ejabberd architecture and challenges in designing a modern messaging service

process one
Leader in Messaging and Push Solutions

17th November 2015

Mickaël Rémond <mremond@process-one.net>

Introduction

ejabberd is scalable and versatile enough to adapt to most of the realtime messaging jobs you will want to handle.

For many tasks, like corporate messaging, you can consider ejabberd as a **standard package**.

However, to reach high level of scalability and flexibility, you need to consider ejabberd as an **XMPP framework**.

As such, to build your modern messaging system, you need to:

- Be familiar with XMPP - Prerequisite.
- Learn ejabberd architecture
- Learn ejabberd API
- Work on solution design.

Take control of your messaging platform and design it !

ejabberd: Routing messages in a statefull world

ejabberd is a message router. Its role is to support real time messaging feature by moving messages from clients to other clients.

In that sense it is **stateless**.

However, to integrate in a statefull world, ejabberd has to support **statefull** features:

- Stateless:
    - ejabberd is an XMPP server: Its goal is to route XMPP packets between JID.
- Statefull:
    - In most case ejabberd depends on data (user base, contact list, ...) or produce data (message archive, ...)

Goal: deploy an ejabberd that is as stateless as possible, by leveraging backends.

What are ejabberd backends ?

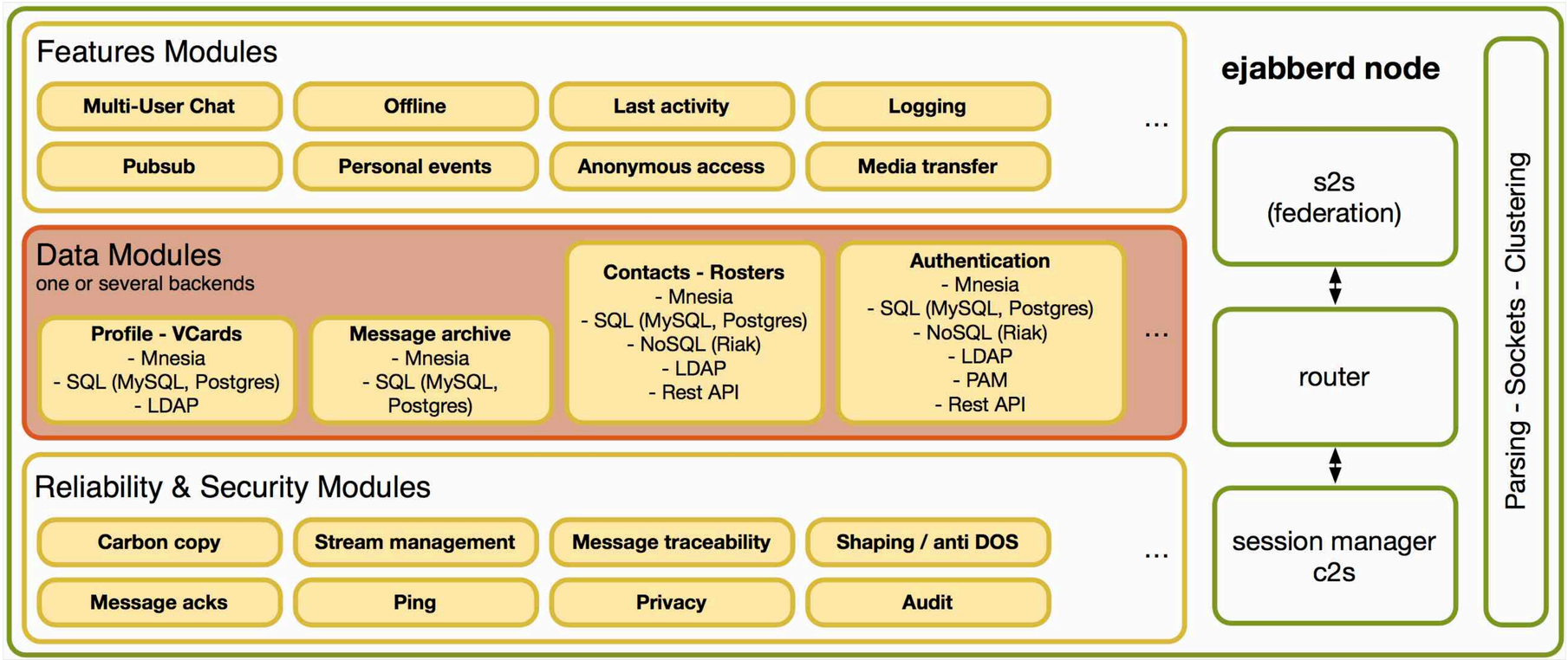
Backends are pluggable modules you can configure to define where you would like to store part or all of your data.

