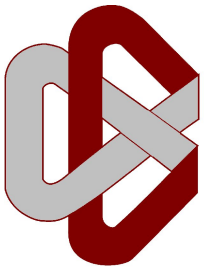


Centro de Investigación en Matemáticas, A.C.

CIMAT

Tesis: Aprovisionamiento automático de infraestructura y de configuración de la infraestructura. El caso Adblock Plus

TESIS

PARA OBTENER EL GRADO EN:
MAESTRÍA EN INGENIERÍA DE SOFTWARE

Presenta
Francisco Josué López Aguilar

Director de Tesis:
José Guadalupe Hernández Reveles

Zacatecas, Zacatecas., 07 de Diciembre de 2015

Índice

Índice de Figuras y Tablas

Abstract

En este documento de investigación se presenta una solución que involucra software de código abierto que permite utilizar las mejores prácticas del desarrollo de software aplicadas al mantenimiento, configuración y administración de la infraestructura. Dicha solución es aplicada en Adblock Plus, la extensión más descargada de internet con más de 350 millones de descargas y 60 millones de usuarios activos al mes, siendo un producto usado por millones de personas es necesario contar con una infraestructura que permita escalar rápidamente así como modificarla de acuerdo a los constantes cambios que se presentan.

Palabras clave: aprovisionamiento automático, infraestructura desechable, configuración de infraestructura, adblock plus, puppet, vagrant, hiera, visible ops, itil.

Capítulo 1. Introducción

> *"Facilitar el cambio es más efectivo que intentar prevenirlo. Aprende a confiar en tu habilidad para responder a cambios imprevisibles; esto es más importante que aprender a confiar en tu habilidad de planear contra el desastre"*
>
> - Manifiesto Ágil. Fowler, M., & Highsmith, J. (2001)

A lo largo de la historia se han desarrollado diferentes metodologías para el desarrollo de software, en las cuales se proponen prácticas para controlar y administrar el desarrollo del mismo, desde metodologías como el desarrollo en cascada (Center for Technology in Government, 1998), hasta algunos más recientes como Scrum o programación extrema (Banerjee, 2012), estos último se centran en adaptarse al cambio en lugar de intentar evitarlo.

Una práctica sugerida para contar con un ambiente de desarrollo ágil, por ejemplo al utilizar Scrum, Kanban o programación extrema, es utilizar entornos limitados durante todas las fases de la construcción del software. (Ambler, S.)

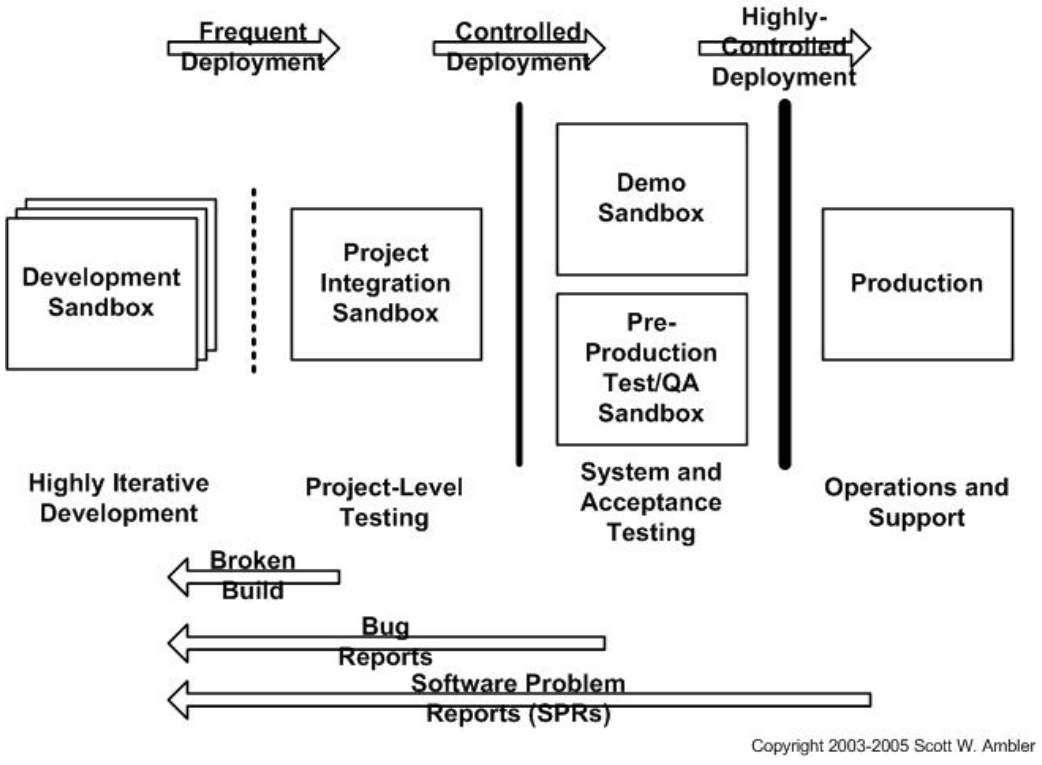


Figura 1. Entornos limitados en las fases de desarrollo

Pero así como se ha evolucionado en prácticas de desarrollo, se debe de evolucionar en prácticas de aprovisionamiento de la infraestructura, pues en el desarrollo de software como en la configuración de la infraestructura del mismo, se encuentran diferencias en los diferentes ambientes con configuraciones que no coinciden y generan retrabajo (Duvall et al.,2007).

Sin embargo, algunas preguntas surgen a la hora de planear utilizar dichas metodologías e incorporarlas a una nueva manera de trabajo para la configuración y administración de la infraestructura, por lo que se proponen los siguientes **objetivos de investigación**:

- Aprovisionamiento tradicional
    - ¿Como se hace el aprovisionamiento de infraestructura en la actualidad?, ¿Qué características tiene la forma actual?
    - ¿Como se puede mejorar?
- Aprovisionamiento automático
    - ¿Qué están haciendo las empresas en la actualidad?
    - ¿Qué ventajas tiene hacer el aprovisionamiento de esta manera?
- Métricas
    - ¿Como se puede medir el caos en la configuración de la infraestructura?
    - ¿Qué métricas se utilizan para medir la efectividad y calidad del aprovisionamiento?
- ¿Existe un método documentado para lograr el aprovisionamiento automático que sea fácil de seguir para empresas que quieran implementarlo?

Existen diferentes soluciones a la hora de crear entornos limitados, que se abordarán más adelante, sin embargo, la respuesta a las tres últimas preguntas es sencilla: infraestructura desechable.

De acuerdo con Fowler (2013), la configuración de los entornos puede resultar caótica, pues se pueden tener servidores que han estado dando servicio durante años y que han sufrido actualizaciones de los sistemas, y que en cierto punto, se ha perdido el control de la configuración, esto no sólo es cierto para desarrolladores pero también para administradores de sistemas. La infraestructura desechable permite generar entornos de desarrollo que son fácilmente replicables y tienen una configuración estandarizada, utilizando herramientas de administración de la configuración que permitan hacer inmutable la forma en la que se configurará dicho servidor. (Fowler, 2013)

Diversas empresas como Google, Facebook y Netflix cuentan con los recursos y personal especializado para implementar infraestructura desechable, aunque no se encontró documentación de que dicha infraestructura sea la misma para las diferentes fases de desarrollo. En este trabajo de investigación se presenta el caso de Adblock Plus, la cuál es la extensión para navegadores web más descargada de Internet; se eligió Adblock Plus debido a que cuenta con repositorios de código abierto, y por lo tanto, se puede apreciar cómo están usando en la práctica la infraestructura desechable tanto en desarrollo local, fase de pruebas y producción. Como valor agregado al desarrollo de este trabajo cabe destacar que el autor del mismo realizó una estancia académica de seis meses en Alemania en la sede de EyeO GmbH, la empresa detrás del desarrollo de Adblock Plus.

Adblock Plus es una extensión para navegadores que permite bloquear publicidad intrusiva, además ayuda a navegar de forma más segura por internet, evita el rastreo de los usuarios y bloquea algunos virus potenciales. La extensión cuenta con más de 300 millones de descargas en el mundo y tiene más de 60 millones de usuarios activos (Adblock Plus, 2015).

El principal problema al que se enfrentan los responsables de infraestructura consiste en la necesidad de crear al menos dos entornos, uno para desarrollo y otro para producción, y además que estos entornos que puedan sincronizarse de una manera automática. Las diferentes fases de desarrollo requieren entornos similares, tanto en software como en hardware, pero existe una brecha conceptual muy grande de las características de las configuraciones que existen en los requerimientos y lo que se implementa en la realidad pues los desarrolladores tienen un ambiente de trabajo distinto al que se utiliza en producción, actualmente dicha brecha es cubierta de manera manual lo que provoca errores y configuraciones distintas en los diferentes entornos que se tienen durante todo el ciclo de desarrollo de software. (Narain, Levin & Kaul, 2008).

De acuerdo con Baart (2011) el desplegar código de ambientes de desarrollo a producción tiene costos ocultos, un desarrollador pasa aproximadamente 200 horas volviendo a hacer el despliegue del código a producción, lo que se traduce a 12,000 dolares empleados en esta tarea por año, costos que podrían reducirse teniendo mejor control de la configuración de la infraestructura, es decir, actualizaciones e instalaciones de dependencias, altos costos para introducir nuevos empleados a la configuración actual así como para probar los cambios en un ambiente de pruebas o producción, entre otros.

Por lo que se plantea la siguiente hipótesis para solucionar el siguiente problema:

Utilizar la misma infraestructura desechable para todas las fases de desarrollo; pues la infraestructura debe ser fácilmente auditable y replicable, ya que, cumpliendo con estas características de calidad, dicha infraestructura será también mantenible y escalable.

Aunque se escuche como algo obvio y natural, en la práctica, es algo más complicado de lo que parece. (Behr, Kim, & Spafford, 2005)

Esta tesis está dirigida para las personas que necesiten proveer infraestructura para los proyectos en los que están involucrados, Administradores de Sistemas, DevOps o a cualquier persona que le pueda resultar útil esta información, si alguna vez se han encontrado con problemas como los antes mencionados.

En los siguientes capítulos se explicará con más detalle la práctica de la infraestructura desechable y el rol que juega en otras metodologías como Continuous Integration, Continuous Delivery y Continuous Deployment, también se hablará del experimento realizado en Adblock Plus y los resultados obtenidos de dicho experimento dando resultados y conclusiones importantes para el trabajo realizado.

Capítulo 2. Estado del arte del aprovisionamiento de infraestructura de servidores

2.1 Aprovisionamiento tradicional

Un sistema de software es un sistema de componentes que se intercomunican entre sí, un "sistema es una colección de componentes organizados para cumplir con una función específica o un conjunto de funciones. El término sistema engloba aplicaciones individuales, sistemas en el sentido tradicional, sistemas de sistemas, líneas de productos, familias de productos, empresas enteras y otros agregados de interés. Un sistema existe para satisfacer una o más misiones en su ambiente" (Eeles, P. 2006)

Así como en el desarrollo de software se han creado e implementado diferentes metodologías para crear sistemas, en el área encargada de las operaciones de los mismos, se cuentan con metodologías tradicionales que permiten tener un control de cómo está evolucionando la infraestructura, la práctica más común es conocida como Desarrollo-Pruebas-Producción, DSP por sus siglas en inglés. (Sabanin, 2012)

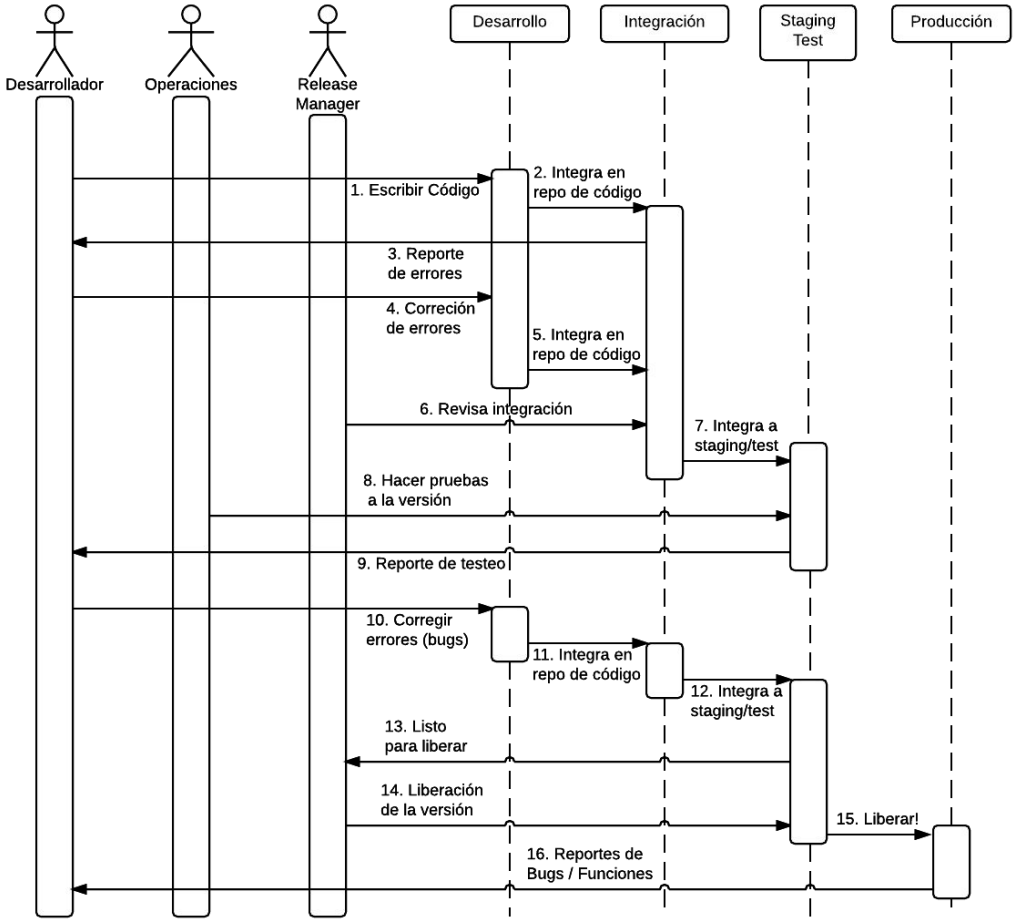


Figura 2. Ciclo de desarrollo. Fuente: Murray, P. (2006)

En el diagrama anterior se puede apreciar cómo se hace el desarrollo de software junto con el desarrollo de la infraestructura, en una narrativa sería la siguiente:

El desarrollador escribe código de acuerdo a los requerimientos que se tienen, esto normalmente sucede en el equipo en el que trabaja el desarrollador y suele ser de forma

aislada del resto del equipo en caso de haber uno (1), cuando el desarrollador considera que el código desarrollado cumple con las características necesarias, se integra en un repositorio de código, y esto puede suceder en el mismo equipo en el que se trabaja o en algún contenedor de código para todo el equipo (2), en caso de haber errores se le reportan al desarrollador (3) el cual los corrige (4) y los vuelve a integrar (5), así el encargado de hacer la liberación del código lo revisa (6) e integra a los servidores de pruebas/pre-producción (7), donde el encargado de operaciones se encarga de revisar que los cambios realizados al código funcionen de manera correcta (8) y reporta cualquier problema que se haya encontrado (9), para que el desarrollador corrija esto (10), lo vuelva a integrar (11) para que a su vez se libere a staging/testing (12), teniendo una versión lista para su liberación a producción (13) del cual el encargado de las liberaciones coloca dicha versión en pruebas/pre-producción (14) para a su vez liberar a producción (15) y por último reportar errores o nuevas características deseadas al desarrollador (16) volviendo a iniciar el ciclo. (Murray, 2006)

Algunas consideraciones se tienen que hacer con este modelo son las siguientes:

1. Los actores descritos pueden ser diferentes personas o la misma.
2. Las fases descritas pueden dividirse como se ve en el diagrama o puede ser que las primeras tres, desarrollo, integración y pruebas/pre-producción, estén en el mismo equipo de cómputo mientras que producción opere de forma aislada.

