INSTITUTO TECNOLÓGICO BUENOS AIRES

PROYECTO FINAL

PROYECTO TIX

Desarrollo e implementación de una arquitectura horizontalmente escalable

Autores
Matías Gabriel
DOMINGUES
Facundo Nahuel
MARTINEZ CORREA
Javier PÉREZ CUÑARRO

Tutor
Dr. Ing. José Ignacio
ALVAREZ-HAMELIN

12 de diciembre de 2017

Índice

2. Arquitectura

2.1. Arquitectura previa

El Proyecto TiX en principio fue pensado como un conjunto de subsistemas. En la Figura 1 se puede ver un diagrama de los servicios que comprendían y que, a grandes rasgos, aún comprenden el actual sistema de TiX. Estos son el Subsistema de Presentación y Administración, el Subsistema de Procesamiento y el Cliente del sistema.

2.1.1. Responsabilidades e implementación de los subsistemas

Subsistema de Presentación y Administración
Este subsistema estaba integrado exclusivamente por la aplicación Web con la que interactuaba el usuario. La misma era una aplicación monolítica hecha en Java [5], usando la combinación de Spring [6], Hibernate [7] y Wicket [8]. Su instalación era manual, usando un Apache Tomcat en el servidor de la infraestructura mediante un archivo WAR [9] que se compilaba en la computadora del desarrollador. Cumplía también el rol de administrar los usuarios y sus distintas instalaciones.

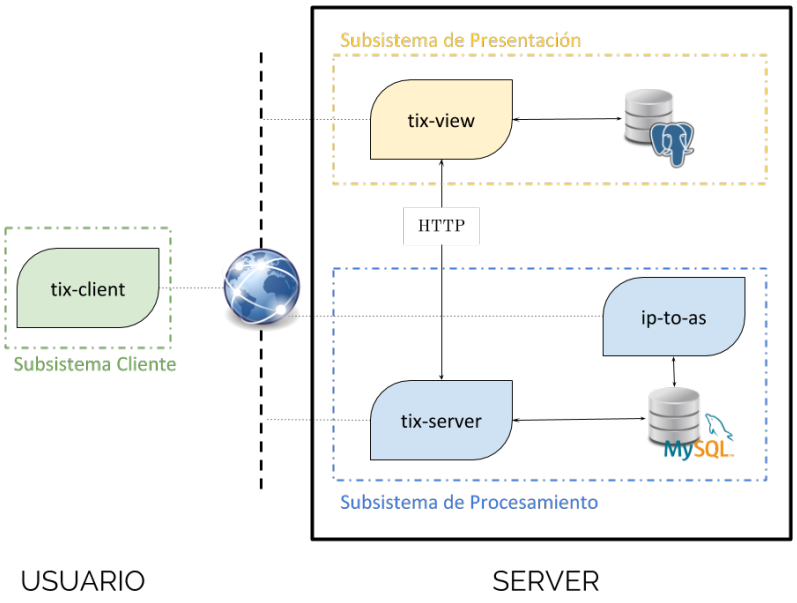

Figura 1: Arquitectura previa del Sistema

Subsistema de Procesamiento
Este subsistema estaba integrado por dos servicios que compartían la misma base de código.

Por un lado estaba el Servicio de Procesamiento de Reportes, el cual era un sistema monolítico, con código principalmente hecho en Python 2.7 [10] y con algunas llamadas externas a código escrito en R [11]. Su instalación se realizaba a través de Capistrano [12], una herramienta escrita en Ruby [13], y de forma remota en el servidor. La función de este servicio era la de interactuar con el cliente, actuando como servicio de respuesta ante los paquetes enviados.

Por otro lado estaba el IP2AS, el cual era un servicio escrito en Python. El mismo se encargaba de mantener actualizada la tabla que relacionaba direcciones IP con Sistemas Autónomos y Proveedores de Servicio de Internet. El mismo funcionaba en base a una tarea agendada que se ejecutaba automáticamente gracias a un Cron [14].

Subsistema Cliente
Este subsistema era un archivo ejecutable de Python que corría en segundo plano. Su principal y única función era la de enviar los distintos paquetes hacia el Subsistema de Procesamiento con el fin de realizar las mediciones.

