality with your own custom widgets.

Overview

With Lightning Web Components in dashboards, widgets can be a custom data visualization, a custom selection control, a richly formatted custom document, and more. The main benefits are:

- Part of the dashboard canvas - Any Lightning Web Component can be next to a dashboard, but it's always offset by spacing. Lightning Web Components in dashboards allow the component to render ON the dashboard canvas, and it can look and feel like a native widget. Also, the dashboard can have one or more Lightning Web Components inside it, so everything can be browsed, embedded, and consumed as a single, cohesive unit.
- No code data querying built-in - Each Lightning Web Component can be optionally associated with a query, also known as a step. This query provides a table of data to the Lightning Web Component, so that the developer writes less code to query and fetch data. Anyone configuring your component can use the CRM Analytics Explorer UI to configure any query against datasets, Salesforce Objects, or Snowflake with clicks and not code. Also, the Analytics Dashboard Designer UI has additional features like Embedded Filters, Global Filters, selective faceting, and bindings. Here you can configure how the widget interacts with other dashboard elements without adding any additional code to your Lightning Web Component.
- Attributes UI - The developer can specify attributes and their types that populate a configuration UI for each setup. These attributes are a great way to abstract:
 - Constants for use case defined parameters
 - Dynamic, calculated data
 - Metadata about how to use the columns of configured query results
 - Labels
 - Colors, sizes, and other styling attributes

By specifying the attribute type, your users see the appropriate configuration UI and validation for that type. And they always have the option to use the Designer IDE to create custom interactions to make any attribute dynamic.

- Set and Get State Functions - These functions provide the same functionality present in the CRM Analytics Dashboard Component. Your custom Lightning Web Component can get the entire state of the parent dashboard and modify it in any way. The benefit of a Lightning Web Component is that it's even easier to connect to the parent dashboard with auto-wiring.

[Architecture](#)

Let's look at how custom Lightning Web Component widgets fit into the existing CRM Analytics dashboard architecture.

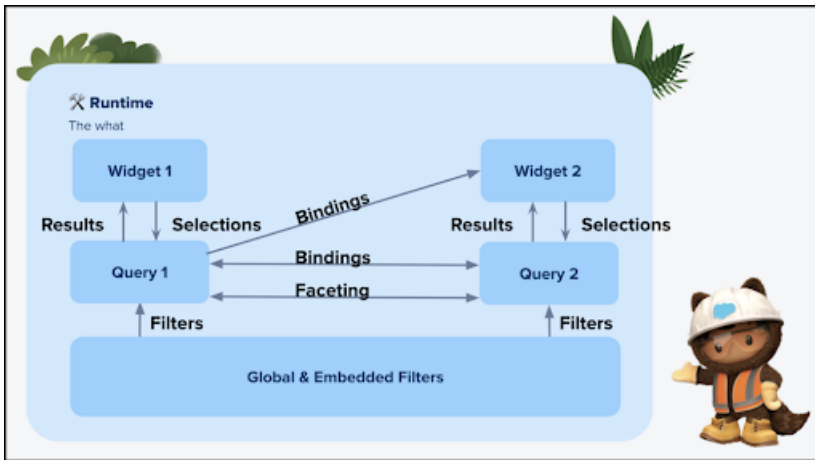
[Setup](#)

Set up your environment to create Lightning Web Component widgets, and then add them to your CRM Analytics dashboards.

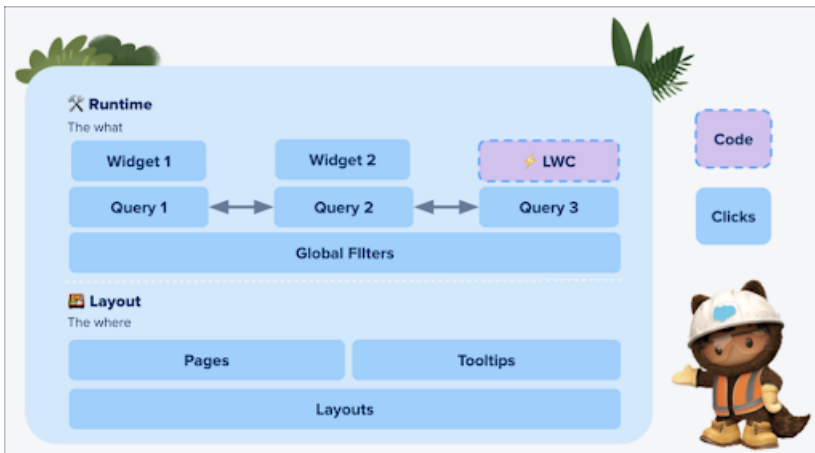
Architecture

Let's look at how custom Lightning Web Component widgets fit into the existing CRM Analytics dashboard architecture.