De igual manera se presentan algunas características importantes, que pueden o no ser obvias a simple vista (Duvall et al, 2007):

1. En la fase de producción se puede tener configuraciones diferentes a las otras tres fases, puesto que opera de manera independiente.
2. El desarrollo hecho puede ser modificado varias veces hasta que éste funcione en producción.
3. El código desarrollado puede variar desde su primer desarrollo hasta que éste alcanza la fase de producción.
4. El código funciona en desarrollo pero no en producción.
5. La configuración es diferente entre los ambientes de desarrollo, pruebas y producción.
6. Diferentes versiones de las dependencias existen entre los entornos.

La principal área de oportunidad que encuentran los responsables de la administración de la infraestructura es unificar el proceso de desarrollo, para de esta manera contar con ambientes

fácilmente replicables que permitan tener una consistencia tanto en desarrollo como en producción y todas las fases intermedias, para así evitar diferencias en las configuraciones de los diferentes ambientes y por consiguiente ahorrar dinero y esfuerzo a la hora de pasar el código desarrollado entre una fase y otra (Narain, Levin & Kaul, 2008).

Es por esto que se han propuesto prácticas para tener menos dificultades: Continuous Integration, Continuous Delivery y Continuous Deployment.

“Continuous Integration es la práctica que permite a los miembros de un equipo integrar el código desarrollado continuamente, usualmente cada miembro integra por lo menos una vez al día obteniendo múltiples integraciones diariamente.” (Fowler, M. 2006) Con esta práctica se pueden encontrar errores en etapas tempranas del desarrollo de software, disminuyendo costos y haciendo más fácil el corregir problemas.

Continuous Delivery es la práctica que permite publicar código a producción en cualquier momento que se desee y Continuous Deployment es la práctica de publicar el desarrollo a producción automáticamente una vez que dicho desarrollo haya pasado todos los requisitos necesarios. (Humble, J., & Farley, D., 2007)

2.2 Mejores prácticas para aprovisionamiento de infraestructura

Este apartado se centra en algunas prácticas de Continuous Integration debido a que es el fundamento de las otras dos prácticas antes descritas.

Cuando se planea tener un ambiente de Continuous Integration, se debe de tener en cuenta que se está hablando tanto del desarrollo de software, como de las configuraciones necesarias de infraestructura, hardware y sistemas operativos.

Existen ciertos requisitos para tener un ambiente de Continuous Integration, según Martin Fowler (2006):

- Mantener un único repositorio de código
- Automatizar la construcción del software
- Automatizar las pruebas
- Publicar el código al repositorio central
- Corregir errores rápidamente
- Hacer pruebas en un entorno local
- Hacer sencillo que todo el equipo cuente con el mismo entorno

- Entre otras

Dichas prácticas tienen algo en común: un entorno local, tanto como para hacer pruebas, como para corregir errores rápidamente, Continuous Integration nos pide que exista una manera de tener todo el desarrollo disponible para todos los miembros del equipo en un ambiente local, de pruebas y producción, idealmente dichos ambientes tendrán las mismas configuraciones entre sí, pero incluso siendo diferentes, es importante contar con la configuración específica de cada uno de ellos.

2.2.1 Infraestructura Desechable

Autores como Kief Morris(2013), Barney Hanlon(2015), Chad Fowler (2013) y Gene Kim(2005), mencionan que a la hora de generar infraestructura que contenga las configuraciones necesarias para los ambientes de producción, es más barato, en términos de tiempo y esfuerzo, crear nueva infraestructura que actualizar la que ya existe.

Cuando se hace una actualización se corre el riesgo de cometer errores, generando entropía dentro del ambiente de producción, por lo que es mejor tener alguna manera de hacer este proceso de manera automática, quitando de la ecuación el factor humano.

La infraestructura desechable es parte primordial para tener un ambiente de Continuous Integration y por consiguiente de Continuous Delivery y Continuous Deployment, pues se tiene la seguridad, cuando se es ejecutado correctamente, de tener una infraestructura, es decir, un conjunto de sistemas operativos con software de utilidad, idempotente que permita trabajar o publicar el código en un ambiente controlado.

2.3 Herramientas para aprovisionar un ambiente de Infraestructura Desechable

Para aprovisionar un ambiente de Infraestructura Desechable es necesario tomar en cuenta dos aspectos, el sistema operativo y el software de aplicación que se utilizará, el primero es la base para permitir que el segundo pueda funcionar correctamente dentro del entorno deseado (Eeles, P. 2006).

Esto no es sólo necesario para Continuous Integration sino para todos los sistemas de software, por lo que es importante, a fin de contar con un ambiente de desarrollo y producción estable, definir cómo lucirán dichos entornos, por lo que se deben definir ambos aspectos.

2.3.1 Herramientas para Sistemas Operativos:

Existen diversas herramientas que permiten virtualizar sistemas operativos, teniendo la ventaja de que se utiliza un ambiente aislado del resto de los componentes del sistema, las más utilizadas son:

- VirtualBox(2011): Es una herramienta de código abierto bajo la licencia GPL v2, es utilizada de forma empresarial como de uso doméstico y puede virtualizar arquitecturas de hardware de 32 y 64 bits, funciona en Linux, Windows, Mac OS y Solaris
- VMWare(2015): Es una herramienta de software propietario que permite la virtualización de arquitecturas de 32 y 64 bits, funciona en Linux, Windows y Mac OS

Así mismo, es importante administrar la configuración de los sistemas operativos, por lo que también se pueden encontrar herramientas que cumplen con esta función, las herramientas más usadas para administrar la configuración son las siguientes:

- Puppet(2011): es una herramienta de la administración de la configuración que permite definir el estado de los diferentes nodos del sistema, aplicando automáticamente dicha configuración a los servidores proveyendo de una infraestructura de tecnologías en ambiente definido de antemano, dicha configuración es definida como “manifiestos”.
- Chef(2014): Es definido como “infraestructura como código”, permitiendo automatizar las tareas de configuración y administración del estado deseado de los servidores, la configuración es almacenada en archivos llamados “recetas” que definen el estatus deseado de la configuración de los servidores.
- Ansible(2015): es un poderoso motor de automatización que permite definir, de manera sencilla y sin código base extra en los nodos, la configuración deseada para cada uno de ellos, se caracteriza por utilizar un lenguaje sencillo que permite manejar los servidores en un lenguaje fácil de aprender.

Otro punto importante a considerar es la forma en la que se puede empaquetar ambas partes, tanto los sistemas operativos, como las herramientas de administración de la configuración, dicha encapsulación, es necesaria para el ambiente de desarrollo, y así, evitar utilizar la configuración del equipo de desarrollo que puede generar diferentes resultados, por lo que es recomendado utilizar máquinas virtuales que permitan simular un ambiente de producción o pruebas, en los ambientes de pruebas y producción, es mejor utilizar servidores privados virtuales o servidores dedicados, ya que, funcionan como una máquina virtual dentro del equipo del desarrollador, pero con más poder de cómputo.

Para lograr el empaquetamiento de los sistemas operativos con las herramientas de administración de la configuración, existen, de igual manera, soluciones que hacen de este proceso algo sencillo:

- Vagrant(2013): es una herramienta que permite configurar y reproducir ambientes, tanto de desarrollo como de pruebas y producción, y que trabaja sobre las herramientas antes mencionadas.
- Docker(2004): Es una herramienta que hace las funciones de un contenedor, es decir, aísla completamente el sistema “dockerizado” del sistema operativo, mientras que en una máquina virtual, algunos procesos son compartidos entre el sistema operativo base y la máquina virtual; en esta herramienta se configura el estatus que se desea tener pero sin depender de una máquina virtual, pues trabaja sobre el mismo sistema huésped para así lograr tener un ambiente que funcione tanto en desarrollo local, como en pruebas y producción.

La combinación de las herramientas anteriores permite flexibilidad a la hora de elegir la que mejor se acomode a las necesidades tanto tecnológicas como de personal capacitado para su uso.

2.4 ¿Qué se está haciendo en la industria?

A la hora de utilizar herramientas de la administración de la configuración, las empresas eligen entre las que se mencionaron anteriormente:

- Puppet: Empresas como Spotify, Sony o Rackspace utilizan Puppet para administrar la configuración de sus servidores, sin embargo no es posible encontrar la configuración completa de dichas empresas, ya que su código es propietario y esta información es basada en los clientes que aparecen en la página de Puppet

- Chef: Empresas como IndieGoGo, CheezBurguer o Facebook utilizan Puppet, debido a la naturaleza propietaria de su código, no se tiene acceso a las configuraciones utilizadas, pero se puede corroborar la información en la página de clientes de Chef

- Ansible: Siendo relativamente nuevo en el mercado, cuenta con clientes como Apple o la NASA, de igual manera, por la naturaleza de su código cerrado no es posible ver las configuraciones de dichas empresas

- Vagrant: Es usado por empresas como Disqus, O'Reilly, Nokia, entre otros, en este caso, se usa de manera local pues los mismo autores de la herramienta no recomiendan utilizarlo en producción aún.

- Docker: Es utilizado por Yelp, Spotify, entre otros, siendo código propietario, no se tiene acceso a ejemplos de cómo se está implementado la solución.

Como se puede apreciar, grandes empresas alrededor del mundo están siguiendo algunas prácticas de implementar Infraestructura Desechable, sin embargo, no se tienen ejemplos de cómo lo están haciendo, y no se ha encontrado documentación clara de cómo crean ambientes de desarrollo local, de pruebas o de producción.

Aún así, se pueden encontrar personas que han decidido utilizar ambientes virtuales para hacer desarrollo local, utilizando las herramientas como VirtualBox o VMware, o utilizando Vagrant o Docker.

2.5 ¿Qué métricas nos permiten saber si el aprovisionamiento de infraestructura es eficiente?

Para saber si la infraestructura utilizada y la forma de aprovisionamiento es eficiente y eficaz, Behr, K., Kim, G., & Spafford, G. (2005) proponen una serie de métricas a fin de medir qué tan competente es una organización cuando a infraestructura se refiere, se eligieron las que son relevantes para aprovisionamiento:

2.5.1 Métricas de liberación de código:

- Tiempo para aprovisionar versiones estables conocidas desde cero
- Tiempo en que una versión está en producción antes de ser reemplazada.

- Porcentaje de sistemas que coinciden con las versiones aceptadas.

2.5.2 Métricas de control:

- Número de cambios por semana
- Número de cambios exitosos
- Número de veces que se dejó de dar servicio por mal funcionamiento.
- Proporción de cambios hechos contra cambios revisados

2.5.3 Métricas de resolución de problemas

- Tiempo medio de reparación(MTTR, por sus siglas en inglés)
- Tiempo medio entre fallas (MTBF, por sus siglas en inglés)

2.6 ¿Qué documentación existe para realizar infraestructura desechable?

Después de la investigación que se hizo para la realización de esta tesis, se descubrió que no existe un método completo o documentado para la automatización del aprovisionamiento de la infraestructura que sea replicable para los diferentes ambientes antes mencionado: desarrollo, testing/staging y producción.

Sin embargo, se puede hacer una recopilación de diferentes textos que describen las mejores prácticas, como es el caso de “The visible ops”, que describe los pasos para la implementación de ITIL en cuatro pasos: 1) Estabilizar al paciente, 2) Encontrar elementos frágiles, 3) Establecer una librería repetible y 4) Establecer la mejora continua. En el caso de esta tesis, se centra específicamente en el tercer paso pero enfocado en la configuración de servidores para las diferentes etapas de desarrollo. (Behr, K. et al, 2005).

También se pueden encontrar libros de integración continua así como artículos que tratan el mismo tema; si bien es cierto que el contenido de esta tesis no es precisamente la integración continua y sus derivados, es importante entender estos conceptos debido a que van relacionados con las prácticas ágiles que se siguen para automatizar el aprovisionamiento de la configuración de la infraestructura. (Duvall, P., 2007).

El blog de uno de los gurús y pioneros del movimiento ágil, Martin Fowler, que además es co-fundador de ThoughtWorks, empresa que sigue, promociona y escribe sobre las prácticas

ágiles e integración continua, es uno de los recursos más abundantes y confiables sobre el tema. (Fowler, M., 2015)

Finalmente, existen diferentes blogs y páginas que hacen mención a las diferentes etapas que se describirán en los siguientes capítulos.

Capítulo 3. Planeación de la implementación de aprovisionamiento automático en Adblock Plus.

3.0 Qué es Adblock Plus y cómo está configurada su arquitectura de servidores

Adblock Plus es una extensión para navegadores que permite bloquear publicidad intrusiva, además ayuda a navegar de forma más segura por internet, evita el rastreo de los usuarios y bloquea algunos virus potenciales. La extensión cuenta con más de 300 millones de descargas en el mundo y tiene más de 60 millones de usuarios activos (Adblock Plus, 2015).

