ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

AUTOMATIZACIÓN DE REDES

AUTOMATIZACIÓN DE REDES

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN TELECOMUNICACIONES

JONATHAN DANIEL CORAIZACA NAULA

jonathan.coraizaca@epn.edu.ec

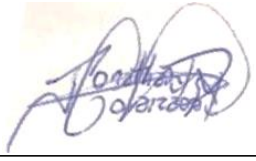
DIRECTOR: CARLOS ALFONSO HERRERA MUÑOZ

carlos.herrera@epn.edu.ec

DMQ, febrero 2023

CERTIFICACIONES

Yo, JONATHAN DANIEL CORAIZACA NAULA declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.



JONATHAN DANIEL CORAIZACA NAULA

Certifico que el presente trabajo de integración curricular fue desarrollado por JONATHAN DANIEL CORAIZACA NAULA, bajo mi supervisión.

CARLOS ALFONSO HERRERA MUÑOZ
DIRECTOR

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el (los) producto(s) resultante(s) del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

JONATHAN DANIEL CORAIZACA NAULA

CARLOS ALFONSO HERRERA MUÑOZ

DEDICATORIA

A mis padres Segundo Daniel y Blanca Inés por su apoyo incondicional, su amor y paciencia hacia mi persona durante toda mi vida y mostrarme el camino hacia la superación. A mis hermanos Fabián y Jessica por su ejemplo, cariño eterno y su ayuda en tiempos difíciles, a mi sobrina Daniela por ser el motivo para seguir adelante.

AGRADECIMIENTO

Agradezco de todo corazón a mis padres. Gracias por el apoyo, paciencia y confianza que con su motivación me impulsaron para convertirme en la buena persona que soy en la actualidad.

Quiero agradecer de manera especial al Ing. Carlos Herrera por su guía, su paciencia y su aporte que me ha dado para el desarrollo de este Trabajo de Integración Curricular.

A mis amigos Juan Carlos, Wilmer, Victor, Ángel, Edison, Carlos Luis, Edgar, Romel que hice en este arduo camino universitario, por acompañarme, y ser un apoyo en muchos de los momentos complicados en mi etapa como estudiante.

De manera especial a Verónica por creer en mi desde el primer día hasta la actualidad y ser tan incondicional, brindándome su apoyo en buenos y malos momentos, por el cariño que me brinda y por su sinceridad.

Por ultimo y más importante, agradezco a Dios por las fuerzas que me ha dado para seguir adelante.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

En este presente Trabajo de Integración Curricular se realiza un estudio sobre los fundamentos básicos de Automatización de Redes y DevOps.

En el primer capítulo se describe las bases teóricas de Automatización de Redes, los beneficios de migrar a una red automatizada, la forma de como interactúan estas redes con los dispositivos de red mediante las APIs (*Aplication Programming Interfaces*), y del software de automatización que se utilizan para la comunicación con las APIs como son XML (Lenguaje de Marcado Extensible) y JASON(Notación de Objetos de JavaScript). A su vez se describen los fundamentos teóricos de las DevOps y las tres vías que se utilizan para poder implementarlo en la Automatización de Redes.

En el segundo capítulo se estudia los conceptos básicos para los diferentes tipos de Automatización de Redes, como son las redes basadas en scripts, y las redes basadas en software. En las redes basadas en scripts se describe el lenguaje de programación más utilizado como lo es *Phyton*, así como la plataforma de software libre para gestionar las redes como lo es *Ansible*. Para la optimización de los procesos que intervienen en las redes automatizadas, se describen las herramientas de DevOps más importantes que utilizan las redes empresariales.

El tercer capítulo muestra comparaciones entre las herramientas más importantes usadas en la Automatización de Redes y también entre herramientas más utilizadas en las DevOps.

PALABRAS CLAVE: XML, Json, DevOps, Ansible, Netbrain, Jenkins, Docker, Puppet, Chef.

ABSTRACT

In this present Curricular Integration Work, a study is carried out on the basic fundamentals of Network Automation and DevOps.

The first chapter describes the theoretical bases of Network Automation, the benefits of migrating to an automated network, the way these networks interact with network devices through APIs (Application Programming Interfaces), and the automation software that They are used for communication with APIs such as XML (Extensible Markup Language) and JASON (JavaScript Object Notation). At the same time, the theoretical foundations of DevOps and the three ways that are used to implement it in Network Automation are described.

In the second chapter, the basic concepts for the different types of Network Automation are studied, such as script-based networks and software-based networks. In script-based networks, the most widely used programming language such as Python is described, as well as the free software platform to manage networks such as Ansible. For the optimization of the processes involved in automated networks, the most important DevOps tools used by business networks are described.

The third chapter shows comparisons between the most important tools used in Network Automation and also between the most used tools in DevOps.

KEYWORDS: XML, Json, DevOps, Ansible, Netbrain, Jenkins, Docker, Puppet, Chef.

1 INTRODUCCIÓN

Durante décadas, la gestión de redes se ha basado en su totalidad en la interfaz de línea de comandos (CLI) y protocolos heredados como SNMP (Simple Network Management Protocol). Estos protocolos y métodos están severamente limitados. La CLI, por ejemplo, es específica del proveedor, carece de una jerarquía de datos unificada (a veces incluso para plataformas del mismo proveedor) y se diseñó principalmente como una interfaz humana. SNMP sufre importantes problemas de escalado, no es apto para escribir la configuración en los dispositivos y, en general, es muy complejo de implementar y modificar a gusto del administrador.

En esencia, la automatización tiene como objetivo disminuir en lo que más se pueda, la mayor cantidad de trabajo posible de los humanos y delegar ese trabajo a las máquinas. Pero con las interfaces y protocolos heredados antes mencionados, la comunicación de máquina a máquina no es efectiva ni eficiente; y a veces, casi imposible.

Además, la configuración del dispositivo y los datos operativos tradicionalmente han carecido de una jerarquía adecuada y no han seguido un modelo de datos. Los flujos de trabajo de administración de red siempre han estado lejos de ser maduros, en comparación con los flujos de trabajo de desarrollo de software en términos de versiones, colaboración, pruebas e implementaciones automatizadas.

El presente Trabajo de Integración Curricular describe los principales conceptos, herramientas y métodos utilizados en la Automatización de las Redes. También se enfoca en las DevOps, utilizando las herramientas en este entorno organizado de Automatización.

1.1 Objetivo general

Estudiar los conceptos Automatización de Redes mediante metodologías de DevOps.

1.2 Objetivos específicos

1. Describir los fundamentos básicos de la Automatización de Redes.
2. Estudiar los diferentes tipos de Automatización de Redes.
3. Describir las características de DevOps.
4. Describir las herramientas más utilizadas en DevOps para redes empresariales.

1.3 Alcance

Se realizará el estudio de los conceptos básicos de la Automatización de Redes, también se realizará la descripción de los diferentes tipos de soluciones como son la Automatización de redes basada en scripts, automatización de redes por lenguaje heredado, por código abierto y a través de software. Se incluirá un estudio de las características de DevOps y como este conjunto de prácticas influye en el aumento de la capacidad de una organización para brindar servicios y aplicaciones de TI de forma más rápida que los tradicionales procesos de desarrollo, permitiendo cada vez más la integración del Desarrollo y la parte Operativa en una infraestructura de TI. Se analizará las características, beneficios y las herramientas que más se utilizan en las prácticas de DevOps.

El trabajo se lo realizará en tres fases:

1.3.1 Fase de diseño

Se realizará una revisión bibliográfica acerca de los fundamentos y tipos de Automatización de Redes, fundamentos y características de DevOps.

1.3.2 Fase teórica

Se describirá los diferentes tipos de soluciones para automatizar las redes y se describirá las herramientas más utilizadas en DevOps.

1.3.3 Fase de evaluación

Se describirán comparaciones entre los diferentes tipos de soluciones de Automatización de Redes y también de las principales herramientas que se utilizan en el desarrollo de DevOps.

1.4 Marco teórico

En esta sección se describe los conceptos de Automatización de Redes, las características de este tipo de redes, las soluciones que se ofrecen tanto en hardware como en software y también la influencia actual en la Automatización de Redes con las denominadas DevOps.

1.4.1 Automatización de Redes

La automatización, en palabras simples, significa usar software para realizar una tarea que de otro modo se lo realizaría de forma manual. La automatización no es un concepto nuevo que antes no se lo ha utilizado. Los protocolos de enrutamiento, por citar un ejemplo, son programas de automatización que le ahorran el trabajo de ingresar rutas de forma manual en cada nodo de la red. Por dar otro ejemplo, DNS es un programa de automatización que le evita tener que buscar la dirección IP de cualquier destino con el que necesite comunicarse.

