The **Dashboard System** in Niagara Cloud allows users to visually represent and interact historical data, alarms, points live read data through different types of widgets. The dashboard system is highly flexible, enabling users to create and configure dashboards with various widgets, each representing different data types and visualizations. The system allows users to create customizable, interactive dashboards to display device data, points, alarms, telemetry. A dashboard is a collection of **widgets** that display various kinds of information. Widgets are individual UI components that display data in a graphical form (e.g., charts, maps, gauges, tables, etc.). Widgets are placed on the dashboard and connected to data sources (e.g., devices, sensors, or other entities).

Dashboard System architecture can be broken down into three main layers: Client, Server, and Data Sources. Each layer plays a crucial role in delivering the user experience and handling data flow.

Below is high level architecture of Dashboard System that is cloud-native.

**Widget Container (WidgetContainer.tsx)**

* Manages lifecycle, state, and layout.
* Supports drag-and-drop reordering.
* Handles resizing and dynamic loading
* Role is Handles the rendering, layout, and real-time data for a single widget.
* Handles drag, resize, and settings for its assigned widget.

Widget Manager

The role of this to Acts as a registry for widgets, enabling add, remove, update actions.

Allows dynamic widget creation, deletion, and reordering.

Handles drag, resize, and settings for its assigned widget.

* **Dashboard Service** is in charge of the overall dashboard configuration and data flow.
* **Widget Service** manages the rendering, configuration, and data updates for individual widgets.

These 2 services work together to create dynamic, and customizable dashboards for users, often integrating with databases (PostgreSQL) and caching mechanisms (Redis) to ensure fast and responsive UI interactions.

**Overview of the Dashboard System in ThingsBoard**

1. **Dashboard**: A dashboard is a collection of **widgets** that display various kinds of information (e.g., sensor readings, data visualizations, and controls) in a layout. Dashboards can be created, edited, and configured by users with appropriate permissions.
2. **Widgets**: Widgets are individual UI components that display data in a graphical form (e.g., charts, maps, gauges, tables, etc.). Widgets are placed on the dashboard and connected to data sources (e.g., devices, sensors, or other entities). Widgets can be customized to fit the user’s requirements, including layout, style, and data source.
3. **Widget Template**: A widget template is a predefined configuration of a widget, which can be reused across multiple dashboards. Templates save time by providing standard widget configurations and designs.
4. **Dashboard Configuration**: Each dashboard has a configuration that includes settings for layout, widget placement, and data sources. Users can create or modify dashboards, change widget configurations, and define interactivity between widgets (e.g., linking widgets to update when data changes).
5. **Data Source**: The data source refers to the origin of the data that a widget will display. In **ThingsBoard**, data sources can include **devices**, **timeseries data**, **attributes**, **telemetry**, and more. The data can be retrieved from different databases (e.g., **PostgreSQL**, **Cassandra**) or real-time streaming mechanisms (e.g., **WebSockets**).
6. **Real-time Data Updates**: The dashboard system is designed to support **real-time data updates**, which allows widgets to automatically update when new data is available. This is achieved through **WebSockets** or **polling** mechanisms, where the dashboard subscribes to data streams that update the widgets dynamically.

In the **ThingsBoard** architecture, the backend consists of several services that work together to deliver dashboard and widget functionality. Here’s a breakdown of how the **Dashboard Service** and **Widget Service** function:

**1. Dashboard Service (DS)**

The **Dashboard Service** is responsible for managing dashboards, which are collections of widgets and settings that users interact with. The dashboard service is tasked with:

* **Dashboard Creation**: It handles the creation of new dashboards, including the configuration and storage of widget data.
* **Dashboard Retrieval**: When a user requests a dashboard, it fetches the configuration, widgets, and any necessary data from various data sources.
* **Dashboard Updates**: When a user modifies a dashboard (e.g., adding/removing widgets, adjusting settings), it ensures that changes are persisted to the database and cached.
* **Dashboard Deletion**: It allows the deletion of dashboards when required.
* **Data Aggregation**: It may aggregate data from different sources (e.g., databases, external APIs) to render the dashboard’s data visualizations.
* **Real-time Updates**: The service handles real-time data updates for dashboards, especially in cases where widgets need to reflect live data (e.g., sensor telemetry).