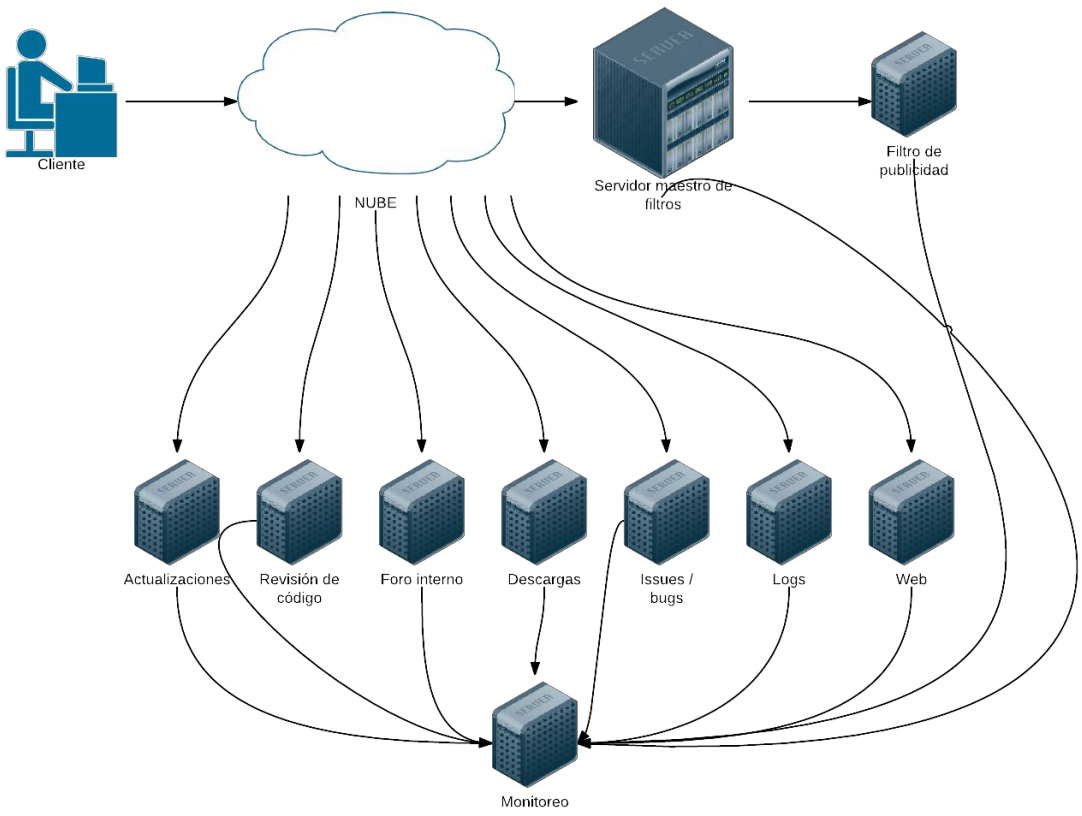


Figura 3.. Diagrama general de la infraestructura de Adblock Plus. Elaboración propia.

A continuación se detalla el funcionamiento general de cada uno de los servidores mencionados en el diagrama general de infraestructura:

- Servidor Maestro de Filtros que hace las veces de balanceador de cargas así como de recolector de información sobre los anuncios bloqueados.
- Servidores de Filtros de Publicidad. Detrás de este servidor se tienen servidores que hacen la función de filtros, es decir, la función principal de la extensión, que consiste en:
    - Verificar que la extensión no bloquee más de lo debido,
    - Verificar que la extensión esté actualizada así como sus listas de bloqueo y,
    - Generar logs de los anuncios bloqueados, los cuales son enviados posteriormente al “Servidor Principal de Logs”.
- Servidor Principal de Logs. Aquí se registran los logs de los filtros para su posterior análisis, cabe resaltar que dichos logs son manejados antes de ser enviados para proteger la privacidad de los usuarios.
- Servidor de Monitoreo. A este servidor están conectados todos los demás, tiene la función de reportar los estatus de salud general de todo los servidores.
- Servidor de Actualizaciones. Este servidor tiene la función de buscar y descargar actualizaciones para el software interno.
- Servidor de Revisión de Código. Este servidor tiene la función de revisar y mantener el código de la organización.
- Servidor de Foro Interno. En este servidor se administran los foros de discusiones internas donde se habla desde la organización de una fiesta o alguna reunión hasta el la generación de ideas para nuevas funcionalidades de la extensión o ideas para nuevos productos y servicios de empresa.
- Servidor de Descargas. Este servidor sirve para que los usuarios descarguen la extensión cuando no desean hacerlo desde las páginas oficiales de Apple, Android, etc.
- Servidor de Bugs/Issues. Aquí se registran problemas reportados sobre el funcionamiento de la extensión. Este servidor es público por lo que personas que incluso no tengan conocimientos técnicos pueden reportar problemas.

- Servidor Web. Aquí se tienen las paginas, tanto la de la extensión, la de la empresa en sí y un pequeño foro para usuarios que no son técnicos.

Todo esto se encuentra en producción pero para hacer pruebas no se tienen servidores dedicados a esta tarea, toda las pruebas se hacen en ambiente local para lo que se tiene un ambiente de desarrollo, el cual, desafortunadamente, no usan los desarrolladores.

3.0.1 Propiedad de módulos

La forma de trabajo del equipo de Adblock Plus se divide en módulos y se basa en la forma de trabajo de Mozilla; esta forma de trabajo está pensada para que cada parte del proyecto tenga un responsable que se encargará de: revisar y aprobar los cambios al módulo, aprobar o revocar acceso de escritura, mantener y mejorar la calidad del código, establecer la prioridad de los issues reportados así como ver que los mismos se corrijan.

Cada módulo puede tener o no una persona llamada “Super reviewer” el cual se encarga de revisar cambios a niveles más altos, es decir, cuando un cambio afecta a otros módulos, cuando se haran grandes refactorizaciones, etc.

3.1 El aprovisionamiento en Adblock Plus

Al contar con una infraestructura relativamente grande, Adblock Plus decidió utilizar un proceso definido para administrar la configuración de la misma, dicho proceso consistía en:

1. Identificar el cambio a realizar, utilizando el sistema de issues/bugs.
2. Generar la receta utilizando Puppet como software de la administración de la configuración.
3. Se revisaban los cambios de acuerdo al proceso de módulos antes descrito.
4. Desplegar el cambio a producción.

El responsable del desarrollo del módulo en ese momento era el líder de desarrollo y el CTO era el “super reviewer”, rol que autorizaba o rechaza el cambio.

Este proceso no se llevaba a cabo de manera constante, y no se seguía siempre debido a:

1. La carga de trabajo de los responsables.
2. El perfil de los responsables no era el mejor para desarrollar la posición.
3. Había problemas que se resolvían pero no se incluían en las recetas por la gravedad de los mismos.

Así mismo, las recetas desarrolladas no cumplian con todos los estándares de calidad por los problemas antes mencionados, sin embargo, se consiguió que se tuviera cierto control en la infraestructura debido a las ventajas que la herramienta ofrece.

La configuración dependía de los errores que encontraban y era cambiada lentamente pues las pruebas se intentaban hacer en desarrollo pero los cambios importantes tenían que hacerlos directamente en producción, ya que, al contar con ambientes diferentes, sólo se podían hacer cambios que no dependieran del ambiente en el que se encontraban, esto representa ciertas ventajas y áreas de oportunidad:

Ventajas	Áreas de oportunidad
● Cambios grandes podían revisarse antes de pasarlos a producción. ● Se tenía una idea de la infraestructura general.	● No se tenía certeza de que lo que funcionase en desarrollo sería igual en producción. ● Cambios importantes se tenían que revisar en producción. ● El código tenía que mantenerse manualmente en muchas de sus partes. ● Los desarrolladores no utilizaban la herramienta para probar antes de subir el código a producción

Tabla 1. Ventajas y desventajas del proceso inicial de aprovisionamiento de Adblock Plus.

A simple vista, esto fue algo que se descubrió al empezar a analizar la forma en que mantenían la infraestructura, sin embargo, y para tener mejores bases sobre las cuales encontrar los problemas más comunes, se recogieron requerimientos que se describen a continuación.

3.2 El proceso de aprovisionamiento automático propuesto

Basado en las ventajas y áreas de oportunidad que se mencionaron en las secciones anteriores, se propone una solución para la extensión Adblock Plus, la cual deberá contener los siguientes requerimientos, tanto funcionales como no funcionales, obtenidos por el líder de desarrollo y el CTO de la empresa.

ID	Nombre	Descripción	Prioridad	Flexibilidad	Verificabilidad	Necesario
RF-01	ENC	Utilizar un clasificador de nodos externo (ENC por sus siglas en inglés)	Alta	No	Fácil	Sí
RF-02	Sistema Operativo Desarrollo	El sistema operativo debe ser el mismo que el que se utiliza en el ambiente de producción	Media	Sí	Fácil	Sí
RF-03	Configuración	La configuración se deberá hacer en un administrador de la configuración, Puppet.	Media	No	Fácil	Sí
RF-04	Virtualización	Se debe tener un ambiente local virtualizado para el ambiente de desarrollo	Alta	No	Fácil	Sí
RF-05	Repositorios	Se deberán usar los repositorios de código oficiales de la empresa.	Alta	No	Fácil	Sí
RF-06	Privacidad	Se deberán mantener privadas las llaves y contraseñas de acceso de la empresa	Alta	No	Fácil	Sí
RF-07	Libre	El software utilizado deberá ser open source y permanecer así	Alta	No	Fácil	Sí
RNF-01	Pruebas	Se debe poder hacer pruebas de la configuración del software desarrollado	Alta	No	Difícil	Sí
RNF-02	Usabilidad	Debe ser fácil de usar para los desarrolladores	Alta	No	Fácil	No
RNF-03	Nodos	Deben poder agregar nuevos nodos (servidores) a la configuración de manera sencilla	Alta	No	Fácil	Sí
RNF-04	Portabilidad	Debe funcionar en cualquiera de los siguientes sistemas operativos: windows, mac, linux.	Alta	No	Fácil	Sí

Tabla 2. Requerimientos de Adblock Plus

ID	RF -> Requerimiento funcional RNF -> Requerimiento no funcional Seguido de un número
Nombre	Una palabra para identificar el requerimiento
Descripción	Explicación breve del requerimiento
Prioridad	Importancia del requerimiento (Alta, media, baja)
Flexibilidad	Posibilidad de que el requerimiento cambie (sí, no)
Verificabilidad	Facilidad de corroborar que el requerimiento existe (fácil, difícil)
Necesario	Necesidad de que el requerimiento exista (sí, no)

Tabla 3. Nomenclatura de Requerimientos. Elaboración propia

En resumen, con los requerimientos anteriores se deberá tener como resultado final 1) una infraestructura desechable 2) la cual funcione en cualquiera de los tres sistemas operativos mencionados anteriormente y 3) que replique el ambiente utilizado en producción, de esta forma los desarrolladores podrán verificar que el código desarrollado funcione y que los cambios en la configuración de la infraestructura sean revisados y aprobados antes de hacerse en producción, cumpliendo con las características encontradas en el estado del arte:

1. Utilizando las prácticas del aprovisionamiento tradicional así como
2. Las mejores prácticas detectadas para este propósito,
3. Utilizando las herramientas que la empresa ya emplea y que cumplen con el estado del arte descubierto y que a su vez,
4. Son herramientas que la industria utiliza y con las cuales se pueden obtener
5. Métricas que permitan comparar los resultados obtenidos.

3.3 Fases para la implementación de la primer solución

La solución propuesta consta de tres fases: centralizar la información de los nodos o servidores, integrar las configuraciones con su software de puppet y presentar la configuración al ambiente de producción.

Fase 1: Centralizar la información

La importancia de centralizar la información específica de los nodos reside en tener un control más sencillo de los servidores que existen en la infraestructura, pudiendo así tener una independencia entre la configuración que se les aplicará así como de la información específica de cada uno de ellos, de esta manera se podría cambiar el administrador de la configuración usado, agregar nodos a la red, auditar los nodos y su información y replicar la infraestructura en un ambiente más pequeño, todo de manera sencilla ya que dicha información se encontrará en un solo lugar.

Fase 2: Integración de la configuración

La configuración de cada uno de los nodos deberá ser independiente de su información específica, para de esta manera poder reutilizarla independientemente del ambiente en el que se encuentre, esto sirve para poder hacer pruebas de manera local en un solo servidor virtual aunque en producción existan dos o más servidores con la misma configuración.

Dicha configuración se puede ver como una base de datos de recetas que dictan cómo deben de lucir los servidores después de aplicar dichas recetas, así pues, un nodo que sea marcado como servidor web, por ejemplo, seguirá ciertas instrucciones hasta alcanzar un estado en el que coincida con la receta especificada.

Fase 3: Presentar la configuración

Una vez que la configuración de los servicios que pertenecen a cada servidor esté terminada y verificada en un ambiente de desarrollo o pruebas, ésta será presentada a los servidores de producción, los cuales no serán reconfigurados, sino que se desecharon los que tengan una configuración antigua para crear uno nuevo con la configuración nueva.

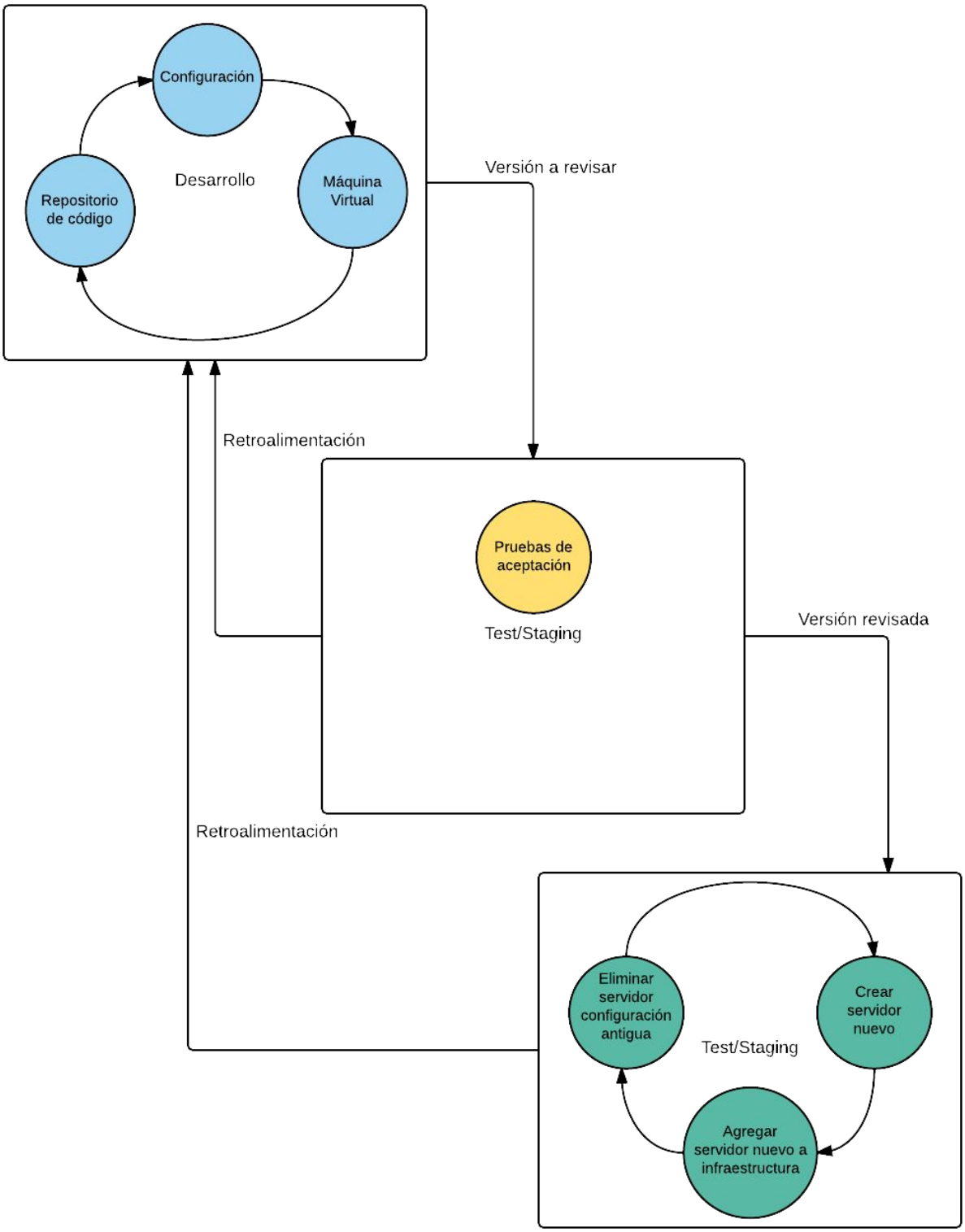


Figura 4. Ciclo de infraestructura desechable. Elaboración propia.