2.1.2. Funcionamiento conjunto del sistema

El sistema, para un usuario determinado, comenzaba a funcionar tras la creación de la cuenta y la posterior descarga del cliente a su computadora a través del Subsistema de Presentación y Administración. En ese momento se creaba una instalación desde la cual se enviaban paquetes UDP y reportes al Subsistema de Procesamiento. Éste recibía los reportes con las mediciones tomadas, comprobaba su integridad y los almacenaba en un directorio del Sistema Operativo. Tras contar la cantidad de reportes recibidos, si éstos llegaban a una cantidad suficiente, se procedía a procesarlos con el fin de estimar el uso y calidad de la conexión para el intervalo de tiempo correspondiente. Una vez hecho el procesamiento, se cotejaba el Sistema Autónomo y Proveedor al que pertenecía la IP de donde habían llegado los reportes. Luego el Subsistema de Procesamiento enviaba el resultado al Subsistema de Presentación y Administración a través de una pequeña interfaz REST [15] expuesta. Así es que finalmente el usuario podía visualizar los resultados a través del Subsistema de Presentación y Administración.

2.1.3. Fortelazas de esta arquitectura

La principal fortaleza de esta arquitectura era el buen balance que había entre la distribución de los distintos componentes y el diseño monolítico de los mismos. Esto permitía que estén bien separadas las capas y que pudieran ser montadas o desplegadas en distintos servidores de manera autónoma en infraestructuras

que cumplan con los requerimientos propios a cada estrato. A su vez, el hecho de que cada una de estas capas sea monolítica en sí misma confería una gran facilidad en el seguimiento de errores y su eventual solución.

El hacer el cliente exclusivamente en Python tenía como principal idea permitir iteraciones rápidas, y la entrega de actualizaciones constantes y minimalistas al usuario. Esto sería a través de actualizar exclusivamente los archivos de Python modificados, en una suerte de repositorio constante.

2.1.4. Problemas y debilidades de esta arquitectura

Tras trabajar durante unos meses con este sistema, se hicieron evidentes varios problemas y posibles dificultades futuras.

La forma en que el Subsistema de Presentación y Administración había sido diseñado mostraba problemas sustanciales al momento de interactuar con el resto de los subsistemas, así como para evolucionarlo. Estaba construido con una versión muy vieja de Spring y pensado exclusivamente como una aplicación Web, no como un sistema que pudiera interactuar con otros otros. La pequeña interfaz REST que se exponía fue una adición que excedía al diseño original, con lo que se convertía en código con mucha deuda técnica y de muy difícil modificación o mejora.

El haber empaquetado todo el subsistema en un único archivo, hacía que fuera muy difícil referenciar ficheros que estuvieran por fuera del paquete. Por este motivo, se terminó decidiendo empaquetar en el mismo archivo WAR los binarios para los clientes. Esto trajo como obvias consecuencias que dicho archivo fuera muy grande y que debía ser re-empaquetado ante una actualización del cliente.

El Subsistema de Procesamiento era un servicio en Python que se levantaba y corría de forma autónoma. Sin embargo, su diseño estaba pensado para que corriese exclusivamente en la máquina del desarrollador y en el servidor de pruebas de TiX. Hacerlo correr en un entorno distinto a estos implicaba tocar variables definidas en el código de la aplicación para que su configuración sea la adecuada. Estas variables estaban distribuidas a lo largo de toda la aplicación y con poca o nula documentación.

La aplicación Cliente que se instalaba en las computadoras de los usuarios estaba pensada para tener una fácil actualización. Sin embargo, dependía de que el usuario volviese a empaquetar la aplicación. Algo que no se le podía pedir.

Finalmente, el protocolo de comunicación, implementado en su totalidad en Python, estaba escrito dos veces - en el cliente y en el Subsistema de Procesamiento - haciéndolo muy difícil de mantener. Además, la implementación no era clara ni mantenía las especificaciones dadas por el protocolo.

2.2. Arquitectura actual

En base a las observaciones presentes en el capítulo anterior, se decidió hacer una evaluación de la arquitectura. Se hizo evidente que para continuar con el desarrollo y las subsecuentes mejoras se debían hacer cambios profundos en el sistema. Se llegó a la conclusión de que los siguientes tres pilares eran necesarios para finalizar el proyecto exitosamente:

1. El Proyecto tiene que poder ser encarado y progresado por terceros ajenos al desarrollo inicial y con escasa o nula ayuda.
2. El Sistema debe ser fácil de medir, para poder ser mejorado de manera continua.
3. El Sistema debe ser robusto, confiable y performante, para cumplimentar con los objetivos del presente Proyecto de Grado.

Como consecuencia de los tres postulados anteriores, se destiló que:

- El sistema debía ser sencillo, evitando la repetición de código donde fuera posible, aislando las funcionalidades en módulos concretos y usando un único servicio externo por tipo, a menos que hubieran razones de performance que lo justificaran.
- Todas las partes debían contar con pruebas comprensivas, que ayuden a formalizar el código y a evidenciar su intención.
- Todas las partes debían tener una instalación automatizada que garantizaran el correcto despliegue de las mismas independientemente del entorno donde se encontrasen.

