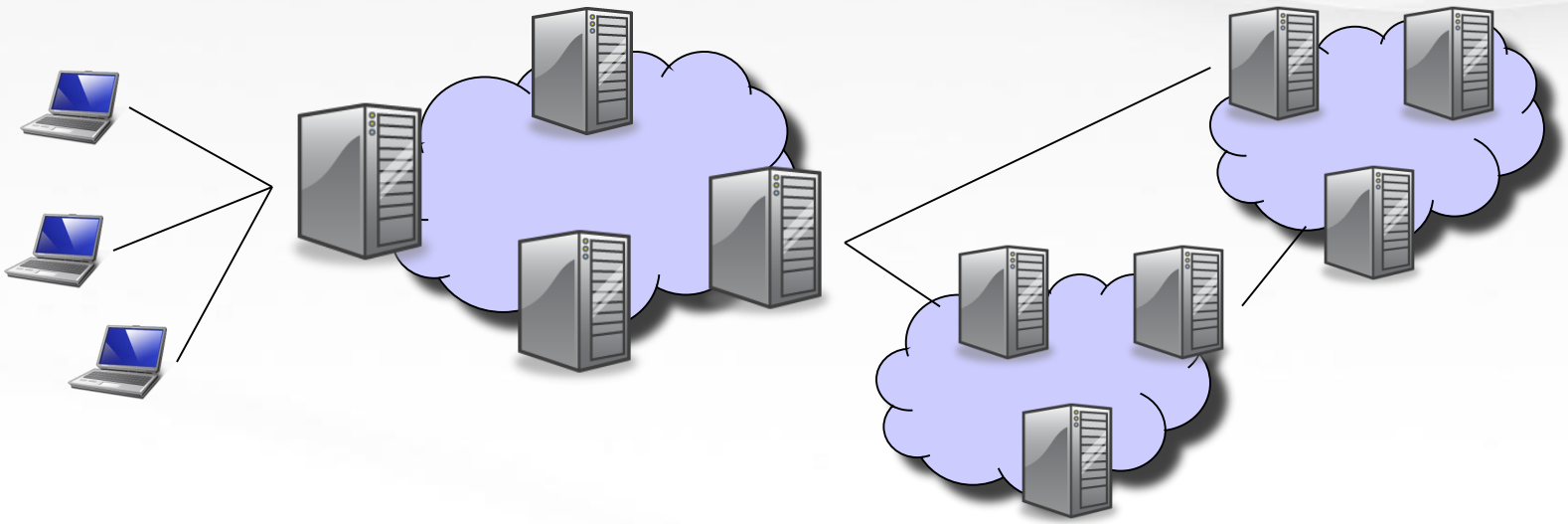
XMPP for Cloud Services

Cloud Camp – Milan
10 september 2009

Alessandro Malgaroli (Bluendo)
alex@jabber.bluendo.com
Fabio Forno (Bluendo)
ff@jabber.bluendo.com
Salvatore Loreto (Ericsson Research)
salvatore.loreto@ieee.org

What is Cloud Computing?

- Resources are provided as a service over the Internet
  - Dynamic Scalability
  - High Availability
- Exploding in popularity – a critical trend of software architectures
- Started with simple services; now growing complex
  - Now predominately based on REST architectures, web applications, web services



Cloud Architecture Problems

- REST is exclusively based on the Request/Response pattern
  - No push and asynchronous interactions
  - Polling doesn't scale and isn't real-time;
  - Need of two-way data exchange