En el diagrama anterior se puede apreciar el ciclo propuesto para generar infraestructura desechable que permite el aprovisionamiento automático de ambientes, los cuales sirvan igual en desarrollo, staging/testing y producción.

La primer etapa, de color azul, se encuentra en el ambiente de desarrollo, donde se desarrollará y probará localmente la configuración deseada, toda la configuración y sus respectivos cambios deberá ser almacenada en repositorios de código para tener un control de lo que se está desarrollando y poder generar y cambiar fácilmente las versiones a utilizar, así mismo, se puede ver que la configuración se implementará en una máquina virtual para que las configuraciones propias del equipo de cómputo usado no interfieran con las configuraciones que se desean.

La segunda etapa, de color amarillo, puede ser una etapa opcional, ya que, esta fase puede estar en un diferente equipo o en el mismo que el de la etapa de desarrollo, si se encuentra en un lugar diferente que el equipo de desarrollo, normalmente las pruebas son más formales y es donde se ejecutan para comprobar que la configuración funcione en un ambiente más parecido al del de producción.

La tercer etapa, de color verde, es cuando se liberará la nueva versión de la configuración, en donde se crearán nuevas instancias de los nodos, en lugar de actualizar las existentes, así pues, agregando el nuevo nodo a la infraestructura para después eliminar el nodo con la configuración antigua.

En este capítulo se explicará de manera sencilla la solución desarrollada, para una explicación más a profundidad revisar el anexo A.

Tecnologías usadas

Las tecnologías empleadas son:

- Puppet, que es un sistema de administración de configuración, el cual permite definir cómo lucirá la infraestructura del producto, asegurando que el estado definido sea aplicado. https://puppetlabs.com/puppet/what-is-puppet
- Hiera, que es una herramienta de configuración de datos basado en la búsqueda “llave-valor” que permite almacenar toda la información de la configuración en un mismo

lugar, especificando los nodos de la infraestructura, siguiendo el principio “no te repitas” (don’t repeat yourself) . https://docs.puppetlabs.com/hiera/1/

- Vagrant, que es una herramienta utilizada para crear y configurar ambientes virtuales de desarrollo, que a su vez es independiente de sistema operativo, sistemas de administración de configuración y proveedores de virtualización. https://www.vagrantup.com/

Solución Propuesta

En el siguiente diagrama se presenta la forma en la que se hacía el aprovisionamiento local y en producción, después de la implementación ambos ambientes se aprovisionan de la misma manera.

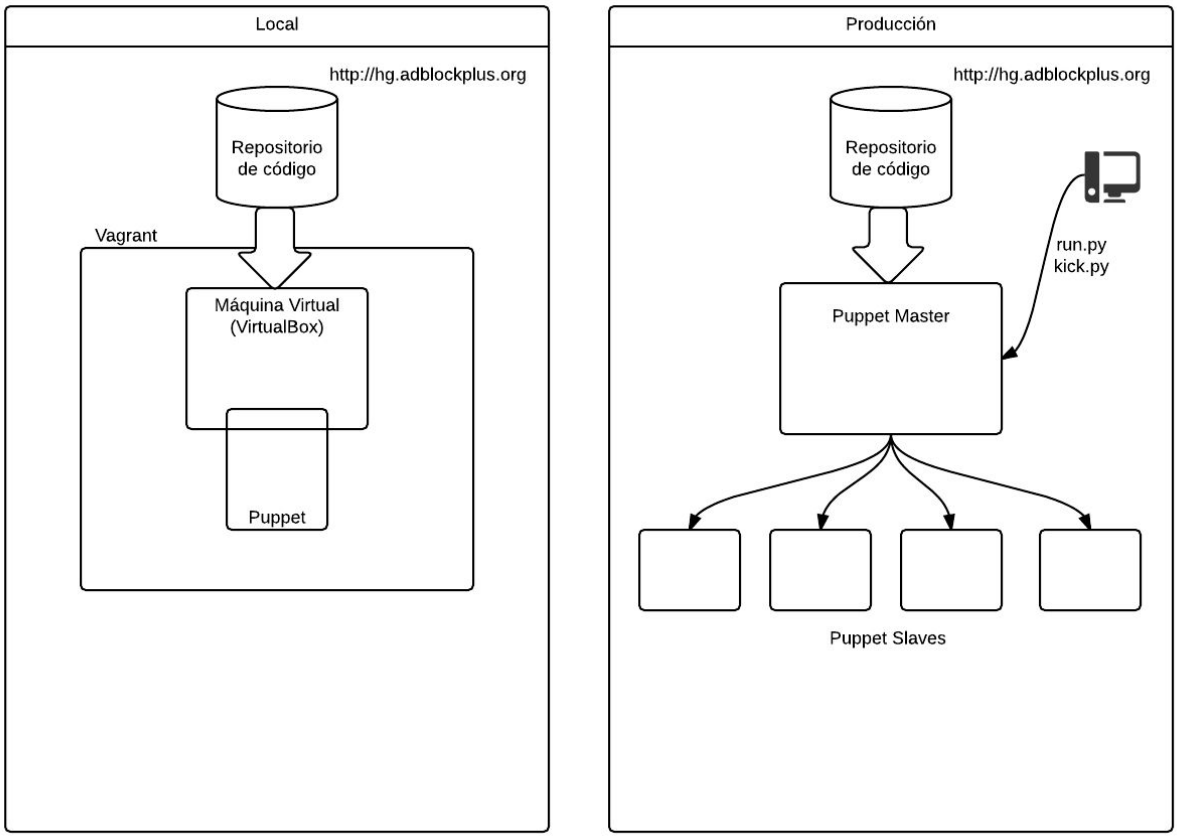


Figura 5. Aprovisionamiento base vs. aprovisionamiento automático. Elaboración propia

La solución propuesta consiste en utilizar las tecnologías antes mencionadas de la siguiente manera:

Primero se debe separar la configuración de los servicios de la información específica de cada nodo, es decir, si se va a aprovisionar un servidor con los servicios específicos que pueden ser el servicio web, el código que se utilizará y la base de datos, por ejemplo, y la información específica sería la dirección IP, el nombre del nodo, entre otra información que sólo le concierne a ese servidor específicamente, se debe tener en cuenta que un nodo o servidor puede tener la misma configuración de servicios pero nunca compartirán la información específica, este paso se hizo con Puppet debido a que permite especificar la configuración necesaria.

Después se debe utilizar un clasificador de nodos externo a la herramienta, el cual permitirá contener toda la información específica de cada uno de los nodos, toda esta información se debe poner en un mismo lugar para poder así escalar y mantener rápidamente la información, para este paso se utilizó Hiera que es la herramienta que tiene soporte nativo para Puppet.

Por último se debe asegurar que el aprovisionamiento sirva de igual manera en desarrollo como en producción, para esto se usó Vagrant, ya que, permite crear máquinas virtuales que tendrán la configuración creada con Puppet, de igual manera se permite utilizar Hiera para poder recrear la infraestructura de una manera similar a la que existe en producción, pues los nodos tienen características de hardware más pequeños y sólo existirá uno de cada uno.

El primer prototipo fue desarrollado de esta manera con *Puppet 3.1, Hiera 1.2 y Vagrant 1.7*, lo que permitió contar con todas las características deseadas, pudiendo hacer pruebas en desarrollo y con esto tener cierto grado de certeza de que en producción se comportaría de la misma manera, sin embargo, la empresa usa Puppet 2.7, versión que no tiene soporte nativo para Hiera, y por otro lado se necesita contar con un Puppet Master también en desarrollo, haciendo más complicado para los desarrolladores poder utilizar la solución.

El segundo prototipo y solución final fue utilizar Puppet 2.7, con una implementación que permite utilizar Hiera, pero que utiliza las funciones con una implementación hecha por el equipo para hacer el mapeo de las funciones de Hiera a Puppet 2.7, esto permitió no hacer la actualización de la versión ya que por políticas de la empresa esto no era posible, así mismo se utilizó una función de Vagrant para que el mismo software hiciera las veces de Puppet Master sin necesidad de utilizar una máquina virtual nueva.

Lo nuevo que se presenta con la solución implementada es que todos los ambientes tendrán la misma configuración, siendo fácilmente replicables bajo cualquier circunstancia y siguiendo las mejores prácticas tanto de infraestructura como de desarrollo de software, por lo que se cuenta con una solución de alta calidad.

Adicional a lo anterior, la solución final actualmente se encuentra en **producción y funcionando**, a la fecha del término del experimento se migró el 80% de la configuración, el cual cumple con todas y cada uno de los requerimientos listados anteriormente, ya que, permite hacer pruebas en desarrollo como primer paso y después de otros procesos que la empresa tiene, que no tienen nada que ver con el experimento, hacer la liberación de la nueva versión a los servidores de producción.

3.3 Evaluación de la eficiencia y eficacia del proceso de aprovisionamiento de Adblock Plus

En función de las métricas determinadas en el Estado del Arte se creó la siguiente tabla que incluye cada métrica y el valor actual antes de realizar el proceso de aprovisionamiento.

Métrica	Medida
Tiempo para establecer versiones conocidas desde cero	5 minutos aproximadamente por servidor (no cuentan con información precisa)
Tiempo que una versión está en producción antes de ser reemplazada	Una semana para roles que son dinámicos: Servidor ● Maestro de filtros ● Filtros ● Monitoreo ● Actualizaciones ● Revisión de código ● Foro interno ● Descargas

	● Bugs/Issues Cada seis meses para roles que son estáticos: Servidor ● Web ● Logs
Porcentaje de sistemas que coinciden con las versiones aceptadas	100%, todo se maneja con puppet
Número de cambios por semana	1*
Número de cambios exitosos	100%
Número de veces que se dejó de dar servicio por mal funcionamiento	Sitios estáticos: 95%** Sitios con balanceador de carga: 100%** Sitios dinámicos sin balanceador de cargas: 1.5 en promedio al mes.
Tiempo medio de reparación (MTTR)	Alta prioridad: 5 mins Media-Baja prioridad: 3 días
Tiempo medio entre fallas (MTBF)	6 meses

Tabla 4. Métricas de Adblock Plus

*Debido a que algunos datos no eran consistentes se realizó una muestra de los datos con un 95% de confianza y un error del 5%.
**Uptime.

Con las métricas antes mencionadas podemos ver que el caos es muy bajo en la mayoría de la infraestructura, sin embargo, en los servidores que son dinámicos y que no tienen un balanceador de cargas, existen muchos problemas por mes, lo que representa un trabajo relativamente constante, también podemos ver que resulta más sencillo aprovisionar un

servidor desde cero que intentar arreglarlo, práctica que es usada en Adblock Plus, pues el tiempo de aprovisionamiento de una versión conocida es de aproximadamente 5 minutos.

Para medir la efectividad y la calidad del aprovisionamiento podemos fijarnos en el tiempo medio entre fallas, el cual es de una falla cada 6 meses, también podemos medirlo con el número de veces que se deja de dar servicio, que en aproximadamente el 80% de la infraestructura, se cuenta con un uptime de más del 90%

Sin embargo, podemos ver que Adblock Plus tiene un área de mejora muy importante: se tiene un cambio a la configuración cada semana, lo cual refleja una velocidad baja de desarrollo, esta métrica representa la facilidad con la que se pueden realizar cambios dentro del código.

3.5 Plan de trabajo en la implementación

Para la implementación de la solución propuesta se planearon las siguientes etapas:

- Planeación
- Desarrollo de la solución propuesta
    - Integración de hiera
    - Migración de módulos
- Revisión de código
- Pruebas
- Integración

	Tarea	Duración	Inicio	Fin	Q3			Q4			Q1		
					Jul	Aug	Sep	Oct	Nov	Dec	Jan	Feb	Mar
1	Planeación	5d	09/08/14	09/12/14									
2	Integración con Hiera	15d	09/15/14	10/03/14									
3	Script de integración (Hiera)	5d	09/15/14	09/19/14									
4	Migración de módulos	70d	10/06/14	01/09/15									
5	Pruebas	40d	01/12/15	03/06/15									
6	Integración	2d	03/09/15	03/10/15									

Tabla 5. Planeación de la implementación del experimento.

El desarrollo de la solución propuesta no tiene una fecha límite pero se consideran necesarios 5 meses para la misma, en el que integrará toda la información específica de los servidores en un área central la cual será manejada por hiera.

La migración de los módulos es necesaria debido a diferentes factores, primero se tiene que utilizar el método de un clasificador de nodos externo, después se tienen que integrar las variables de dicho clasificador dentro de los módulos para finalmente refactorizar el código.

La revisión de código es realizada tanto por el CTO como por el líder de desarrollo, los cuales se encargaran de hacer las pruebas pertinentes, dicho periodo tampoco es definido por la empresa pero se asume que se tardará un mes para la revisión y otro mes para las pruebas, debido a que es un cambio muy grande que implica todos los servidores de la empresa.

El desarrollo de la solución fue hecho por el líder de DevOps: Mathias J. Hennig y dos internos de DevOps: Alfonso Álvarez Sánchez y Francisco Josué López Aguilar, el ponente de esta tesis.

Capítulo 4. Resultados de la implementación de la solución propuesta.

4.1 Cambios en la solución propuesta.

Como se comentó anteriormente, se pensó en una solución que incluía tres fases:

1. Centralizar la información
2. Integrar la configuración
3. Presentar la configuración

Cada una de las fases debe mantener la funcionalidad hasta ahora establecida y además cumplir con los requerimientos mencionados en el capítulo anterior, para obtener información detallada de los cambios realizados ver: Anexo A. Caso Adblock Plus.

Los cambios finales fueron los siguientes:

4.1.1 Centralizar la información

Para centralizar la información fue necesario utilizar una herramienta que tuviera compatibilidad con Puppet, puesto que es el software de la administración de la configuración utilizado por la empresa. Se decidió utilizar Hiera, ya que tiene compatibilidad nativa con Puppet y cumple con el objetivo de centralizar la información. Se desarrolló un script propio que funciona sobre Hiera y permite utilizar toda su funcionalidad la versión de Puppet 2.7. Ver Anexo C. Script

Esto permitió tener centralizada la información específica de cada nodo como es la dirección IP, el nombre del nodo, sus llaves privadas y públicas, su rol así como al grupo de servidores que pertenecen. En el momento de la implementación había dos tipos de servidores: servidores de filtros y servidores web.

