Tipping the Webscale

with XMPP & WebSockets

Sonny Scroggin

email/xmpp: **sonny@scrogg.in** + github/twitter: **@scrogson**

Nashville, TN

openstack hosted private cloud // hybrid cloud

bluebox.net

WHAT WE'LL COVER

- INTRODUCTION TO THE XMPP PROTOCOL
- SERVERS
- CLIENTS
- COMPONENTS
- HOW TO APPLY

EXTENSIBLE MESSAGING & PRESENCE PROTOCOL

CORE SERVICES

- Channel Encryption
- Authentication
- Presence
- Roster
- Messaging (one-to-one, multi-party)
- Service Discovery
- Notifications (Publish/Subscribe)
- Capabilities Advertisement
- Data Forms
- Peer-to-Peer Media Sessions

Why XMPP?

Open - free, public, and easy to understand

Standard - RFC 6120 / 6121 / 6122

Proven - first developed in 1998 and is very stable

Decentralized - similar to email, run your own server

Secure - SASL and TLS built-in

Extensible - build on top of the core protocols

XMPP WHO USES IT?

Apple - iMessage, Push Notifications, Bonjour

Google - GoogleTalk, Push Notifications

Cisco - Webex

Facebook - chat

WhatsApp - started with ejabberd

Chesspark - online chess game

ROUTING / ADDRESSING

ROUTING / ADDRESSING

JID (Jabber ID) - addressing similar to email

Bare - `<user>@<server>`

Full - `<user>@<server>/<resource>`

Resource - allows multiple sessions per user

- Automatically generated by the server if not specified
- Messages can be addressed to either the bare JID or full JID
- Messages to the bare JID route to the full JID with the highest priority

BASIC CONNECTION LIFECYCLE

BASIC CONNECTION LIFECYCLE

- Client initiates a TCP connection to the server
- Sends stream header
- Negotiates stream features and authentication mechanisms
- The client binds to a resource
- The client sends presence to the server
- The client requests its roster
- The client sends and receives messages to/from others
- The client ends the session and closes the connection

COMMUNICATION PRIMATIVES

XML Stanzas

Fragments of XML sent over an XML stream

```
<message/>
```

```
<presence/>
```

```
<iq/>
```

Common Attributes

to: specifies the JID of the intended recipient

from: specifies the JID of the sender

type: specifies the purpose or context of the message

xml:lang: *specifies the default language of any such human-readable XML character data*

id: used by the originating entity to track any response or error stanza that it might receive in relation to the generated stanza from another entity

The <message/> stanza is a "push" mechanism.

- One entity pushes information to another entity, similar to the communications that occur in a system such as email.
- The payload in the case of instant messaging is a <body/> element, but it can also include any other custom elements to extend the protocol.

```
<message
  to="romeo@im.montague.lit"
  id="sa7f8df"
  type="chat"
  xml:lang="en">
  <body>Where art thou, Romeo?</body>
</message>
```

NOTE: Juliet's server will add the "from" attribute when routing the message.

The `<presence/>` stanza is a "broadcast" or "publish/subscribe" mechanism.

- Multiple entities receive information (in this case, network availability information) about an entity to which they have subscribed.
- Sets and shows the availability of the entity that is connected.

Announce availability on the server

```
<presence/>
```

STANZAS: PRESENCE

Announce you're away

```
<presence>
  <show>away</show>
</presence>
```

Do not disturb

```
<presence>
  <show>dnd</show>
  <status>working hardcore...</status>
</presence>
```

Going offline

```
<presence type="unavailable"/>
```

Subscriptions

Juliet requests a subscription to Romeo's presence

```
<presence to="romeo@im.montague.lit" type="subscribe"/>
```

Romeo gladly excepts

```
<presence to="juliet@im.capulet.lit" type="subscribed"/>
```

STANZAS: IQ

The <iq/> stanza is a "request/response" mechanism, similar in some ways to HTTP.