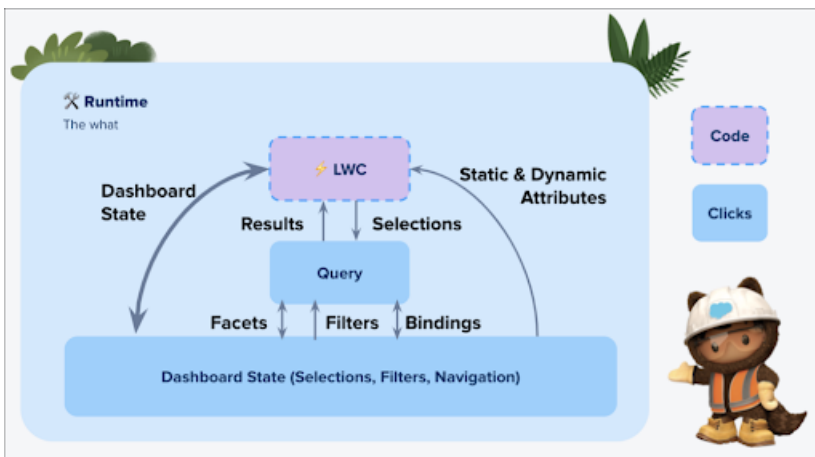
CRM Analytics dashboards work with a clean separation of concerns. Widgets get data from queries, and pass selection back to them. Queries can pass selections to each other through faceting and receive global filters from the core runtime. Steps can also pass selections and data to other components via bindings.



Now you have a way to put your code into the mix while taking advantage of this entire system.



Mixing custom code in with widgets creates a powerful dynamic, and you have many ways to get state and configuration into your code. Meanwhile, you can communicate with the parent dashboard by setting selections on the associated query or using the `setState` method.



Setup

Set up your environment to create Lightning Web Component widgets, and then add them to your CRM Analytics dashboards.

CRM Analytics provides a collection of easy-to-digest code examples for Lightning Web Components that are specific to CRM Analytics features. These examples demonstrate how to code charts, graphs, and hierarchies using third-party JS libraries and CRM Analytics features.

[Setup Instructions](#)

Use these instructions to set up your Salesforce org for development of Lightning Web Components and to use the CRM Analytics Lightning Web Component examples provided.

[Getting Started with Lightning Web Components](#)

Before continuing with this guide, we recommend you familiarize yourself with Lightning Web Components.

Setup Instructions

Use these instructions to set up your Salesforce org for development of Lightning Web Components and to use the CRM Analytics Lightning Web Component examples provided.

1. Create or use a Developer Edition (DE) org enabled for CRM Analytics. Sign up for a new CRM Analytics enabled DE org at developer.salesforce.com/promotions/orgs/analytics-de.
2. If you plan to publish any of your own Lightning Web Component work to App Exchange, enable your DE org as a dev hub. For more information about enabling dev hubs, see [Enable Dev Hub Features in Your Org](#).
3. Create a Connected App, a record client key, and a secret. DO NOT use the JWT flow. For more information on Connected Apps, see [Create a Connected App for Your Dev Hub Org](#).
4. Install the Salesforce CLI. For detailed instructions, see [Install Salesforce CLI](#). Alternatively, you can install Visual Studio Code to perform steps 6–10. See [Install Salesforce Extensions for Visual Studio](#).
5. Log in to your DE org via the CLI. For `org login web` and other command details, see the [CLI command reference guide](#).
6. Clone the GitHub repository containing the CRM Analytics Lightning Web Component example components. This repository is structured as a Salesforce DX project for use with CLI commands.


```
git clone https://github.com/forcedotcom/sfdx-analytics.git
```

7. See the [README file](#) in the repository for more information on the CRM Analytics Lightning Web Component example components.
8. Deploy the CRM Analytics Lightning Web Component examples in the project from the command line with

```
sf project deploy start --source-dir quick-start/main/default/lwc --target-org <USERNAME>
```

or use the SFDX: Deploy Source to Org command in Visual Studio code.

9. Use and adapt the CRM Analytics LWC examples to create your own custom Lightning Web Component widgets for CRM Analytics dashboards.
10. Create your own component using the SFDX: Create Analytics Dashboard LWC command in Visual Studio Code or a standard Lightning Web Component with the `sf lightning generate component` CLI command.

 **Note:** The SFDX: Create Analytics Dashboard LWC command requires installation of the `analyticsdx-vscode` extension pack.


Getting Started with Lightning Web Components

Before continuing with this guide, we recommend you familiarize yourself with Lightning Web Components.