Teniendo en cuenta estos últimos dos ítems, es que se decidió utilizar tres herramientas fundamentales en la arquitectura y el trabajo de operaciones del sistema.

- **TravisCI [16]** como sistema para automatizar no sólo las pruebas en cada uno de los repositorios, sino también el despliegue.
- **Docker [17]** como forma de mantener congeladas las dependencias y el estado de cualquiera de las partes del sistema que se instalasen en el servidor.
- **DockerHub [18]** como repositorio para guardar y distribuir todas las partes con el fin de que más gente lo quisiera adoptar y mejorar, además de como facilidad para la instalación en el servidor.
- **Docker-Compose [19]** como mecanismo básico de orquestación para los servicios del sistema.

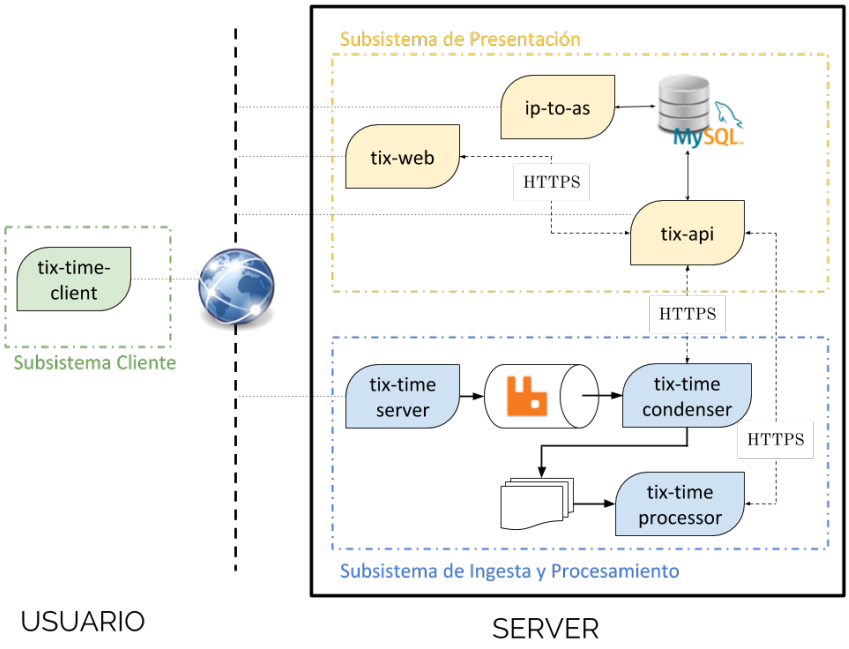

Figura 2: Arquitectura actual del Sistema

En lo que respecta al primer punto, se tuvieron en consideración los siguientes ítemes:

- La arquitectura y la separación de responsabilidades entre los subsistemas existentes era buena, aunque en su implementación se filtrasen algunas entre los distintos módulos. por ejemplo, el IP2AS.
- No era necesario el uso de dos bases de datos.
- Se debía formalizar el protocolo de comunicación entre el Subsistema Cliente y el Subsistema de Procesamiento mediante una biblioteca que fuera común a ambos.
- La información de negocio, y en consecuencia, la de visualización debía estar contenida exclusivamente en el Subsistema de Presentación.
- Cada una de las partes debía ser homogénea en sí misma, evitando ejecutar procedimientos que pudieran vulnerar la integridad del sistema.

En base a todo esto, se entendió que el Sistema en general debía tener la arquitectura que se muestra en la imagen de la Figura 2.

2.2.1. Responsabilidades y consideraciones de los subsistemas

Generalidades
Como se explaya en la subsección previa, se tuvieron en cuenta dos consideraciones generales y que tocan a más de un subsistema.

La primera es la necesidad de una biblioteca que formalice el protocolo de TiX. Esta se expone a través de un repositorio central, del cual puede ser descargado como cualquier administrador de dependencias. Dicha biblioteca, tix-time-core, posee una gran batería de pruebas automatizadas debido a que es una pieza central de toda la arquitectura y por ese motivo se intenta que sea lo más confiable posible. Por otro lado, la idea de hacerla en Java [5] y exponerla a través de Maven [20] como dependencia responde a las necesidades tanto del Subsistema Cliente como del Subsistema de Ingesta y Procesamiento. Java es un lenguaje multiplataforma, que óptimo para hacer un cliente enfocado a distintos Sistemas Operativos. Por otro lado, la JVM [21] es una Máquina Virtual con grandes prestaciones y excelente performance en relación a otros lenguajes de programación. Además, Java cuenta con un rico ecosistema de bibliotecas y frameworks que permiten el fácil desarrollo de sistemas.