- Enables an entity to make a request of, and receive a response from, the server or another entity.
- Request type is either "get" or "set".
- Response type is either "result" or "error".
- Service discovery

Juliet requests Romeo's vCard

```
<iq to="romeo@im.montague.lit" type="get" id="s8f7">
  <vCard xmlns="vcard-temp"/>
</iq>
```

Romeo's server responds

```
<iq
  to="juliet@im.capulet.lit/balcony"
  from="romeo@im.montague.lit"
  type="result"
  id="s8f7">
  <vCard xmlns="vcard-temp">
    <EMAIL>
      <USERID>romeo@montague.lit</USERID>
    </EMAIL>
    <FN>Romeo Montague</FN>
  </vCard>
</iq>
```

Roster request

```
<iq to="im.capulet.lit" type="get" id="s8f7">
  <query xmlns="jabber:iq:roster"/>
</iq>
```

Server response

```
<iq to="juliet@im.capulet.lit/balcony" type="result" id="s8f7"
  from="im.capulet.lit">
  <query xmlns="jabber:iq:roster">
    <item jid="romeo@im.montague.lit" name="Romeo Montague"
     subscription="both">
      <group>Lover</group>
    </item>
  </query>
</iq>
```

SERVICE DISCOVERY

Query clients or servers about features/capabilities

Enables you to find out which features are supported by another entity, as well as any additional entities that are associated with it (e.g., rooms hosted at a chat room service).

disco#info query

```
<iq
  to="juliet@im.capulet.lit/balcony"
  type="get"
  id="s8f7">
  <query xmlns="http://jabber.org/protocol/disco#info"/>
</iq>
```

disco#info response

```
<iq
  from='juliet@im.capulet.lit/balcony'
  to='romeo@im.montague.lit/library'
  type='result'
  id='a2a3'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <identity category='client' type='pc' name='Adium'/>
    <feature var='http://jabber.org/protocol/caps'/>
    <feature var='http://jabber.org/protocol/chatstates'/>
    <feature var='http://jabber.org/protocol/muc'/>
    ...
    <feature var='http://jabber.org/protocol/muc#user'/>
    <feature var='http://jabber.org/protocol/xhtml-im'/>
    <feature var='http://jabber.org/protocol/commands'/>
  </query>
</iq>
```

SERVICE DISCOVERY

disco#items query

```
<iq
  to="im.capulet.lit"
  type="get"
  id="s8f7">
  <query xmlns="http://jabber.org/protocol/disco#items"/>
</iq>
```

disco#items response

```
<iq
  from='im.capulet.lit'
  to='juliet@im.capulet.lit/balcony'
  id='1396'
  type='result'>
  <query xmlns='http://jabber.org/protocol/disco#items'>
    <item jid='conference.im.capulet.lit'/>
    <item jid='pubsub.im.capulet.lit'/>
    <item jid='vjud.im.capulet.lit'/>
    <item jid='im.capulet.lit' node='announce' name='Announcements'/>
    <item jid='im.capulet.lit' name='Configuration' node='config'/>
    <item jid='im.capulet.lit' name='User Management' node='user'/>
    <item jid='im.capulet.lit' name='Online Users' node='online users'/>
    <item jid='im.capulet.lit' name='All Users' node='all users'/>
 </query>
</iq>
```

SERVERS

ejabberd

https://github.com/processone/ejabberd

The World's Most Popular XMPP Server Application

- Alexey Shchepin started ejabberd in November 2002
- Written in Erlang
- Version 1.0 in December 2005
- Supports most popular XMPP services
- Extremely flexible (gen_mod, ejabberd_hooks)

MongooseIM

https://github.com/esl/MongooseIM

- Erlang Solutions forked ejabberd at version 2.1.8
- OTP compliant project structure
- Improved the build process
- Removed obsolete/rarely used modules to reduce maintenance burden
- Reduction of runtime memory consumption
- Test coverage of the system according to corresponding RFCs and XEPs.

