CS-310 Scalable Software Architectures

Lecture 12: Push Notifications

Steve Tarzia

Last Time: Architecture Example

- Showed National Gun Violence Memorial architecture design case study.
- It's another article publishing system, so arch is like Wikipedia.
  - Caching and load balancers on frontend,
  - Stateless app,
  - SQL DB with read-replicas.
  - S3 file store was used for large media files (photos).
- Like Wikipedia, the design scales.

Limitations of Client-Server Architectures

- So far, everything we have talked about is a Client-Server architecture.
- Client (web browser, smartphone app, desktop app) makes requests, and the Server gives responses.
- The client starts all interactions. For example:
  - User clicks web link or app button
  - Javascript running on browser makes a REST request.
- In what situation would a **server** should start an interaction?
  - Deliver a WhatsApp message to a user.
  - Notify an Uber customer that their driver has arrived.
  - Notify an Ebay user that they were outbid.
  - Notify an Ebay user that they won an auction.

Caused by another user's action.

Email is a simple way to send notifications

- Many services notify users by sending an email.
- To send an email just connect to an SMTP server. SMTP services are offered by every cloud provider and other 3rd parties.
- SMS messages can also be programmatically sent by connecting to a service like Twilio or SNS.

- Email/SMS notifications suffice for many apps, but they're limited.
- These can include links to your app, but cannot directly interact with your app.
- What we really want is some way to send a request to a client app.

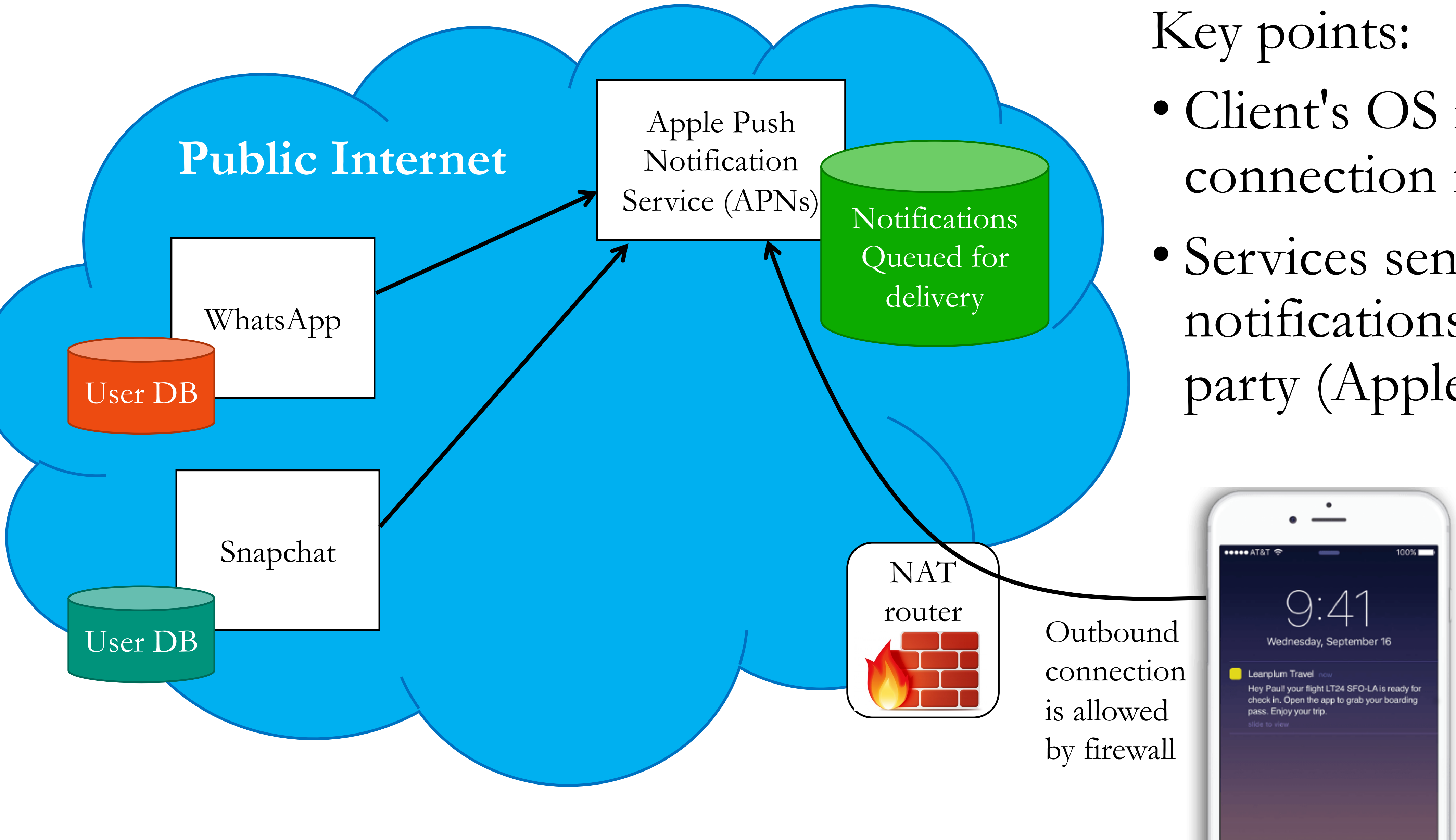
The Internet is not really a network of peers