There are many resources to help you get started, such as:

- [Introducing Lightning Web Components](#): The official Lightning Web Components Developer Guide and reference
- [Build Lightning Web Components](#): A Trailhead trail with QuickStart, basics, building, testing, and using events
- [Lightning Web Components - Episode 1: An Introduction](#): A video series on Lightning Web Components
- [Lightning Web Components Open Source](#): Documentation on the Lightning Web Components open-source UI framework

When you're comfortable with the basics and have your development environment set up, continue to the [Hello World Component Example](#).

 **Note:** If you create a Lightning Web Component that uses Lightning events, the events work when your CRM Analytics dashboard is embedded in Salesforce pages, but not from Analytics Studio. Analytics Studio is a separate Lightning app.

HELLO WORLD COMPONENT EXAMPLE

Create a simple Lightning Web Component widget to display Hello World in your CRM Analytics dashboard.

Component Attributes

You can convert any Lightning Web Component into a CRM Analytics Lightning Web Component by adding specific attributes.

CRM Analytics Configuration

Now that the Lightning Web Component code is ready to go, it's time to bring the component into a CRM Analytics dashboard.

Query Data

Let's learn how to inject query results into your running Lightning Web Component code.

Other Widgets

The Hello World example is a simplified scenario. The most common real world use cases are a custom data visualization, a selection widget, or even better, a widget that does both. With practice, you can add anything you want. To see a list, a third-party data visualization integration, a hierarchy, and more widgets, check out the example GitHub repository.

Component Attributes

You can convert any Lightning Web Component into a CRM Analytics Lightning Web Component by adding specific attributes.

The Lightning Web Component framework requires a special tag to know your component is compatible for use in CRM Analytics dashboards. Let's start by adding this XML to your `js-meta.xml` file:

```
<targets>
  <target>analytics__Dashboard</target>
</targets>
```

The `target` attribute informs the designer UI that this widget is compatible with CRM Analytics and allows the component to show up in the component selector. This stage in the process is also a good time to make sure your component widget is visible. To enable visibility, check out this line in the `js-meta.xml`:

```
<isExposed>true</isExposed>
```

Now any CRM Analytics dashboard author can add your widget after it's published to an org, but you also want to expose configuration options. To do that, create an attribute:

```
<targetConfigs>
  <targetConfig targets="analytics__Dashboard">
    <hasStep>false</hasStep>
    <property name="title" type="String" label="Title" description="Title of the
component" required="true" />
  </targetConfig>
</targetConfigs>
```

Here, you set `hasStep` to `false` because you aren't using any query data. You're adding a single attribute `title`, which is type `String`, so that the dashboard author can type any free text to configure. Set `required="true"` so that authors can't make blank components. When you're done, the whole file looks something like:

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
  <apiVersion>53.0</apiVersion>
  <isExposed>true</isExposed>

  <masterLabel>Hello LWC</masterLabel>
  <description>Test project for LWC.</description>

  <targets>
    <target>analytics__Dashboard</target>
  </targets>

  <targetConfigs>
    <targetConfig targets="analytics__Dashboard">
      <hasStep>false</hasStep>
      <property name="title" type="String" label="Title" description="Title of the
component" required="true" />
    </targetConfig>
  </targetConfigs>
</LightningComponentBundle>
```

 **Note:** Your API version can be different so that it matches the version of the Salesforce org that you're developing your Lightning Web Component for.

To wire up the component, start by adding these attributes with `@api` annotations in your `.js` file. After you add them, the `.js` file looks like:

```
import { LightningElement, api } from 'lwc';

export default class HelloWorld extends LightningElement {
  @api title;
}
```

You don't use any custom logic in this example, so no additional code is needed. You do need your component to say hello, so edit the HTML template in the `.html` file.