Para automatizar, se requiere de un software que va incorporado en los nodos de la red, se puede comprar una plataforma de software o a su vez, se puede implementar un programa o script que cree usted mismo. [1]

Además del beneficio obvio de hacer la vida más sencilla, la automatización ofrece las siguientes ventajas:

• Implementación rápida de cambios en la red.

• Alivio de realizar tareas rutinarias repetitivas.

• Cambios en el sistema consistentes, confiables, probados y que cumplen con los estándares.

• Reducción de errores humanos y malas configuraciones de la red.

• Mejor integración con las políticas de control de cambios.

• Mejor documentación de red y análisis de cambios.

De todas estas ventajas, se puede afirmar que la velocidad de implementación es de las más importantes. Ser capaz de implementar un cambio de red con la simple acción de presionar un botón (si se lo quiere simplificar de forma sencilla) definitivamente es menos costoso que visitar cada nodo de red y reconfigurar manualmente. El ahorro de tiempo aumenta drásticamente a la vez que se aumenta el número de nodos afectados. [1]

Sin embargo, los cambios de red consistentes y confiables, junto con la reducción del error humano son aún más beneficiosos que la velocidad de implementación. La importancia de la precisión se vuelve imprescindible a medida que aumenta la cantidad de veces que se debe implementar un cambio. La implementación de un cambio de red en pocos dispositivos se puede hacer con bastante precisión utilizando herramientas primitivas y una supervisión elevada. Lo más probable es que este mismo cambio no sea posible al implementar en miles de dispositivos. La velocidad ahorra gastos operativos durante la implementación, pero la precisión brinda beneficios acumulativos durante la vida útil de la red. [1]

Dentro de las definiciones de la Automatización de Redes se puede identificar varias herramientas y comprender las relaciones entre éstas. En la Figura 1.1, se puede observar la clasificación de las herramientas necesarias que un ingeniero de redes o desarrollador va a necesitar, para automatizar las redes. [1]

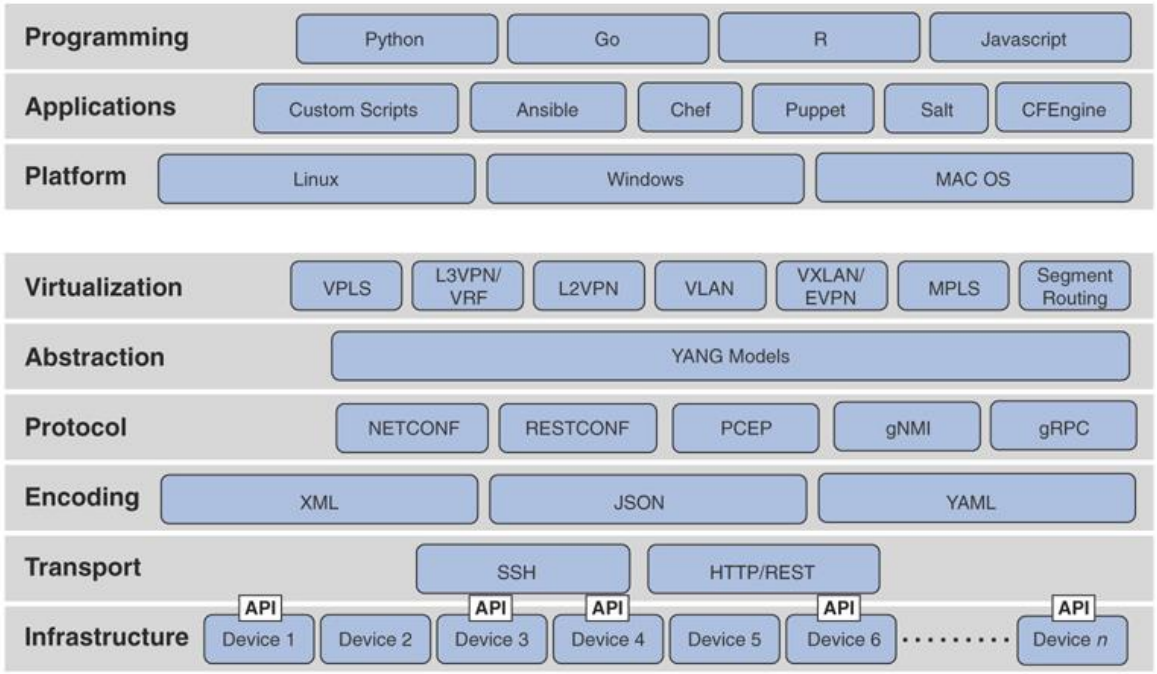


Figura 1.1 El ecosistema de programación y automatización de redes [1]

Es importante señalar que la Figura 1.1 sólo se muestra una perspectiva. El orden en que aparecen las capas de Aplicación, Automatización, Plataforma, Virtualización y Abstracción y cómo interactúan puede variar según el entorno de red. Lo que es más importante son las herramientas disponibles dentro de las distintas capas.

Actualmente, las herramientas de automatización interactúan con los dispositivos de red a través de las API (*Application Programming Interfaces*), que en sí mismas son

abstracciones del dispositivo físico subyacente. La diferencia es que las APIs residen en los dispositivos individuales y son específicas del software de su propio dispositivo. El software de automatización generalmente se comunica con las API a través del Lenguaje de Marcado Extensible (XML) o la Notación de Objetos de JavaScript (JSON), "Lenguaje de Marcado Extensible (XML) y XML Definición de esquema (XSD)", y "Notación de objetos JavaScript (JSON) y Definición de esquema JSON (JSD)". Incluso las CLI de los enrutadores y conmutadores modernos, son en realidad aplicaciones que se ejecutan sobre las API locales. en lugar de interfaces directas a los sistemas operativos. [1]

1.4.1.1 Application Programming Interfaces (APIs)

Las APIs son mecanismos que se utilizan para comunicarse con aplicaciones y otros programas. También son usados para comunicarse con varios componentes de una red a través de software. Un desarrollador puede usar API para configurar o monitorear componentes específicos de una red. [2]

Para el enfoque de Automatización de Redes existen dos tipos: API hacia el norte y API hacia el sur. La Figura 1.2 muestra las operaciones básicas típicas de las APIs hacia el norte y hacia el sur. [3]

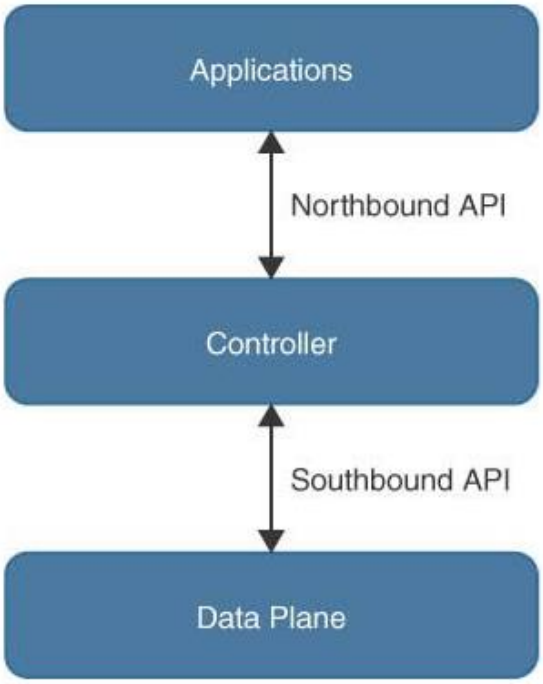


Figura 1.2 Operaciones básicas de las APIs [3]

Las APIs de dirección norte se utilizan por lo general para la comunicación desde un controlador de red a su software de gestión. Por ejemplo, Cisco DNA Center tiene una interfaz gráfica de usuario (GUI) de software, que se utiliza para administrar su propio controlador de red. [3]

Si un operador de red realiza un cambio en la configuración de un conmutador en el software de administración del controlador, esos cambios se reflejan a los dispositivos individuales mediante una API hacia el sur. Estos dispositivos pueden ser enrutadores, conmutadores o incluso puntos de acceso inalámbrico. Las API interactúan con los componentes de una red mediante el uso de una interfaz programática. [3]

1.4.1.2 Lenguaje de Marcado Extensible (XML)

El Lenguaje de Marcado Extensible (XML) es un formato de datos muy común que se usa mucho en la automatización de la configuración. El análisis de XML es similar al uso de otros formatos de datos, ya que Python comprende de forma nativa y puede admitir la codificación y decodificación de XML. La Figura 1.3 muestra un ejemplo simple de cómo se ve la estructura XML. [3]

```
<device>
    <Hostname>Rtr01</Hostname>
    <IPv4>192.168.1.5</IP4>
    <IPv6> </IPv6>
 </device>
```

Figura 1.3 Estructura simple XML [3]

XML se parece un poco a la sintaxis HTML; fue diseñado para trabajar de la mano con HTML para el transporte y almacenamiento de datos entre servicios web y API. XML tiene una estructura de árbol, con el elemento raíz en la parte superior. Hay una relación padre/hijo entre los elementos. En la figura anterior, el dispositivo es el elemento raíz que tiene *Hostname*, IPv4 e IPv6 como elementos secundarios. Al igual que con HTML, una etiqueta tiene significado y se usa para encerrar las relaciones de los elementos con una etiqueta de inicio (<>) y una etiqueta de cierre (</>). No es tan diferente de JSON en que una etiqueta actúa como una clave con un valor.