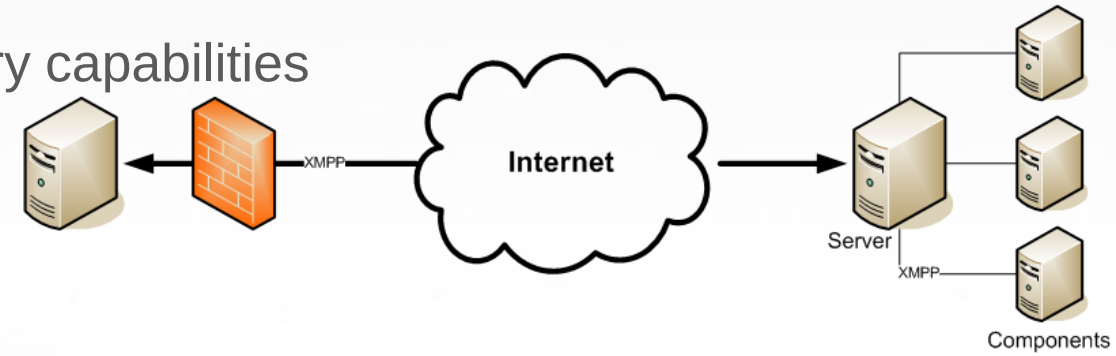
  (Comet/Bosh/WebSocket could be used to make some improvements)
- Web services (SOAP) are overly complicated when handling:
  - Presence (availability) and discovery of remote services
  - Federation with third party services, access behind NAT and firewalls
  - Many-to-Many distribution pattern, asynchronous or multi step calls
  - Binary data, data streaming

Thesis: *web services are great for simple cloud services or direct p2p interactions; XMPP is better for complex cloud services*

XMPP & Cloud Computing

XMPP with its extensions is a powerful protocol for cloud services that demonstrate several adavantages over HTTP-based Web Services:

- Realtime capabilities
  - Eg. Heartbeats, alarms, asynchronous webservices
- Efficient distribution of data: publish/subscribe, direct push
  - Eg. Configuration distribution, push RSS/Atom, data collection, log processing, results delivery to clients
- Federated services & discovery
  - Easy interdomain RPCs
  - Access control based on JIDS and JID domains
  - Advanced discovery capabilities



XMPP Architecture

- XMPP provides a technology for asynchronous, end-to-end exchange of structured data.
- XMPP architectural style builds an overlay network having:
  - Global addresses (JIDs)
  - Network availability (presence)
  - Concurrent information transactions
  - Distributed, federated network
  - Structured Data with XML payloads
- The architecture similar to that of the email network, but it introduces several modifications to facilitate near-real-time communication

”Availability for Concurrent Transactions”(ACT)

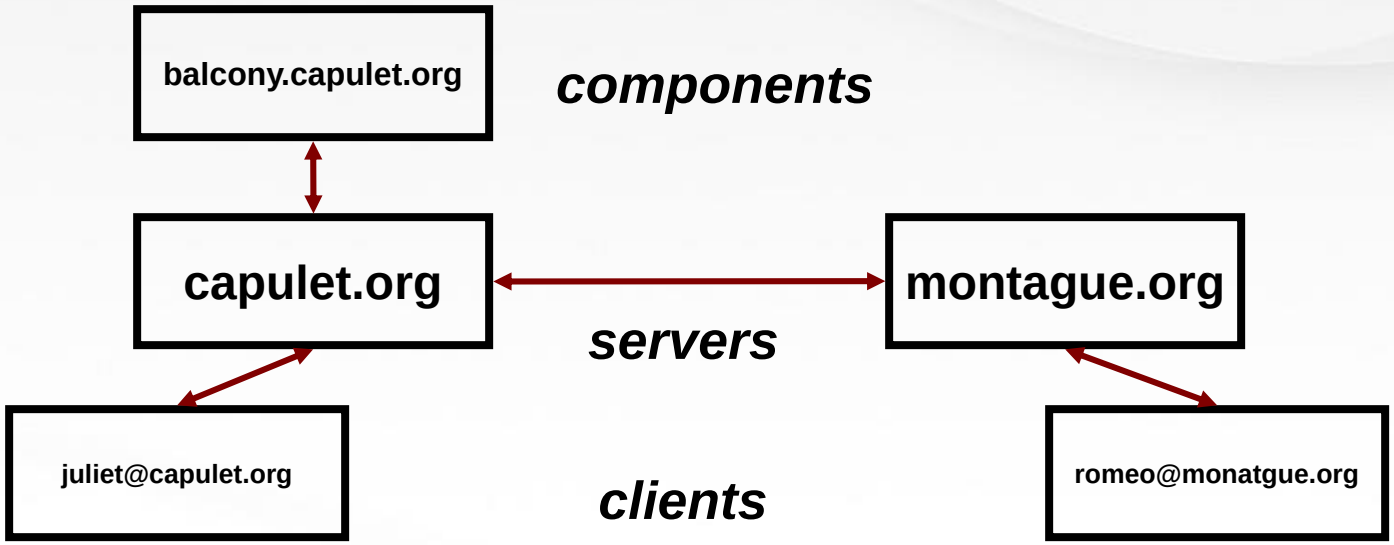
Peter Saint-Andre
https://stpeter.im/index.php/2009/09/01/active-architectures