- **Client** cannot implement a REST API beca
it is not easily reachable. Why?

  - IP addresses change when devices move.
  - IP addresses are usually *private* (NATed).
  - Device or network may have a firewall.
  - Client does not control a DNS server.
  - May be powered off, or out of radio contact.
  - App may not always be running.
- So, most services rely on clients initiating all requests themselves, sent to always-listening services with well-known hostnames.
  - Server actions are *synchronous* with client.
  - But this is not always sufficient!
- *Push Notifications:* hacks to send msgs to clients.

Motorola DynaTAC ~1984

Solution: Push Notification Service

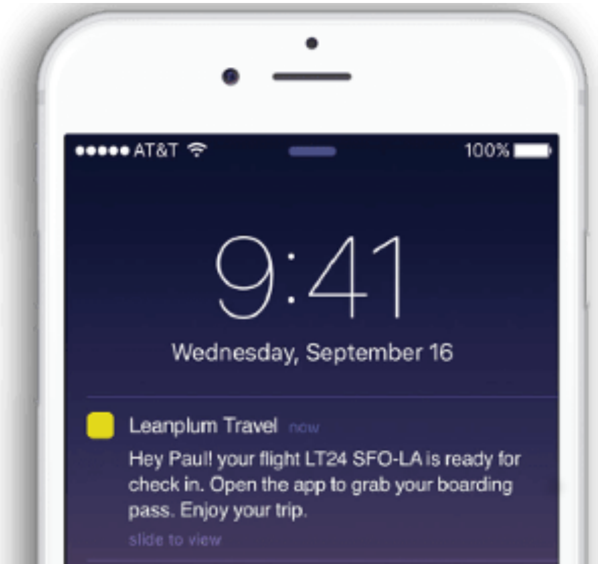


Key points:

