Phoenix Framework

Proyecto de Red Social en 7 días

MANUEL ÁNGEL RUBIO JIMÉNEZ

Phoenix Framework

Proyecto de Red Social en 7 días

Manuel Angel Rubio Jiménez

Phoenix Framework
Proyecto de Red Social en 7 días

Manuel Angel Rubio Jiménez

Resumen

Elixir está siendo una revolución para el desarrollo web de aplicaciones gracias al uso de librerías como Ecto, Plug y sobre todo Phoenix Framework que aúna todas las buenas prácticas recogidas de muchos otros entornos de programación y lenguajes para proporcionar un entorno actual, rápido, tolerante a fallos (***thread-safe***) y escalable. Poco a poco se comienza a oír cada vez más dentro de empresas como Bleacher Report, Remote, Pepsico, Cabify o Financial Times entre otras.

Este libro intenta acercar de una forma práctica cómo desarrollar empleando Phoenix Framework a través del desarrollo de un proyecto de red social. Este proyecto nos permitirá abordar y resolver problemas típicos que podemos encontrar en otros desarrollos así como introducir cada aspecto del framework. Crearemos la lógica de negocio de nuestra red social almacenándola en PostgreSQL, implementaremos controladores para definir el flujo de la aplicación, las vistas y plantillas revisando cómo Webpack puede ayudarnos en esta tarea y la puesta en producción de nuestra solución.

Phoenix Framework ha sido y está siendo noticia dentro del mundo del desarrollo de software tanto en sus inicios soportando 2 millones de conexiones Websocket simultáneas como después con la publicación de LiveView y LiveDashboard que nos permiten desarrollar aplicaciones web con una respuesta mucho más rápida y fluida así como una interfaz donde observer qué sucede dentro de nuestra aplicación web. ¿Te unes a la aventura?

Depósito legal CO-XXX-2021.

ISBN 978-84-945523-9-7

[1] http://books.altenwald.com/elixir-i
[2] https://creativecommons.org/licenses/by-nc-sa/3.0/deed.es

¿Qué es Phoenix Framework?

La verdadera potencia del framework es haber escrito el código para mil usuarios y cambiar unas 10 líneas de código y tenerlo preparado para soportar millones de usuarios.
—Chris McCord

En 2013 comencé con la idea de emplear Erlang para el desarrollo web. Quería experimentar la capacidad de desarrollar una solución escalable para soportar millones de conexiones. El famoso gráfico de Joe Armstrong (ver la Figura 1, "Comparación Apache vs Yaws") de la comparación de Yaws y Apache ha sido una revelación para la mayoría de nosotros. Había algunas opciones disponibles en ese entonces pero la opción más desarrollada y probada fue ChicagoBoss[1].

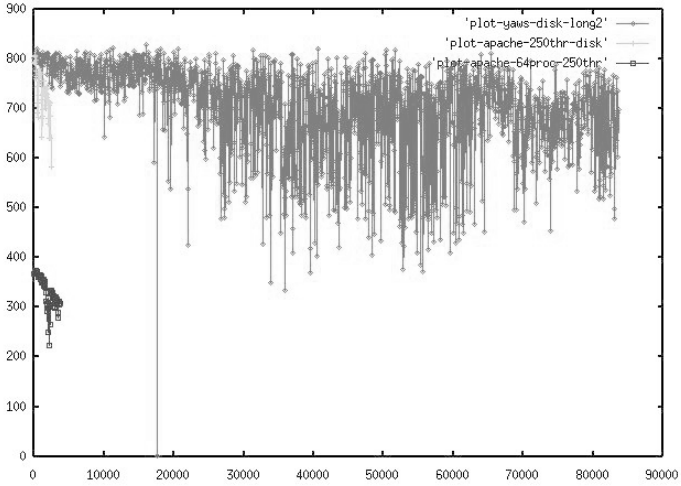

Figura 1. Comparación Apache vs Yaws

En diciembre de 2013 ofrecí una charla en BetaBeers Córdoba[2] donde comentaba cómo emplear el framework para el desarrollo web. No obstante, el empleo de ChicagoBoss nos dejaba a todos los usuarios con la sensación de ***aún queda mucho por hacer***.

