Cloud Voting Report

Introduction

Cloud computing is a model for on-demand access to a shared pool of configurable resources. We intend to bring user owned devices that weren't designed for cloud operation in this shared resource pool. Once these devices can communicate with each other, they will enable more smart processing and services and can be part of a larger infrastructure (IaaS). Cloud help to connect to the devices (IaaS) around us so that we can access anything at any time and any place in a user friendly manner using customized portals and in built applications (SaaS).

As a test application for the cloud model we will have a social choice voting implementation. Choosing the most preferred alternative when more than two choices are available (elections, popularity contests, even the menu for an event) is a complex issue if we want to choose fairly. Condorcet cycles can be present, or the straightforward anonymity, neutrality or strategyproofness conditions cannot be satisfied. Solutions like maximal lotteries or Kemeny's rule have been developed and will be implemented over the cloud system. Voting and determining the voting results will happen on different, loosely coupled cloud nodes - at the least one voting client node type and one node that centralizes and calculates results.

State-of-the-art

On the pure cloud computing side, the existing solutions are legion. Some of those are even specifically targeted as IoT devices, such as:

- **Google Cloud Platform** (https://cloud.google.com/solutions/iot/): Google Cloud Platform gives you the tools to scale connections, gather and make sense of data, and provide the reliable customer experiences that hardware devices require.

- **Oracle internet of things** (https://cloud.oracle.com/iot): Gain new data-driven insights and drive actions from IoT by connecting, analyzing and integrating device data into your business processes and applications, enabling your business to deliver innovative new services faster and with less risk.

- **Xively** (https://xively.com): simplifies the way companies securely and robustly connect their products and users, manage IoT data at scale, and engage more closely with their customers,users and partners.

On the Social Choice side, there is one publicly available solution, Pnyx () . It allows defining polls and obtaining results via standard plurality rules and also more advanced rules like Borda's, Fishburn's (maximal lottery) or Kemeny's (maximize pairwise agreement).

Our Solution

What we planned

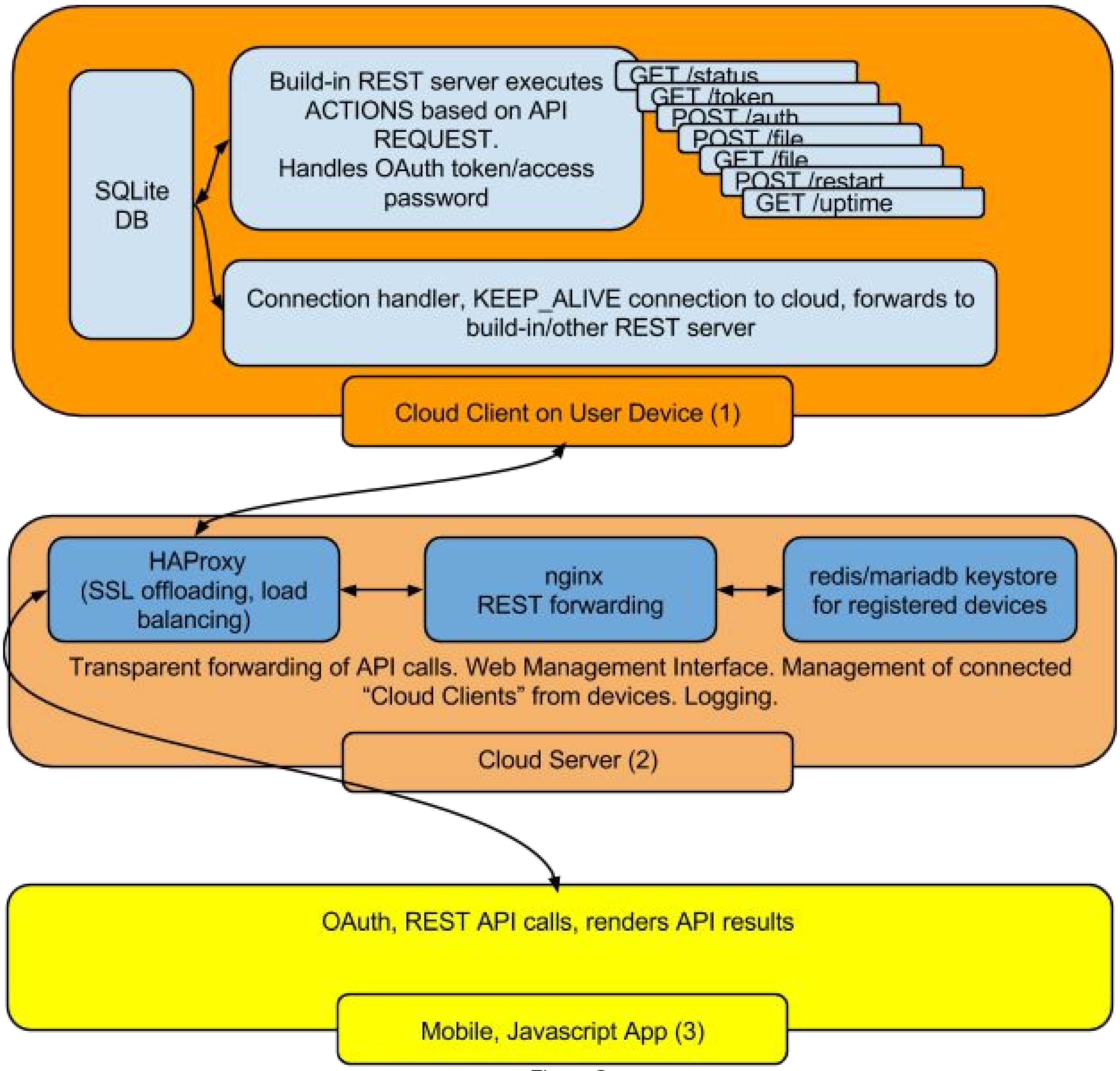
SQLite DB
Build-in REST server executes ACTIONS based on API REQUEST. Handles OAuth token/access password
GET /status
GET /token
POST /auth
POST /file
GET /file
POST /restart
GET /uptime
Connection handler, KEEP_ALIVE connection to cloud, forwards to build-in/other REST server
Cloud Client on User Device (1)
HAProxy (SSL offloading, load balancing)
nginx REST forwarding
redis/mariadb keystore for registered devices
Transparent forwarding of API calls. Web Management Interface. Management of connected "Cloud Clients" from devices. Logging.
Cloud Server (2)
OAuth, REST API calls, renders API results
Mobile, Javascript App (3)