También puede asignar atributos a una etiqueta utilizando la siguiente sintaxis, como se observa en la Figura 1.4: [3]

```
attribute name="some value"
```

Figura 1.4 Asignación de atributos de una etiqueta XML [3]

1.4.1.3 Notación de Objetos de JavaScript (JSON)

La Notación de Objetos JavaScript (JSON) es una estructura de datos que se deriva del lenguaje de programación Java, pero se puede usar como una estructura de datos portátil para cualquier lenguaje de programación. Fue creado para ser una forma estándar y fácil de leer para transportar datos entre aplicaciones. JSON se usa mucho en los servicios web y es uno de los formatos de datos principales que necesita saber cómo usar para interactuar con la infraestructura de Cisco. La estructura de datos se basa en pares clave/valor, que simplifican el mapeo de datos y su recuperación. La Figura 1.5 muestra un ejemplo de JSON. [3]

```
{
    "interface": {
        "name": "GigabitEthernet1",
        "description": "Router Uplink",
        "enabled": true,
        "ipv4": {
            "address": [
                {
                    "ip": "192.168.1.1",
                    "netmask": "255.255.255.0"
                }
            ]
        }
    }
}
```

Figura 1.5 Ejemplo de JSON [3]

En la Figura 1.5, puede ver la estructura que proporciona JSON. “interface” es el objeto de datos principal, y puede ver que su valor son varios pares clave/valor. Se observa que los

datos se ven similares a un diccionario de Python. Esta capacidad de anidamiento le permite estructurar modelos de datos muy sofisticados.

1.4.2 Development and Operations (DevOps)

DevOps es una combinación de los términos en inglés *Development* y *Operations* y son prácticas de intercambio en la que los desarrolladores de software y la gente de operaciones conforman una unidad que puede crecer y trabajar en conjunto, en este marco de trabajo se promueve un desarrollo mejorado de aplicaciones en un menor tiempo, garantizando la calidad y las operaciones. [4]

Es una aplicación práctica "*Lean*" and "*Agile*", dos métodos muy populares en la gestión de proyectos. "*Lean*" está enfocado en conseguir un proceso capaz de entregar al cliente el mayor valor posible de un proceso con la mejor calidad. "*Agile*" se enfoca más en el personal y sus relaciones, tanto en los grupos de trabajo como en el cliente. [3]

El objetivo de DevOps es ayudar a las organizaciones a producir más en menor tiempo y por consecuencia de esto mejorar el rendimiento en las todas sus etapas, desde el diseño hasta la implementación. DevOps tiene cinco principios fundamentales: [5]

- **Cultura**: para que DevOps funcione, la cultura organizacional debe cambiar. Este es, con mucho, uno de los aspectos más difíciles de adoptar, pero es el factor con más importancia para alcanzar el éxito. DevOps requiere una cultura de compartir.
- **Automatización**: si bien DevOps es más que un simple conjunto de herramientas de software, la automatización es el beneficio más fácil de identificar. Las técnicas de automatización aceleran en gran medida el proceso de implementación, permiten detectar y corregir defectos antes y eliminan la necesidad de intervención humana en tareas repetitivas.
- **Lean**: Los objetivos de Lean son reducir los esfuerzos desperdiciados y agilizar el proceso. Es una filosofía de gestión de mejora continua y aprendizaje.
- **Medición**: Para obtener una mejora en los resultados, se necesita realizar mediciones continuas. El éxito con DevOps requiere la medición de las métricas de rendimiento, procesos y personas con la frecuencia que sea posible.
- **Compartir**: DevOps requiere una cultura de comentarios y de intercambio. El objetivo final es romper barreras, creando un entorno de destino compartido inclusivo.

La Figura 1.6, muestra los componentes principales de DevOps y cómo están interrelacionados. [3]

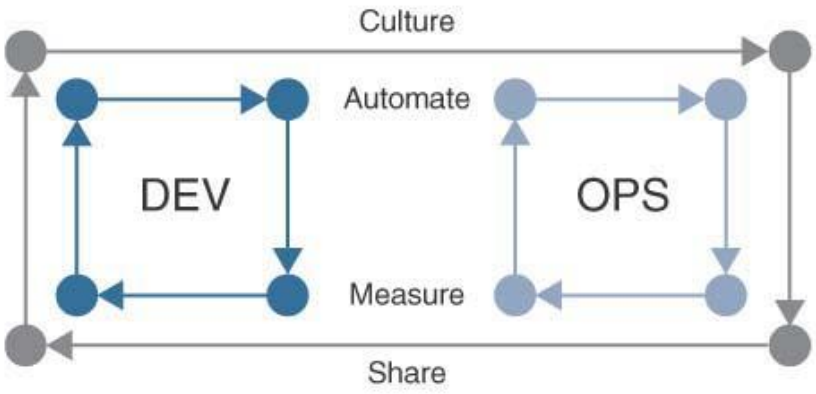


Figura 1.6 DevOps [1]

1.4.3 Prácticas de DevOps

Existen tres formas de DevOps que nos ofrecen un marco para la implementación en su propia organización.

1.4.3.1 Primera Vía: Sistemas y Flujo

Se asemeja mucho a tener una vista panorámica del flujo de tráfico de autos en una carretera. El tener una vista panorámica de cómo fluye el trabajo desde el desarrollo hasta la producción es esencial para descubrir formas óptimas de llegar al objetivo final de una aplicación funcional que proporcione valor. Muchos equipos usan tableros Kanban para visualizar el trabajo, ya sea a través de una aplicación como Jira o simplemente colocando notas adhesivas con el flujo de trabajo en una pizarra para que todos puedan ver lo que se está haciendo y lo que se debe hacer. [3]

La Figura 1.7, muestra la primera forma de DevOps, que se centra en los sistemas y el flujo. [3]

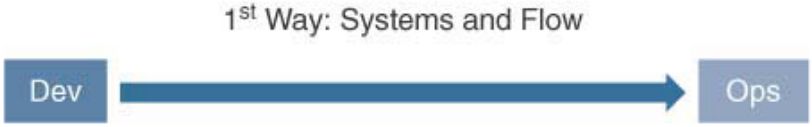


Figura 1.7 Primera Vía: Sistemas y Flujo [3]

Para lograr los objetivos de esta vía, se debe reducir las unidades o los grupos de trabajo, de modo que sean más eficientes y confiables, evitando atascos de tráfico de carga de trabajo. Un sprint ágil es un corto periodo de tiempo de trabajo para mejorar un software. En un sprint ágil, es común trabajar en un pequeño conjunto de funciones durante un período de dos semanas para reducir la cantidad de trabajo en un intervalo de tiempo determinado. Esto les da tiempo a los desarrolladores para determinar la capacidad sin verse atareados en múltiples direcciones. Esto también se aplica al lado de las operaciones, ya que ambos lados son parte integral del éxito. [5]

Los desarrolladores no pueden simplemente ofrecer actualizaciones de aplicaciones en sus equipos de operaciones sin tener una visión sólida de cómo se implementa cada actualización y los equipos de operaciones tienen que construir un sistema que sea capaz de manejar la velocidad y la agilidad que necesitan los desarrolladores. Al hacer que tanto los desarrolladores como las operaciones trabajen en conjunto, para comprender dónde están las restricciones. [3]

La calidad debe estar integrada en todo el proceso, desde el desarrollo hasta la implementación. Cuanto antes sea abordada una situación desafiante, más se reducirá su esfuerzo laboral general. Esto equivale a ahorrar dinero y tiempo. El objetivo es reducir el tiempo perdido en el trabajo repetitivo.

Las siguientes son las características clave de la primera forma:

- Hacer visible el trabajo
- Reducir el tamaño de los grupos.
- Reducir los intervalos de trabajo.
- Incorporar calidad al evitar que los defectos pasen aguas abajo.
- Optimizar constantemente para los objetivos comerciales.

1.4.3.2 Segunda Vía: Circuito de Retroalimentación

Una de las cosas frecuentes en DevOps son las analogías con la fabricación. Dado que gran parte de esta metodología de gestión se deriva de *Lean* y del Sistema de producción de Toyota, conceptos como la prevención de defectos, ocupan un lugar central en esta propuesta. La idea de un circuito de retroalimentación es proporcionar orientación y dirección sobre lo que funciona y lo que no funciona. En el caso de que algo no funcione, se debe asegurar de que no vuelva a suceder.

DevOps requiere que tome la retroalimentación como la mejor oportunidad para realizar las correcciones.