La popularización de Elixir poco a poco y la salida de Phoenix Framework en mayo de 2014 abrió una nueva posibilidad. A diferencia de

[1] https://altenwald.org/2020/04/28/chicagoboss-por-que-no-triunfo/
[2] https://altenwald.org/2013/12/17/betabeers-cordoba-x/

ChicagoBoss, el trabajo hecho en Phoenix Framework parecía tener una cohesión mucho más fluida a través de la implementación de diferentes elementos: Plug, Ecto, EEx y por supuesto Elixir.

La base de Elixir y sus pilares puestos en la extensibilidad, la documentación y las herramientas aportaron todo lo necesario para el triunfo. Todos los elementos son claros ejemplos de esta extensibilidad proporcionando una forma de empleo fácil para la creación de rutas en Plug, o consultas en Ecto o la transformación de las páginas dinámicas HTML mediante EEx.

La documentación es un punto clave en la difusión y expansión de la plataforma. Además, tener disponible la documentación a través de la plataforma de distribución de paquetes Hex[3] facilita el estudio y reutilización de código de manera más fácil.

Por último, la herramienta de construcción **mix** permite definir tareas para cada uno de los elementos. Phoenix provee tareas para crear un proyecto desde cero así como diferentes generadores. Ecto provee la generación y ejecución de migraciones así como la creación y eliminación de base de datos.

Vamos a revisar cómo surgió Phoenix Framework, quiénes son los creadores, sus raíces y hasta dónde ha llegado hasta el momento su desarrollo. El uso de Ecto y su gran cantidad de backends para acceso a base de datos actuales, la aparición de ***LiveView*** y el futuro del framework.

1. Las raíces de Phoenix Framework

Como decíamos antes, Phoenix Framework se basa mucho en el uso de algunas librerías desarrolladas en Elixir. Una de ellas y para mi una de las más importantes, es Plug. Esta librería fue desarrollada por Jose Valim en 2013. Jose inició desarrollando inicialmente Dynamo[4], un framework web completo donde probar muchas ideas con pretensión de competir al mismo nivel de otros como Sinatra[5] (en Ruby) o Express[6] (de JavaScript).

Como librería Plug cumplió su objetivo. Permitió a desarrolladores agregar un sistema servidor web muy sencillo y extensible. A día de hoy además dispone de muchas extensiones publicadas en Hex para agregar autenticación básica a nivel de HTTP, parseadores de JSON y XML, impresión de logs en formato JSON o enviándolos directamente a Prometheus, almacenaje de las sesiones en Mnesia y cientos de posibilidades más.

[3] https://hex.pm
[4] https://github.com/dynamo/dynamo
[5] http://sinatrarb.com/
[6] https://expressjs.com/es/

Por otro lado tenemos otra de las grandes librerías en las que se apoya Phoenix Framework: Ecto. Aunque es cierto que podemos crear proyectos de Phoenix Framework en los que no empleemos bases de datos (por ejemplo en sitios corporativos), la mayoría de proyectos que crearemos con Phoenix Framework emplearán base de datos y Ecto nos proporciona una capa de abstracción que nos permite emplear cualquier base de datos soportada y realizar consultas de forma simple. Es más, incluso nos permite realizar validaciones y un control bastante detallado de los datos de entrada desde formularios, incluso cuando estos no irán a parar a una base de datos.

Ecto fue desarrollada principalmente por Eric Meadows-Jönsson a partir de mediados de 2013. Eric no solo ha aportado código a Ecto, sino también aporta en la creación y evolución de Elixir.

Con esta base, Chris McCord comenzó el desarrollo de Phoenix Framework a principios de 2014. En principio incluyendo solo Plug y un esbozo de cómo serían los controladores. La descripción inicial era: ***Realtime Elixir Web Framework***; y uno de sus principales objetivos era hacer de los WebSockets el elemento principal del framework y una distribución en anillo para el sistema.

Poco a poco se fueron sumando cada vez más contribuidores, la base de código comenzó a evolucionar muy rápido incluyendo las vistas, las plantillas, las tareas para **mix** y todos los elementos que hacen a Phoenix Framework uno de los frameworks más completos a día de hoy.