La otra gran consideración es la de mover el servicio de IP2AS del Subsistema de Ingesta y Procesamiento al Sistema de Presentación. Dos fueron las ideas detrás de esta decisión:

1. Simplificar el enriquecimiento de los datos para la visualización del usuario, dejando esta tarea exclusivamente al Subsistema de Presentación.
2. Usar un único servicio de base de datos para guardar la información.

Con respecto a esta última idea, se decidió usar el Sistema de Manejo de Bases de Datos Relacional MySQL por ser altamente confiable, de código libre, con grandes prestaciones y sencillo de usar.

Subsistema de Presentación
El Subsistema de Presentación fue pensado bajo los mismos principios que se repiten a lo largo del proyecto. Se buscó generar una arquitectura que fuera altamente escalable y de fácil instalación, por lo que se dividió el subsistema en dos módulos.

Capa de presentación
Esta capa permite al Usuario:

- Visualizar los reportes de sus distintas instalaciones
- Visualizar y editar las instalaciones creadas por el usuario
- Editar sus preferencias de usuario (email/contraseña)
- Visualizar sus reportes mensuales por instalación

También le permite al Administrador:

- Visualizar las instalaciones de todos los usuarios del sistema y sus reportes
- Editar el rol de los demás usuarios, así como deshabilitar cuentas.
- Visualizar gráficos de utilización generales filtrados por Proveedor de Servicios de Internet (ISP), así como también descargar los informes en un archivo con formato de Valores Separados por Comas (CSV) para su posterior análisis.

La principal tecnología de desarrollo es ReactJS [22], la cual permite, mediante el uso de empaquetadores como Webpack [23], servir el código ya compilado mediante una Red de Distribución de Contenidos (CDN). Esto asegura que la aplicación pueda llegar a un gran número de usuarios de forma efectiva y a muy bajo costo.

Capa de de Administración y Reglas de Negocio
Esta capa expone una interfaz, la cual puede ser alcanzada por cualquier cliente que sea capaz de autenticarse vía Json Web Token (JWT) [24] o Basic Auth [25]. Esto permite que si en el futuro se decidiera implementar una aplicación mobile, entonces el desarrollador podría de manera muy simple conectarla el servicio y consultar o agregar reportes.

Existen dos roles definidos para los usuarios del sistema, el de Usuario Común y el de Administrador. Cuando el usuario se autentica mediante usuario y contraseña, se le confieren ciertos permisos de acuerdo al rol.

Los permisos para el rol Usuario Común son:

- Consultar sus reportes
- Editar su email
- Editar su contraseña
- Consultar, editar o borrar sus instalaciones.

Mientras tanto, si el usuario autenticado tiene rol de Administrador, además de tener los mismos permisos que un usuario regular, este puede:

- Consultar los reportes basado en ISP
- Consultar los reportes basado en un usuario particular
- Consultar, editar o borrar instalaciones de otros usuarios
- Generar nuevos reportes para un usuario.

El Rol de Administrador también es usado por el Subsistema de Ingesta para que pueda ingresar datos del usuario sin tener que depender de las credenciales del mismo.

En busca nuevamente de escalabilidad, esta capa se encuentra detrás de un servidor Nginx. Esto permite que, de ser necesario, se puedan instalar múltiples instancias de este servicio y realizar un balanceo de carga.

Finalmente, los datos consultados y creados por esta capa son almacenados en MySQL, en junto con los datos de IP2AS.

Subsistema de Ingesta y Procesamiento
El Subsistema de Ingesta y Procesamiento fue pensado con las siguientes especificaciones en mente:

- Debe ser capaz de ingerir y procesar la información de 5000 clientes en simultáneo
- Debe ser altamente escalable
- Debe poder validar los paquetes recibidos desde el cliente y asegurar que pertenezcan a instalaciones activas usuarios activos del sistema

Y además debe bajo las especificaciones generales del Sistema TiX presentes al principio de este capítulo.

Para logralo se decidió ir por una arquitectura de microservicios. Esto confería tres grandes ventajas:

1. Fácil escalamiento horizontal y selectivo de acuerdo al cuello de botella en cualquier momento dado
2. Alta especialización de cada uno de sus componentes, permitiendo usar las tecnologías que mejor se ajusten al trabajo que realicen
3. Sencillez en el diseño de cada uno de los componentes

Con esto en mente, se definieron cuatro servicios con funcionalidades bien definidas.

tix-time-server
Es el primer microservicio y la puerta de entrada al sistema. Su trabajo consiste en recibir paquetes, validar que sean paquetes íntegros del protocolo de TiX y devolverlos con las marcas de tiempo correspondientes. Además, si estos paquetes cuentan con reportes completos, enviarlos al siguiente servicio en la tubería de ingesta.