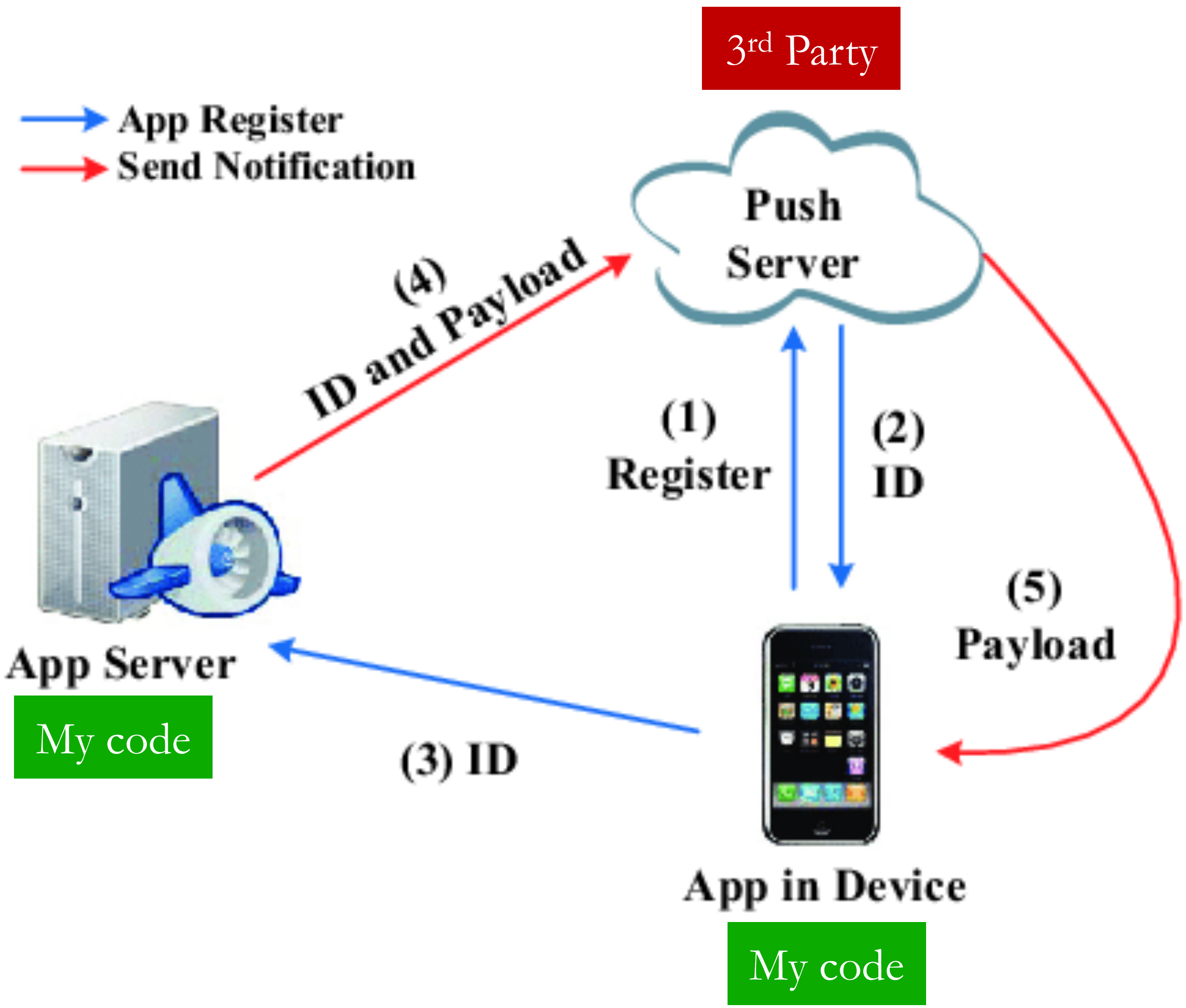
- Client's OS makes one connection for all apps.
- Services send notifications through 3rd party (Apple or Google).

Smartphone Push Notifications

- *Location registration* is used for iOS and Android push notifications.
  - Apple & Google run a **push notification service** (**PNS**) for apps on their OS.
    - *Called:* Apple Push Notifications (APN) & Google Cloud Messaging (GCM)
- Whenever phone gets a new IP address, the OS opens a long-lived connection to the PNS. PNS stores: ⟨user id, IP address⟩
- Smartphone apps like WhatsApp or Snapchat cannot contact user's phone directly; send user notifications to the PNS: ⟨user id, message⟩
  - The PNS relays the message to the user's current IP address
  - OS can show notification even if app is not running.
- On iOS, to protect users' privacy, different apps have different user ids (called device tokens).
- *If you took CS-340*: How to deal with NAT?
  - OS sends keepalive msgs. Just one port is needed for all apps.



Device Registration



Every times device connects to the network, OS creates a long-lived connection to the PNS.

1. App registers for notifications.
2. PNS returns a unique push ID.
3. App sends push ID to its backend service. Backend service stores the user's push ID in a database for later use.

Much time passes …

The backend app finally has a notification for the user!

4. Backend gets the push ID for that user from its database. Sends notification request to PNS.
5. PNS uses push ID to identify the long-lived connection to the client. It relays the notification.