Backends provide the data to the feature modules of ejabberd.

They can be read-write or read-only if you do not need some of the ejabberd associated features:

- For example, if you handle user management elsewhere in your infrastructure, you can use a user back-end that can only authenticate users but not create them.

ejabberd internals: Feature modules rely on data modules - backends



Available backends

- Modules that need data backends generally support:
    - Internal database: Mnesia
    - RDBMS: MySQL / MariaDB, Postgres, SQLServer / Azure SQL, SQLite
    - NOSQL: Riak
- Some of them support additional backends (i.e authentication, shared roster):
    - LDAP / Active Directory / Azure Directory
- ejabberd SaaS provides ReST API description to implement HTTPS backend for:
    - User authentication,
    - Contact list (rosters),
    - Message archiving
- Custom backends in ejabberd-contribs: i.e. HTTP authentication.

Properties of ejabberd backends

Backend can be combined: Different features can use different backends to leverage different backend properties:

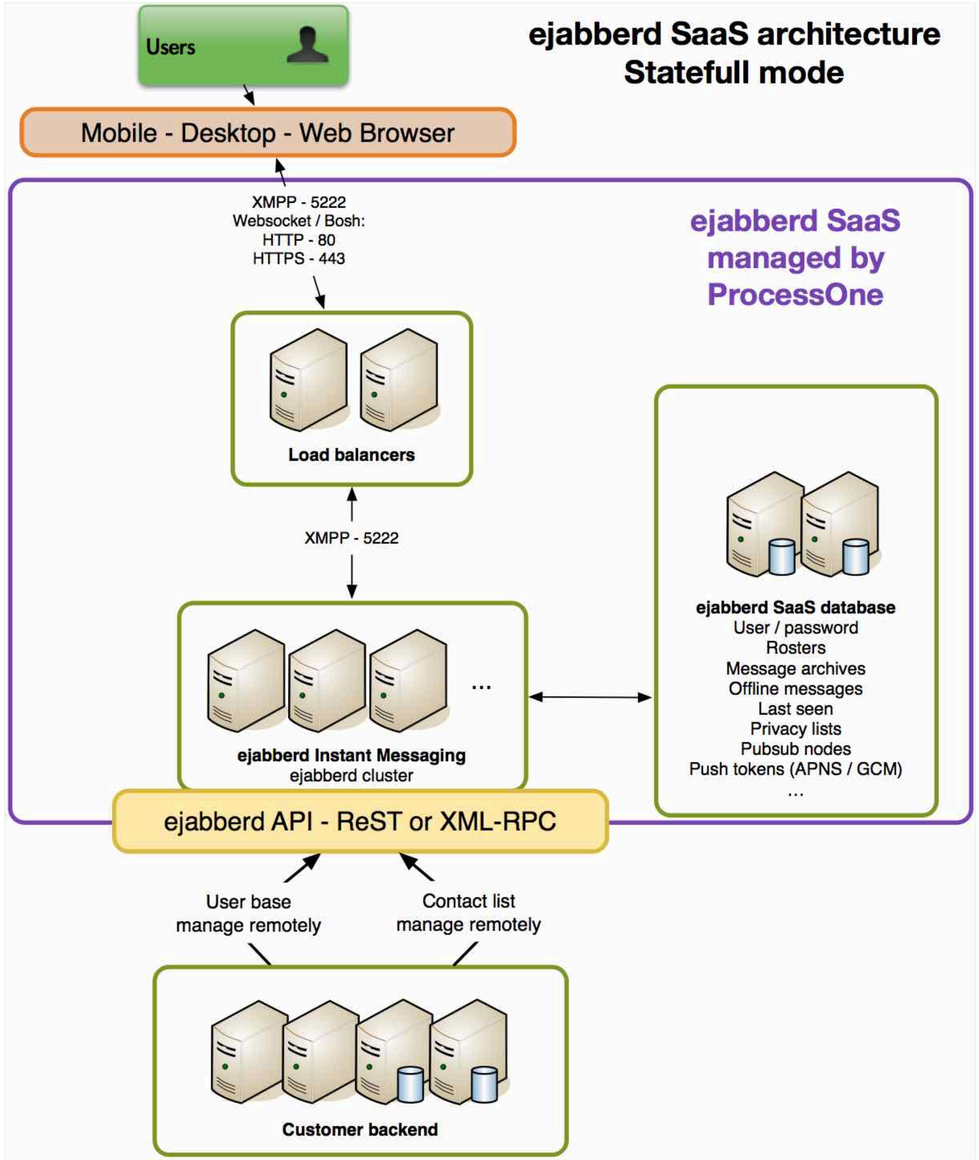
- Massive amount of data
- Speed
- ...