Sobre sus objetivos podemos decir que a través de los canales cumplió con mantener los WebSockets como elemento fácil de emplear en la infraestructura. Desgraciadamente no existe ningún sistema de distribución en anillo que nos permita distribuir por defecto nuestro código. No obstante, para ello podemos emplear Elixir y otras librerías como Horde[7]. Pero esto será otra historia.

2. Soportando 2 Millones de Usuarios

Nota

Esta historia no es nueva y ya la comenté también en el libro Elixir: Introducción para Alquimistas[8], no me gusta repetirme, pero la historia merece la pena ser contada y creo que es digna de mención.

A finales de 2015 Chris McCord[9] comenzó a publicar una serie de tuits donde mencionaba cómo estaba probando e intentando alcanzar 2

[7] https://github.com/derekkraan/horde
[8]
[9] https://github.com/chrismccord

millones de usuarios concurrentes[10] conectados por WebSocket en Elixir y más específicamente en Phoenix Framework.

Todo comenzó cuando Gary Rennie[11] estaba haciendo pruebas sobre cuántos canales simultáneos podría soportar Phoenix Framework. Conseguía un máximo de mil (1000) en su máquina local. Al comentarlo vía IRC y ante la falta de bancos de pruebas, los desarrolladores del núcleo de Phoenix Framework quisieron arrojar un poco de luz sobre cómo realizar las pruebas y aportar números.

Rackspace aportó tres máquinas de 15GB I/O v1 con 15GB de RAM y 4 núcleos cada una. También tuvieron acceso a un OnMetal I/O de 128GB con 40 núcleos.

Las pruebas para generar el número crítico de clientes se realizaron a través de Tsung[12], un sistema para realizar bancos de pruebas en diferentes protocolos y generar una alta carga. La primera configuración probada fue con un servidor configurado para aceptar las peticiones y los otros dos para generar el tráfico de cliente usando Tsung.

Se alcanzaron 27 mil conexiones simultáneas. José Valim hizo una corrección en el código de Phoenix para acelerar la entrada de usuarios. Se repitieron las pruebas y esta vez se alcanzaron 50 mil conexiones simultáneas.

A través de una visualización de los procesos mediante ***observer***[13] Chris pudo detectar otro cuello de botella. Chris eliminó el sistema de heartbeat de la conexión de websocket al ver que se duplicaba el número de temporizadores en uso. Tanto el sistema de websocket como cowboy implementaban un temporizador. Dejó únicamente el provisto por cowboy en una nueva modificación de Phoenix Framework. De nuevo ejecutaron las pruebas y esta vez llegaron a 100 mil conexiones simultáneas.

Debido a las limitaciones de tener solo dos máquinas para clientes y solo poder generar 40 mil y 60 mil conexiones simultáneas respectivamente, emplearon otra máquina de 128GB disponible para conseguir más clientes y seguir las pruebas. Esta vez consiguieron 330 mil conexiones simultáneas.

Gabi Zuniga[14] echó un vistazo y agregó una corrección al sistema. Empleando otro tipo de dato para las tablas ETS consiguieron llegar a 450 mil conexiones simultáneas.

[10] http://phoenixframework.org/blog/the-road-to-2-million-websocket-connections
[11] https://github.com/Gazler
[12] http://tsung.erlang-projects.org/
[13] Una aplicación de Erlang/OTP que permite obtener información en tiempo real del sistema en ejecución a través de una interfaz gráfica de usuario (GUI).
[14] https://twitter.com/gabiz

La empresa Live Help Now[15] aportó a la prueba 45 máquinas en Rackspace para proseguir gracias a la participación de Justin Schneck[16] en las pruebas. Configuraron Tsung en unas cuantas máquinas y lo limitaron a 1 millón de conexiones simultáneas para hacer las pruebas de nuevo. Terminaron de configurar el resto de las 45 máquinas para probar si era posible llegar a 2 millones de conexiones en una sola máquina. Desafortunadamente se quedaron en 1,3 millones de conexiones. Nada mal no obstante.

Pero algo no estaba bien. A esos niveles de carga los mensajes para los suscriptores se demoraban más de 5 segundos. Tras varios análisis Chris tuvo la idea de generar un concentrador (pool) para el sistema de publicación/suscripción (pubsub) y Justin la de fraccionar (shard) la tabla ETS donde se almacenaba todo. Esta combinación permitió reducir el tiempo de difusión a un segundo y alcanzar 2 millones de conexiones simultáneas.