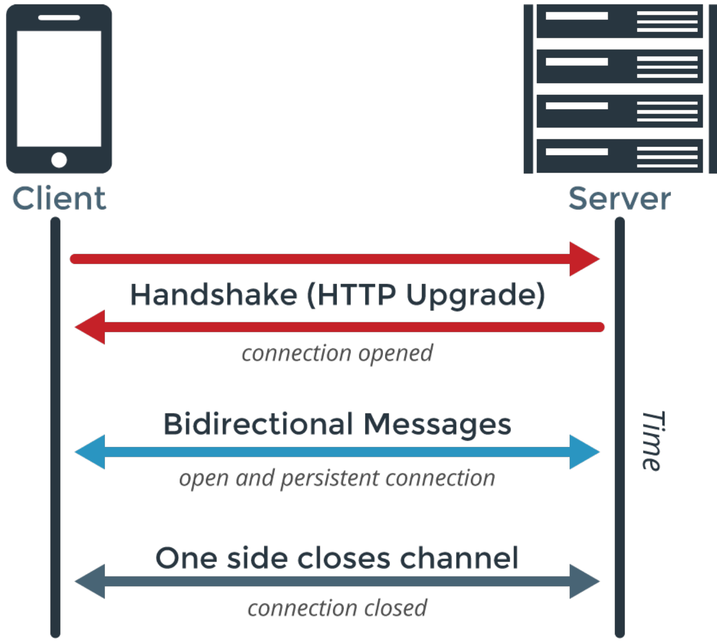
Web Browser Push Notifications

- Web browsers were designed to **pull** data from servers.
  - Server implements REST api, browser makes REST request to fetch data.
- Modern applications also desire **push**ed updates from the service.

  Eg., there is a new message for you, an edit occurred on a shared document, …
  - Client can make repeated requests for new data (**polling**), but this is a poor solution. Requires a tradeoff between latency and network overhead.
- **Websockets** are the preferred modern solution.
- **Long-polling** was the solution prior to websockets.
  - Both present some architectural challenges (similar to smartphone PNS).

