Liquefy the cloud

Etienne Brodu
PhD student, poster presenter
S
etienne.brodu@insa-lyon.fr

Stéphane Frénot
INRIA - CITI
stephane.frenot@insa-lyon.fr

Frédéric Oblé
Worldline
frederic.oble@worldline.com

Fabien Cellier
Worldline
fabien.cellier@worldline.com

Abstract

Web services handle a fluctuating number of connected users implying variation of resources usage. These connections are usually processed by web services as a stream of user requests.

To be able to handle these variations in the stream, classical solutions offer developers an API to split the web service into parts, and dynamically distribute them onto multiple machines to balance the load.

We offer to automatically transpile a web service from regular code to multiple atomic stateless parts instead of constraining developers to adhere to a specific syntax.

Our destination model of transpilation is composed of atomic stateless parts containing the logic, listening for messages, and sending modified messages to other parts. A messaging system delivers messages between the parts.

To allow these stateless parts to store states, the messaging system binds a memory context with the message just before delivering it to the parts, and stores back modifications.

The messaging system then would be able to distribute these atomic parts between machines to balance the load from each part, and move memory contexts accordingly.

Using transpilation instead of a specific API, this approach keeps the scalability distinct from development, allowing developers to focus on business logic, and not en scalability issue, thus allowing small business to rapidly grow a user base.

Presentation of this poster

Etienne Brodu is a PhD student working part time in the DICE team, in the INRIA - CITI laboratory, and in the High Processing and Volumes (HPV) team at Worldline. He will present the poster, and the results we worked on during his first year as a PhD student.

Liquefy the cloud

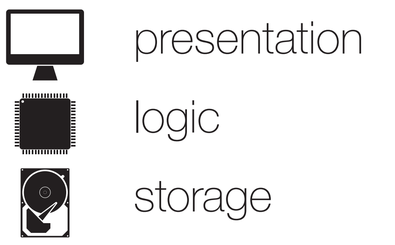
Etienne Brodu, Stéphane Frenot, Frédéric Oblé, Fabien Cellier

How to abstract web services' usage variation from developpement ?

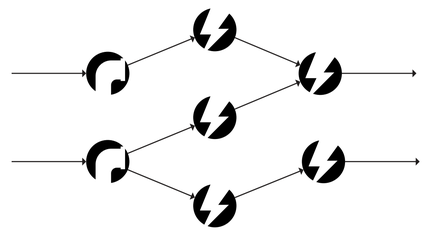
A popular web service might grow from thousands to millions of users in a matters of days.

To react to such variation of usage, they have to be scalable.

The classical approaches - the **three tiers** architecture, frameworks like **storm** or langages like **erlang** - allow developer to split web services into well defined parts in order to be scalable.



three tiers architecture



Storm

Instead we want to **automatically** split a web service into **stateless parts**, and making them communicate by **volatile data streams**, thus making the web service scalable.

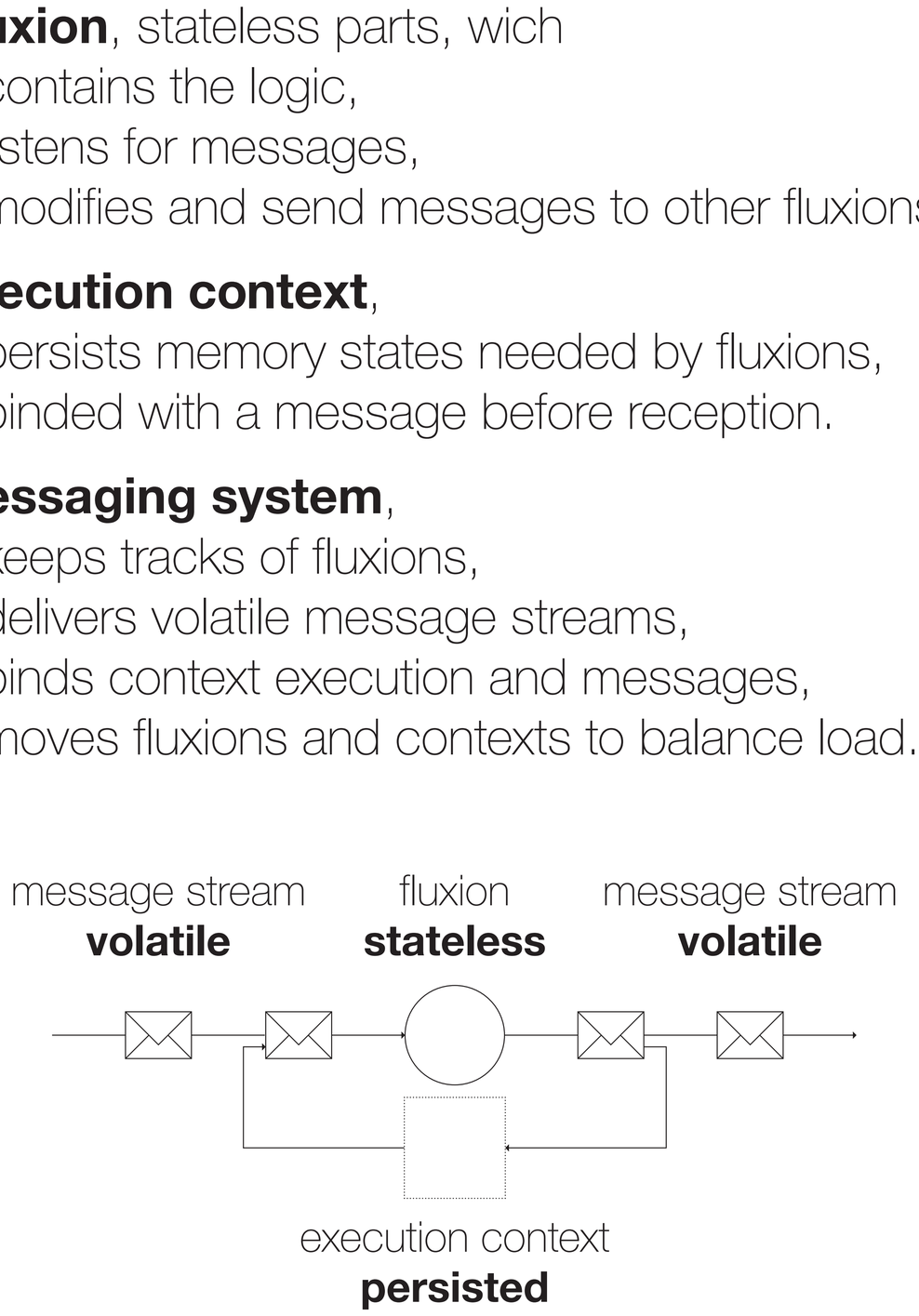
We want to abstract the scalability from developement constrains.

Our approach is composed of :

Fluxion, stateless parts, wich
+ contains the logic,
+ listens for messages,
+ modifies and send messages to other fluxions.

Execution context,
+ persists memory states needed by fluxions,
+ binded with a message before reception.

Messaging system,
+ keeps tracks of fluxions,
+ delivers volatile message streams,
+ binds context execution and messages,
+ moves fluxions and contexts to balance load.



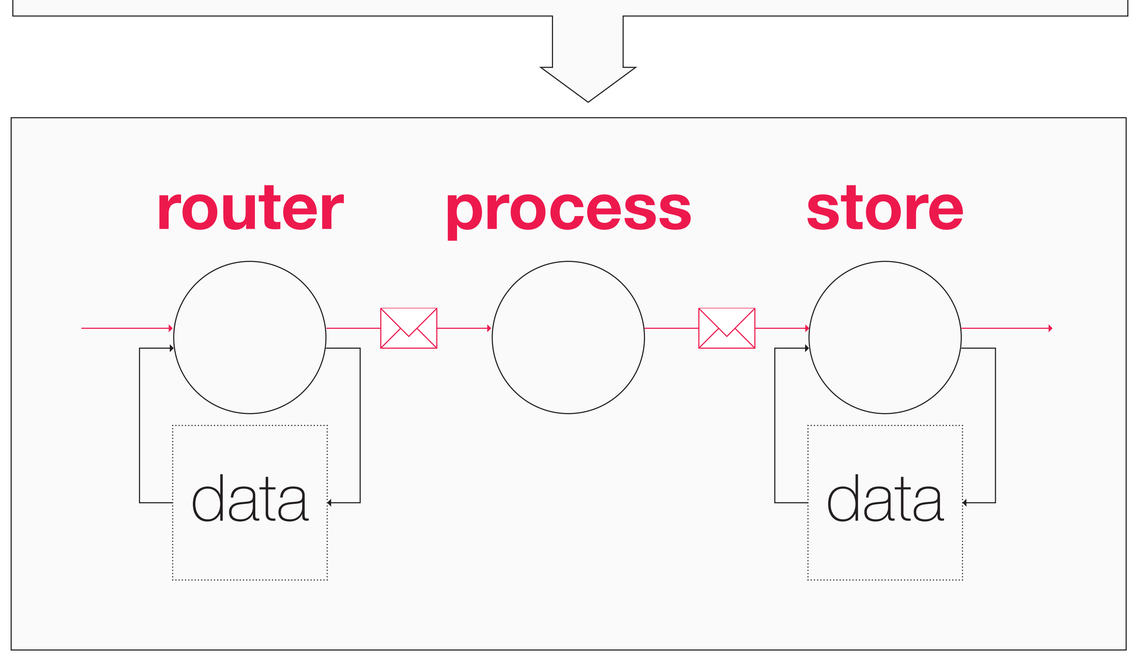
node.js express 3.0.0

```
function router(req, res) {
  // gather data
  next()
}

function process(req, res) {
  // process request
  next()
}

function store(req, res) {
  // store data
  next()
}

...
```



We aim to be able to handle more request than a monolithic approach, but without the constraints on developpement.