La diferencia con la implementación anterior era que cuando se necesitaba cambiar algo específico de un nodo o agregar uno nuevo, se tenían que modificar manualmente diferentes archivos a lo largo de todo el código, lo que provocaba que el mantenimiento fuera lento y propenso a errores.
Por poner un ejemplo, ahora se puede agregar un nuevo filtro al archivo de hosts.yaml que se encuentra en el módulo private-stub/hiera/ y poniendo las siguientes líneas:

```
filter1:
  ip: [10.8.0.120]
  role: filterserver
  ssh_public_key: [llave pública]
  groups: [filter-servers]
```

Haciendo que toda la configuración específica quede en un solo lugar.

4.1.2 Integración de la configuración

Después de centralizar la información específica de los nodos, es necesario integrar la configuración genérica para obtener la funcionalidad deseada, para esto fue indispensable refactorizar los módulos dedicados a proveer la configuración para cada nodo.

La configuración genérica se refiere a cómo lucirá un nodo con ciertas características de funcionalidad y es independiente de la información mencionada anteriormente, por lo que en esta configuración se puede encontrar información sobre servidores web, marcos de desarrollo de software, paquetes a instalar dentro de los servidores, configuración de los paquetes y cuestiones de seguridad, entre otras cosas.

La integración hecha alcanzó el 80% de la configuración total puesto que algunos servicios requieren mucho trabajo para poder integrarlos de manera correcta por lo que se decidió no migrar esa configuración aún. La configuración que no fue integrada con Hiera es 1) Servidor de Issues/Bugs, 2) Servidor de Descargas, 3) Servidor de Actualizaciones y 4) Servidor Web. Los servidores no migrados representan el 40% del total de servidores en cantidad. Por tanto, la integración realizada impactó al 60% de servidores de Adblock Plus. Este alcance fue determinado por el CTO y el líder de proyecto en función de la importancia de los servidores. Para el CTO y el líder de proyecto, por poner un ejemplo, es más importante un servidor de filtros que un servidor Web.

Sin embargo, en cuanto a configuración se refiere y de acuerdo a los análisis del CTO y líder de proyecto, la configuración migrada representa un 80% del total, aunque el porcentaje de los servidores sea del 60%.

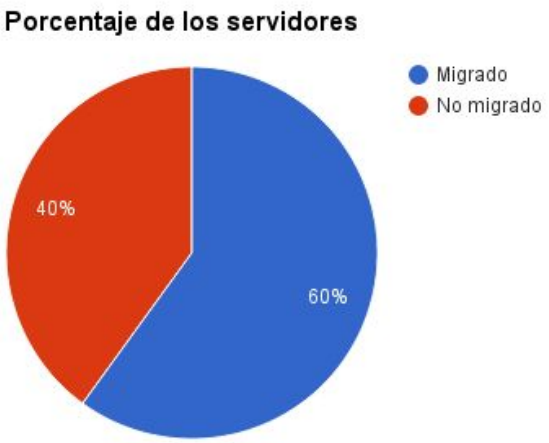


Figura 6. Porcentaje de servidores migrados.

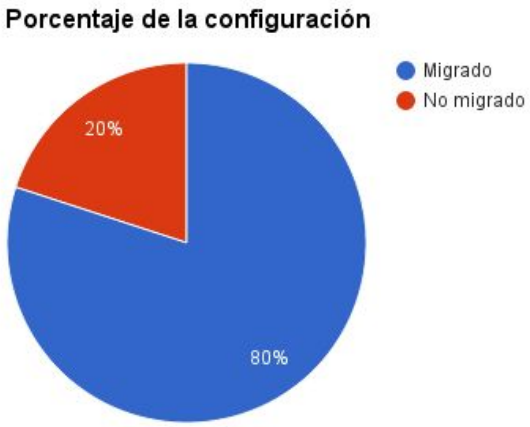


Figura 7. Porcentaje de configuración migrada.

4.1.3 Presentar la configuración

Después de hacer las pruebas necesarias en un ambiente local y debido a que no se cuenta con acceso a los servidores de producción, los encargados del aprovisionamiento revisan y aprueban los cambios propuestos después de algunas revisiones técnicas así como juntas para aclaración de dudas.

Este paso fue necesario debido al proceso de cambios que se tiene implementado en la empresa, para más información ver: 3.0.1 Propiedad de Módulos.

De los cambios propuestos el 100% fue aprobado después de realizarse las revisiones correspondientes, para posteriormente subirse al repositorio de código en el cual figura el cambio, se puede revisar dicho cambio en la siguiente liga con el identificador 7b541c38184e.

4.2 Evaluación de la eficiencia y eficacia del proceso de aprovisionamiento de la solución propuesta para Adblock Plus

Métrica	Antes	Después
Tiempo para establecer versiones conocidas desde cero	5 minutos	5 minutos
Tiempo que una versión está en producción antes de ser reemplazada	Una semana para roles dinámicos Cada seis meses para roles estáticos	Una semana para roles dinámicos Cada seis meses para roles estáticos
Porcentaje de sistemas que coinciden con las versiones aceptadas	100%	100%
Número de cambios por semana	1	5.83
Número de cambios exitosos	100%	100%
Número de veces que se dejó de dar servicio por mal funcionamiento	Sitios estáticos: 95%* Sitios con balanceador de carga: 100%*	Sitios estáticos: 95%* Sitios con balanceador de carga: 100%*

	Sitios dinámicos sin balanceador de cargas: 1.5 en promedio al mes.	Sitios dinámicos sin balanceador de cargas: 1.5 en promedio al mes.
Tiempo medio de reparación (MTTR)	Alta prioridad: 5 mins Media-Baja prioridad: 3 días	Alta prioridad: 5 mins Media-Baja prioridad: 3 días
Tiempo medio entre fallas (MTBF)	6 meses	6 meses

Tabla 7. Comparativa de métricas antes y después del experimento.

*Uptime.

Los resultados obtenidos muestran que todas las métricas se mantuvieron como antes excepto en la de “número de cambios por semana”, para visualizar mejor los datos se presenta la siguiente gráfica:

Figura 8. Número de commits por semana. Elaboración propia.

- La línea azul representa el número de commits que se hicieron antes de la solución propuesta.
- La línea roja representa los commits hechos después de la implementación de la solución propuesta.

El resultado obtenido al hacer la comparación de la información nos muestra que hubo un incremento del 583% del número de commits por semana, para ver el ejercicio completo de la información obtenida ver: Anexo B. Análisis de datos.

Capítulo 5. Discusiones, hallazgos y trabajo futuro

En los capítulos anteriores se detalló la implementación de la solución para automatizar el aprovisionamiento de la configuración de la infraestructura de Adblock Plus, debido a que la extensión cuenta con millones de usuarios es indispensable tener mecanismos que permitan realizar tareas de forma rápida y automática, sin embargo, esta solución tiene ventajas y áreas de oportunidad. Como consecuencia del desarrollo de la tesis, se realizaron hallazgos que no se habían planteado como objetivos iniciales.

5.1 Discusión

Si bien en el capítulo anterior se presentaron los datos recolectados a través de las métricas propuestas, en este apartado se analizarán las ventajas y áreas de oportunidad que se encontraron después de la implementación y la operación en un periodo de seis meses.

5.1.1 ¿Qué se hizo bien?

En el periodo de operación mencionado previamente, así como con la experiencia del ponente y los involucrados de la empresa, se encontraron las siguientes ventajas respecto a la forma previa del aprovisionamiento:

¿Qué?	¿Cómo?
Velocidad de desarrollo	Gracias a la solución propuesta se verificó que la velocidad a la que se integran nuevas características a la infraestructura aumentó en un 583% (de 1 cambio por semana a 5.83 cambios por semana en promedio). Esto significa que la velocidad promedio de desarrollo de nueva infraestructura se incrementó, aunque es importante considerar otros factores como: ● Ahora se cuenta con una persona dedicada específicamente al desarrollo de infraestructura, mientras que antes el CTO y el líder de desarrollo eran los encargados de realizar dichas tareas. ● La persona encargada de la infraestructura ahora cuenta con un “super reviewer” lo que agiliza también el desarrollo, ya que antes esta tarea era dividida entre el CTO y el líder de desarrollo por lo era más tardado por la carga de trabajo de estos dos personas. Tomando en cuenta estas consideraciones aún se puede apreciar un incremento de la velocidad de desarrollo, tanto por las métricas mostradas como por la experiencia del encargado de la infraestructura, el cual trabajó antes y después de la solución implementada y puede confirmar que la solución aumentó la facilidad de realizar cambios y por lo tanto la velocidad de desarrollo.
Pruebas	Antes de implementar la solución no se tenía forma de saber si los cambios funcionaban en producción sin mayor esfuerzo debido a la forma en la que estaba desarrollada la configuración, ahora el ambiente de desarrollo es igual al ambiente de producción por lo que se tiene gran confianza a la hora de presentar los cambios a producción y hasta el momento no ha habido ningún problema. Hubiera sido deseable la existencia de métricas que validaron esta aseveración, sin embargo, Adblock Plus no cuenta con datos para generar esta métrica. Sin embargo, el

	CTO, líder de proyecto y el encargado de la infraestructura validan la existencia de problemas cuando los entornos eran diferentes y lo no existencia de este problema con la solución actual.
Facilidad de mantenimiento	La implementación de la solución permite agregar, eliminar o modificar nuevos nodos a la infraestructura de una manera sencilla. Antes se tenía que modificar la configuración en cuatro distintos lugares para poder hacer un cambio a un nodo de la red, ahora sólo es necesario hacer dicha modificación en un solo lugar, que además se encuentra bien definido. Por poner un ejemplo, si se requería poner un nuevo Servidor de Filtros se deben realizar cuatro pasos: 1. Se tenía que modificar el Vagrantfile, que sirve para hacer el desarrollo local, 2. Agregar la llave pública del archivo en el módulo privado, 3. Modificar la configuración del módulo de monitoreo 4. Así como modificar el archivo sites.pp para agregarlo a la red de nodos de la infraestructura. Ahora sólo es necesario un solo paso: 1. agregarlo al módulo privado de Hiera dentro del archivo hosts donde toda la información será accedida automáticamente, incluyendo la llave pública del nodo.

Tabla 9. Ventajas de la solución.

5.1.2 ¿Qué se puede mejorar o cambiar para obtener mejores resultados?

Así como hay cosas buenas, también se pudieron identificar algunas áreas de oportunidad de la solución que podrían mejorar la experiencia de desarrollo para los encargados de la infraestructura.

¿Qué?	¿Cómo?

Configuración	Si bien es cierto que la configuración de los nodos es más sencilla ahora, la configuración genérica también podría serlo. En la actualidad se utiliza Puppet que podría considerarse una barrera de entrada ya que en la experiencia del ponente se necesitaron dos años para desarrollar recetas que son de calidad así como hacerlo rápidamente, sin embargo existen otros lenguajes como Ansible que podrían ser más sencillos de aprender.
Usabilidad	Actualmente la solución sólo es utilizada por las personas que modifican la configuración de la infraestructura, debido a que la solución fue pensada para este propósito, sin embargo se podría hacer un poco más sencilla la forma en la que se establezca la solución de manera local y los desarrolladores puedan utilizarla también para contar con entornos iguales a la producción y eliminar este factor de falla.
Virtualización	Debido a que la virtualización ocupa recursos de la máquina huésped se limitan las características de hardware que tienen tanto el huésped como la máquina virtual. Se podría implementar una solución que utilice contenedores como Docker, de esta manera se elimina una capa de complejidad de la solución.

Tabla 9. Áreas de oportunidad de la solución.

5.2 Hallazgos

Durante todo el desarrollo de la solución así como de las pruebas realizadas antes, durante y después de la implementación y con los datos históricos con los que se cuentan se detectó lo siguiente:

5.2.1 Resultados obtenidos que no fueron establecidos como objetivos iniciales

1. Se logró que el tiempo necesario para agregar un nuevo nodo a la red fuera reducido de aproximadamente de dos días de duración a dos horas, como se mencionó anteriormente antes el proceso manual era propenso a errores por falta de integridad en los cuatro archivos mencionados, ahora se modifica sólo un archivo de configuración que maneja la información de los nodos.

5.2.2 Métricas no contempladas

1. Número de bugs introducidos en producción antes de tener una versión estable. Previamente se generaban errores a la hora de implementar la configuración en el ambiente de producción, sin embargo no existe información para generar esta métrica. Con la nueva configuración después de seis meses no ha habido problemas.
2. Número de personas que modifican exitosamente la configuración. Después de analizar los cambios hechos al código durante tres años antes de la implementación y los seis meses después de la misma, se descubrió que antes había seis personas modificándola exitosamente, mientras que ahora hay 8 personas haciendo cambios, pues es más fácil de corroborar que la configuración funcione y los cambios se hacen más rápido.

5.3 Trabajo futuro

Debido al tiempo con el que se contaba y a la complejidad de la migración quedan tareas pendientes, de igual manera, se encontraron áreas de oportunidad que podrían implementarse en un futuro en Adblock Plus.

- Benchmarking de herramientas de administración de la configuración. En la actualidad se usa Puppet 2.7, que como ya se dijo, es usado por grandes empresas alrededor del mundo, sin embargo, una persona sin conocimiento previo a la herramienta puede tener problemas al inicio debido a la complejidad de la misma. Por lo que se puede hacer este mismo ejercicio con otros metalenguajes y comparar la rapidez en el desarrollo, la implementación y el mantenimiento.
- Migrar el 20% restante de la infraestructura a la tecnología propuesta. Del total de la configuración se migró sólo el 80% de la misma, por lo que es necesario migrar el resto para tener una solución integral que contenga todos los servidores y todas las configuraciones.
- Promover el uso de infraestructura desechable entre los propios desarrolladores de Adblock Plus. Generar material y herramientas de capacitación para los desarrolladores con el objeto de aumentar la velocidad de aprendizaje de los mismos, así todo el desarrollo se haría con infraestructura desechable.
- Integración continua de infraestructura. El desarrollo de software ha tenido un gran auge en cuanto a herramientas para hacer más sencillo y eficaz el ciclo entero, entre ellas existe la integración continua que permite probar el software en todas sus fases lo que

permite encontrar errores rápidamente, esta misma metodología podría ser implementada en la administración de la configuración de la infraestructura.

Capítulo 6. Conclusiones

“La única constante es el cambio”.

- Heráclito.

Cuando se analiza la forma en la que ha evolucionado el desarrollo de software se puede apreciar la importancia que tiene el cambio. El proyecto realizado contribuye con método práctico para realizar aprovisionamiento automático que demuestra un impacto en la eficiencia así nos adaptamos al cambio constante siendo congruentes con los principios del manifiesto ágil.