```
<template>
  Hello World: {title}
</template>
```

CRM Analytics Configuration

Now that the Lightning Web Component code is ready to go, it's time to bring the component into a CRM Analytics dashboard.

Basic Configuration

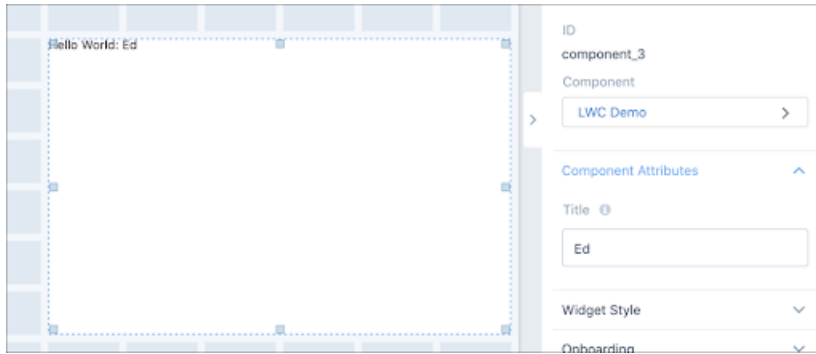
After pushing your code into a Salesforce org, go to the CRM Analytics dashboard designer and create a blank dashboard.


1. To create a dashboard component, drop a component widget () onto a dashboard.

2. Select the **Lightning Components** tab.

 **Note:** If you don't see this tab, you must enable this feature. For enablement information, see the [Setup](#) page.


3. Choose the Hello World Lightning Web Component that you created from the list.
4. To see the `title` attribute, expand the Component Attributes on the right-hand side. Type your name in the box, and then watch your code in action.



 **Note:** You can use **Preview** to run the dashboard, but many attributes update during design time, for faster testing.

Advanced Configuration - SOQL Query

Let's make your component dynamic by adding some user attributes. You can use SOQL with a filter by the current user token to grab any field from your user object. For this example, use the `FirstName` field.

1. On your dashboard, click **Edit**.
2. In the dashboard editor, click a blank space.
3. At the top-right, click the **Create Query** button.
4. Select the **Salesforce Object** tab.
5. Select the **User** object.
6. To edit the SOQL, click .
7. Paste this SOQL into the editor:

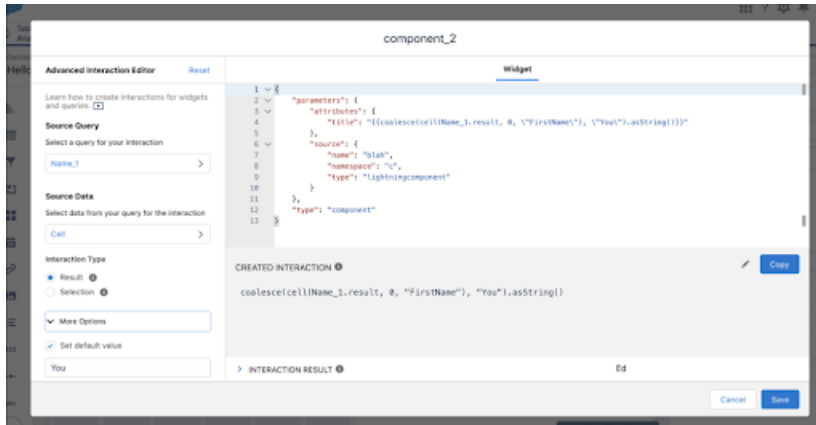
```
SELECT FirstName, COUNT(Id) FROM User WHERE Id= '!{User.Id}' GROUP BY FirstName ORDER BY FirstName ASC LIMIT 1
```

8. Name the query and click **Done**.`import chartJsPluginSubtitle from '@salesforce/resourceUrl/chartJsPluginSubtitle';`

Now you have a query that returns one row containing the current user's first name. Use the interactions UI to wire it into your step.

1. To bring up the query panel to the right, click a blank space on the dashboard.
2. Find your new SOQL query, and then from its dropdown menu, select **Properties**.
3. Click **Advanced Editor**.
4. Under **Source Query**, select your query.
5. For **Source Data**, choose `cell, 0`, and `FirstName`.
6. For **Interaction Type**, keep `Result`.

7. Optionally, add a default value in case no results are found. This example uses `You`.
8. Copy the interaction.