Figure 2
anythingOnCloud Arhitecture

What we implemented

90% of great ideas end in the trashcan of history.
Ours made it to the proof of concept stage.

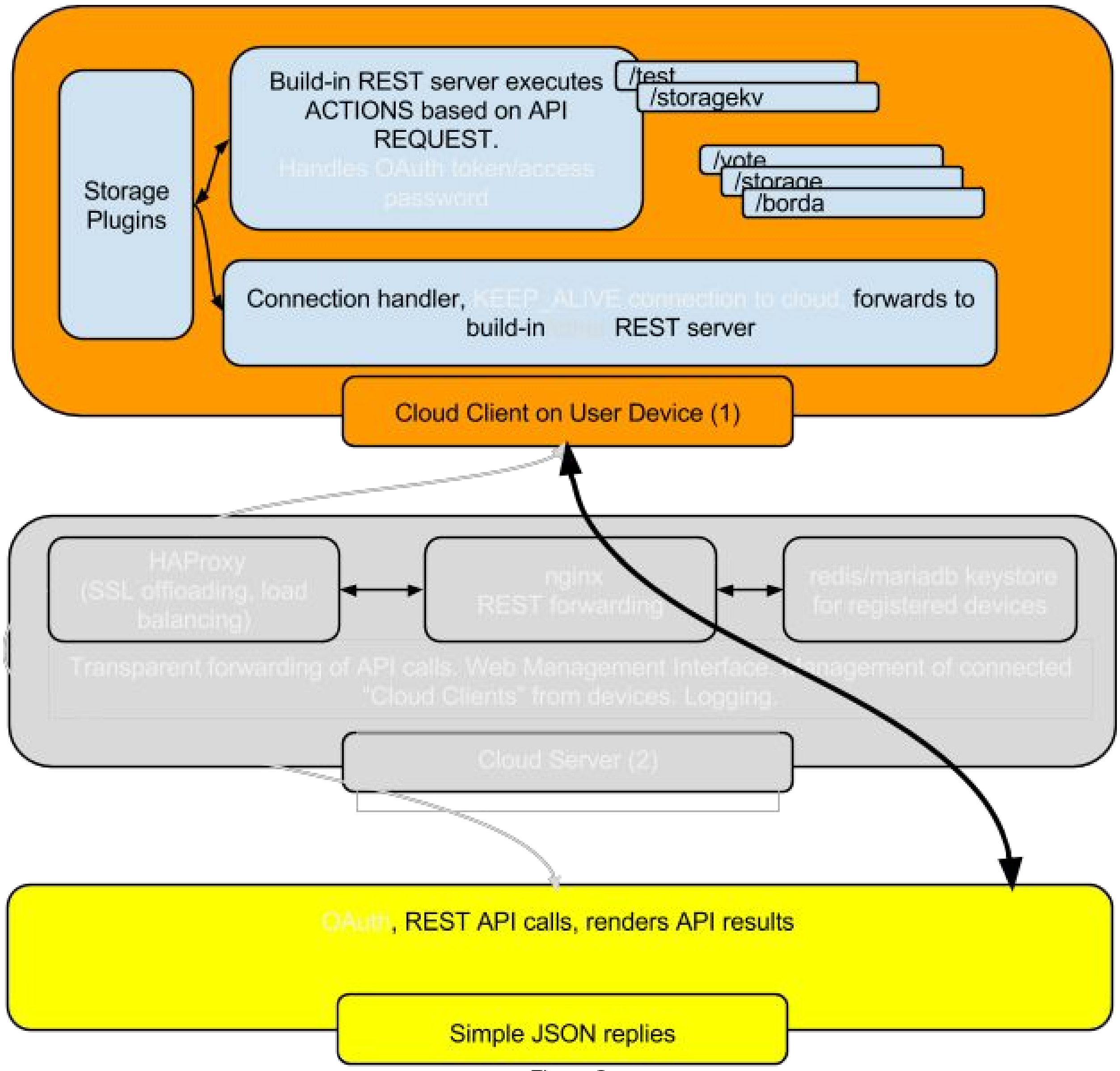
Storage Plugins
Build-in REST server executes ACTIONS based on API REQUEST.
Handles OAuth token/access password
/test
/storagekv
/vote
/storage
/borda
Connection handler, KEEP_ALIVE connection to cloud, forwards to build-in REST server
Cloud Client on User Device (1)
HAProxy (SSL offloading, load balancing)
nginx REST forwarding
redis/mariadb keystore for registered devices
Transparent forwarding of API calls. Web Management Interface. Management of connected "Cloud Clients" from devices. Logging.
Cloud Server (2)
OAuth, REST API calls, renders API results
Simple JSON replies

Figure 2
anythingOnCloud Arhitecture

We implemented a dynamic cloud plugin loading framework with automatic mapping from external URL requests to plugins loading them, and a Service oriented Architecture via said system of plugins.

The social choice (or polling) plugins/services rely on a storage service that could conceivably be run on a different machine.

Implemented components:

- Storage plugin - a REST interface to a key/value store. Offers the usual get, set and a getAll call that will return everything stored, which is used by the Borda plugin
- Test plugin - A simple RunTime test that check if the other plugins responds to queries
- Poll plugin - a proof of concept polling plugin limited to one poll. It allows users to register their vote and records it via the storage plugin.
- Borda plugin - aggregates the stored polling results via Borda's rule and displays the choice hierarchy based on the currently recorded votes

The latter two plugins offer a direct web interface for quick testing instead of a REST API. Incidentally, that was more effort than implementing an API accessible via another application, but it's a lot more useful for testing our framework.

Results, Evaluation

Due to major time constraints, the project was a disaster that never made it past the proof of concept stage. We believe that it did prove that our architecture makes sense though.

Choosing Python as the implementation language proved to be a very good decision though, as it enabled us to do a lot more in the available time than what a less flexible language would have allowed.

Comparison With Other Solutions

Our solution offers expandability, we mixed a service oriented architecture approach with plugins that actually made up the service architecture, each plugin can choose to use other services/plugins, and new services/plugins can be easily added. While in our implementation these services/plugins run locally, we could implement a plugin that can handle meta-information about other plugins running on other servers that would make this architecture distributed. Message passing in this architecture would be basic REST calls.

Future Work

- Insert message queue between REST API and the actual plugin
- Make message queue work with intermittent connectivity; store incoming/outgoing messages on both the client and the server and batch send them when a direct connection is available
- Authentication and per user plugin state (for example storage plugin would store data separately for each user)
- Generic polling plugin configurable from the data store
- Pretty clients

- Advanced choice aggregation methods, like Kemeny’s rule
- Service directory, for example the social choice plugins could use it to locate the storage plugin on a different machine

Conclusions

Human time is the most precious resource. We didn’t have enough of it.

Bibliography

- Fremantle, P. (2015) - A reference architecture for the internet of things
- Neisse, R., Steri, G. , Baldini, G (2015) - Dynamic context-aware scalable and trust based IoT Security Privacy Framework
- Huss P., Wigertz N (2014) - Flexible Architecture for Internet of Things Utilizing an Local Manager
- Felix Brandt (2014) - Axiomatic Social Choice Tutorial

Links

- Social Choice introduction: http://www.sigecom.org/ec14/EC%202014%20Tutorial.pdf
- Social Choice implementation: https://pnyx.dss.in.tum.de
- https://cloud.google.com/solutions/iot/
- https://xively.com
- https://imgs.xkcd.com/comics/crowdsourcing.png
- http://xkcd.com/1137/
-