La decisión de implementarlo en Java sigue la idea que se explica anteriormente respecto de su gran performance. Es un lenguaje que cuenta con un amplio ecosistema de bibliotecas. Siendo Netty[26] una de ellas. Ésta permite la creación

de servidores de alta performance. Además, es una biblioteca que se encuentra disponible tanto para Java para Computadoras como para dispositivos móviles.

tix-time-condenser
Es el segundo microservicio del sistema. El objetivo de éste es validar que los mensajes de reportes sean correctos y pertenezcan a instalaciones activas de usuarios activos del sistema. Una vez hecha esta validación debe dejar los archivos de reportes en el directorio correspondiente al usuario y la instalación para su eventual procesamiento.

Este servicio también está implementado en Java, pero mediante el *framework* Spring Boot [27]. El cual permite la creación de micro-servicios de una forma sencilla y rápida, ofreciendo gran flexibilidad de configuración y un rico entorno para hacer pruebas. Todo esto sin dejar de ofrecer la performance natural de Java y una baja marca de tamaño en memoria.

tix-time-processor
Es el tercer y último microservicio del sistema. Fue creado con la idea de ser un servicio que corra de forma programada en un cierto periodo de tiempo. Toma los archivos de tal como los haya dejado el anterior servicio, para luego calcular las principales métricas para cada instalación.

La decisión de implementarlo en Python se debe a que este servicio cumple un rol más bien analítico y cercano a lo que es la Ciencia de Datos. Python cuenta con un ecosistema de bibliotecas y entornos de trabajo muy favorecido en este sentido, como lo son NumPy [28], SciPy [29], Matplotlib [30] y Scikit-learn [31]. Además, dado su rol especial, algunas pruebas del sistema no pueden ser ejecutadas de forma automática. Estas pruebas son las que se refieren al análisis y la mejora de la calidad en la estimación de las métricas. Para eso se tiene en cuenta el uso de Notebooks de Jupyter [32], un entorno de trabajo natural para la Ciencia de Datos y Python, principalmente para el análisis exploratorio de los datos.

Por último y no por ello menos relevante, el hacerlo en Python también cumple con la motivación de hacer un servicio sencillo y escalable mediante el uso de Celery [33]. Celery es un entorno de trabajo y una cola de trabajos implementada enteramente en Python. La misma cuenta con varios modos de uso. El elegido por nosotros es el modo programado, el cual se ejecuta de forma autónoma en un determinado período. De esta manera, se puede paralelizar el trabajo de procesar los datos en varios servicios replicados.

Cola de mensajes
Es el cuarto servicio del subsistema. Tiene la responsabilidad de comunicar el tix-time-server con el tix-time-condenser. También sirve como soporte en la comunicación del maestro y los esclavos de la cola de trabajos del tix-time-processor.

La decisión de usar el servicio RabbitMQ [34] para esto fue natural. Es una cola de mensajes con una gran cantidad de conectores para varios lenguajes, y

cuenta con funcionalidad directa con Celery y con Spring Boot. Además, es una cola de mensajes muy performante y fácil de administrar.

Como se puede ver, cada componente del Subsistema es autónomo en sí mismo e idempotente. Lo que confiere una gran escalabilidad horizontal.

A su vez, el que el tix-time-server sea sumamente sencillo y que las dos operaciones más pesadas sean la de procesar el paquete UDP como un paquete válido del protocolo TiX, y poner el paquete de reportes en la cola de mensajes, hace que el sistema entero tenga una gran capacidad de respuesta.

Subsistema Cliente
Para este subsistema, la consideración primaria se resume en que un único cliente se pueda usar en distintas plataformas, o por lo menos, que el código permita su reuso y adaptación simple para diversas plataformas. Al referirse a plataformas se piensa en computadoras, celulares, consolas, smartwatches, smart TVs, etc.

Tomando esto como punto de partida, se decide programar el cliente en lenguaje Java pensando en su gran ventaja: es un lenguaje multiplataforma. Existen intérpretes del lenguaje para casi todas las plataformas conocidas, incluyendo Windows, Linux, Mac y Android. El código resultante es independiente de la plataforma en la que terminará corriendo, ya que corre sobre la JVM (Java Virtual Machine) que se encarga de interpretar el conjunto de instrucciones al lenguaje de máquina donde se esté ejecutando el programa.

Otra consideración importante es que sea fácil de instalar para un usuario promedio, especialmente usuarios de sistemas operativos populares. Se decide no asumir que el usuario sabe usar una terminal de comandos, y proveer una interfaz visual y cotidiana para la instalación del cliente. Siguiendo el mismo proceso que cualquier otra aplicación, el usuario no tiene dificultades para comenzar a usar este software.