9. Save your work.
10. Click on your component and then click **Advanced Editor** in the Widget menu.
11. In the editor, paste the interaction as the `title` value and click **Save**.

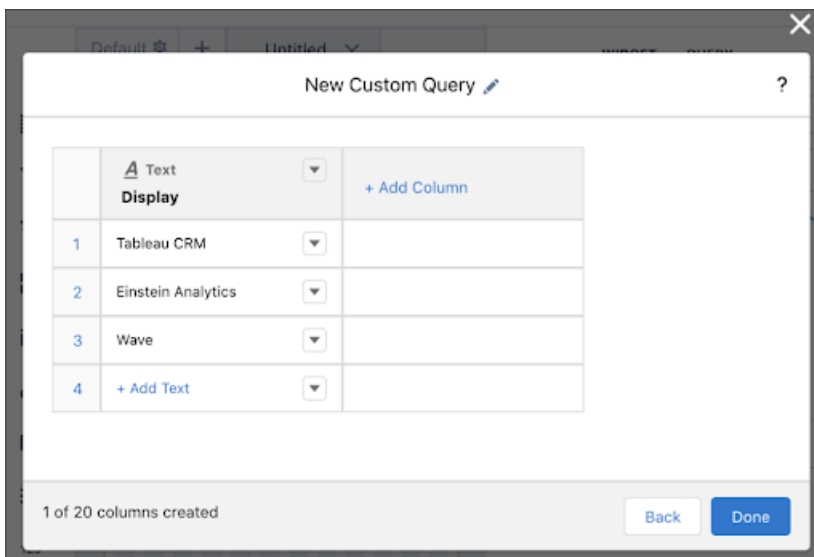


Next. Try it out. You have a dynamic value automatically set by the current user viewing your content. If it's not working, preview your results from the query panel to verify the data is coming back correct. To debug for typos, use the Advanced Editor, or `cmd+e / CTRL+e` to view the dashboard JSON.

Advanced End-User Configuration - Static Step UI


Using bindings, you can create any dynamic value from existing queries and selections. Let's make a set of discrete values that an end user can select that drives an attribute in your LWC code.

1. Drag a toggle widget onto the dashboard page.
2. To configure the widget, click the **Toggle** button.
3. At the bottom of the configuration menu, choose **Create Custom Query**.
4. Under the **Display** column, add a few options. This example uses the many names of Salesforce Analytics:

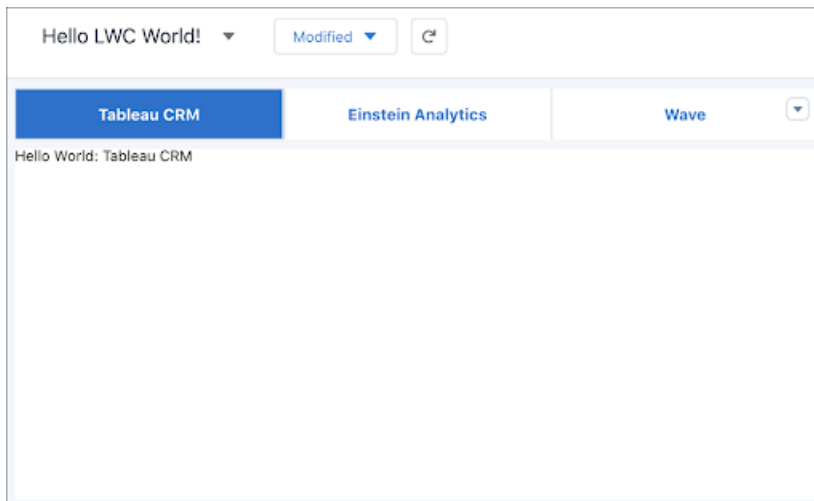


5. Click **Done**.
6. To show the **Properties** panel on the right, click your LWC Component.
7. Click **Advanced Editor**.
8. To create an interaction, choose the static query you created.
9. For **Source Data**, select `cell, 0`, and `Display`.
10. Choose a selection binding.
11. Optionally, add a default value in case no results are found. This example uses whoever the user is.
12. Copy and paste the interaction as the `title` value.
13. Save your changes.



 **Note:** Instead of setting a default value, you can go into the query properties and set the selection mode to `SingleRequired`. This setting ensures there's always a selection for the query.

And there you have it. Preview your dashboard to try it out. Select a name from the toggle and then watch your Lightning Web Component say "Hello" to that name.



Now that you have an understanding of attributes and configuration, feel free to add more to your widget. You can add style and CSS options to your component and expose them as attributes, such as `Text Size`, `Text Color`, or `Background Color`.

Query Data

Let's learn how to inject query results into your running Lightning Web Component code.

To start, modify your Hello World component to use a step by setting `hasStep` to `true` in the `js-meta.xml`.