La Figura 1.8, muestra la segunda forma de DevOps, que se centra en el circuito de retroalimentación. [3]

Figura 1.8 Segunda vía: circuito de retroalimentación [3]

Cuantas más muestras de retroalimentación, más rápido se puede detectar problemas y recuperarse de ellos. En la línea de fabricación, dentro del sistema de producción de Toyota hay un parámetro llamado Andon Cord, que cualquier persona puede interceder en cualquier momento para detener la línea de producción. Esta forma de retroalimentación es inmediata y muy visible. También inicia el proceso de corrección de un problema, donde todos en el área se enfocan en resolver el problema antes de que se reinicie la línea. En DevOps, significa que cuando se detectan problemas con el software, todo el equipo colabora para solucionar la causa anterior. [5]

Una vez superado un problema, el siguiente paso es documentar y mejorar los procesos que llevaron al problema en cuestión, para aprender de esos errores y poder seguir adelante.

Las siguientes son las características clave de la segunda vía:

- Ampliar la retroalimentación para evitar que los problemas vuelvan a ocurrir.
- Habilitar una detección y recuperación más rápida.
- Ver los problemas a medida que ocurren y agregarlos hasta que se solucionen.
- Maximizar las oportunidades de aprender y mejorar.

1.4.3.3 Tercera Vía: Experimentación y Aprendizaje Continuos

Esta vía utiliza la prueba y el error como método de experimentación. Es muy raro que el éxito ocurra en el primer intento. Cuando ocurran problemas, se debe evitar señalar con el dedo y culpar a las personas; en cambio, hay que descubrir qué salió mal y qué podría

mejorarse para hacerlo mejor. La Figura 1.9, muestra la tercera forma de DevOps, que se centra en la experimentación y el aprendizaje continuos. [3]

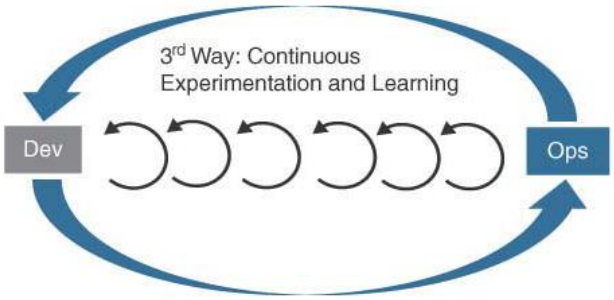


Figura 1.9 Tercera Vía: Experimentación y Aprendizaje Continuos [3]

DevOps hace referencia a la mejora continua y la solución de problemas para que el sistema funcione mejor. Su objetivo es programar tiempo para que las personas se concentren en mejorar el sistema y prueben nuevos métodos y tecnologías.

Finalmente, se debe construir una cultura de compartir y aprender. Un método simple para conseguirlo es crear repositorios de códigos compartidos y poner a disposición esa información valiosa de las computadoras al resto de la organización. [3]

Las siguientes son las características claves de la Tercera Vía:

- Llevar a cabo una experimentación dinámica y disciplinada y asumir riesgos.
- Defina el tiempo para solucionar problemas y mejorar el sistema.
- Cuando las cosas van mal, no apuntar con el dedo.
- Crear repositorios de código compartido.

2 METODOLOGÍA

En este capítulo se muestran los diferentes tipos de Automatización de Redes como son la Automatización de Redes basada en scripts, por lenguaje heredado, por código abierto y basados en software. [6]

La Automatización de Redes es un método para que el software configure, provea, administre y pruebe automáticamente los dispositivos de red. Las empresas y los proveedores de servicios lo utilizan para aumentar la eficiencia y reducir los errores humanos y los costos operativos. [7]

Las funciones admitidas por las herramientas de automatización de redes van desde el mapeo básico de redes y el descubrimiento de dispositivos hasta flujos de trabajo más complicados, como la gestión de configuración de redes y el suministro de recursos de redes virtuales. [7]

La Automatización de Redes también juega un papel clave en las redes definidas por software, la virtualización de redes y la orquestación de redes, lo que permite la configuración automática de usuarios y funciones de redes virtuales, como el equilibrio de carga virtual. [7]

2.1 Tipos de Automatización de Red

La automatización se puede utilizar en cualquier tipo de red, incluidas las redes de área local (LAN), las redes de área extensa (WAN), las redes de centros de datos, las redes en la nube y las redes inalámbricas. En contexto, cualquier recurso de red intervenido a través de una interfaz de línea de comandos (CLI) o una interfaz de programación de aplicaciones (API) puede automatizarse. [1]

La automatización de red basada en scripts y sus tareas son ejecutadas utilizando lenguajes de programación y por guiones de comandos interpretados por un motor de escritura, a esta acción se la denomina *scripting*. Los lenguajes heredados, como Tcl y Perl, son populares en la automatización de redes debido a su familiaridad. Pero, a medida que las redes se volvieron más complejas, los nuevos lenguajes de programación de código abierto como Python y Ruby ganaron popularidad por su facilidad de uso y flexibilidad. [3]

La Automatización de Red basada en software o comúnmente denominada automatización de red inteligente se reorganiza a través de un portal de gestión, lo que elimina la necesidad

de ejecutar comandos de forma manual. Estas plataformas por lo regular proporcionan plantillas para crear y realizar tareas basadas en políticas de lenguaje simples. [8]

2.1.1 Automatización de Red basada en scripts

A menudo se encuentra que la programación no es necesaria para crear sistemas automatizados útiles. Algunas herramientas, como *Gluware*, proporcionan una interfaz de usuario que los administradores de red pueden usar para automatizar la configuración de la red sin necesidad de programación. Asimismo, los fabricantes de equipos de red obtienen productos o socios que proporcionarán las herramientas para simplificar y automatizar la configuración de los equipos. [3]

Existen varias formas de automatizar mediante scripts, una de las más conocidas y de fácil entendimiento de lo que es scripting es Ansible. Su lenguaje de control se asemeja más a una sintaxis de gestión de red que a un lenguaje de programación. Una gran variedad de funciones útiles es posible: [3]

- Ejecuta comandos y recopila la salida en archivos, idóneo para la solución de problemas.
- Valida la operación de la red lo cual es verificado contra una fuente de verdad.
- Crea configuraciones a partir de plantillas. Esto lo hace añadiendo variables Jinja2.
- Carga configuraciones en dispositivos de red.

Una vez creado los scripts y la documentación, también llamada fuente de verdad, se debe aprender a utilizar un repositorio de código fuente basado en la web como Git, GitHub, GitLab o un repositorio central semejante en donde los desarrolladores pueden almacenar el código fuente para probarlo y seguir colaborando.

2.1.1.1 Python

En la cima del ecosistema de programabilidad y automatización se encuentran los lenguajes de programación. Python, Go, R y JavaScript se dan como ejemplos. Existen lenguajes de programación que se podrían utilizar, siendo C y C ++ los más destacados, aunque son más utilizados para el desarrollo de software que para la gestión de redes. que necesitan poder escribir programas y scripts para facilitar su trabajo. También hay una serie de lenguajes que podríamos agregar a la lista, como Perl, Expect y Tcl que todavía existen en mayor o menor medida, pero que han sido eclipsados por lenguajes más nuevos y poderosos. [1]

Python es el lenguaje de programación más utilizado para la automatización de redes, y admite una gran cantidad de bibliotecas, módulos y paquetes específicos para redes. Python es de fácil aprendizaje, fácil uso y también fácil de depurar, lo que se ajusta a los requisitos de los usuarios de redes que solo necesitan hacer su trabajo sin tener que convertirse en programadores profesionales. Python está lejos de ser un lenguaje "principiante". Es ampliamente utilizado por compañías como Facebook, Netflix, Instagram, Reddit, Dropbox y Spotify. Los desarrolladores de software de Google incluso tienen un dicho: "Use Python donde podamos, C++ donde debamos". Python es versátil, funciona igual de bien para secuencias de comandos y como lenguaje de unión (para unir módulos escritos en otros lenguajes). También es altamente portátil a diferentes plataformas. [3]

Cuanto más utilice productos de automatización empaquetados como Ansible o la solución de Cisco (denominado Cisco ACI) o interactúe con dispositivos de red a través de sus API en lugar de directamente con su CLI, más encontrará que Python es una herramienta esencial en su caja de herramientas.