MongooseIM provides high availability, ease of deployment, development, and reliability in production. It's aimed at large, complex enterprise level deployments where real-time communication is critical for business success.

CLIENTS

Stanza.io

https://github.com/otalk/stanza.io

Modern XMPP in the browser, with a JSON API.

Stanza.io is a library for using modern XMPP in the browser, and it does that by exposing everything as JSON. Unless you insist, you have no need to ever see or touch any XML when using stanza.io.

Useful when connecting directly to a websocket enabled XMPP server and you have built custom server modules to handle your business logic.

Example Echo Client

```
var XMPP = require('stanza.io'); // if using browserify

var client = XMPP.createClient({
    jid: 'echobot@example.com',
    password: 'hunter2',

    transport: 'websocket',
    wsURL: 'wss://example.com:5281/xmpp-websocket'
});

client.on('session:started', function () {
    client.getRoster();
    client.sendPresence();
});

client.on('chat', function (msg) {
    client.sendMessage({
      to: msg.from,
      body: 'You sent: ' + msg.body
    });
});

client.connect();
```

Borrowed from the stanza.io README

Hedwig

https://github.com/scrogson/hedwig

XMPP Client/Bot Framework

Hedwig is an XMPP client and bot framework written in Elixir. It allows you to build custom response handlers to fit your business logic.

Simply configure Hedwig with credentials and custom handlers and you're set!

Hedwig Client Configuration

```
use Mix.Config

alias Hedwig.Handlers

config :hedwig,
  clients: [
    %{
      jid: "juliet@im.capulet.lit",
      password: "R0m30!"
      nickname: "romeosgirl",
      resource: "balcony",
      rooms: [
        "lobby@conference.im.capulet.lit"
      ],
      handlers: [
        {Handlers.Help, %{}},
        {Handlers.GreatSuccess, %{}}
      ]
    }
  ]
```

Example Handler

```
defmodule Hedwig.Handlers.GreatSuccess do
  use Hedwig.Handler

  @links [
    "http://mjanja.co.ke/wordpress/wp-content/uploads/2013/09/borat_great_success.jpg",
    "https://www.youtube.com/watch?v=r13riaRKGo0"
  ]

  def handle_event(%Message{} = msg, opts) do
    cond do
      hear ~r/great success(!)?/i, msg -> process msg
      true -> :ok
    end
    {:ok, opts}
  end

  def handle_event(_, opts), do: {:ok, opts}

  defp process(msg) do
    :random.seed(:os.timestamp)
    link = Enum.shuffle(@links) |> List.first
    reply(msg, Stanza.body(link))
  end
end
```

COMPONENTS

COMPONENTS

XEP-0114: Jabber Component Protocol

- Allows you to extend server functionality
- External - connects to server or vice versa
- Handshakes and authenticates with the server
- Can be written in any language
- Can interact on the whole domain or subdomain
- Can alter stanzas including the to/from attributes