```
<hasStep>true</hasStep>
```

Add the results wiring into your `.js` file.

```
import { LightningElement, api } from 'lwc';

export default class HelloWorld extends LightningElement {
  @api title;

  @api results;

  get stringResults() {
    return JSON.stringify(this.results)
  }
}
```

To view the query results as text, update your `.html` file.

```
<template>
  Hello World: {title}
  Results: {stringResults}
</template>
```

Deploy the changes to your component to your org. Refresh your dashboard to pick up any updates to the component code. After refreshing the dashboard, edit the dashboard and click your component. You're now prompted to select a data source. You can select a dataset to create a query or select an existing query. Make your selection and watch the query results appear in your dashboard.

Other Widgets

The Hello World example is a simplified scenario. The most common real world use cases are a custom data visualization, a selection widget, or even better, a widget that does both. With practice, you can add anything you want. To see a list, a third-party data visualization integration, a hierarchy, and more widgets, check out the example GitHub repository.

GitHub Examples

Clone the [GitHub example repository](https://github.com/forcedotcom/sfdx-analytics.git), `git clone https://github.com/forcedotcom/sfdx-analytics.git`. Open the repository in Visual Studio. The Lightning Web Component examples are in the `quick-start/main/default/lwc` directory and include:

Example	Description	Requirements
<code>breadcrumb</code>	Leaves breadcrumbs as users navigate through a dashboard.	No step or attribute requirements
<code>graph</code>	Render a graph.	Requires 2 attributes from step results: <ul style="list-style-type: none"> <code>sourceField</code> - graph edge sources (Dimension)

Example	Description	Requirements
		<ul style="list-style-type: none"> • <code>destinationField</code> - graph edge destinations (Dimension) • <code>valueField</code> - Optional numeric weight for an edge (Measure)
<code>hierarchy</code>	Render results into a hierarchical structure as a tree grid.	Requires 4 attributes from step results: <ul style="list-style-type: none"> • <code>idColumn</code> - primary key (Dimension) • <code>parentIdColumn</code> - self-reference to parent record (Dimension) • <code>labelColumn</code> - record label (Dimension) • <code>root</code> - root node (String)
<code>libsChartjs</code>	Render results using Chart.js.	Requires 2 attributes from step results: <ul style="list-style-type: none"> • <code>measureColumn</code> - segment size (Measure) • <code>labelColumn</code> - segment label (Dimension)
<code>list</code>	Render results in a list.	Requires a step, but no attributes. Set up the component with a simple step of <code>Count of Rows</code> on 1 field and it lists the field results.
<code>playButton</code>	Play recording of dashboard selection changes.	Requires a step and 1 attribute. <ul style="list-style-type: none"> • <code>delay</code> - delay between selection steps in milliseconds (Integer)
<code>selectNavigate</code>	Navigate to another dashboard page.	Requires a step and 1 attribute. <ul style="list-style-type: none"> • <code>targetPage</code> - ID of page to navigate to on selection (String)
<code>tdxGanttChart</code>	Gantt chart example with multiple attributes exposed for user input in a CRM Analytics dashboard.	Requires 4 attributes from step results, plus formatting attributes: <ul style="list-style-type: none"> • <code>fromColumn</code> - start of segment (Dimension) • <code>toColumn</code> - end of segment (Dimension) • <code>labelColumn</code> - segment label (Dimension) • <code>colorColumn</code> - segment color (Dimension) • <code>axesType</code> - the type of axes: <code>linear</code> or <code>time</code> (String) • <code>title</code> - optional title for the chart (String) • <code>titleFontSize</code> - optional title font size for the chart (Integer) • <code>subtitle</code> - optional subtitle for the chart. (String) • <code>subtitleFontSize</code> - optional subtitle font size for the chart (Integer) • <code>theme</code> - theme to use for the chart (String) • <code>backgroundColor</code> - the background color for the chart (String)

Example	Description	Requirements
wordcloud2	Render a word cloud.	<ul style="list-style-type: none"> label - the label for the chart (String)
		<ul style="list-style-type: none"> Requires 2 attributes from step results, plus formatting attributes: wordColumn - column with words to display (Dimension) measureColumn - numerical value defining word size (Measure) height - height of the canvas (Integer) width - width of the canvas (Integer) minTextSize - minimum size of the text in px (Integer)

Using the Hierarchy Component

To test and play with this component, create a custom static query in the dashboard editor.

	Text Display	Text ID	Text ParentID	+ Add Column
1	Jane Smith	1	0	
2	All Abouli	2	1	
3	Fiona Brown	3	2	
4	Alex Whitefield	4	3	
5	Matt Mendell	5	4	