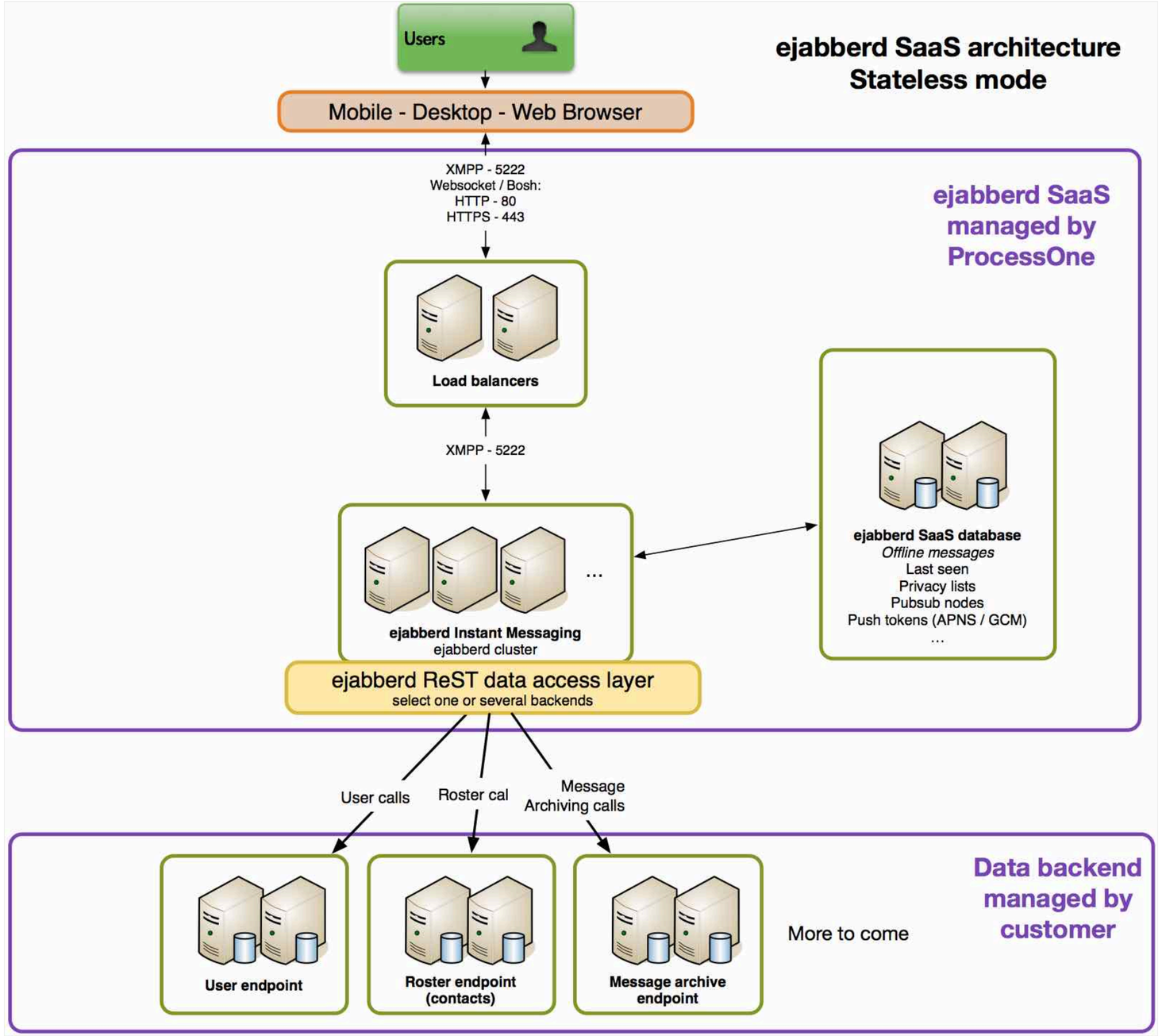
Backends are API based:

- They are developed using ejabberd API.
- It means you can write your own backend to meet special needs.

Selecting the right backend or deciding to write your own for the each module is part of ejabberd solution design.