Websockets

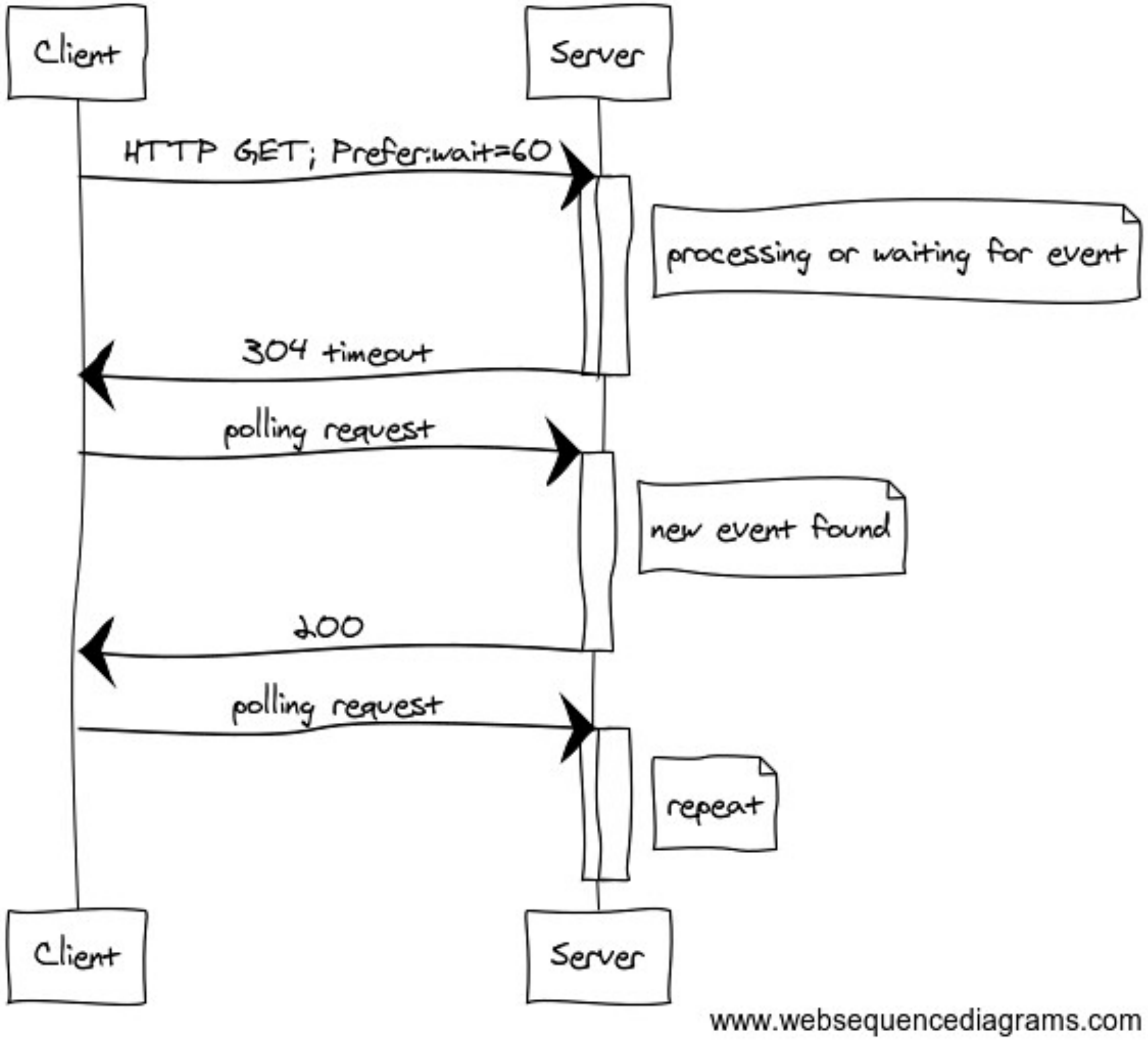
- A Websocket is a **long-lived**, **bi-directional** network connection.
  - It's similar to a TCP socket, but it's available to Javascript code in a browser.
- JS app creates a websocket connection to server.
  - Client can send API requests through the websocket.
  - Responses comes back through the websocket.
- The connection remains open!
- Server can send messages **at any time**, independent of client requests.



Old-style solution – HTTP long-polling / Comet

- Client sends an HTTP request.
- Server waits… sends a response only when new data is ready. If no data is available within 60 seconds, then send an empty response.
- Client then makes another long-polling request (infinite loop).

- **Client instantiates** the request.
- Server controls when the **response** is sent.
- Server always has one outstanding request from the client available to send data.
- *Cons:* Periodic requests every 60 seconds are wasteful. Periodic gap in service.



Comparison

HTTP	WebSocket
Duplex	
Half	Full
Messaging Pattern	
Request-reponse	Bi-directional
Service Push	
Not natively supported. Client polling or streaming download techniques used.	Core feature
Overhead	
Moderate overhead per request/connection.	Moderate overhead to establish & maintain the connection, then minimal overhead per message.
Intermediary/Edge Caching	
Core feature	Not possible
Supported Clients	
Broad support	Modern languages & clients

https://blogs.windows.com/windowsdeveloper/2016/03/14/when-to-use-a-http-call-instead-of-a-websocket-or-http-2-0/

Architectural challenges

- Whether using APN, GCM, Websockets, or HTTP long-polling, the challenge is finding the **one** long-lived connection to the client.
- A network socket (connection) is tied to **one IP address**.
- Notifications originating from anywhere in the large, distributed system must be somehow directed to the **one** appropriate notification server instance that the client is connected to.
- To solve this problem, notifications are often a separate microservice.