6	Xi Chang	6	2	
7	Sara Lind	7	6	
8	Lee Green	8	7	
9	Root	0	+ Add Text	
10	+ Add Text	+ Add Text	+ Add Text	

3 of 20 columns created

Drop a component on the dashboard, select **Hierarchy** from the **Lightning Components** list, and use **Existing Query** to set the custom static query as the component step.

In the component properties menu, set the **Component Attributes** to these values.

- **ID Column** - *ID*
- **Parent ID Column** - *ParentID*
- **Label Column** - *Display*
- **Root Node** - *0*

Save your work and preview the dashboard to see the working hierarchy component. If you make any updates to the component attributes, refresh the dashboard in preview mode to render the changes.

Using the TDX Gantt Chart Demo Component

To test and play with this component, create a custom query in the dashboard editor. Use the Opportunities dataset to create a values table query with six columns. The six columns are **CreatedDate**, **CloseDate**, **Opportunity Name**, **Opportunity Owner**, **Amount**, and **Stage**.

#	Created Date	Close Date	Opportunity Name	Opportunity Owner	Amount	Stage
1	2021-04-21T00:00:00.000Z	2021-12-13	Opportunity for Conard5	Laura Palmer	\$32,400	Qualification
2	2020-11-05T00:00:00.000Z	2020-12-30	Opportunity for Wood8	Catherine Brown	\$397,290	Closed Lost
3	2020-04-19T00:00:00.000Z	2020-08-26	Opportunity for McDonald93	Dennis Howard	\$265,747	Closed Won
4	2020-12-18T00:00:00.000Z	2021-05-03	Opportunity for Jafferson17	Irene McClay	\$21,840	Closed Won
5	2021-06-28T00:00:00.000Z	2021-10-30	Opportunity for McLaughlin30	Eric Gutierrez	\$1,149,000	Negotiation/Review
6	2021-02-16T00:00:00.000Z	2021-09-29	Opportunity for Chandler193	Dennis Howard	\$714,840	Perception Analysis
7	2020-05-02T00:00:00.000Z	2020-07-13	Opportunity for Row134	Kelly Frazer	\$243,450	Closed Won
8	2021-06-27T00:00:00.000Z	2021-10-11	Opportunity for Barnes141	John Williams	\$363,400	Qualification
9	2021-07-14T00:00:00.000Z	2021-10-17	Opportunity for Edwards146	Johnny Green	\$4,212,140	Perception Analysis
10	2021-09-14T00:00:00.000Z	2021-09-28	Opportunity for Williams149	Evelyn Williamson	\$2,305,550	Needs Analysis
11	2021-01-04T00:00:00.000Z	2021-06-01	Opportunity for Wagner150	Bruce Kennedy	\$2,660,660	Closed Lost
12	2020-04-19T00:00:00.000Z	2020-12-08	Opportunity for Lewis158	Laura Garza	\$308,938	Closed Won
13	2021-07-23T00:00:00.000Z	2021-09-29	Opportunity for Dennis260	Bruce Kennedy	\$2,868,400	Needs Analysis

Drop a component on the dashboard, select **TDX Gantt Chart Demo** from the **Lightning Components** list, and use **Existing Query** to set the custom query as the component step.

In the component properties menu, set the **Component Attributes** to these values.

- **From Column** - *CreatedDate*
- **To Column** - *CloseDate*
- **Label Column** - *Opportunity Name*
- **Color Column** - *Stage*
- **Axes Type** - *time*
- **Title** - *Deals over Time*
- **Title Font Size** - *22*
- **Subtitle** - *Dealflow Analysis*
- **Subtitle Font Size** - *11*
- **Theme** - *Sunrise*
- **Background Color** - *#FFFFFF*

Save your work and preview the dashboard to see the working Gantt chart component. If you make any updates to the component attributes, refresh the dashboard in preview mode to render the changes.

Using the Word Cloud 2 Component

To test and play with this component, create a custom query in the dashboard editor. Use the Opportunities dataset to create a compare table query with a column for Count of Rows and a column for **Scaled Size**. Make the **Scaled Size** column a formula of $A/10$. Group the columns by **Account.Industry**.


Account Industry	Count of Rows	Scaled Size
Agriculture	30	3
Apparel	50	5
Banking	54	5.4
Biotechnology	40	4
Communications	24	2.4
Consulting	23	2.3
Education	30	3
Electronics	37	3.7
Energy	70	7
Engineering	41	4.1
Finance	23	2.3
Healthcare	21	2.1
Insurance	45	4.5

Drop a component on the dashboard, select **Word Cloud 2** from the **Lightning Components** list, and use **Existing Query** to set the custom static query as the component step.

In the component properties menu, set the **Component Attributes** to these values.

- **Words** - *Account.Industry*
- **Measure Column** - *Scaled Size*
- **Height** - *400*
- **Width** - *1000*
- **Min Text Size** - *20*

Save your work and preview the dashboard to see the working word cloud component. If you make any updates to the component attributes, refresh the dashboard in preview mode to render the changes.

 **Note:** Creating a query first isn't required. You can also create a query when the component is added to the dashboard. You have the choice to select fields from the existing dataset or to change the data source to another dataset and then select fields.

REFERENCE

Detailed reference for Lightning Web Components in Analytics dashboards.

Component Configuration

Description of attributes for the component `js-meta.xml` file.

analytics__Dashboard Target

To allow the component to be used in Analytics dashboards, add this target to the list of `targets`.