2.1.1.2 Ansible

Ansible es una herramienta de administración y orquestación de configuración que es utilizando para una variedad de propósitos. Esta herramienta se encuentra ubicada en el entorno de herramientas de programación, dentro de las aplicaciones que son más utilizadas en Automatización de Redes. [3]

Se puede usar para configurar y monitorear servidores y dispositivos de red, instalar software y realizar tareas más avanzadas, como implementaciones continuas y actualizaciones sin tiempo de para o inactividad. Fue creado en 2012 y adquirido por RedHat en 2015. Ansible es apropiado tanto para entornos pequeños como grandes y se puede usar para administrar un grupo de servidores y dispositivos de red o para administrar miles de dispositivos. No tiene agente, lo que significa que no hay software o servicio que deba instalarse en el dispositivo administrado. [3]

Ansible se conecta a los dispositivos administrados de la misma manera que un administrador de red o sistema normal se conectaría con fines de administración; en la mayoría de los casos, a través de SSH, pero también se admiten las interfaces NETCONF y REST API. Ansible al ser basada en Python, también es considerada de código abierto. Existe una oferta comercial disponible, llamada Ansible Tower, que incluye una interfaz de usuario web, una API REST hacia el norte, control de acceso basado en roles, estadísticas y mucho más. [3]

También en la categoría de aplicaciones hay una serie de plataformas de automatización prediseñadas que puede descargar de forma gratuita o comprar: Salt, Chef, Puppet y CFEngine son ejemplos, pero hay muchos otros. Lo que tienen en común, es que todos comenzaron como plataformas para automatizar la administración de servidores. Si está en una tienda de DevOps o en cualquier entorno que orqueste una gran cantidad de sistemas finales, su organización probablemente ya tenga una plataforma de automatización favorita de esta lista. [3]

Ansible es el marco de automatización más popular entre los usuarios de redes. Es fácil de aprender y se integra bien como módulo de Python. En muchos casos, se utilizan otros métodos de automatización para una organización, pero tener una base en Ansible es valioso y brinda una ventaja para comprender los conceptos de cualquiera de esta clase de plataformas de automatización.

Ansible se puede instalar de varias maneras diferentes, utilizando el administrador de paquetes del sistema operativo, como se indica en la Figura 2.1 se puede instalar ansible en RedHat y CentOS Linux: [3]

```
sudo yum install ansible
```

Figura 2.1 Línea de comando para instalar ansible en RedHat y CentOS. [3]

Ansible es una herramienta de automatización sin agente, lo que significa que no es necesario instalar nada en el host administrado además de SSH. Si un host es capaz de ejecutar Python, también se debe instalar Python. En cualquier caso, debe instalar Ansible en el host de administración, y en la Figura 2.2 muestra cómo puede hacerlo en una distribución basada en RPM (*RPM Package Manager*) como CentOS. [2]

Cuando Ansible está instalado en el host de administración, puede comenzar a administrar hosts de inmediato. La Figura 2.2 ilustra el flujo de llamadas para un solo comando en Ansible.

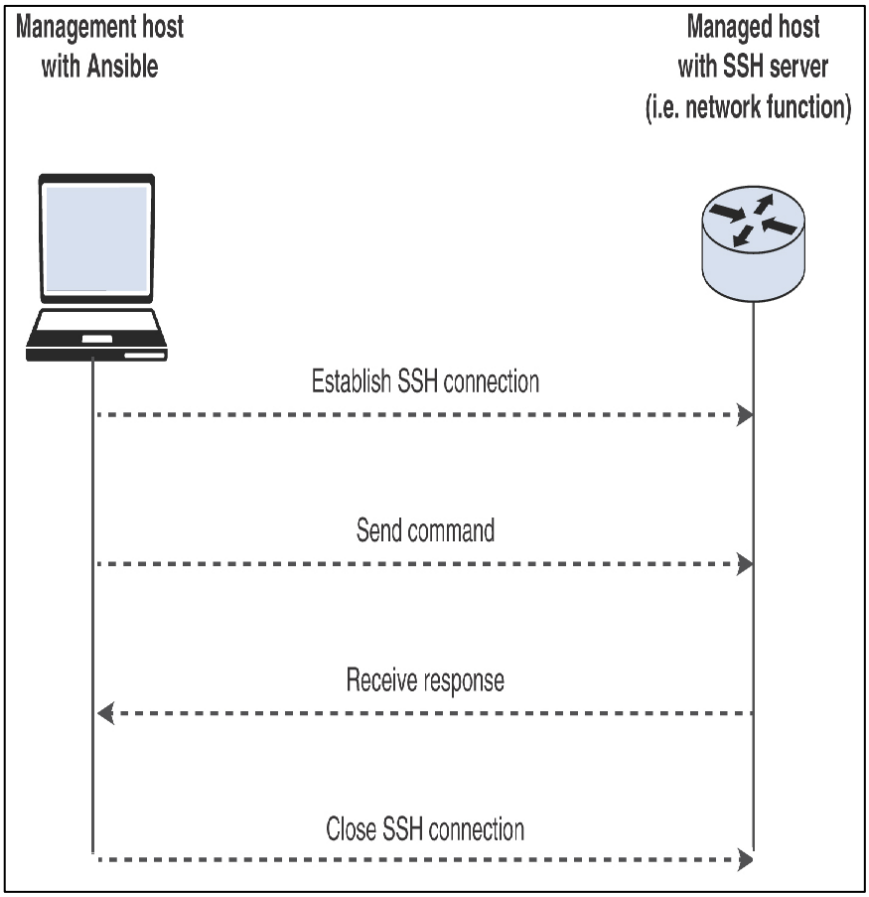


Figura 2.2 Flujo de llamadas de Ansible para un solo comando [2]

La Figura 2.2 muestra la siguiente secuencia de acciones:

- El host administrador con Ansible establece una conexión SSH con el host administrado.
- El host administrador envía los comandos para ejecutar en el host administrado. Por ejemplo, podría ser configuración o mostrar comandos en el contexto de funciones de red.
- El host de gestión recibe el resultado de la ejecución del comando desde el host gestionado. Para un comando *show*, esta es la salida de ese comando (es decir, algunos datos operativos), mientras que, para un comando de configuración, esta es una salida vacía para un éxito y un mensaje de error para una falla.
- El host administrador finaliza la conexión SSH con el host administrado. [2]

2.1.2 NetBrain

NetBrain es una solución de automatización basada en software que previene interrupciones y minimiza el tiempo necesario para aislar y diagnosticar fallos. Mediante la generación de mapas dinámicos de red se puede ver en tiempo real el flujo de comunicación punto a punto incluso en entornos híbridos (SDN, Cloud) y descubrir el diseño de red en cuestión de segundos incluyendo infinitos detalles del diseño para cualquier tecnología (desde capa 1 a capa 4). [8]

NetBrain ofrece un entorno para que los equipos codifiquen el conocimiento y las mejores prácticas, convirtiéndolo en procesos repetibles denomindos *runbooks*. Los *runbooks* ejecutables son totalmente programables ofreciendo una plataforma para que los equipos de red creen flujos de trabajo automatizados, incluyendo guías de solución de problemas, listas de verificación de seguridad y mucho más. Estos *runbooks* pueden ser compartidos con los diferentes grupos de trabajo pudiendo ser alimentados por cada uno de ellos. [8]

Un ejemplo de este tipo de Automatización de Red es la solución de NatBrain para Cisco ACI. Cisco introdujo la Infraestructura Centrada en las Aplicaciones (ACI) como la solución SDN más completa de la industria. Cisco ACI está diseñado para enfatizar la aplicación como el punto focal de la infraestructura, permitiendo una arquitectura ágil, abierta y segura, entregando automatización y gestión de políticas basadas en aplicaciones, para redes físicas y virtuales. [9]

La solución NetBrain para Cisco ACI ofrece un enfoque simple, ayudando a las empresas a realizar la transición a un centro de datos centrado en aplicaciones habilitado por Cisco ACI. Como plataforma de automatización escalable y versátil, NetBrain se integra con Cisco ACI proporcionado una visibilidad profunda y automatización de los flujos de trabajo operativos, como la monitorización, resolución de problemas y visualización de la red heterogénea. [9]

La integración también proporciona automatización de *runbook*, permitiendo a los equipos codificar soluciones a problemas conocidos y colocar estas rutinas de código en un monitor ejecutable. La Figura 2.3, muestra la forma en que NetBrain para Cisco ACI realiza el mapeo y automatización para redes heterogéneas. [9]