La tercera consideración tiene que ver con proveer una facilidad adicional al usuario: alimentar a la aplicación con actualizaciones de forma automática. Este proceso debe ser transparente para el usuario, es decir, no debe requerir que entre al sitio web de TiX para hacer descargas adicionales. Y tiene otra gran ventaja: asegura que un gran porcentaje de usuarios de TiX usan la versión más actualizada del cliente. Esto es muy útil para asegurarse de que cualquier cambio o agregado al software tendrá efecto rápidamente y los usuarios podrán beneficiarse de estos cambios.

Por último, y siguiendo en línea con los puntos anteriores, se toma la premisa de que el cliente sea lo más simple posible en cuanto a la interfaz gráfica y la interacción que requiere de parte del usuario. Lejos de interrumpir su trabajo diario, el cliente debe exigir datos de autenticación e instalación para una configuración inicial, y a partir de ese instante permanecer latente en la barra de estados.

2.2.2. Funcionamiento conjunto del sistema

Al igual que con la vieja arquitectura, el sistema actual comienza a funcionar tras la creación de la cuenta del usuario a través del servicio tix-web en el servicio tix-api, siendo el primero sólo una interfaz amigable del segundo. Luego de la descarga del cliente a su computadora, se crea la instalación y se comienza a enviar reportes al tix-time-server. Cuando el tix-time-server recibe un reporte, lo pone en la Cola de Mensajes para que sea tomado por el tix-time-condenser. Éste valida que el mensaje sea de una instalación activa de un usuario activo contra tix-api. Entonces chequea la integridad del reporte, para asegurar que no haya sido corrompido. Luego deja el mensaje en el sistema de archivos, en el directorio correspondiente al usuario y su instalación. Cuando el momento es correcto, el tix-time-processor toma todos los reportes de la instalación, con los que estima las métricas para esa ventana de tiempo y envía el dato a la tix-api. La misma enriquece los datos del reporte aportando el Sistema Autónomo y el Proveedor de Servicios Internet, cruzando la IP que figura en el reporte con la información que actualiza el servicio IP2AS. Así es que finalmente el usuario puede visualizar los resultados a través del servicio tix-web que consulta al servicio tix-api.

2.2.3. Fortelazas de esta arquitectura

Las principales fortalezas de esta arquitectura nacen de su naturaleza totalmente distribuida.

- Es una arquitectura altamente escalable, incluso dentro de un solo servidor.
- Es una arquitectura robusta, en el sentido que si uno de los componentes del subsistema se cae, el servicio se vería degradado pero no necesariamente interrumpido.
- El uso de tecnologías idóneas para cada servicio en línea con las tendencias actuales y las mejores prácticas presentes.
- La gran performance aparente, consecuencia de poder elegir lo que mejor se comporta en cada caso.
- Cada servicio es medible en sí mismo como una caja negra, permitiendo identificar cuellos de botella de forma preventiva mediante el uso de pruebas de carga aisladas.
- El tener un servicio central con una interfaz estándar como la tix-api, confiere la posibilidad de hacer clientes e interfaces amigables al usuario de una forma más sencilla y rápida además de una excelente integración con otros sistemas.

2.2.4. Debilidades y problemas de esta arquitectura

Naturalmente, las debilidades y problemas también nacen de la forma distribuida de la arquitectura.

Las modificaciones entre Subsistemas (e incluso entre servicios) debido a un cambio en el protocolo de comportamiento entre sí, pueden llevar a tiempos de desarrollo muchísimo más extensos que con otras arquitecturas. Además, requiere de pruebas y ensayos mucho más severos que con una arquitectura monolítica.

Los errores espontáneos o en tiempo de ejecución son mucho más difíciles de encontrar que en una aplicación monolítica. Esto se debe a la distribución de la bitácora de ejecución a lo largo de todo el sistema, y no en un solo lugar.

2.3. Conclusiones de la arquitectura actual

Al principio de la sección previa se declaran tres pilares sobre los cuales se basa esta iteración del proyecto, que buscan satisfacer los objetivos y motivaciones planteadas en la Introducción. Estos son:

- El Proyecto tiene que poder ser encarado y progresado por terceros ajenos al desarrollo inicial y con escasa o nula ayuda.
- El Sistema debe ser fácil de medir, para poder ser mejorado de manera contínua.
- El Sistema debe ser robusto, confiable y performante, para cumplimentar con los objetivos del presente Proyecto de Grado.

Con expuesto, es evidente que la elección de tecnologías de apoyo y operaciones están orientadas a satisfacer el primer punto. En tanto que el cambio de arquitectura y la distribución de servicios estan orientados a satisfacer los últimos dos puntos.