```
<targets>
  <target>analytics__Dashboard</target>
</targets>
```

To customize how the component appears in Analytics dashboards, add a `targetConfig` to `targetConfigs`.

```
<targetConfigs>
  <targetConfig target="analytics__Dashboard">
  </targetConfig>
</targetConfigs>
```

<hasStep> tag

In an `analytics__Dashboard` target config, you can choose to include the `<hasStep>true</hasStep>` attribute.

```
<targetConfigs>
  <targetConfig target="analytics__Dashboard">
    <hasStep>true</hasStep>
  </targetConfig>
</targetConfigs>
```

This attribute indicates that your component requires a dashboard step to function as expected. With this tag, the dashboard builder UI prompts the user to attach an existing step or to create a step to an instance of the component. Components with an attached step have access to step specific API hooks like `results` and `selection`.

Measure and Dimension Attribute Data Types

Attributes specified in an `analytics__Dashboard` target config are displayed in the Analytics dashboard builder UI for configuration. In addition to the common data types, this target also supports `Measure` and `Dimension` data types for components with `<hasStep>true</hasStep>`. The dashboard builder is able to choose a column of the given data type from the results of the attached step.

```
<targetConfigs>
  <targetConfig target="analytics__Dashboard">
    <hasStep>true</hasStep>
    <property name="labelColumn" type="Dimension" label="Label Column"
description="Segment label." required="true"/>
  </targetConfig>
</targetConfigs>
```

API Hooks Overview

Name	Description	Available for Use in LWC
results	An array of objects.	Requires <hasStep>true</hasStep>.
metadata	An object with three arrays: <code>groups</code> , <code>strings</code> , and <code>numbers</code> . Each array is the field name of each type from the associated query.	Requires <hasStep>true</hasStep>.
selectMode	Returns the selection mode of the associated query. Valid values are: <ul style="list-style-type: none"> • <code>single</code> • <code>multi</code> • <code>singlerequired</code> • <code>multirequired</code> • <code>none</code> 	Requires <hasStep>true</hasStep>.
selection	The current selection of the associated step.	Requires <hasStep>true</hasStep>.
setSelection	Use this function to modify the current selection on the associated step and inform the parent dashboard.	Requires <hasStep>true</hasStep>.
getState	Use this function to retrieve the entire parent dashboard state as a JSON document.	Always available.
setState	Use this function to modify the dashboard state.	Always available
refresh	Use this function to refresh the dashboard state.	Always available

API Hooks Details

Results

The results are rows returned by the query represented as an array of maps.

```
[
  {columnOne: 'one', columnTwo: 123},
  {columnOne: 'two', columnTwo: 456}
]
```

Metadata

Metadata describes the shape of the results using lists of column developer names organized into `groups`, `numbers`, and `strings`.

- `Groups` - Grouped dimensions. Typically, all grouped columns can be used together as a primary key for the row.
- `Strings` - All text fields, also known as `Strings` or `Dimensions`.
- `Numbers` - All numeric fields, also known as `Measures`.

```
{
  groups: [],
  strings: ['columnOne'],
  numbers: ['columnTwo']
}
```

selectMode

Returns the select mode specified for the associated step. Valid values are:

- `single`
- `multi`
- `singlerequired`
- `multirequired`
- `none`

```
isMultiSelect() {
  return this.selectMode.includes('multi');
}
```

selection

The current selection of the associated step as an Array of objects, with each object being one or more selected rows from `results`. Use the `setSelection` function to update the dashboard selection.

```
return new Map((this.selection ?? []).map((row) => [this.hash(row), row]));
```

setSelection

The `setSelection` function is a callback passed in that allows the component to update the attached step's selection in Analytics. In doing so, it potentially applies filters to the rest of the dashboard's contents depending on how the queries are configured.

```
this.setSelection(this.isMultiSelect() ? [...selectedRowsByHash.values(), row] : [row]);
```

getState and setState

Use `getState` to retrieve the current state of the dashboard. Use `setState` to modify the current state of the dashboard. Fetch and modify all the current selections, filters, and the currently viewed page. Use these functions to add custom logic and behavior that's not standard in the dashboard runtime.

```
if (!this.getState().pageId == this.targetPage) {
  this.priorPage = this.getState().pageId;
  this.setState({...this.getState(), pageId: this.targetPage, replaceState: true});
}
```

Learn more about selection syntax with [Filter and Selection Syntax for Embedded Dashboards](#).

refresh

Use `refresh` to rerun either all the queries on the dashboard or a particular query. To rerun a specific query, use the `config` argument with the step id of the query.

```
this.refresh({step: this.step});
```

Custom Lifecycle Methods

stateChangedCallback

Use `stateChangedCallback`, a built-in callback method, to inform the component whenever the state of the dashboard changes. Use `stateChangedCallback` to compare the current state with the new state and then rerender if necessary.

```
stateChangedCallback(prevState, newState) {  
  // perform comparison logic and set the new state to the current state to trigger a  
  // rerender of the component  
  this.currentState = newState;  
}
```

For an example of this callback in action, see the [Breadcrumb](#) javascript in the GitHub example repository.



Note: This callback method is available in v55 and later.

RELEASE NOTES

Use the Salesforce Release Notes to learn about the most recent updates and changes to Lightning Web Components in CRM Analytics Dashboards.

For a list of all current developer changes, including Lightning Web Components in CRM Analytics Dashboards, see [CRM Analytics](#) in the Salesforce Release Notes.



Note: If the Analytics Development section in the Salesforce Release Notes isn't present, there aren't any updates for that release.