Real Time Dashboard - Design

Primary Owner ots-dse-hw-team (LDAP)

Last modified a few seconds ago by traykeli.

Overview

The Realtime Dashboard (RTD) displays data gathered from Deadbolt Lockers on a Warehouse ID (WHID) level . There are four main tables that help leaders manage their team's activity:

- Total Device Count by Locker
- Device Guardrail
- Live Device Status
- Historical Device Activity

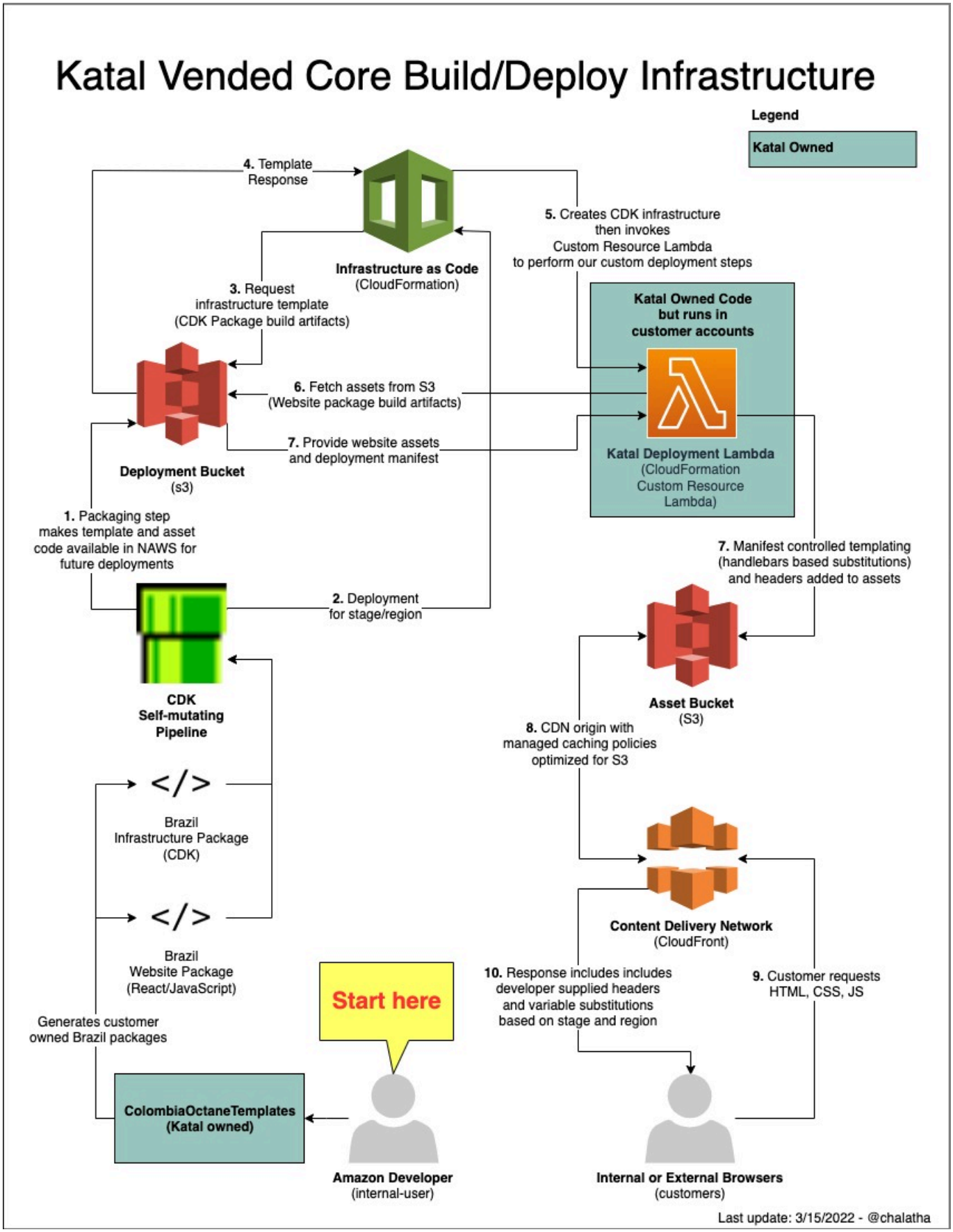
The purpose of this dashboard is to get insights in realtime on the devices stored in Deadbolt Lockers. There is also a link for convenience to the Harmony Dashboard if further analysis is needed outside of the capabilities of the RTD.

Software Architecture

The Provisioning Website architecture is based off of Katal Midway React app. Katal is an internal Amazon framework to streamline frontend development. More information on Katal can be found here. The architecture comes with the following code packages straight OOTB:

- React Application: here
- CDK of the CICD hosting the React Application: here

An overview of the Katal Architecture OOTB:



Frontend Design

- **Styling:** RTD leverages the Katal React Components library for styling

- **State:** State management of the frontend is maintained via React Context
- **Authentication:** Site Authentication & API Authentication is validated through Midway

Styling

For each of the tables RTD leverages KatDataTables and the overall theme of the application is following the color guide found here. In terms of keeping the development DRY, RTD also leverages the concept reusable components and the styling of said components are done via styled-components .