Phoenix Framework

Phoenix

https://github.com/phoenixframework/phoenix

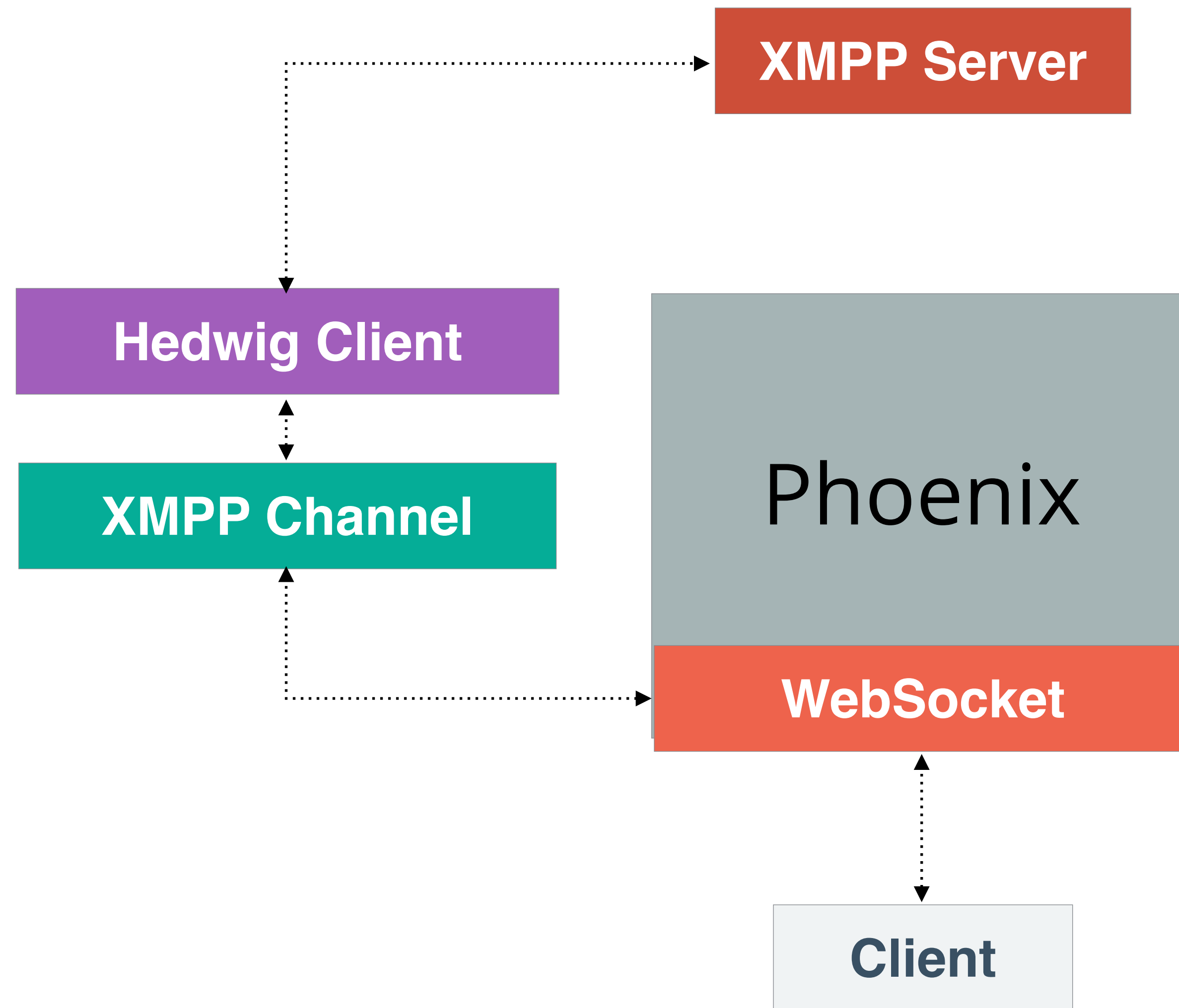
Elixir Web Framework

Phoenix is a web framework targeting full-featured, fault tolerant applications with realtime functionality. It has support for WebSockets baked-in.

Router

```
defmodule MyApp.Router do
  use Phoenix.Router

  pipeline :browser do
    plug :accepts, ~w(html)
    plug :fetch_session
  end

  pipeline :api do
    plug :accepts, ~w(json)
  end

  scope "/", alias: MyApp do
    pipe_through :browser

    get "/pages/:page", PageController, :show

    resources "/users", UserController do
      resources "/comments", CommentController
    end
  end

  scope "/api", alias: MyApp.Api do
    pipe_through :api

    resources "/users", UserController
  end

  channel "xmpp", XMPPChannel
end
```

IDEAS


XMPP Server
Hedwig Client
XMPP Channel
Phoenix
WebSocket
Client

DEMO?

Fragen?

Danke!