Referencias

[1] J. Postel, "User datagram protocol," Internet Engineering Task Force, Tech. Rep., aug 1980. [Online]. Available: https://www.ietf.org/rfc/rfc768.txt

[2] D. Mills, "Network time protocol (NTP)," Internet Engineering Task Force, Tech. Rep., sep 1985. [Online]. Available: https://tools.ietf.org/html/rfc958

[3] M. E. Crovella and A. Bestavros, "Self-similarity in world wide web traffic: evidence and possible causes," *IEEE/ACM Transactions on networking*, vol. 5, no. 6, pp. 835–846, 1997.

[4] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, "On the self-similar nature of ethernet traffic (extended version)," *IEEE/ACM Transactions on networking*, vol. 2, no. 1, pp. 1–15, 1994.

[5] Oracle, "Java Language Documentation." [Online]. Available: https://docs.oracle.com/en/java/

[6] R. Johnson, J. Hoeller, K. Donald, C. Sampaleanu, R. Harrop, A. Arendsen, T. Risberg, D. Davison, D. Kopylenko, M. Pollack, T. Templier, E. Vervaet, P. Tung, B. Hale, A. Colyer, J. Lewis, C. Leau, M. Fisher, S. Brannen, R. Laddad, A. Poutsma, C. Beams, T. Abedrabbo, A. Clement, D. Syer, O. Gierke, and R. Stoyanchev, "Spring Framework Documentation." [Online]. Available: https://docs.spring.io/spring/docs/3.1.1.RELEASE/spring-framework-reference/html/

[7] Red Hat JBoss Middleware, "Hibernate ORM Documentation." [Online]. Available: http://hibernate.org/orm/documentation/

[8] Apache Software Foundation, "Apache Wicket Framework Documentation." [Online]. Available: https://ci.apache.org/projects/wicket/apidocs/1.5.x/index.html

[9] R. Mordani, Oracle, N. Abramson, Apache Software Foundation Art Technology Group Inc.(ATG), K. Avedal, BEA Systems, H. Bergsten, Boeing, Borland Software Corporation, Developmentor, J. Hunter, IBM, InterX PLC, R. Johnson, Lutris Technologies, New Atlanta Communications, LLC, Novell, Inc., Persistence Software, Inc., Pramati Technologies Progress Software, SAS Institute, Inc., Sun Microsystems, Inc., Sybase, and G. Wilkins, "JavaTM Servlet 2.4 Specification," Java Community Process, techreport 154, Nov. 2003. [Online]. Available: https://jcp.org/en/jsr/detail?id=154

[10] Python Software Foundation, "Python 2 Language Documentation." [Online]. Available: https://docs.python.org/2/

[11] R Development Core Team, "The R Language Manuals." [Online]. Available: https://cran.r-project.org/manuals.html

[12] J. Buck and L. Hambley, “Capistrano Documentation.” [Online]. Available: http://capistranorb.com/

[13] Y. Matsumoto, “Ruby Language Documentation.” [Online]. Available: https://www.ruby-lang.org/en/documentation/

[14] The Open Group and IEEE Std, “crontab,” The Open Group, techreport, 2001. [Online]. Available: http://pubs.opengroup.org/onlinepubs/9699919799/utilities/crontab.html

[15] R. T. Fielding, “Architectural styles and the design of network-based software architectures,” phdthesis, University of California, 2000. [Online]. Available: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[16] Travis-CI Community, “Travis-CI.” [Online]. Available: http://travis-ci.org/

[17] Docker, Inc., “Docker.” [Online]. Available: https://www.docker.com/

[18] ——, “Dockerhub.” [Online]. Available: https://hub.docker.com/

[19] ——, “Docker Compose Documentation.” [Online]. Available: https://docs.docker.com/compose/

[20] Apache Software Foundation, “Apache Maven.” [Online]. Available: https://maven.apache.org/

[21] T. Lindholm, F. Yellin, G. Bracha, and A. Buckley, *The Java Virtual Machine Specification, Java SE 8 Edition*, A.-W. Professional, Ed. Addison-Wesley Professional, 2014. [Online]. Available: https://docs.oracle.com/javase/specs/jvms/se8/html/index.html

[22] Facebook, Instagram, and Community, “React framework (reactjs).” [Online]. Available: https://reactjs.org/

[23] T. Koppers, S. Larkin, J. Ewald, J. VepsÃ€lÃ€inen, K. Kluskens, and W. contributors, “Webpack.” [Online]. Available: https://webpack.js.org/

[24] M. Jones, J. Bradley, and N. Sakimura, “JSON web token (JWT),” IETF, Tech. Rep., may 2015. [Online]. Available: https://tools.ietf.org/html/rfc7519