**Key Components of Dashboard Service:**

* **Data Sources**: It connects to various data sources (e.g., **PostgreSQL**, **Cassandra**) for storing and retrieving data.
* **Cache Layer**: It works with a **Cache Server** (e.g., **Redis**) to store frequently accessed dashboard data, improving performance and reducing load on the database.
* **User Interface Integration**: The service supplies the necessary data to the **API Gateway**, which then serves it to the **Web Browser (UI)**.

**2. Widget Service (Widget Framework)**

The **Widget Service** in ThingsBoard is responsible for managing the individual widgets within a dashboard. Each widget represents a specific type of data visualization or control component on the dashboard (e.g., charts, gauges, maps, etc.).

**Key Responsibilities of Widget Service:**

* **Widget Configuration**: It provides configuration options for widgets, such as data sources, visualization styles, and widget-specific settings.
* **Widget Rendering**: The **Widget Renderer** component in the service is responsible for taking the widget configuration and rendering the widget in the **Web Browser (UI)**. It can generate charts, graphs, or any other custom visual components.
* **Widget Data Handling**: Widgets often rely on real-time data from various sources (e.g., sensors, databases). The widget service helps manage how data is sent to widgets for rendering. It may subscribe to data streams (e.g., via **WebSockets**) or periodically poll the server.
* **Widget Customization**: The service enables users to customize widgets by selecting data, adjusting the appearance, and adding interaction features (e.g., drilldowns, alerts).
* **Widget Updates**: When the underlying data changes (e.g., new telemetry data), the widget service ensures the widget gets updated in real time.

**Widget Components:**

* **Widget Container**: It is responsible for holding multiple widgets on the dashboard and determining how they should be displayed.
* **Widget Configurator**: The configurator allows users to customize the widgets by choosing data sources and adjusting the appearance and behavior.
* **Widget Renderer**: It handles the logic to render the widget, either statically or dynamically (for real-time updates).

**Interaction Between Dashboard and Widget Services**

* **Widget Configuration**: When creating a dashboard, the user selects or configures the widgets they want to add. This configuration is stored in the **Dashboard Service** and can include widget-specific settings (data sources, visual styles, etc.).
* **Widget Rendering**: The **Widget Service** takes the configuration and renders the widgets within the **Web Browser**. It may interact with the **API Gateway** and **Dashboard Service** to retrieve data for rendering.
* **Real-time Data**: Both the **Widget Service** and **Dashboard Service** interact with **WebSocket Connections** or polling mechanisms to push real-time updates to the UI as telemetry data changes.
* **State Management**: Widgets can store their state (e.g., user settings, dynamic data) either within the **Dashboard Service** or in external storage (like **Cache** or **Database**). This ensures that user preferences and interactions are persistent across sessions.

**Key Backend Services:**

1. **API Gateway**: Acts as the middle layer that communicates with the frontend (Web Browser) and backend services. It handles HTTP requests to access dashboards, widgets, and other services (e.g., data, user management).
2. **Security Service**: Manages authentication and authorization, ensuring only authorized users can access and modify dashboards and widgets.
3. **Cache Server (Redis)**: Stores frequently used data to reduce load on the database and improve performance.
4. **PostgreSQL & Cassandra**: Store persistent data, including dashboards, widgets, and user settings. PostgreSQL is used for structured data, while Cassandra may be used for time-series or large-scale telemetry data.

**Backend Flow Example:**

1. **User Requests a Dashboard**:
   * The **API Gateway** receives the request and checks if the dashboard data is cached.
   * If the data is cached, it’s returned directly to the **UI**.
   * If the data is not cached, the **API Gateway** fetches the dashboard data from the **Dashboard Service**.
   * The **Dashboard Service** queries the **PostgreSQL** and **Cassandra** databases for widget configurations and any historical data needed.
   * The data is returned to the **API Gateway**, cached, and sent to the **UI**.
2. **User Modifies a Widget**:
   * The **UI** sends an API request to update a widget (e.g., changing data source or appearance).
   * The **API Gateway** forwards this to the **Dashboard Service**.
   * The **Dashboard Service** saves the updated configuration in the **Cache** and databases (PostgreSQL, Cassandra).
   * The **Widget Renderer** is updated accordingly, and the changes are reflected in real-time on the UI.

**Summary**

* The **Dashboard Service** is in charge of the overall dashboard configuration and data flow, whereas the **Widget Service** manages the rendering, configuration, and data updates for individual widgets.
* These services work together to create dynamic, real-time, and customizable dashboards for users, often integrating with databases (PostgreSQL, Cassandra) and caching mechanisms (Redis) to ensure fast and responsive UI interactions.

Would you like more detail on any specific service or component?