La metodología utilizada permite agilizar la manera en la que se trabaja logrando una mayor calidad en general. De acuerdo a resultados obtenidos respecto a los objetivos planteados después del experimento implementado en Adblock Plus se pueden llegar a las siguientes conclusiones:

6.1 Aprovisionamiento tradicional

Respecto al aprovisionamiento tradicional establecimos inicialmente las siguientes preguntas:

- ¿Cómo se hace el aprovisionamiento de infraestructura en la actualidad?
- ¿Qué características tiene la forma actual?
- ¿Cómo se puede mejorar?

Después del trabajo presentado se puede concluir que el modelo Desarrollo-Pruebas-Producción (DSP por su siglas en inglés) es un modelo ampliamente aceptado que intenta separar la manera en la que se trabaja para encontrar errores en etapas tempranas del aprovisionamiento de la infraestructura, sin embargo, se puede mejorar haciendo que dichos ambientes tengan exactamente las mismas características en todas las fases asegurando así que en caso de que se eviten la mayor cantidad de problemas de configuración como sucedió en el caso de Adblock Plus.

El aprovisionamiento automático de todos los ambientes surge como una necesidad natural de mejorar el tiempo de desarrollo de nuevas características en la infraestructura de la misma manera que otras metodologías han surgido de la necesidad de asegurar mejor calidad en el ciclo de desarrollo de software como lo son Continuous Integration y sus derivados.

6.2 Aprovisionamiento automático

Respecto al aprovisionamiento automático se estableció inicialmente las siguientes preguntas:

- ¿Qué están haciendo las empresas en la actualidad?
- ¿Qué ventajas tiene hacer el aprovisionamiento de esta manera?

También se encontró que existen diferentes enfoques para hacer de esta tarea algo más sencillo como la llamada Infraestructura Desechable, la cual sigue el principio de que es más sencillo aprovisionar desde cero una configuración estable que intentar solucionar los problemas encontrados, de esta forma la industria ha empezando a utilizar herramientas como Puppet, Ansible y Docker, por mencionar algunas que permiten trabajar bajo este principio,

Así pues, si se unen herramientas como las mencionadas con otras soluciones como Vagrant, se puede implementar una solución de aprovisionamiento automático que permita establecer ambientes de configuración iguales en todos los ambientes de trabajo, obteniendo ventajas como encontrar errores en etapas tempranas del desarrollo y permitir que los desarrolladores de los servicios o productos tengan la confianza de que lo que se desarrolla cumple con las características que se tendrán en el ambiente de pruebas y posteriormente en producción.

6.3 Métricas

Respecto al métricas para evaluar el status del aprovisionamiento establecimos inicialmente las siguientes preguntas:

- ¿Como se puede medir el caos en la configuración de la infraestructura?
- ¿Qué métricas se utilizan para medir la efectividad y calidad del aprovisionamiento?

Como se mencionó en capítulos anteriores existen ciertas métricas que proponen los autores de la metodología de Visible Ops, sin embargo la métrica que resultó ser la más significativa para los propósitos de esta tesis fue la de el número de cambios por semana, pues se pudo apreciar un incremento del 583%.

Sin embargo, se pudieron identificar dos métricas que no se tenían contempladas y que de igual manera son importantes para medir la efectividad y calidad del aprovisionamiento:

1. Número de bugs introducidos en producción antes de tener una versión estable.
2. Número de personas que modifican exitosamente la configuración.

De esta manera se puede tener un mejor control de la manera en que se está trabajando y así mejorar el aprovisionamiento de la infraestructura.

6.4 Método

¿Existe un método documentado para lograr el aprovisionamiento automático que sea fácil de seguir para empresas que quieran implementarlo?

Aunque existen procesos específicos como Visible Ops, no se encontró documentación específica que permita hacer el aprovisionamiento automático de la infraestructura en todas sus fases por lo que el método propuesto aquí puede ser un parteaguas para mejorar la calidad del desarrollo de la infraestructura, para esto se debe tener en cuenta que es indispensable separar la configuración genérica de la información específica de los nodos que conforman la red de la infraestructura, siguiendo un proceso iterativo primero en ambiente de desarrollo, posteriormente pruebas y finalmente producción.

6.5 Conclusiones generales

Es importante recalcar que la solución propuesta es un experimento exitoso en el sentido que está siendo usada por la extensión de navegadores más usada del mundo: Adblock Plus.
Así mismo se propone un marco nuevo de integración de la configuración para la administración y aprovisionamiento de infraestructura de servidores que da como resultado, en el caso de Adblock Plus, un incremento del 583% de commits al código de infraestructura semanal.

Así pues, el propósito de esta tesis cumple con un objetivo más importante, el cual es presentar una forma definida de trabajo que permita a los especialistas de infraestructura, tener una forma con la cual asegurar la calidad en la configuración en la que se está trabajando, utilizando las mejores prácticas propuestas por los líderes de la industria.

ANEXO A. Caso Adblock Plus.

Adblock Plus es una extensión web que bloquea la publicidad intrusiva, disponible para la mayoría de los navegadores existentes e incluso se cuenta con una aplicación para Android.

Durante la estancia en la empresa que desarrolla dicha extensión se desarrolló un proyecto enfocado en refactorizar el código existente encargado de la infraestructura, para que dicho código fuera más estable, mantenible, escalable y sirviera tanto en desarrollo como en producción.

El principal objetivo del proyecto fue hacer un entorno que fuera igual al de producción, con sus limitaciones obvias, debido al rápido crecimiento de la empresa, existen problemas de escalabilidad y mantenibilidad de la infraestructura existente, por lo que el ambiente propuesto serviría para que permitiera tener una forma sencilla de hacer las pruebas necesarias antes de subir un nuevo servicio a producción, así como poder escalar rápidamente la infraestructura.

En el presente anexo se mostrarán las similitudes y diferencias del código escrito, y se hará la comparativa del por qué se hizo de esa manera, así mismo se mostrará la forma en la que funciona y, gracias a que el proyecto es open source, se podrá descargar el código para futuros experimentos.

A la fecha del presente, los archivos fueron obtenidos del repositorio oficial de Adblock Plus (https://hg.adblockplus.org/infrastructure) así como del repositorio del líder de DevOps en la misma empresa (https://github.com/mjhennig/adblockplus-infrastructure) el cual es usado como repositorio de desarrollo.

Las tecnologías empleadas son:

- Puppet, que es un sistema de administración de configuración, el cual permite definir cómo lucirá la infraestructura del producto, asegurando que el estado definido sea aplicado. https://puppetlabs.com/puppet/what-is-puppet

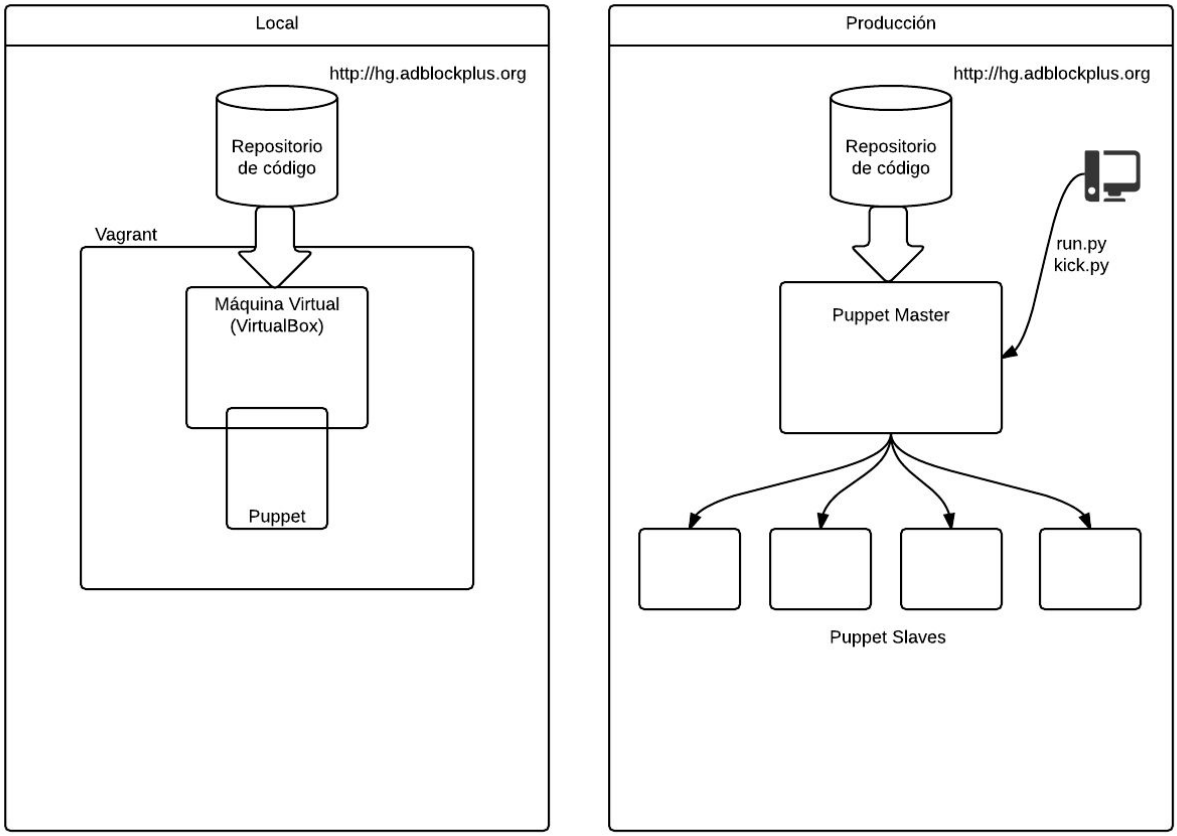
- Hiera, que es una herramienta de configuración de datos basado en la búsqueda “llave-valor” que permite almacenar toda la información de la configuración en un mismo

lugar, especificando los nodos de la infraestructura, siguiendo el principio “no te repitas” (don’t repeat yourself) . https://docs.puppetlabs.com/hiera/1/

- Vagrant, que es una herramienta utilizada para crear y configurar ambientes virtuales de desarrollo, que a su vez es independiente de sistema operativo, sistemas de administración de configuración y proveedores de virtualización.

- VirtualBox, que es un software para virtualización que permite emular una sistema operativo dentro del equipo físico.

Al momento de iniciar el proyecto se tenía una vista general del entorno de desarrollo local y de producción como muestra el diagrama siguiente:



En el ambiente de desarrollo, el cual se ejecuta dentro del equipo personal, se utiliza Vagrant envolviendo tanto a Puppet como a VirtualBox, ejecutando así la configuración dentro de la máquina virtual y obteniendo el resultado deseado.

En el ambiente de producción, se tiene un servidor de configuración central llamado “Puppet Master” el cual recibe la configuración general de todos los servidores para a su vez enviar dicha configuración a los “esclavos” los cuales realizarán los cambios (en caso necesario) y notificarán al “master” del resultado, dicha operación se ejecuta en un inicio desde la computadora personal del encargado de la operación, notificando al “master” qué servidores se necesitan aprovisionar para que a su vez éste ejecute las tareas necesarias.

Una diferencia muy importante que se puede observar en el diagrama anterior es el uso de un “Puppet Master” en el ambiente de producción, mientras que en el ambiente local, Vagrant es el encargado de aplicar los manifiestos a cada una de las máquinas virtuales.

También es importante observar que en el ambiente de producción se utilizan dos scripts código hecho en Python, los cuales sirven para aplicar la configuración de los manifiestos a los servidores elegidos, para el objetivo del proyecto no es necesario analizar el código de dichos archivos, sin embargo es importante hacer notar que dichos archivos no se podían probar en el ambiente local, por lo que cualquier error encontrado en el código de dichos scripts era encontrado en el ambiente de producción.

El primer archivo a revisar es llamado “Vagrantfile”, la configuración del ambiente virtual de sistemas operativos es realizada en dicho archivo, mientras que la configuración de los servicios en dichas máquinas virtuales puede hacerse con diferentes softwares de administración de la configuración, en el caso de Adblock Plus el software utilizado es Puppet.

Un “Vagrantfile” puede ser considerado básicamente como código de ruby que ejecuta las librerías de Vagrant, siendo código Ruby se puede hacer las operaciones normales o más complicadas que se necesiten, en este caso se tiene un función llamada “define_standard_vm” que precisamente define cómo lucirá un máquina virtual estándar dentro de nuestro ambiente de desarrollo.

En dicha función se instala Puppet para poder utilizarlo como software de administración de la configuración así como se especifican los valores de IP, memoria RAM, procesadores, archivos de donde se obtendrá la configuración para se ejecutada.

```
> require 'yaml'
>
< def define_standard_vm(config, host_name, ip)
> def define_standard_vm(config, host_name, ip, role=nil)
>
>       setup_path = File.join(REPOSITORY_DIR, "hiera", "roles", "#{role}.yaml")
>       setup = YAML.load_file(setup_path) rescue {}
>       requirements = setup.fetch("requirements", {})
>
>       requirements.each do |key, value|
>       vb.customize ['modifyvm', :id, "--#{key}", "#{value}"]
>       end
>
>       # The repository location in the production system's puppet master
>       config.vm.synced_folder ".", "/etc/puppet/infrastructure"
>
< if ! test -f /usr/bin/puppet; then
<   sudo apt-get update && sudo apt-get install -y puppet
< fi'
<
<       manifest_files = ['vagrant.pp', 'nodes.pp']
<       manifest_files.each do |manifest_file|
<       config.vm.provision :puppet do |puppet|
<       puppet.options = ['--environment=development']
<       puppet.manifests_path = 'manifests'
<       puppet.manifest_file = manifest_file
<       puppet.module_path = 'modules'
<       end
>       sudo /etc/puppet/infrastructure/hiera/install-precise.py
>       '
>
>       config.vm.provision :puppet do |puppet|
>       puppet.options = [
>       '--environment=development',
>       '--external_nodes=/etc/puppet/infrastructure/hiera/puppet-node-classifier.rb',
>       '--node_terminus=exec',
>       '--verbose',
>       '--debug',
```

```
>       ]
>       puppet.manifests_path = 'manifests'
>       puppet.manifest_file = 'nodes.pp'
>       puppet.module_path = 'modules'
>       # Requires Puppet 3.x or later
>       #puppet.hiera_config_path = 'hiera/vagrant.yaml'
>
<   define_standard_vm config, 'server4', '10.8.0.99'
<   define_standard_vm config, 'server5', '10.8.0.100'
<   define_standard_vm config, 'server6', '10.8.0.101'
<   define_standard_vm config, 'server7', '10.8.0.102'
<   define_standard_vm config, 'server10', '10.8.0.105' do |config|
<       config.vm.provider :virtualbox do |vb|
<       vb.customize ["modifyvm", :id, "--memory", 1024]
<       end
>   config_path = File.join(REPOSITORY_DIR, "modules/private/hiera/hosts.yaml")
>   config_data = YAML.load_file(config_path)
>   servers = config_data["servers"]
>   servers.each do |server, items|
>       ip = items["ip"][0]
>       role = items.fetch("role", "default")
>       define_standard_vm(config, server, ip, role)
<   define_standard_vm config, 'server11', '10.8.0.106'
<   define_standard_vm config, 'server12', '10.8.0.107'
<   define_standard_vm config, 'server15', '10.8.0.110'
<   define_standard_vm config, 'server19', '10.8.0.114'
<   define_standard_vm config, 'notification1', '10.8.0.118'
<   define_standard_vm config, 'notification2', '10.8.0.119'
<   define_standard_vm config, 'filter1', '10.8.0.120'
<   define_standard_vm config, 'filter2', '10.8.0.121'
<   define_standard_vm config, 'filter3', '10.8.0.122'
<   define_standard_vm config, 'filter4', '10.8.0.123'
<   define_standard_vm config, 'filter5', '10.8.0.124'
<   define_standard_vm config, 'filter6', '10.8.0.125'
<   define_standard_vm config, 'download1', '10.8.0.126'
<   define_standard_vm config, 'filtermaster1', '10.8.0.127'
<   define_standard_vm config, 'update1', '10.8.0.128'
<   define_standard_vm config, 'web1', '10.8.0.129'
<   define_standard_vm config, 'stats1', '10.8.0.130'
<   define_standard_vm config, 'issues1', '10.8.0.131'
<   define_standard_vm config, 'codereview1', '10.8.0.132'
<   define_standard_vm config, 'filter7', '10.8.0.133'
<   define_standard_vm config, 'filter8', '10.8.0.134'
```

```
<   define_standard_vm config, 'filter9', '10.8.0.135'
<   define_standard_vm config, 'filter10', '10.8.0.136'
<   define_standard_vm config, 'filter11', '10.8.0.137'
<   define_standard_vm config, 'filter12', '10.8.0.138'
<   define_standard_vm config, 'filter13', '10.8.0.139'
<   define_standard_vm config, 'filter14', '10.8.0.140'
<   define_standard_vm config, 'filter15', '10.8.0.141'
<   define_standard_vm config, 'filter16', '10.8.0.142'
<   define_standard_vm config, 'filter17', '10.8.0.143'
<   define_standard_vm config, 'filter18', '10.8.0.144'
<   define_standard_vm config, 'web2', '10.8.0.145'
```

Para entender el código anterior y subsecuente, el símbolo “<” (menor que) significa que la línea fue eliminada, “>” (mayor que) significa que la línea fue agregada.

Como se puede apreciar, se creó una función llamada “define_standard_vm” en la cual se define el tipo de servidor a provisionar y la dirección IP a utilizar en la red, esta información tenía que ser especificada en este archivo así como en en el manifiesto del servicio de nagios, el cual es un servicio que permite monitorear la salud y estado de los servidores, con la finalidad de crear el ambiente de desarrollo así como para tener el servicio de monitoreo.

Parte del manifiesto para la configuración del servidor de monitoreo era la siguiente:

```
nagios_host {'server_4.adblockplus.org': use => 'generic-host'}
nagios_host {'server_5.adblockplus.org': use => 'generic-host'}
nagios_host {'server_6.adblockplus.org': use => 'generic-host'}
nagios_host {'server_7.adblockplus.org': use => 'generic-host'}
nagios_host {'server_10.adblockplus.org': use => 'generic-host'}
nagios_host {'server_11.adblockplus.org': use => 'generic-host'}
nagios_host {'server_12.adblockplus.org': use => 'generic-host'}
nagios_host {'server_15.adblockplus.org': use => 'generic-host'}
nagios_host {'server_16.adblockplus.org': use => 'generic-host'}
nagios_host {'server_19.adblockplus.org': use => 'generic-host'}
nagios_host {'download1.adblockplus.org': use => 'generic-host'}
nagios_host {'filter1.adblockplus.org': use => 'generic-host'}
nagios_host {'filter2.adblockplus.org': use => 'generic-host'}
nagios_host {'filter3.adblockplus.org': use => 'generic-host'}
nagios_host {'filter4.adblockplus.org': use => 'generic-host'}
nagios_host {'filter5.adblockplus.org': use => 'generic-host'}
nagios_host {'filter6.adblockplus.org': use => 'generic-host'}
```

```
nagios_host {'filter7.adblockplus.org': use => 'generic-host'}
nagios_host {'filter8.adblockplus.org': use => 'generic-host'}
nagios_host {'filter9.adblockplus.org': use => 'generic-host'}
nagios_host {'filter10.adblockplus.org': use => 'generic-host'}
nagios_host {'filter11.adblockplus.org': use => 'generic-host'}
nagios_host {'filter12.adblockplus.org': use => 'generic-host'}
nagios_host {'filter13.adblockplus.org': use => 'generic-host'}
nagios_host {'filter14.adblockplus.org': use => 'generic-host'}
nagios_host {'filter15.adblockplus.org': use => 'generic-host'}
nagios_host {'filter16.adblockplus.org': use => 'generic-host'}
nagios_host {'filter17.adblockplus.org': use => 'generic-host'}
nagios_host {'filter18.adblockplus.org': use => 'generic-host'}
nagios_host {'filtermaster1.adblockplus.org': use => 'generic-host'}
nagios_host {'notification1.adblockplus.org': use => 'generic-host'}
nagios_host {'notification2.adblockplus.org': use => 'generic-host'}
nagios_host {'update1.adblockplus.org': use => 'generic-host'}
nagios_host {'web1.adblockplus.org': use => 'generic-host'}
nagios_host {'web2.adblockplus.org': use => 'generic-host'}
nagios_host {'stats1.adblockplus.org': use => 'generic-host'}
nagios_host {'issues1.adblockplus.org': use => 'generic-host'}
nagios_host {'codereview1.adblockplus.org': use => 'generic-host'}
```

Como se puede apreciar, tanto en el archivo ‘Vagrantfile’, usado sólo en desarrollo, como en el manifiesto de monitoreo, usado en desarrollo y producción, se utilizaba la misma configuración, repitiendo información y haciéndolo más difícil de mantener.

La solución, que abarcaría tanto desarrollo como producción fue utilizar ‘Hiera’, para concentrar la información individual de los nodos en un solo lugar, así como la configuración genérica del tipo de servidor en otro, de tal manera sólo se mantendría un sólo archivo para cada tipo de servidor así como un archivo para la información de cada servidor.

A continuación, se puede observar la configuración para un nodo de monitoreo, toda la configuración específica de ese servicio se encuentra en el archivo ‘yaml’, el cual es el formato usado en ‘Hiera’.

```
classes:
 puppetmaster:
 nagios::server:
        domain: "monitoring.adblockplus.org"
```

```
        certificate: "monitoring.adblockplus.org_sslcert.pem"
        private_key: "monitoring.adblockplus.org_sslcert.key"
        is_default: "true"
        htpasswd_source: "puppet:///modules/private/nagios-htpasswd"
        admins: ["*"]
        zone: "adblockplus.org"

nagios_commands:
 check_nrpe_timeout:
        command_line: '/usr/lib/nagios/plugins/check_nrpe -H $HOSTADDRESS$ -c $ARG1$ -t $ARG2$'
 check_easylist_http:
        command_line: '/usr/lib/nagios/plugins/check_http -S -I $HOSTADDRESS$ -H
easylist-downloads.adblockplus.org -u /easylist.txt -k "Accept-Encoding: gzip,deflate" -e "HTTP/1.1 200
OK"'
 check_notification_http:
        command_line: '/usr/lib/nagios/plugins/check_http -S -I $HOSTADDRESS$ -H
notification.adblockplus.org -u /notification.json -k "Accept-Encoding: gzip,deflate" -e "HTTP/1.1 200 OK"'

nagios_hostgroups:
 all:
        members: '*'
 filter-servers:
        members: ','
 http-servers:
        members: ','

nagios_services:
 current-load:
        use: 'generic-service'
        hostgroup_name: 'all'
        service_description: 'Current Load'
        check_command: 'check_nrpe_1arg!check_load'
 disk-space:
        use: 'generic-service'
        hostgroup_name: 'all'
        service_description: 'Disk Space'
        check_command: 'check_nrpe_1arg!check_disk'
 total-processes:
        use: 'generic-service'
        hostgroup_name: 'all'
        service_description: 'Total Processes'
        check_command: 'check_nrpe_1arg!check_total_procs'
 zombie-processes:
```

```
        use: 'generic-service'
        hostgroup_name: 'all'
        service_description: 'Zombie Processes'
        check_command: 'check_nrpe_1arg!check_zombie_procs'
ssh:
        use: 'generic-service'
        hostgroup_name: 'all'
        service_description: 'SSH'
        check_command: 'check_ssh'
http:
        use: 'generic-service'
        hostgroup_name: 'http-servers'
        service_description: 'HTTP'
        check_command: 'check_http'
easylist-http:
        use: 'generic-service'
        hostgroup_name: 'filter-servers'
        service_description: 'HTTP'
        check_command: 'check_easylist_http'
notification-http:
        use: 'generic-service'
        hostgroup_name: 'filter-servers'
        service_description: 'HTTP'
        check_command: 'check_notification_http'
bandwidth:
        use: 'generic-service'
        hostgroup_name: 'all'
        service_description: 'Bandwidth'
        check_command: 'check_nrpe_timeout!check_bandwidth!20'
        first_notification_delay: '15'
connections:
        use: 'generic-service'
        hostgroup_name: 'all'
        service_description: 'Connections'
        check_command: 'check_nrpe_1arg!check_connections'
memory:
        use: 'generic-service'
        hostgroup_name: 'all'
        service_description: 'Memory'
        check_command: 'check_nrpe_1arg!check_memory'
```

Además, se muestra el archivo de 'hosts' con formato 'yaml', en el cual podemos observar la configuración individual de cada nodo (servidor), donde se asigna un nombre, una IP, un rol, una llave pública y un grupo donde se indica qué tipo de servidor es, de esta forma se elimina la duplicidad en el archivo 'Vagrantfile' y en el manifiesto de monitoreo.

```
---
servers:
 filter1:
        ip: [10.8.0.120]
        ssh_public_key:
AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBAUhHiFdqQBATyyOE9XPR7v2+b
ghp3r1QY0O62d/klV7yhAOBKQuI6MtTu43cYQE60M10OzVfeeMfmc0dPP97UM=
        groups: [filter-servers]
 server4:
        ip: [10.8.0.99]
        role: monitoringserver
        dns: server_4.adblockplus.org
        groups: [http-servers]
 codereview1:
        ip: [10.8.0.132]
        role: codereviewserver
        groups: [http-servers]
 download1:
        ip: [10.8.0.126]
        ssh_public_key:
AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBE9gWnTMFJkJ6DqrmnmQvOHyK
/yaWvz6SKPetMX/A6avRLm4AzinvD+F7z4mIqRbJxhJgRljOxZWVgWCOQEWYaY=
        dns: downloads.adblockplus.org
        groups: [http-servers]
 filtermaster1:
        ip: [10.8.0.127]
 server10:
        ip: [10.8.0.105]
        role: intraforumserver
        ssh_public_key:
AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBPoK6neBTjJIzWn6xhJozABPJoZ
7809RmSOgBHKgbSMztjJD839Vbi263An4YgAhYijclPDX78zxjO0XgvW1aMc=
        dns: server_10.adblockplus.org
        groups: [http-servers]
 issues1:
        ip: [10.8.0.131]
        groups: [http-servers]
```

```
 stats1:
        ip: [10.8.0.130]
        groups: [http-servers]
 update1:
        ip: [10.8.0.128]
        ssh_public_key:
AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBNQ4C/hghKhYSQHiz3cquPVbbM
P5tycwHChe5gbY03jVR9dynO6Ldb5MWL02r83zFisPqM6lvtnjuGA65dzVJYE=
        groups: [http-servers]
 web1:
        ip: [10.8.0.129]
        ssh_public_key:
AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBMu0mGhPAhIbAv7H0iN74h6fC9
H2576AxI9cXFj3FAQxHI2ItLIibrViThm0luOJiRIaLc1GxavBKK0dR4bzFR8=
        groups: [http-servers]
 web2:
        ip: [10.8.0.145]
        ssh_public_key:
AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBDhC6a20iZh6VurNrRC3ZzdilTp6y
elnzJ8/49ujHvkAC38Rj6870re3Yvkg+8AfxdyVNP8owqC6x7KQD0zJ6Nk=
        groups: [http-servers]
```

De esta manera podemos agregar un nuevo rol, diferente al de monitoreo, con la configuración deseada y crear tanto nodos con la misma configuración del servicio pero con diferentes direcciones IP o llaves públicas, manteniendo sólo un archivo por rol y un archivo por nodos.

Esto también nos permite escalar rápidamente los servicios o servidores, pues la configuración puede ser cambiada a placer y afectará a todos los nodos con dicha configuración, así como fácilmente se puede agregar un nodo a la red y asignar un servicio (o varios) sin necesidad de cambiar la configuración de los manifiestos, sólo la información de dichos servicios o servidores(nodos).

En la forma tradicional de implementar lo anterior, se utiliza una configuración ‘master-slave’ donde el master cuenta con la configuración de todos los nodos así como la configuración de todos los servicios, los esclavos hacen una petición al maestro para verificar que la configuración cuente con el estado correcto, el maestro envía únicamente la configuración correspondiente al esclavo y éste la aplica en caso de haber algún cambio.

Para implementar ‘Hiera’ de la forma tradicional, se necesita tener Puppet 3.4+, sin embargo, la versión utilizada en Adblock Plus es 2.7, por lo que no se puede utilizar de forma natural ‘Hiera’ y todas las ventajas que esto representa, por lo que se implementó un mecanismo de instalación y uso de ‘Hiera’ para versiones anteriores, implementado y probado sólo en Puppet 2.7, en el cual se permite el uso de las funciones de ‘Hiera’ pero sin ocupar el código base de Puppet 3.4+.