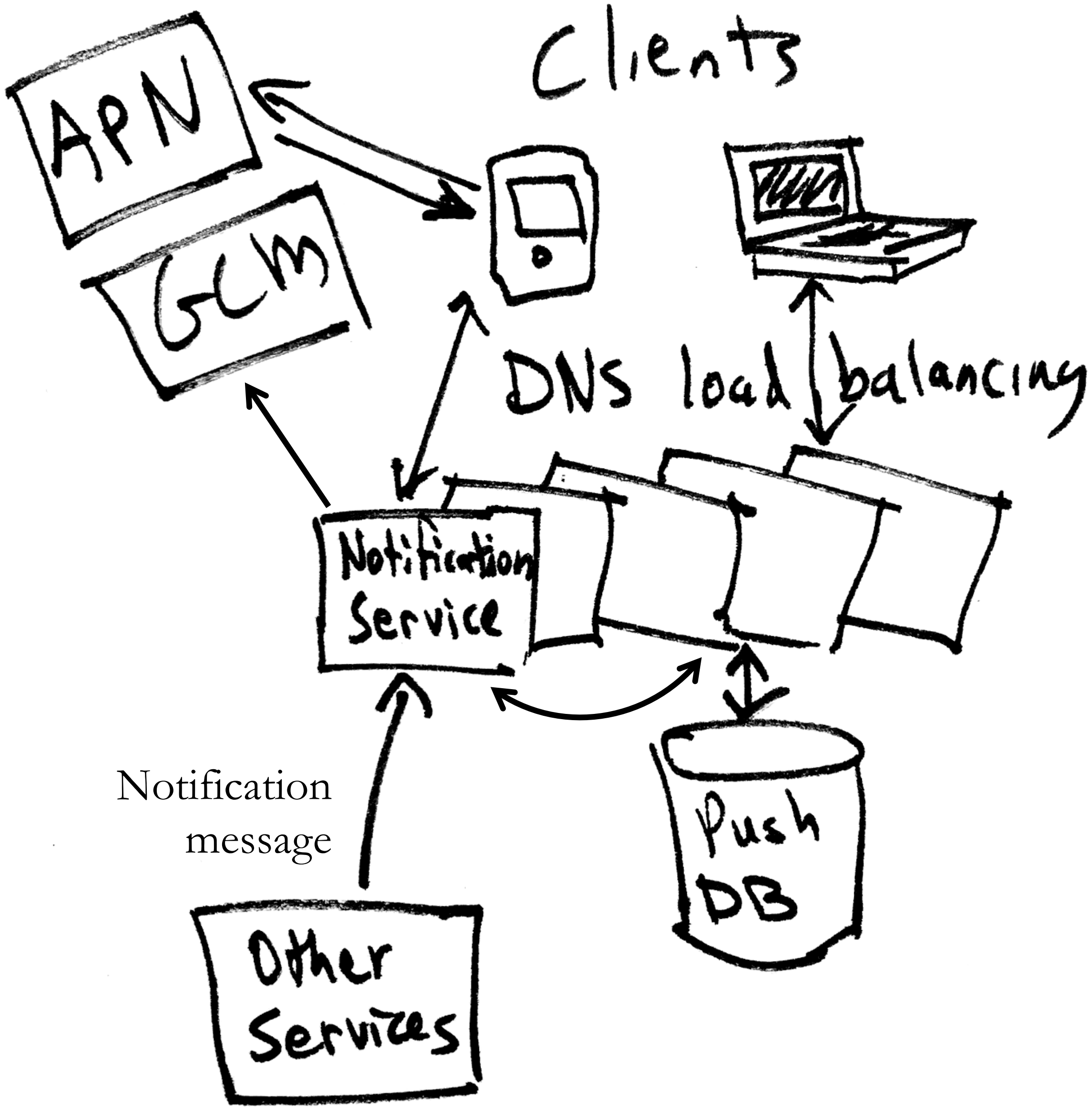
Notification Service API

Clients connect themselves in two ways:

- Opening a websocket.
- Making an API request providing a push ID usable on APN or GCM.
- In both of the cases above, the user's location is stored in a database.

Other microservices send a notification through an api call:

- POST /notification/[user_id] + *JSON notification body*
- Implementation looks up the location of the connection and relays the message.



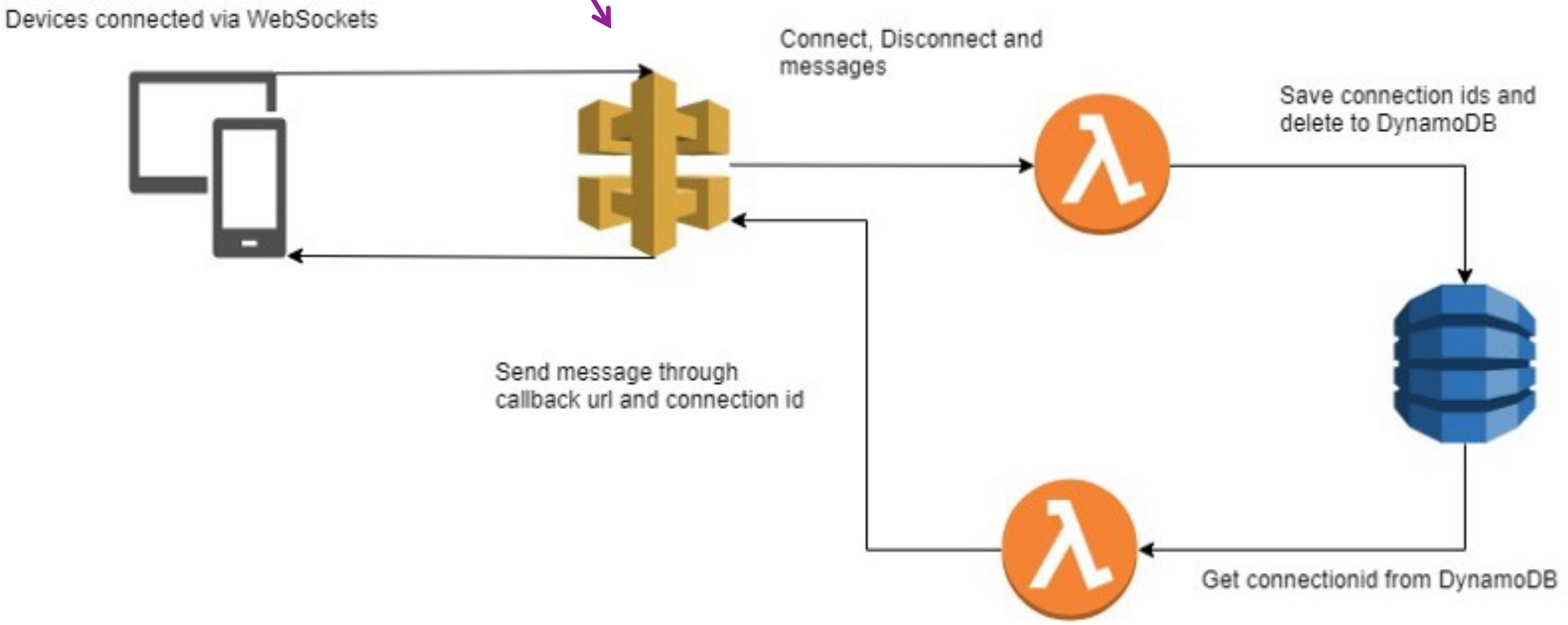
Notification Service Database Example

User	Type	Address
Alice	Android/GCM	device_id:3902390823
Steve	iOS/APNs	device_id:498092390
Steve	Websocket	5.29.193.4 : 129.29.3.2 : 29392
Steve	Websocket	5.29.193.1 : 129.29.3.2 : 9002

(notification server address : client address : client port)

- Steve will receive notifications on three different devices.
- He's running the app in two different browser tabs, and each tab is connected to a different instance of the notification service.

AWS API Gateway with websockets



Details at: https://hackernoon.com/websockets-api-gateway-9d4aca493d39

- Clients have long-lived websocket connections to gateway.
- Requests are handled by Serverless Functions (Lambdas). When connection is established, save connection id. Later use connection id to push data to clients.

Recap

- Traditional web/app design uses a **client-server** model, but sometimes we want to **push** data to client instead of client always **pulling**.
- Asynchronously sending data to clients can be a challenge.
- Mobile OSes have special **push notification services**.
  - Allows a single connection to be shared by all apps on the phone.
  - Allows notifications to be delivered even if app is not running.
- Web browsers can use **Websockets** or **Long-polling.**
  - In both cases, client is connected to one machine and service must somehow relay messages to that connection.