XMPP Architecture

- Global Addresses:
  - As with email, XMPP uses globally-unique addresses.
  - All XMPP entities are addressable on the network:
- clients
- servers
- additional services accessible by clients and servers.
- Presence:
- the ability for an entity to advertise its network availability or "presence" to other entities via dedicated communication primitives (the <presence/> stanza).

XMPP Architecture

- Distributed Network, comprising servers, clients and components (server extensions)
- End-to-end communication in XMPP is logically peer-to-peer but physically client-to-server-to-server-to-client
- Federation across different domains



XMPP Architecture

- Data is sent through persistent XML streams
- The basic unit of meaning in XMPP is an “XML stanza”:
  - a fragment of XML that is sent over a stream
  - the root element of a stanza includes routing attributes (such as "from" and "to" addresses),
  - there are three basic stanzas with different behavior
  - the child elements of the stanza contain an extensible payload for delivery to the intended recipient

XMPP Stanzas

	Pattern	Examples

Presence	Broadcast
	to subscribed JIDs
		`<presence from='juliet@capulet.org'>`
		`  <c xmlns='http://jabber.org/protocol/caps' …/>`
		`</presence>`

Message	One way
	Asynchronous
		`<message from='romeo@montague.org'`
		`  to='juliet@capulet.org'>`
		`<event`
		`  xmlns='http://jabber.org/protocol/pubsub#event'>`
		`…`
		`</event>`
		`</message>`

IQ	Request
	Response
		`<iq from='romeo@montague.org'`
		`to='balcony.capulet.org'`
		`  type='get' id='123'>`
		`<query xmlns='jabber:iq:disco#items"/></iq>`
		
		`<iq from='balcony.capulet.org'`
		`to='romeo@montague.org'`
		`  type='result' id='123'>`
		`<query xmlns='jabber:iq:disco#items"><items>…<items>`
		`</query></iq>`

An example: Publish/Subscribe

- Observer pattern:
  - Data is not directly sent to consumers
  - Producers publish data to an intermediate service, into topics or nodes
  - Consumers subscribe to intermediate nodes
  - Pubsub usually implemented as a dedicated component in servers, any JID can be subscriber or publisher
- Applications
  - Log monitoring, data collection and processing, atom, *wave style applications*
- Why not (only) JMS?
  - Global naming / Federation / Presence / Sophisticated access model

PubSub & REST

- Each node is a resource were it is possible to do actions and it is identified by URIs:
  - xmpp:pubsub.acme.org;/atom/sport
  - xmpp:user@example.org;/pep/location
- Each node contains items, which are stateful representation of the resource
- Each node support actions:
  - PUBLISH: publish (PUT) an item or update (POST) and item
  - DELETE: retract an item
  - GET: retrieve a published item
  - LIST: retrieve the available items
  - SUBSCRIBE: be notified of any change in node items
- REST + Realtime notifications + many-to-many
  - Delivery of notifications is bound to presence (no messages lost)

PubSub Applications

- Implementations
  - Builtin in most servers: tigase, ejabberd, openfire
  - Separate components: idavoll (http://idavoll.ik.nu/)
  - Clients: psi, lampiro, iphone3g, buddycloud, …
  - Libs: wokkel (python), smack (limited support, java)
  - Application servers: DEM (http://freedem.sf.net/)