Cada escenario concreto tiene sus propias limitaciones y como hemos visto pasar de mil conexiones simultáneas a dos millones ha tenido una travesía de análisis y correcciones permitiendo a los desarrolladores ir mejorando cada vez más el sistema. José, Chris, Gary y todos los implicados quedaron satisfechos con la cifra alcanzada aunque comenta también que tras la publicación siguieron encontrando otras mejoras para poder aplicar.

La buena noticia para todos es que estas mejoras y las futuras que se han ido descubriendo han ido formando parte de Elixir y/o Phoenix Framework y eso nos garantiza un sistema robusto y con un alto grado de escalabilidad horizontal.

3. La aparición de LiveView

Otro gran hito en la historia de Phoenix Framework e incluso por extensión en la propia historia de Elixir es la salida de ***LiveView***. Es posible que hayáis oído hablar de esta librería o extensión de Phoenix Framework. En pocas palabras permite desarrollar qué sucede en el navegador a través de código escrito en el servidor en Elixir. La librería escrita en ***JavaScript*** que provee ***LiveView*** es la encargada de conseguir este efecto reduciendo la fragmentación de código entre la parte escrita en cliente y la parte escrita en el servidor.

La idea se dió a conocer a través de un post titulado Phoenix LiveView: Aplicaciones en Tiempo-Real, Interactivas. Sin Necesidad de Escribir JavaScript[17] por Chris McCord en la página web de DockYard en

[15] http://www.livehelpnow.net/
[16] https://twitter.com/mobileoverlord
[17] https://dockyard.com/blog/2018/12/12/phoenix-liveview-interactive-real-time-apps-no-need-to-write-javascript

diciembre de 2018. Este fue el punto de partida de la idea. En abril de 2019 fue presentado en Elixir Conf EU 2019[18].

En esa misma conferencia dio a conocer una de las mejores campañas de márketing para dar a conocer aún más la librería. Un concurso a través de la página Phoenix Phrenzy[19]. De esta forma consiguieron en muy pocos meses cientos de ejemplos de cómo desarrollar código con ***LiveView*** así como gran cantidad de feedback que sirvió para avanzar en el desarrollo tanto en nuevas características como en estabilidad.

Hoy en día, ***LiveView*** está integrado con Phoenix Framework desde el lanzamiento de la versión 1.5 y tan solo agregando un parámetro de configuración es configurado para poder emplearlo desde el principio.

4. El futuro de Phoenix Framework

Phoenix Framework es una plataforma novedosa, como hemos visto desde el principio hace uso de un lenguaje actual, enfocado en el desarrollo a gran escala gracias a BEAM[20], con grandes herramientas y librerías y además siempre presentando nuevas innovaciones como es el caso reciente de ***LiveView*** y LiveDashboard.

El equipo de desarrollo es joven y está enfocado en proporcionar soluciones para el mundo del desarrollo web. Podemos encontrar además muchas empresas que se suman para aportar soluciones en base a completar el entorno de desarrollo, pruebas, trabajo con base de datos y un largo etcétera.

La siguiente gran adición está siendo desde marzo de 2020, una solución para autenticación oficial dentro de Phoenix Framework. Tal y como comentaba José Valim en un artículo titulado una nueva solución de autenticación para Phoenix[21], se anunciaba el trabajo realizado por Aaron Renner[22] creando esta librería y se ha ido avanzando para poder tener una instalación fácil de la librería en proyectos usando Phoenix Framework 1.5 o superior.

Esta es solo una de entre muchas más novedades que nos depara tanto el equipo de desarrollo de DockYard encabezado por Chris McCord, como el equipo formado dentro de Dashbit por José Valim.

[18] https://www.youtube.com/watch?v=txk4WAlabvI
[19] https://phoenixphrenzy.com/
[20] La máquina virtual de Erlang donde pueden ejecutarse otros lenguajes que compilen en su formato de bytecode.
[21] https://dashbit.co/blog/a-new-authentication-solution-for-phoenix
[22] https://github.com/aaronrenner