[25] E. R. Fielding and E. J. Reschke, “Hypertext transfer protocol (HTTP/1.1): Authentication,” IETF, Tech. Rep., jun 2014. [Online]. Available: https://tools.ietf.org/html/rfc7235

[26] Netty Project Community, “Netty.” [Online]. Available: https://netty.io/

[27] P. Webb, D. Syer, J. Long, S. Nicoll, R. Winch, A. Wilkinson, M. Overdijk, C. Dupuis, S. Deleuze, and M. Simons, “Spring Boot Documentation,” 2017. [Online]. Available: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/

[28] The Scipy Community, "NumPy Reference Guide." [Online]. Available: https://docs.scipy.org/doc/numpy-1.13.0/reference/

[29] ——, "Scipy reference guide." [Online]. Available: https://docs.scipy.org/doc/scipy-1.0.0/reference/

[30] J. D. Hunter, M. Droettboom, T. A. Caswell, and The Matplotlib Community, "Matplotlib user's guide." [Online]. Available: http://matplotlib.org/users/index.html

[31] The scikit-learn Community, "scikit-learn documentation." [Online]. Available: http://scikit-learn.org/stable/documentation.html

[32] Jupyter Team, "Jupyter documentation." [Online]. Available: https://jupyter.readthedocs.io/en/latest/contents.html

[33] A. Solem, "Celery user manual." [Online]. Available: http://docs.celeryproject.org/en/latest/

[34] Pivotal, "Rabbitmq documentation." [Online]. Available: https://www.rabbitmq.com/documentation.html

[35] J. Postel, "Transmission control protocol," DARPA Internet Program, Tech. Rep., sep 1981. [Online]. Available: https://tools.ietf.org/html/rfc793

[36] J. Nagle, "Congestion control in IP/TCP internetworks," Internet Engineering Task Force, Tech. Rep., jan 1984. [Online]. Available: https://tools.ietf.org/html/rfc896

[37] F. Yergeau, "UTF-8, a transformation format of ISO 10646," Internet Engineering Task Force, Tech. Rep., nov 2003. [Online]. Available: https://tools.ietf.org/html/rfc3629

[38] R. L. Rivest, A. Shamir, and L. M. Adleman, "Cryptographic communications system and method," United States Patent Patent 4,405,829, 1983. [Online]. Available: http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PALL&p=1&u=%2Fnetahtml%2FPTO%2Fsrchnum.htm&r=1&f=G&l=50&s1=4405829.PN.&OS=PN/4405829&RS=PN/4405829

[39] M. Cooper, Y. Dzambasow, P. Hesse, S. Joseph, and R. Nicholas, "Internet x.509 public key infrastructure: Certification path building," Internet Engineering Task Force, Tech. Rep., sep 2005. [Online]. Available: https://tools.ietf.org/html/rfc4158

[40] Q. H. Dang, "Secure hash standard," National Institute Of Standards and Technology, Tech. Rep., jul 2015. [Online]. Available: http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf

[41] B. Kaliski, "PKCS #1: RSA encryption version 1.5," Internet Engineering Task Force, Tech. Rep., mar 1998. [Online]. Available: https://tools.ietf.org/html/rfc2437

[42] S. Aiyagari, A. Richardson, M. Arrott, M. Ritchie, M. Atwell, S. Sadjadi, J. Brome, R. Schloming, A. Conway, S. Shaw, R. Godfrey, M. Sustrik, R. Greig, C. Trieloff, P. Hintjens, K. van der Riet, J. O'Hara, S. Vinoski, and M. Radestock, "Advanced message queuing protocol," , techreport, 2008. [Online]. Available: https://www.rabbitmq.com/resources/specs/amqp0-9-1.pdf

[43] T. Bray, "The JavaScript object notation (JSON) data interchange format," Internet Engineering Task Force, Tech. Rep., mar 2014. [Online]. Available: https://tools.ietf.org/html/rfc7159

[44] F. Wasilewski, "Pywavelets documentation." [Online]. Available: https://pywavelets.readthedocs.io/en/latest/#license

[45] Oracle, "Javafx." [Online]. Available: http://www.oracle.com/technetwork/java/javase/overview/javafx-overview-2158620.html

[46] ——, "Swing." [Online]. Available: https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html

[47] ——, "Fxml." [Online]. Available: https://docs.oracle.com/javafx/2/api/javafx/fxml/doc-files/introduction_to_fxml.html

[48] E. Syse and F. contributors, "Fxlauncher." [Online]. Available: https://github.com/edvin/fxlauncher

[49] Azureus Software, Inc., "Vuze bittorrent client web page." [Online]. Available: https://www.vuze.com/

[50] H. Hope, "inxi documentation." [Online]. Available: https://smxi.org/docs/inxi-man.htm