Reusable Styled Component example:

```
interface BlueButtonProps {
  buttonFunction: Function;
  buttonText: string;
  disabled?: boolean;
}

export const BlueButton = (props: BlueButtonProps) => {
  return (
    <BlueButtonStyled
      onClick={() => props.buttonFunction()}
      disabled={props.disabled}
    >
      {props.buttonText}
    </BlueButtonStyled>
  );
};

const BlueButtonStyled = styled.button`
  width: 325px;
  margin-top: 2%;
  height: 50px;
  border: 0pt;
  border-radius: 3pt;
  font-size: 1.25em;
  color: white;
  background-color: #008296;
  cursor: pointer;
  opacity: ${(props) => (props.disabled ? 0.5 : 1)};
`;
```

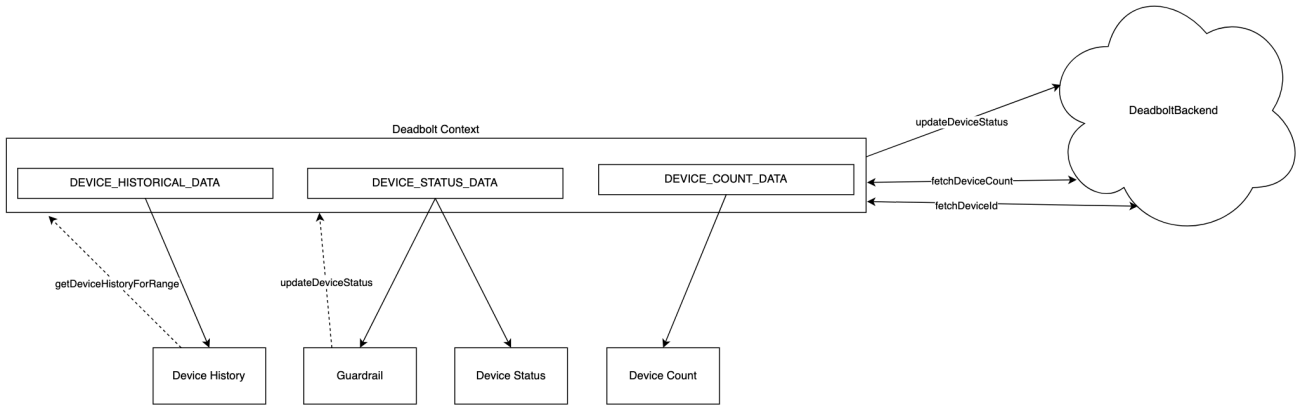
Implementation example:

```
export const MyParentComponent = () => (
  <div>
    <BlueButton buttonFunction={onClickFunction} buttonText={"My Button"} />
  </div>
);
```

RTD's frontend code base uses a combination of Styled-Components along with Katal's React Components to create reusable components which adopts a "build once, use many" mindset.

State

Using React's Context as the source of truth for the application instance running on the browser allows for a pub/sub relationship across the different tables.



Each of the tables in the UI receives updates from the Context object. The context is the only one that makes the API calls and ultimately updates the state. What makes this application "realtime" is the ability for the context to make an API call every 5 seconds for the device count (per locker) and the device status.

Authentication

Since midway comes OOTB with Katal, we leveraged the jwt returned and ensured that the Deadbolt Backend gave the proper access to the bindle associated with the RTD.

High level step by step process how on this looks:

1. User logs in via midway
2. Take the JWT returned and store that in Cookies (view code here for more details)
3. Attach that JWT as an `Authorization: Bearer ${JWT}` in each request to our backend
4. Since the JWT has a expiration time of 15 minutes, RTD refreshes the page before the expiration goes out to prevent creating API calls with bad credentials.

Also, any user which is not apart of this bindle will not have access to the site.

Table UI & Data Sources

The following tables:

- Total Device Count by Locker
- Device Guardrail
- Live Device Status
- Historical Device Activity

Are populated from Deadbolt backend, this section goes into detail regarding the data for each table and what each table looks like. Each table leverages the KatDataTable

Total Device Count by Locker

Source: Device Count Row (Dynamo DB PK: `DEVICECOUNTID#<UUID>`)

Frequency: RTD polls the backend every 5 seconds, the backend data itself is updated from the Lockers on a 5 minute cadence

Total Device Count by Locker

Count Table

Locker ID	Available Device Count	Damaged Device Count	Total Device Count	Date Reported	Time Reported
dtn6-1	11	2	13	3/15/2024	1:29:26 PM
dtn6-2	15	4	19	3/15/2024	1:31:12 PM
dtn6-3	25	6	31	3/15/2024	1:29:53 PM
dtn6-4	25	1	26	3/15/2024	1:30:44 PM

Device Guardrail

Source: Device ID Row (Dynamo DB PK: `DEVICEID#<deviceId>`)

Frequency: Every 5 seconds, the backend data itself is updated after each user activity.