The example of ejabberd SaaS




ejabberd SaaS architecture
Stateless mode
Users
Mobile - Desktop - Web Browser
XMPP - 5222
Websocket / Bosh:
HTTP - 80
HTTPS - 443
ejabberd SaaS
managed by
ProcessOne
Load balancers
XMPP - 5222
ejabberd SaaS database
Offline messages
Last seen
Privacy lists
Pubsub nodes
Push tokens (APNS / GCM)
...
ejabberd Instant Messaging
ejabberd cluster
ejabberd ReST data access layer
select one or several backends
User calls
Roster cal
Message
Archiving calls
User endpoint
Roster endpoint
(contacts)
Message archive
endpoint
More to come
Data backend
managed by
customer

The XMPP client is part of the equation

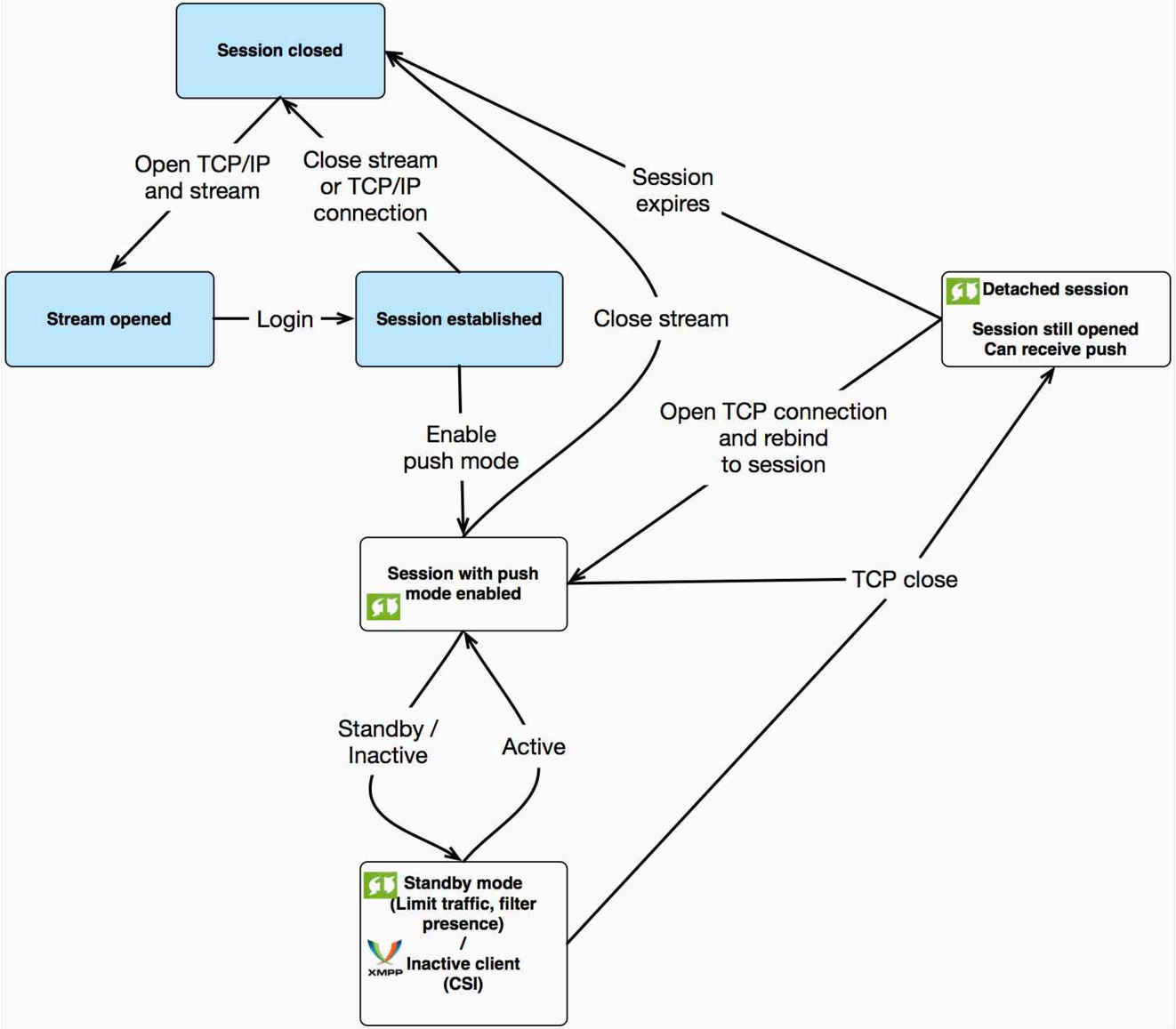
When designing a messaging the client is part of the equation:

- Mobile devices are now **powerful**: It can take a part of the server job.
- **Mobile limitations**: It cannot run all the time: Server has to handle part of the client job.
- Badly designed clients can generate enormous load on server and become a burden.

  Think:

  - exponential back-off.
  - Long running session on the server.
  - Avoid useless extra load: XMPP Ping vs lightweight Keep-alive.

The XMPP session becomes stateful as well:



Conclusion

Don't build your messaging platform without a clear vision of what you are building !

You need to become an architect of your "Messaging Zen Garden" !