En la implementación se utilizaron llamadas a ‘Hiera’ directamente, utilizando un clasificador de nodos de implementación propia. Si bien es cierto que para utilizar la configuración mencionada, es necesario crear un clasificador de nodos, dicho clasificador es utilizado por Puppet y no por el clasificador mismo. La implementación creada es la siguiente:

```
#!/usr/bin/env ruby
# This script is a node classifier for Puppet that operates on top of Hiera
# and uses a custom hosts.yaml config to map host roles.

require 'getoptlong'
require 'hiera'
require 'socket'
require 'yaml'

# Where to search for the Hiera configuration
HIERA_CONFIG = ENV.fetch('PUPPET_HIERA_CONFIG', '/etc/puppet/hiera.yaml')
# Where to search for the Hosts configuration
HOSTS_CONFIG = ENV.fetch('PUPPET_HOSTS_CONFIG',
'/etc/puppet/infrastructure/modules/private/hiera/hosts.yaml')

# For logging and usage hints
BASENAME = File.basename($0)

# There's no need for any options beside the commonly exepected ones yet
GetoptLong.new(
 ['--help', '-h', GetoptLong::NO_ARGUMENT]
).each do |opt, arg|
 case opt

        when '--help'
        puts <<-END
Usage:  #{BASENAME} [hostname]
        #{BASENAME} example.com
```

```
        #{BASENAME} --help

Options:

        --help, -h
        Display this help message and exit gracefully.

Environment:

        PUPPET_HIERA_CONFIG=#{HIERA_CONFIG}
        Where to find the hiera configuration file.
        PUPPET_HOSTS_CONFIG=#{HOSTS_CONFIG}
        Where to find the hosts configuration file.

END
        exit 0

 end
end

# Only one additional non-option argument is allowed, in order to explicitly
# specify a hostname to use instead of the default:
case ARGV.length
 when 0
        hostname = Socket.gethostname
 when 1
        hostname = ARGV[0][/^[^.]+/]
 else
        STDERR.puts <<-END
#{BASENAME}: unknown option: #{ARGV[0]}
#{BASENAME}: try #{BASENAME} --help
END
        exit 1
end

# Extract the server -> hostname -> role information from the hosts
# configuration file:
begin
 config = YAML.load_file(HOSTS_CONFIG)
 servers = config.fetch("servers", {})
 host = servers.fetch(hostname, {})
 role = host.fetch("role", "default")
rescue Exception => error
```

```
 STDERR.puts "#{BASENAME}: #{error.message}: #{HOSTS_CONFIG}"
 exit 1
end

# Map Hiera data into the structure Puppet expects an ENC to generate (see
# https://docs.puppetlabs.com/guides/external_nodes.html for more info):
begin
 hiera = Hiera.new(:config => HIERA_CONFIG)
 scope = {'::hostname' => hostname, '::role' => role}
 classes = hiera.lookup('classes', {}, scope, nil, :hash)
 parameters = hiera.lookup('parameters', {}, scope, nil, :hash)
 parameters['role'] = role
 result = { 'classes' => classes, 'parameters' => parameters }
rescue Exception => error
 STDERR.puts "#{BASENAME}: #{error.message}: #{HIERA_CONFIG}"
 exit 1
end

puts result.to_yaml
```

Para hacer pruebas, es necesario contar mínimo con:

- 8Gb de memoria ram.
- Procesador con 2.4 Ghz de procesamiento y 4 núcleos
- Conexión a internet
- Vagrant
- Virtualbox
- Sistema operativo basado en Debian

Anexo B. Análisis de datos

Debido a que la cantidad de datos era bastante grande y el grado de dificultad para analizarlos era alto, ya que implicaba revisión detallada de uno por uno. Se decidió tomar una muestra del segundo grupo, es decir, de los commits antes de la implementación de la solución, esto permitió obtener información estadísticamente representativa, ahorrando tiempo. Los datos pueden ser revisados en la siguiente liga: https://hg.adblockplus.org/infrastructure/shortlog utilizando los commits que comprenden el periodo desde el inicio del repositorio de código hasta el cambio llamado “Issue112 - Introduce External Node Classifier”

La muestra fue obtenida a través de la fórmula de población finita con proporciones usando la siguiente fórmula:

$$n = \frac{N * Z^2 * p * q}{d^2 * (N - 1) + Z^2 * p * q}$$

Donde:
n= Tamaño de muestra
N = Tamaño de población
Z = Valor de z, nivel de confianza
p = Proporción esperada
q = 1 - p
d = Precisión, error estimado

Utilizando un nivel de confianza del 95% y un error del %5 se tiene la siguiente fórmula:

$$n = \frac{511 * 1.965^2 * 0.5 * 0.5}{0.05^2 * (511 - 1) + 1.965^2 * 0.5 * 0.5}$$

por lo tanto

n = 220.18

Una vez calculada la muestra, se procedió a seleccionar los elementos que serían analizados. Para esto se tomaron en cuenta dos métodos de muestreo probabilístico:

- Sistemático
- Aleatorio simple

El método sistemático consiste en seleccionar datos cada enésimo elemento de la lista. Por otro lado, en el muestreo aleatorio, todos los elementos tienen la misma probabilidad de ser elegidos.

1. Se mezclaron todos los elementos de población, de tal forma que el muestreo fuera aleatorio.
2. Se aplicó el muestreo sistemático, el cual consistió en elegir un elemento al azar sin reemplazo cada 5 observaciones en la primera ronda, 7 en la segunda, 10 en la tercera y 12 en la cuarta, hasta completar el número de la muestra.

Cabe mencionar que algunos elementos elegidos no cumplían los requerimientos para ser analizados, ya que no eran parte de la población objetivo debido a que no eran "tickets que solucionaran algún problema". Dichos elementos también fueron descartados en caso de ser seleccionados. De esta forma aumentaban los elementos que tendrían que ser elegidos en la siguiente ronda.

Al finalizar la selección de la muestra se trabajó con los 220 elementos restantes obteniendo así las fechas de los commits realizados, con lo que se obtuvieron los siguientes resultados:

Antes		Después	
Fecha	Commits	Fecha	Commits

11/2/2012	114	11/2/2012	0
5/3/2013	42	5/3/2013	0
11/1/2013	33	11/1/2013	0
5/2/2014	23	5/2/2014	0
10/31/2014	8	10/31/2014	0
3/17/2015	0	3/17/2015	140

En la tabla anterior se puede apreciar en las primeras dos columnas los commits hechos en un rango de fecha, que en este caso se presenta cada seis meses, en las últimas dos columnas se ven los commits hechos después de la implementación de la solución a partir de la fecha implementada y en un rango de seis meses, que es el tiempo que pasó desde que se hizo la implementación hasta que se analizaron los datos.

- La línea azul representa el número de commits que se hicieron antes de la solución propuesta, la línea azul claro representa la tendencia de acuerdo a una

regresión polinomial con una confianza del 92% en caso de que no se hubiera hecho ningún cambio al código.

- La línea roja representa los commits hechos después de la implementación de la solución propuesta, la línea rojo claro representa la tendencia con una confianza del 71.3% después de la implementación.
- Cabe resaltar que la tendencia del número de commits después de la implementación tiene poca confianza, estadísticamente hablando, debido a que no se cuentan con los suficientes datos históricos para la misma.

Anexo C. Script

```
#!/usr/bin/ruby
# This script is a node classifier for Puppet that operates on top of Hiera
# and uses a custom hosts.yaml config to map host roles.

require 'getoptlong'
require 'hiera'
require 'socket'
require 'yaml'

# Where to search for the Hiera configuration
HIERA_CONFIG = ENV.fetch('PUPPET_HIERA_CONFIG', '/etc/puppet/hiera.yaml')
# Where to search for the Hosts configuration
HOSTS_CONFIG = ENV.fetch('PUPPET_HOSTS_CONFIG',
'/etc/puppet/infrastructure/modules/private/hiera/hosts.yaml')

# For logging and usage hints
BASENAME = File.basename($0)

# There's no need for any options beside the commonly exepected ones yet
GetoptLong.new(
  ['--help', '-h', GetoptLong::NO_ARGUMENT]
).each do |opt, arg|
  case opt

    when '--help'
      puts <<-END
Usage:  #{BASENAME} [hostname]
        #{BASENAME} example.com
        #{BASENAME} --help

Options:

    --help, -h
        Display this help message and exit gracefully.
```

```
Environment:

    PUPPET_HIERA_CONFIG=#{HIERA_CONFIG}
      Where to find the hiera configuration file.
    PUPPET_HOSTS_CONFIG=#{HOSTS_CONFIG}
      Where to find the hosts configuration file.

END
    exit 0

  end
end

# Only one additional non-option argument is allowed, in order to explicitly
# specify a hostname to use instead of the default:
case ARGV.length
  when 0
    hostname = Socket.gethostname
  when 1
    hostname = ARGV[0][/^[^.]+/]
  else
    STDERR.puts <<-END
#{BASENAME}: unknown option: #{ARGV[0]}
#{BASENAME}: try #{BASENAME} --help
END
    exit 1
end

# Extract the server -> hostname -> role information from the hosts
# configuration file:
begin
  config = YAML.load_file(HOSTS_CONFIG)
  servers = config.fetch("servers", {})
  host = servers.fetch(hostname, {})
  role = host.fetch("role", "default")
rescue Exception => error
  STDERR.puts "#{BASENAME}: #{error.message}: #{HOSTS_CONFIG}"
  exit 1
```

```
end

# Map Hiera data into the structure Puppet expects an ENC to generate (see
# https://docs.puppetlabs.com/guides/external_nodes.html for more info):
begin
  hiera = Hiera.new(:config => HIERA_CONFIG)
  scope = {'::hostname' => hostname, '::role' => role}
  classes = hiera.lookup('classes', {}, scope, nil, :hash)
  parameters = hiera.lookup('parameters', {}, scope, nil, :hash)
  parameters['role'] = role
  result = { 'classes' => classes, 'parameters' => parameters }
rescue Exception => error
  STDERR.puts "#{BASENAME}: #{error.message}: #{HIERA_CONFIG}"
  exit 1
end

puts result.to_yaml
```

Referencias

"GNU General Public License v2.0 - GNU Project - Free ..." 2010. 17 Jun. 2015 <https://www.gnu.org/licenses/old-licenses/gpl-2.0.en.html>

(2004). Docker - Build, Ship, and Run Any App, Anywhere. Retrieved June 9, 2015, from http://www.docker.com/.

(2011). Downloads – Oracle VM VirtualBox. Retrieved June 9, 2015, from https://www.virtualbox.org/wiki/Downloads.

(2011). Puppet Labs: IT Automation Software for System ... Retrieved June 9, 2015, from https://puppetlabs.com/.

(2013). Vagrant. Retrieved June 9, 2015, from https://www.vagrantup.com/.

(2014). Chef | IT automation for speed and awesomeness | Chef. Retrieved June 9, 2015, from https://www.chef.io/chef/.

Adblock Plus (2015)- Surf the web without annoying ads!. Adblockplus.org. Retrieved 16 June 2015, from https://adblockplus.org/

Ambler, S. Development Sandboxes: An Agile 'Best Practice'. Agiledata.org. Retrieved 16 June 2015, from http://www.agiledata.org/essays/sandboxes.html

Ansible is Simple IT Automation. Retrieved June 9, 2015, from http://www.ansible.com/.

Banerjee, U. (2012). Brief History of Agile Movement. Technology Trend Analysis. Retrieved 30 une 2015, from https://setandbma.wordpress.com/2012/03/23/agile-history/

Behr, K., Kim, G., & Spafford, G. (2005). *The visible ops handbook*. Eugene, OR: Information Technology Process Institute.

Behr, K., Kim, G., & Spafford, G. (2005). The visible ops handbook. Eugene, OR: Information Technology Process Institute.

Center for Technology in government,. (1998). A survey of System Development Process Models. Retrieved 30 June 2015, from http://www.ctg.albany.edu/publications/reports/survey_of_sysdev/survey_of_sysdev.pdf

Duvall, P., Matyas, S. and Glover, A. (2007). Continuous integration. Upper Saddle River, NJ: Addison-Wesley.

Eeles, P. (2006). *What is a software architecture?*. *Ibm.com*. Retrieved 9 June 2015, from http://www.ibm.com/developerworks/rational/library/feb06/eeles/

Fowler, C. (2013). *Trash Your Servers and Burn Your Code: Immutable Infrastructure and Disposable Components - Chad Fowler*. *Chadfowler.com*. Retrieved 9 June 2015, from http://chadfowler.com/blog/2013/06/23/immutable-deployments/

Fowler, C. (2013). Trash Your Servers and Burn Your Code: Immutable Infrastructure and Disposable Components - Chad Fowler. Chadfowler.com. Retrieved 9 June 2015, from http://chadfowler.com/blog/2013/06/23/immutable-deployments/

Fowler, M. (2006). *Continuous Integration*. *martinfowler.com*. Retrieved 9 June 2015, from http://martinfowler.com/articles/continuousIntegration.html

Fowler, M. Martin Fowler, Retrieved 14 October 2015, from: http://www.martinfowler.com/

Fowler, M., & Highsmith, J. (2001). Agile Manifesto. Agile Alliance. Retrieved 16 June 2015, from http://andrey.hristov.com/fht-stuttgart/The_Agile_Manifesto_SDMagazine.pdf

Hanlon, B. (2015). *Throwing away the bricks: treating your infrastructure as disposable - Session Digital*. *Session Digital*. Retrieved 9 June 2015, from http://www.sessiondigital.com/blog/disposable-infrastructure/

Humble, J., & Farley, D. *Continuous Delivery*. *Martinfowler.com*. Retrieved 9 June 2015, from http://martinfowler.com/books/continuousDelivery.html

Morris, K. (2013). bliki: ImmutableServer. martinfowler.com. Retrieved 9 June 2015, from http://martinfowler.com/bliki/ImmutableServer.html

Murray, P. (2006). Traditional Development/Integration/Staging/Production Practice for Software Development. Disruptive Library Technology Jester. Retrieved 7 July 2015, from http://dltj.org/article/software-development-practice/

Narain, S., Levin, G., & Kaul, V. (2008). Declarative Infrastructure Configuration Synthesis and Debugging (Ph.D). Princeton University.

Narain, S., Levin, G., & Kaul, V. (2008). Declarative Infrastructure Configuration Synthesis and Debugging (Ph.D). Princeton University.

Sabanin, I. Deployments Best Practices • Beanstalk Guides. Guides.beanstalkapp.com. Retrieved 7 July 2015, from http://guides.beanstalkapp.com/deployments/best-practices.html

VMware Virtualization for Desktop & Server, Application ... Retrieved June 9, 2015, from http://www.vmware.com/.

Agradecimientos

Al Consejo Nacional de Ciencia y Tecnología (CONACyT) y al Centro de Investigación en Matemáticas (Cimat) por el apoyo financiero para la realización de mis estudios así como para el desarrollo del trabajo presentado.

A Pepe Hernández y Alejandro García, maestros en ingeniería de software y mentores que me ayudaron a empezar, realizar y terminar mis estudios en la misma maestría, al Dr. Lemus, Arturo y Juan por las revisiones y comentarios.

A EyeO GmbH, la empresa que está detrás de Adblock Plus y en la cual hice las estancias durante seis meses, en especial a Mathias Hennig, que gracias a sus consejos, regaños y bromas se implementó la solución descrita en el presente trabajo, a Felix y Wladimir que me dieron la oportunidad y confianza para participar en el proyecto de Adblock Plus, y en general a todo el equipo.

A mis compañeros y amigos Tony, Leo y Potter por ayudarme siempre que lo necesité y ser un gran equipo de trabajo.

A mi familia por siempre estar ahí, a mi madre que siempre me regaña por desvelarme a veces jugando y a veces trabajando, a mi hermana por siempre preguntarme qué es lo que hago y darme ánimos para seguir haciéndolo.

A María Luisa por el apoyo incondicional durante todo este tiempo :)

Licencia