Device Guardrail

Guardrail Table

User Alias Filter

begin typing user alias

Guardrail Count: 36

User Alias	Devices Taken	Date Reported	Time Reported	Override
kasiatal	dbaa1921,dbaa1985,dbaa2161,dbaa1993,dbaa1933,dbaa2112,dbaa2190	3/15/2024	1:09:32 PM	Override
jameskuy	dbaa2125	3/15/2024	1:04:18 PM	Override
gsemaje	dbaa2172	3/15/2024	1:01:41 PM	Override
jasonaki	dbaa1995	3/15/2024	1:01:17 PM	Override

Clicking `Override` set the device to a status of `LOST`

Live Device Status

Source: Device ID Row (Dynamo DB PK: `DEVICEID#<deviceId>`)

Frequency: Every 5 seconds, the backend data itself is updated after each user activity or whenever the Device ID row is updated.

Live Device Status

Live Device Status Table

Status Filter

begin typing status

Device ID Filter

begin typing device ID

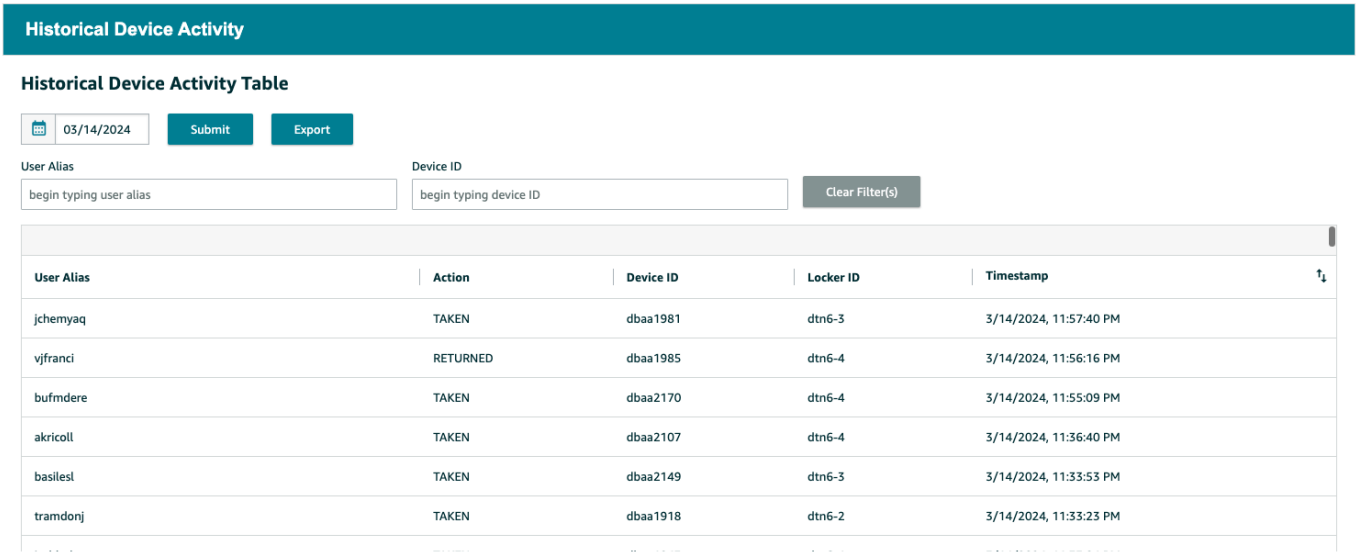
Clear Filter(s)

Device ID	User Alias	Status	Date	Time	Locker
dbaa1933	kasiatal	TAKEN	3/15/2024	1:09:32 PM	dtn6-2
dbaa1921	kasiatal	TAKEN	3/15/2024	1:08:40 PM	dtn6-1
dbaa1985	kasiatal	TAKEN	3/15/2024	1:08:40 PM	dtn6-1
dbaa2161	kasiatal	TAKEN	3/15/2024	1:08:40 PM	dtn6-1
dbaa1993	kasiatal	TAKEN	3/15/2024	1:08:40 PM	dtn6-1

Historical Device Activity

Source: Historical Device ID Row (Dynamo DB PK: `DEVICEID#<deviceId>`)

Frequency: On date select (no polling)



Accessing the site

1. Must be apart of this bindle
2. Navigate to https://dashboard.deadbolt.gsf.a2z.com/#/?whid=dtn6
    1. Note the `dtn6` can be swapped out for any supported WHID. Essentially the whid query string parameter is the trigger for where the dashboard knows to populate the data
3. Login with your midway credentials (if needed)
4. Select a table you wish to view

Issues & Roadmap

There are known issues and plans in store the RTD

Issues

- Selecting clear filter on some tables does not actually remove the text from the filter itself, but it does clear the table data
- High API usage from polling
- No notification that a WHID is not selected if no query string parameter is provided

Roadmap

- Subscription based data updates instead of a 5 second polling
    - Deadbolts backend is created via AppSync and there is support of subscriptions natively, this would allow RTD to be updated immediately instead of running a poll
- Adding Filter and sorts to all columns of tables (refactor component logic to be reusable and easier to refactor

Tags: