Datto Disaster Tracking

Design/Architecture Document

Masters of Disaster
Nsama Chipalo, Brandon Cole, Aaron Damrau, Jhossue Jimenez, Jacob Peterson

Last Updated
April 9th, 2015

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Masters of Disaster	**10/6**	**Creation of document**	**1.0**
Masters of Disaster	**10/13**	**Expansion on API Design**	**1.1**
Masters of Disaster	**11/3**	**Added Design of Mock Systems**	**1.2**
Masters of Disaster	**11/17**	**Updated DB tables** **Updated API Design**	**1.3**
Masters of Disaster	**4/9**	**Update to high-level and mock stack diagrams**	**1.4**

Purpose

This document provides a comprehensive architectural overview of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system.

High Level Architecture Diagram

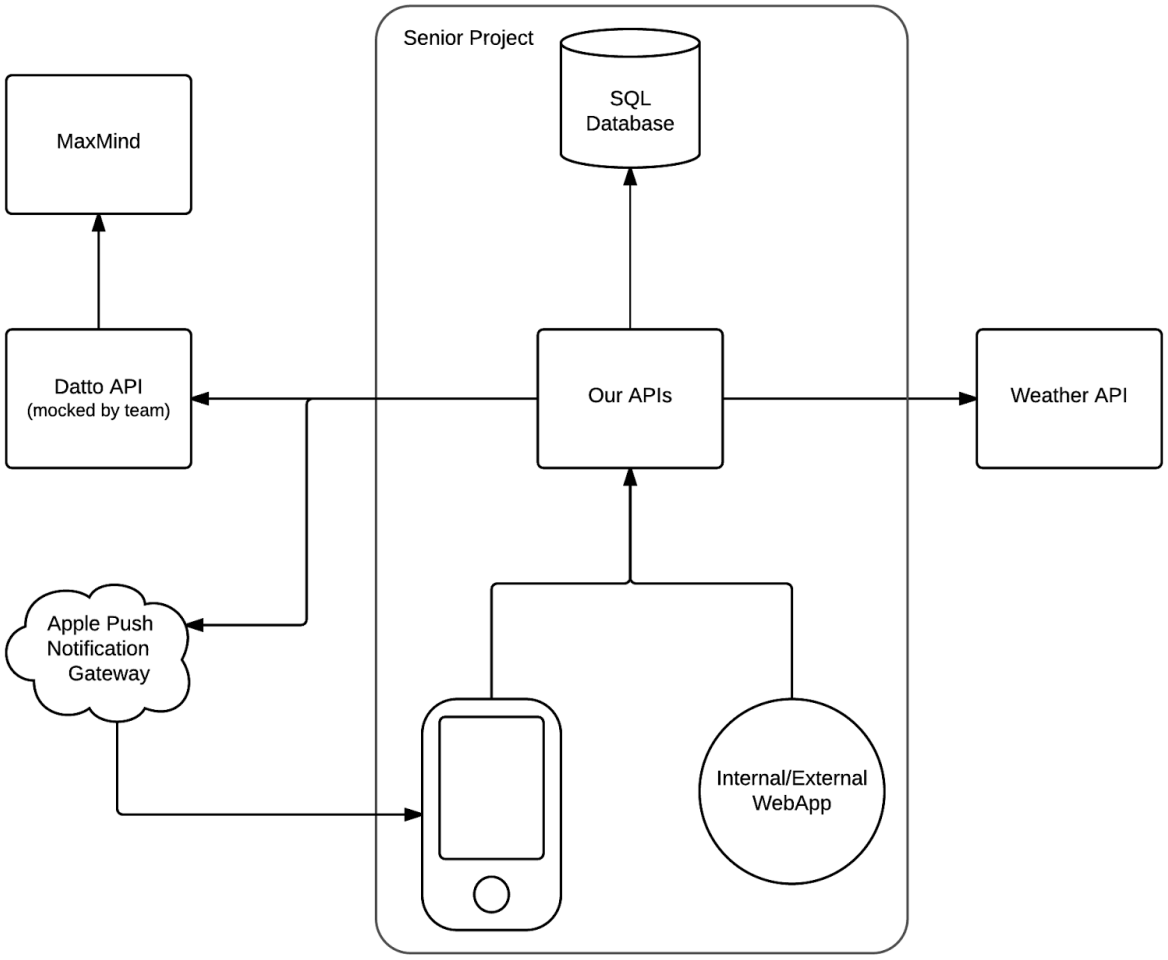

Diagram 1

External:

- MaxMind - provides geolocation information based on IP addresses to Datto.
- Datto API - provides APIs for our project to get a list of customers and provides login functionality.
- Weather API - provides weather information to be used for risk calculation.
- Apple Push Notification Gateway - responsible for sending push notifications to the corresponding MSP owned Apple devices.

Internal:

- Mobile Device - used by MSPs, shows a list of their customers. Notifies the MSP when one or more of their devices are above a risk threshold.
- Internal/External Web App - Used by Datto and MSPs, renders a map view of devices.

- Our APIs - Used by the Web App and Mobile Devices to render views and lists. Responsible for all logic that occurs (calculating risk levels) and coordinating communications with external APIs.
- SQL Database - Used as a cache that is updated periodically with new weather and device information.

Architecture of Mock Systems

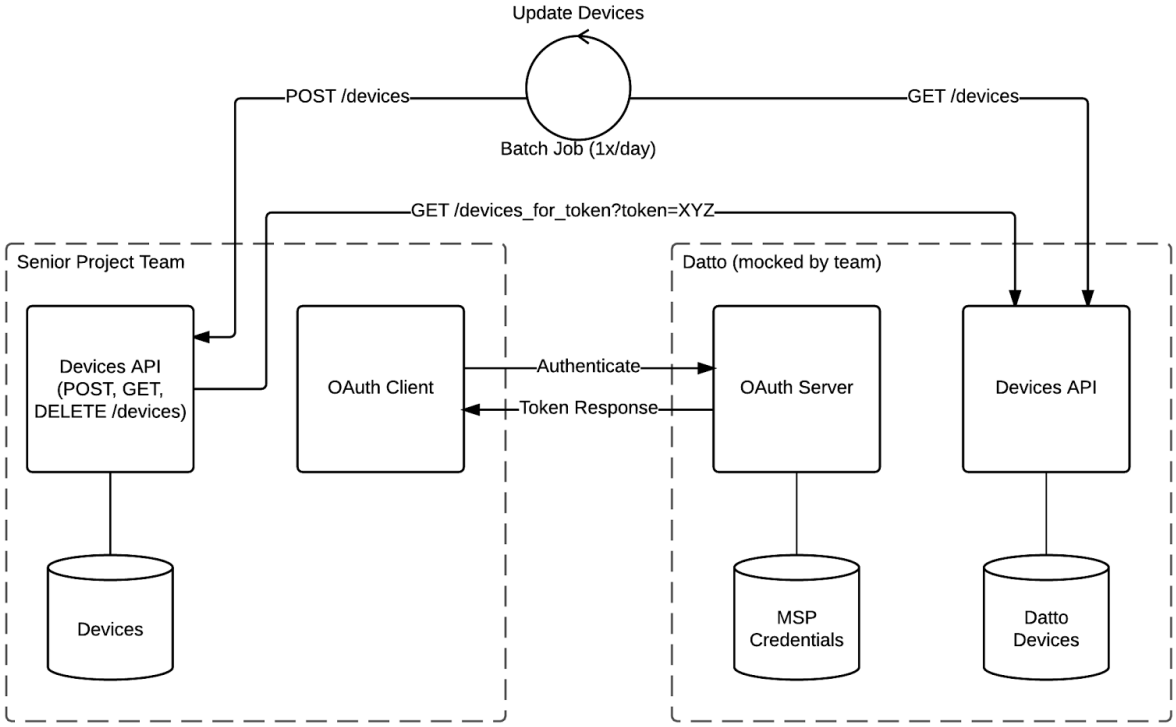

Diagram 2

Due to constraints on the resources Datto currently has available, we have decided to implement several of the systems they will provide for production as mock systems. This will allow us to be better prepared come integration time with Datto's systems. The OAuth Server, Devices API, MSP Credentials database, and Datto Devices database will all be created on our end to mimic the Datto implementations as closely as possible.

OAuth Server: This module does not currently exist in Datto's system. The team will implement what is expected to exist once this product is integrated with Datto. The OAuth Server is responsible for authenticating login credentials against the existing MSP database.

Devices API: This module does not currently exist in Datto's system. The API provides a way to receive Datto's devices in a RESTful way. This allows the Batch Job to receive the devices from Datto, and update the Devices database that exists in the Disaster Tracker project, keeping the databases in sync with each other. As a note, this API may not exist forever, if Datto decides to implement an observer implementation or other design to resolve the issue of keeping multiple databases in sync.

MSP Credentials: This module currently exists in Datto's system. We are not able to hit it directly, so we will be mocking it to use in combination with login through the OAuth Server.

Datto Devices: The module currently exists on Datto's end. We are not able to access the official Datto devices, so they must be mocked on our end in our own server. These devices will be fed into the Devices API to obtain their geographic location.

Database

The following diagram details the database schema for our system.

- Devices - The Devices table will act as a cache that stores device information that will be used to populate the risk map. Everyday at 12:00am EST, a job will run that will update the list of devices. Every 10 minutes, another job will run that will update the Risk rating for each device, based on current and upcoming weather patterns around that device's location. This 10 minute interval will coincide with updates the Weather API receives. Notifications will be sent at this time to MSP's whose device's are at risk. The status of sent notifications will also be stored to ensure we do not send multiple notifications for the same device disaster event.
- Cities - At 12:00am EST when the list of devices is being updated each device will also be linked to the Cities table. This will allow us to display relevant location information for devices to users without having to make repeated calls to get the city and state via coordinates. This will cut down on the amount of calls we must make, increasing the performance of the system.
- MSPToAppleDevice - The MSPToAppleDevice table will store user device tokens (apple_id) that will allow us to send out targeted push notifications to the appropriate MSP when a device becomes at risk.

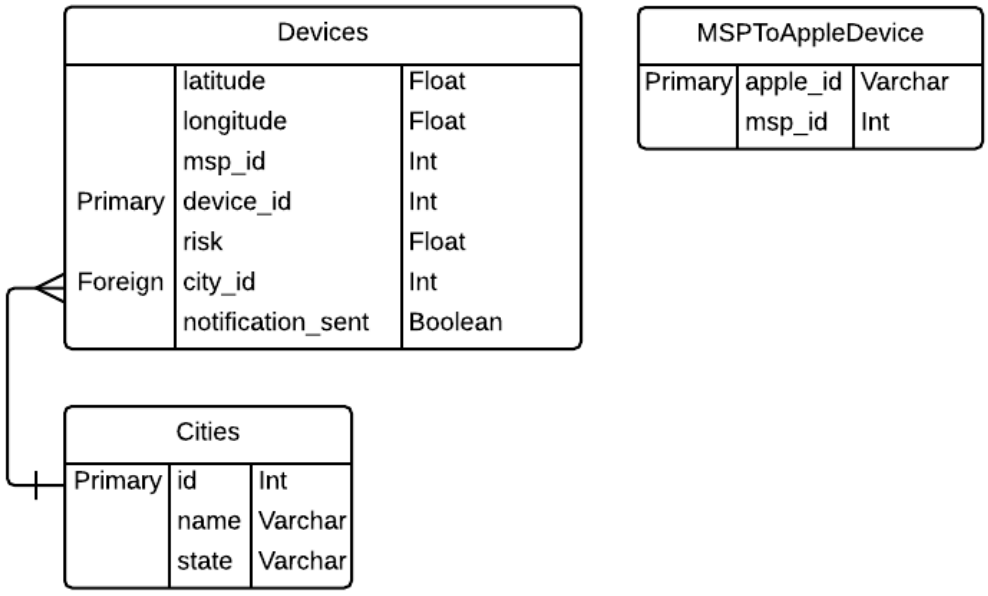

Diagram 3

Use Case Diagram

Basic overview of system functions and actors interacting with the Datto Disaster Tracker.

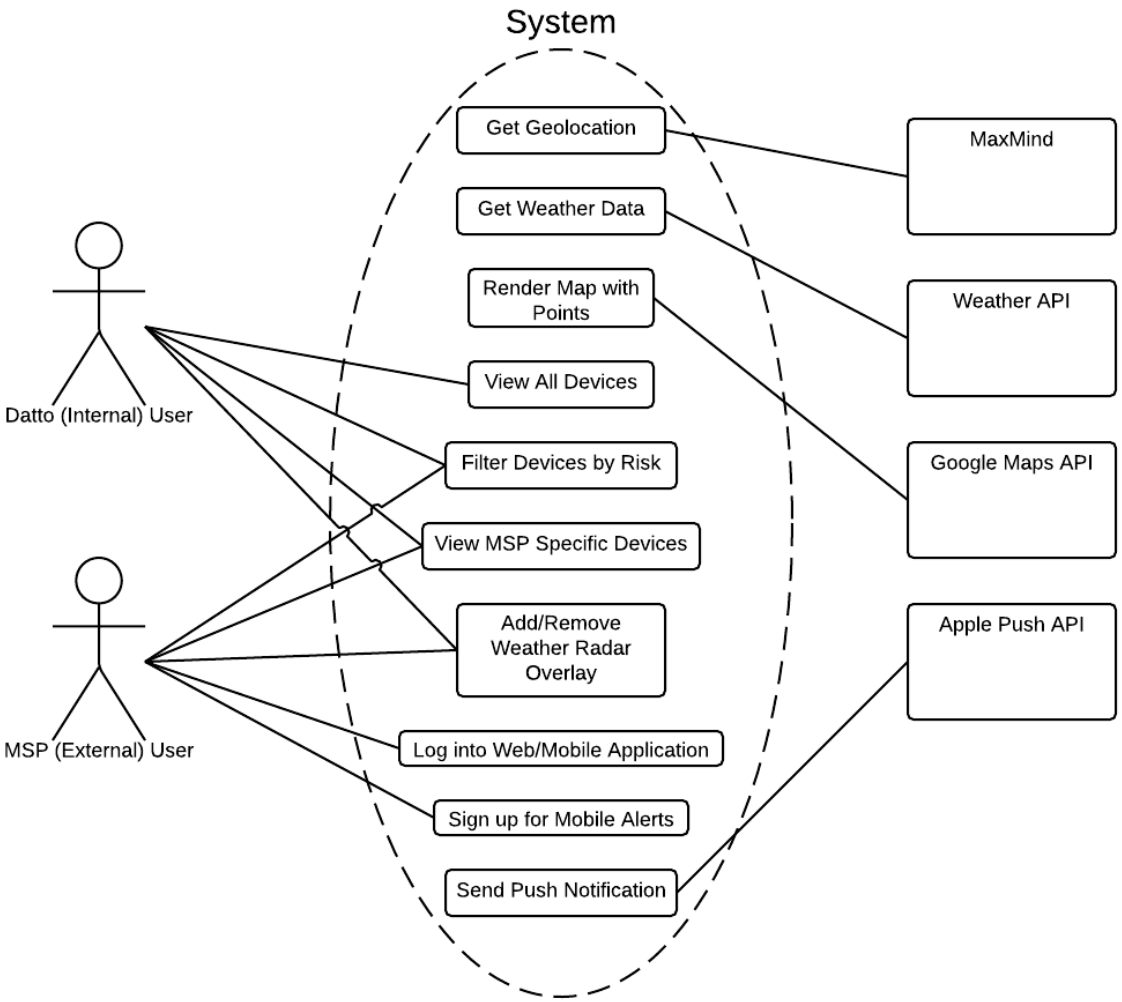

Diagram 4

Get Geolocation: Device information from Datto needs to be sent to the MaxMind API to obtain geo-location information. The points will then be stored within the system database after being instantiated.

Get Weather Data: Every ten minutes, the system will retrieve weather information using our weather API. Weather information is used to calculate the risk of devices in disaster areas.

Render Map with Points: Using the Google Maps API, we generate a map of the entire planet, populating it with the various points we received from Datto which represent devices and weather patterns from the weather API.

View All Devices: Filter the map so that all Datto devices are present. This is a Datto user interaction only because MSPs will only be provided with their devices.
Filter Devices by Risk: Changes the way the devices are listed on the side bar. All devices will be ordered from most at risk to least at risk.
View MSP Specific Devices: An interaction shared between internal and external users of the Datto Disaster Tracker system. Devices from a specific MSP will be displayed on the map instead of all devices. In the case of the external user, this is the only way to view devices on the rendered map.
Add and Remove Weather Radar Overlay: The user of the web application will have the ability to add and remove a weather radar overlay from the device dot map.
Log into Web/Mobile Application: An external user will log into the system using their Datto MSP credentials.
Sign up for mobile alerts: When logged into the mobile application, an external MSP user can sign up for push notifications to alert them on the status of any devices that are at high risk.
Send Push Notification: If the mobile external user signs up for mobile alerts and one or more of their appliances is in an at risk area, the apple push notification API will send them an alert to notify them that their devices are at risk.

API Design

API Sequence Diagrams

Note: The sequence described in Diagram 5 is the suggested approach. However, due to resource constraints, Datto has indicated that they will not be able to implement this observer approach. This section will remain in this document to describe the ideal approach, however, the description of mock systems above represents the actual implementation at this time.

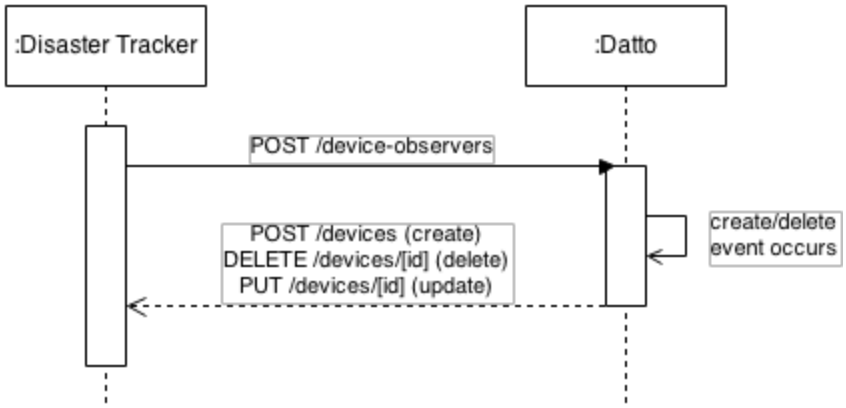

Diagram 5

Sequence Diagram 5 illustrates the process related to a Datto device being registered, modified, or unregistered from Datto's system. When such an event occurs, "device observers" must be notified of the change. In specific, the disaster tracker is a device observer that is observing the subject, Datto. After registering with Datto as a device observer (by sending a POST request to /device-observers; details of this implementation are outside of the scope of disaster tracker and are described in a separate document), Datto is responsible for notifying disaster tracker (and any other observers) of a device-related event. The notification should be achieved by calling the appropriate HTTP POST, DELETE, or PUT operation on the resource endpoint provided during observer registration (for disaster tracker, this endpoint is /devices).

API Specification

- **GET /devices**
  - Output Format: HTTP 200 (JSON)

```
{
      device_id: Integer,
      msp_id: Integer,
      latitude: Float,
      longitude: Float,
      risk: Float,
      city: {
            id: Integer,
            name: String,
            state: String
      }
}
```

- **POST /devices**
  - Description: Internal Use Only. Used by Datto to create a new device.
  - Authenticated Users: Datto Intranet
  - Input Format: (application/x-www-form-urlencoded)

    POST parameters:
    - `device_id: Integer`
    - `msp_id: Integer`
    - `latitude: Float`
    - `longitude: Float`
    - `city: String`
    - `state: String`

    Output Format: HTTP 201 (Created)

```
{
      id: Integer,
      uri: String
}
```

    Error Output Format: HTTP 400 (Bad Request)

```
{
      error: String
}
```

    Example error: "Device with device_id '123' already exists."

- **DELETE /devices/:id**
    - Description: Internal Use Only. Used by Datto to delete a device.
    - Authenticated Users: Datto Intranet
    - Input Format: Empty (HTTP Delete)
    - Output Format: HTTP 204 (No Response) - empty body

To Be Defined: Login API