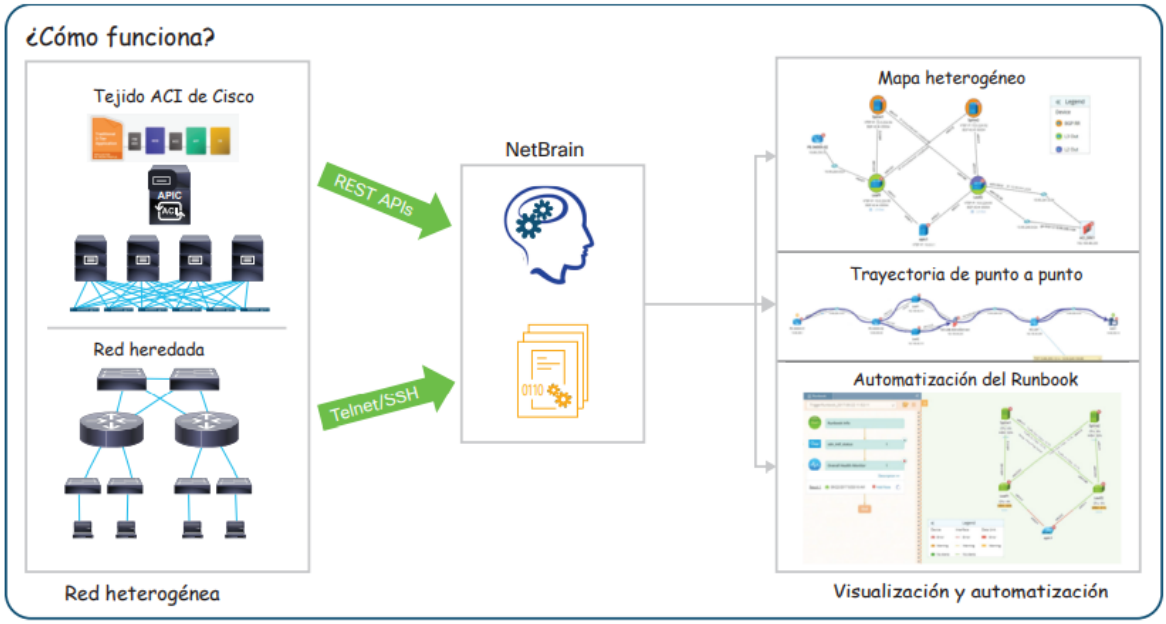


Figura 2.3 NetBrain en Cisco ACI: Mapeo y automatización para redes heterogéneas. [9]

2.2 Implementación de DevOps

La implementación de prácticas técnicas de DevOps dentro de una organización generalmente involucra tres etapas. Estas etapas siguen una progresión natural que conduce a un modelo operativo completamente automatizado desde el código hasta la implementación: [3]

- **Integración continua (CI)**: esta etapa implica fusionar el trabajo de desarrollo con el código base constantemente, para que las pruebas automatizadas puedan detectar problemas temprano. La integración continua ofrece solo un lugar donde un ser humano puede impulsar cambios, y ese es el canal de CI, específicamente la primera parte del canal conocido como revisión por pares. La Figura 2.4 muestra la implementación de un software en una canalización de CI. [2]

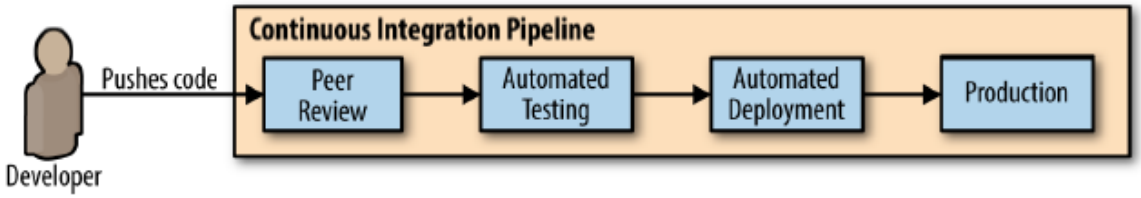


Figura 2.4 Implementación de software en una canalización de CI [2]

- **Entrega continua (CD)**: esta etapa involucra un mecanismo de entrega de paquetes de software para liberar el código a la preparación para su revisión e inspección.
- **Implementación continua**: esta etapa se basa en la integración continua, y la entrega continua para liberar automáticamente el código en producción tan pronto como esté listo.

Se puede utilizar una gran cantidad de herramientas técnicas para automatizar un entorno DevOps, desde aplicaciones comerciales hasta proyectos de código abierto de vanguardia. Debe conocer esta cadena de herramientas de DevOps, si se le pide que administre un entorno de DevOps. [3]

Independientemente de las plataformas o herramientas de DevOps que se utilicen o de si crea las suyas propias o las compra, existen algunos puntos en común para todas las canalizaciones de DevOps. Es más fácil comprender el funcionamiento de una canalización de DevOps viéndola en acción. La Figura 2.5 es una representación gráfica de lo que sucede en cada paso de una canalización de DevOps de muestra. [3]

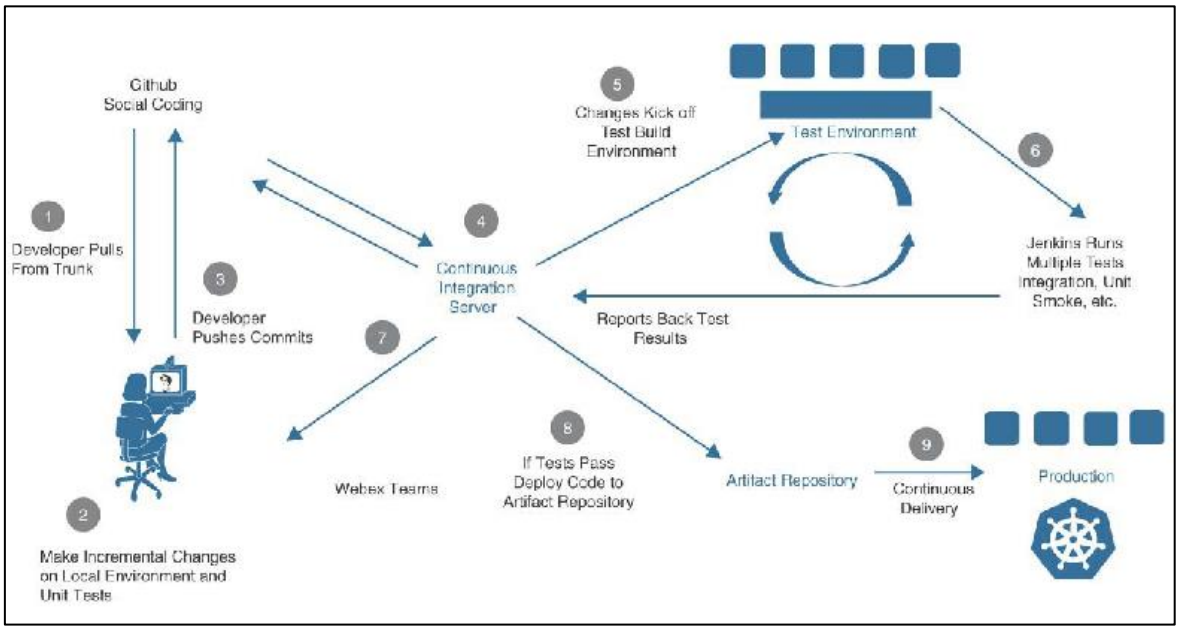


Figura 2.5 Canalización de DevOps en acción [3]

Así es como funciona la canalización:

Paso 1. El desarrollador extrae el último código del sistema de control de versiones con Git. Esto garantiza que el desarrollador tenga los cambios más recientes y no esté trabajando con una versión anterior. [3]

Paso 2. El desarrollador realiza cambios en el código, agrega nuevas funciones o corrige errores. El desarrollador también escribe casos de prueba que se utilizarán para probar automáticamente el nuevo código según los requisitos funcionales del software. El desarrollador eventualmente organiza los cambios en Git para su envío. [3]

Paso 3. El desarrollador usa Git para enviar el código y las pruebas al sistema de control de versiones (por ejemplo, GitHub), sincronizando la versión local con el repositorio de código remoto almacenado en el sistema de control de versiones. [3]

Paso 4. El servidor de integración continua, como Jenkins, tiene un inicio de sesión que supervisa GitHub en busca de nuevos envíos de código. Cuando Jenkins ve una nueva confirmación, puede extraer la última versión e iniciar un proceso de compilación automatizado utilizando los casos de prueba. [3]

Paso 5. Jenkins inicia la compilación automatizada de un entorno de prueba para la nueva compilación de software. Es posible usar secuencias de comandos de Python, Ansible u otras herramientas de automatización de infraestructura para crear el entorno de prueba lo más cerca posible de la producción. [3]

Paso 6. Jenkins ejecuta varias pruebas automatizadas, incluidas pruebas unitarias, pruebas de integración, pruebas de humo (pruebas de estrés) y pruebas de seguridad. Cuando finalizan todas las pruebas, los resultados se envían a Jenkins para su compilación. [3]

Paso 7. Jenkins envía los resultados de la prueba al desarrollador para su revisión. Se pueden enviar por correo electrónico, pero una forma más moderna de alertar al desarrollador es a través de una herramienta de colaboración como Webex Teams. Jenkins tiene complementos para todas las principales herramientas de administración de equipos y permite una fácil integración. Si la compilación falla, el desarrollador puede usar enlaces a la información sobre qué falló y por qué, realizar cambios en el código y reiniciar el proceso. [3]

Paso 8. Si el proceso de compilación es exitoso y todas las pruebas pasan, Jenkins puede implementar el nuevo código en un repositorio de artefactos. El software no se puede implementar en producción. [3]

Paso 9. Jenkins indica a la infraestructura que está lista una versión actualizada del software. Por ejemplo, puede haber un nuevo sistema operativo virtual llamado contenedor listo para implementarse en una plataforma de código abierto como *Kubernetes*. El contenedor actualizado reemplaza los contenedores existentes con el nuevo código

completamente probado y listo para usar. Ahora la aplicación se actualiza con una mínima (o ninguna) interrupción. [3]

La Figura 2.6 muestra en etapas la canalización completa de Integración y Entrega Continua (CI/CD) se pueden ver los pasos necesarios para automatizar todo el ciclo de lanzamiento del software, desde la construcción hasta la prueba y la implementación. [10]

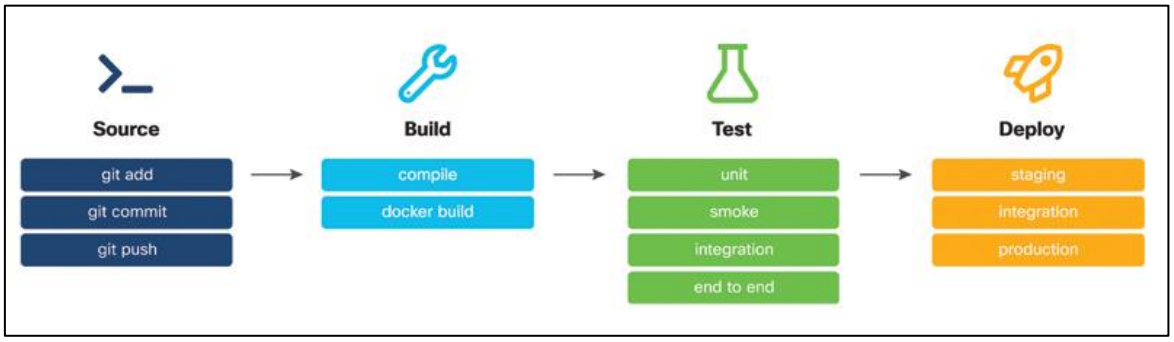


Figura 2.6 Canalización completa de CI/CD [10]

2.3 HERRAMIENTAS UTILIZADAS EN DEVOPS

El modo de trabajo de DevOps es multifuncional, involucra varias herramientas de varios tipos y propósitos, en lugar de una sola herramienta. Estas herramientas son conocidas como cadenas de herramientas DevOps.

Durante todo el ciclo de vida de la producción de software, se necesita de la ayuda de herramientas, incluyendo esta ayuda al desarrollo, la administración y la entrega.

El grupo u organización que adopta DevOps integra diversas herramientas y aplica cada una a diferentes etapas de producción incluyendo la planificación, desarrollo, verificación, empaquetado, lanzamiento, configuración, monitoreo control de versiones. [5]

2.3.1 Servidor de compilación

Un servidor de compilación es un ejemplo de una herramienta de automatización y permite que el código del repositorio de código fuente, se compile en una base de código ejecutable. Los ejemplos populares incluyen Jenkins, SonarQube y Artifactory. [4]

2.3.1.1 Jenkins

Jenkins es una plataforma de automatización de código abierto y gratuita que ayuda a mejorar el proceso de desarrollo de software, incluyendo la construcción, integración y entrega continua (CI/CD), y pruebas. [11]

Esta herramienta DevOps permite a los equipos supervisar y mejorar la eficiencia de tareas repetidas, integrar cambios con facilidad y detectar problemas rápidamente.

Al ser una herramienta DevOps basada en Java, se puede distribuir fácilmente entre las máquinas para acelerar compilaciones, pruebas y despliegues. En la figura 2.7, se muestra la página de inicio de Jenkins. [12]

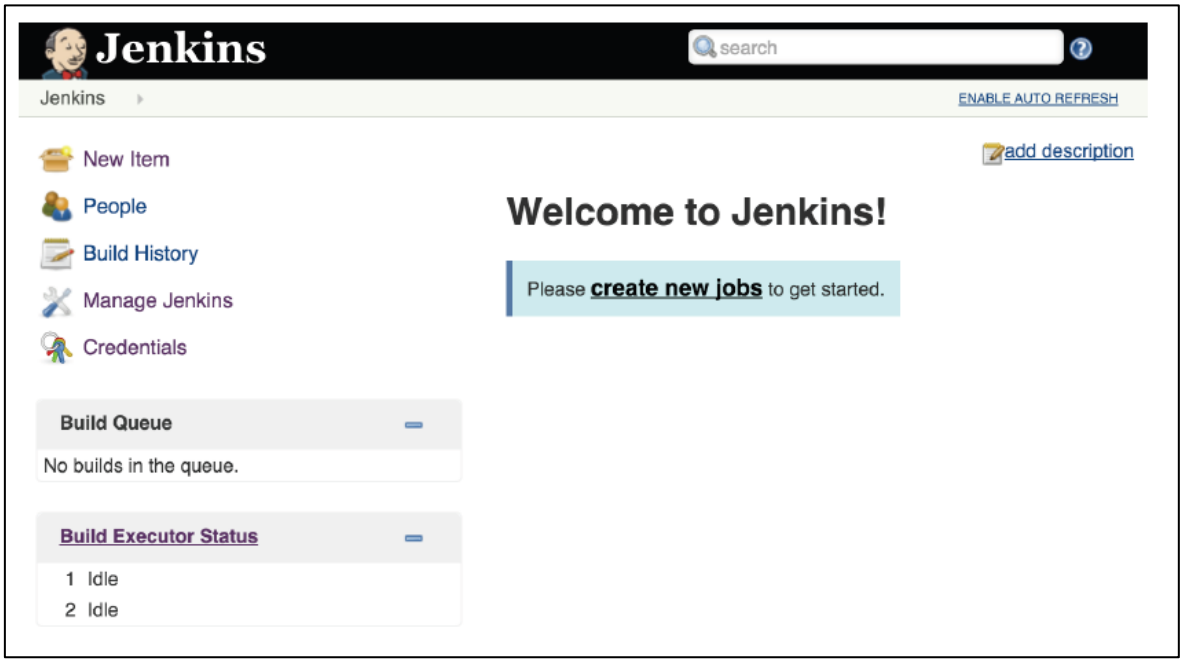


Figura 2.7 Pantalla de inicio de Jenkins después de la instalación standard [12]

Ventajas:

- Jenkins es compatible con más de 100 plugins que permiten integrarse con prácticamente cualquier herramienta de la cadena de CI/CD, como Git, Amazon EC2, Maven, etc.
- Es una aplicación autónoma escrita en Java que funciona en los sistemas operativos más utilizados, incluyendo Windows, macOS, Linux y Unix.
- Se puede configurar y establecer Jenkins de manera fácil a través de su interfaz web intuitiva, con ayudas y comprobaciones de errores integradas.

- Jenkins es altamente personalizable a través de plugins, lo que le permite realizar una amplia gama de funciones adicionales.
- Gracias a su extensibilidad, Jenkins puede ser utilizado como herramienta de CI/CD para proyectos de desarrollo de software de diversa índole. [11]

2.3.1.2 Docker

Docker es una herramienta líder en la implementación de contenedores de software utilizada por millones de desarrolladores en todo el mundo. Fue creada por Solomon Hykes y lanzada por Docker Inc. en 2013. Un contenedor está formado por capas, cada capa incluye solo lo nuevo y no se duplica nada de una capa anterior. Las capas son de solo lectura, lo que significa que están bloqueadas en su sitio. Hay una capa de lectura/escritura que se encuentra en la parte superior que puede usar para interactuar con el sistema de archivos, pero nada de lo que haga se mantendrá a menos que guarde los cambios. Si desea que los cambios sean permanentes, debe colocar otra capa encima del sistema de archivos de unión. Esto se denomina operación de copia en escritura. La Figura 2.8 muestra un ejemplo de una imagen de contenedor y sus capas. [3]

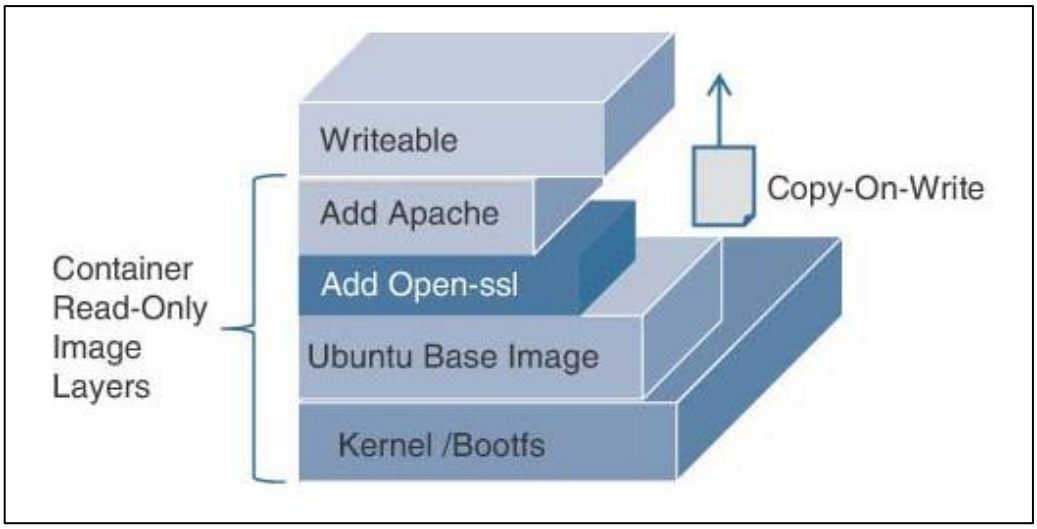


Figura 2.8 Capas de la imagen de un contenedor [3]

Como una herramienta DevOps, Docker permite a los desarrolladores construir, empaquetar y desplegar su código de manera fácil y rápida a través de contenedores que incluyen las dependencias necesarias en lugar de utilizar máquinas virtuales. Al eliminar la necesidad de configuraciones repetitivas, esta herramienta fomenta la colaboración efectiva dentro del equipo. [11]

Docker garantiza que el mismo entorno de desarrollo de software sea mantenido a lo largo de todas las etapas del ciclo de DevOps, desde el desarrollo hasta la implementación en producción. Los desarrolladores crean imágenes, Docker que se ejecutan en el entorno de desarrollo, mientras que los equipos de operaciones pueden realizar testeos y despliegues. [11]

La arquitectura Docker se compone de tres partes primarias: el cliente, el host Docker y el registro de Docker. La Figura 2.9 muestra la Arquitectura de Docker. [3]

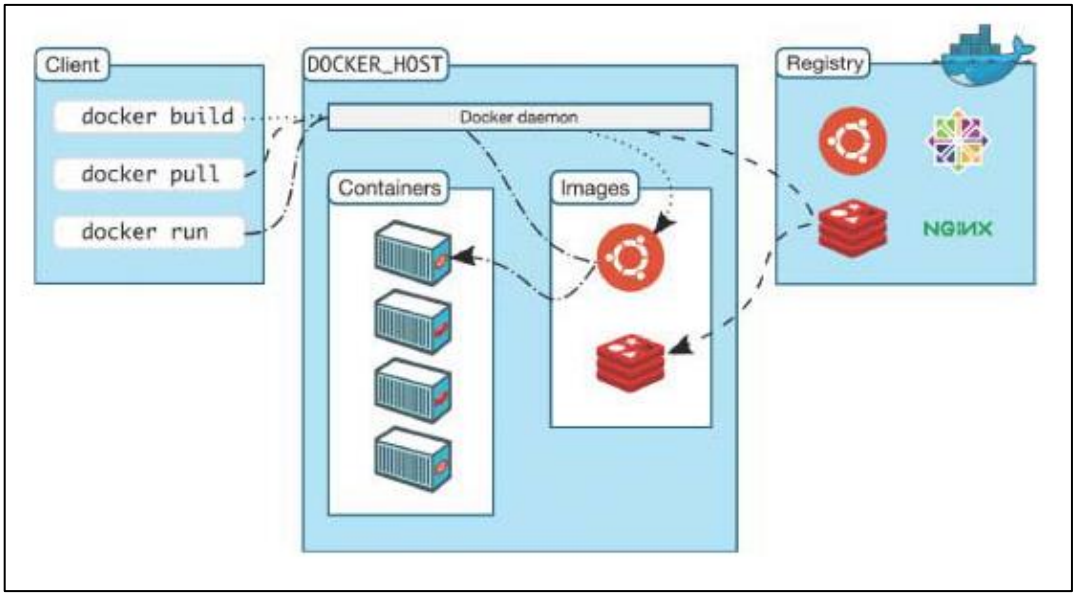


Figura 2.9 Arquitectura de Docker [3]

Ventajas:

- Docker emplea virtualización en el nivel del sistema operativo para entregar aplicaciones en paquetes llamados contenedores. Estos contenedores están separados entre sí y contienen el software, los archivos de configuración y las bibliotecas, lo que los hace transferibles y más seguros.
- Es fácil de usar con GCP y AWS y simplifica la migración a la nube.
- Docker fomenta el desarrollo distribuido.
- La herramienta hace más sencillo agregar funciones y la realización de correcciones.
- Funciona en Windows, macOS y Linux.
- Se integra bien con las herramientas de implementación como CircleCI, GitHub, etc.
- Ofrece soluciones tanto de código abierto como de pago.

- Docker es utilizado por empresas reconocidas como Netflix, Adobe, AT&T, PayPal, etc. [11]

2.3.2 Repositorio de código fuente

Un repositorio de código fuente es un elemento clave de la integración continua y sirve como un lugar donde los desarrolladores pueden administrar varias versiones de código y realizar cambios en el código sin afectar el trabajo de los demás. Las opciones para las herramientas de repositorio de código fuente incluyen Git, Cloudforce, Bitbucket y Subversion. [4]

2.3.2.1 Git

Git es una herramienta muy conocida en el mundo de DevOps, es un software de control de versiones de código abierto y gratuito. Fue lanzado en 2005 y fue creado por Linus Torvalds, que es conocido como el padre de Linux. Se escribió utilizando diversos lenguajes como C, Perl, Shell y Tcl. Creó Git como una alternativa al sistema SCM BitKeeper, debido a que el propietario de este sistema eliminó la gratuidad de su sistema para el desarrollo del kernel de Linux. [3]

Git se creó para ser rápido y escalable, con un flujo de trabajo distribuido para admitir una gran cantidad de colaboradores de kernel de Linux. En la actualidad Git se ha convertido en el sistema de gestión de fuentes más utilizado del mundo. [11]

Git es un sistema de código fuente distribuido (SCM – *Source Code Management*) que se utiliza para rastrear cambios en archivos durante el desarrollo de software y coordinar el trabajo de varios programadores de manera eficiente. Su enfoque es mejorar la velocidad, compatibilidad y la integridad de los datos en flujos de trabajo distribuidos no lineales. [11]

Cada directorio de Git en un equipo es una estructura compleja y autónoma con la capacidad de rastrear versiones y mantener un historial. Git está formado por tres estructuras:

1. **Espacio de trabajo local**: aquí es donde almacena archivos de código fuente, binarios, imágenes, documentación y cualquier otra cosa que necesite.
2. **Área de ensayo**: Esta es un área de almacenamiento intermedia para sincronizar elementos (cambios y elementos nuevos).

3. **Repositorio principal o local**: aquí es donde almacena todos los elementos confirmados.

La Figura 2.10 muestra el seguimiento de las 3 estructuras principales de Git: [3]

Figura 2.10 Estructura de árbol Git [3]

Otro concepto muy importante con Git es el ciclo de vida del archivo. Cada archivo que agrega a su directorio de trabajo y tiene un estado que se le atribuye. Este estado determina cómo Git maneja el archivo. Consta de cuatro estados y son los siguientes:

1. **Sin seguimiento**: cuando crea un archivo por primera vez en un directorio que está administrando Git, se le asigna un estado sin seguimiento. Git ve este archivo, pero no realiza ningún tipo de operación de control de versiones en él. Para todos los efectos, el archivo es invisible para el resto del mundo. Algunos archivos, como los que contienen configuraciones, contraseñas o archivos temporales, pueden almacenarse en el directorio de trabajo, pero es posible que no desee incluirlos en el control de versiones. Si desea que Git comience a rastrear un archivo, debe indicarle explícitamente que lo haga con el comando *git add*; una vez que haga esto, el estado del archivo cambia a rastreado. [3]

2. **Sin modificar**: se incluye un archivo rastreado en Git como parte del repositorio y se observan los cambios. Este estado significa que Git está observando cualquier cambio de archivo que se realice, pero aún no ve ninguno. [3]

3. **Modificado**: cada vez que agrega algún código o realiza un cambio en el archivo, Git cambia el estado del archivo a modificado. El estado modificado es donde Git ve que está trabajando en el archivo, pero no ha terminado. Debe decirle a Git que está listo para agregar un archivo cambiado (modificado) al índice o al área de preparación emitiendo el comando *git add* nuevamente. [3]

4. **En etapas**: ya agrega un archivo modificado al índice, Git debe poder agrupar sus cambios y actualizar el repositorio local. En este punto, el estado de su archivo vuelve al estado de seguimiento y permanece allí hasta que realice cambios en el archivo en el futuro y comience todo el proceso una vez más. [3]

La Figura 2.11 muestra el ciclo de vida del estado del archivo Git. [3]

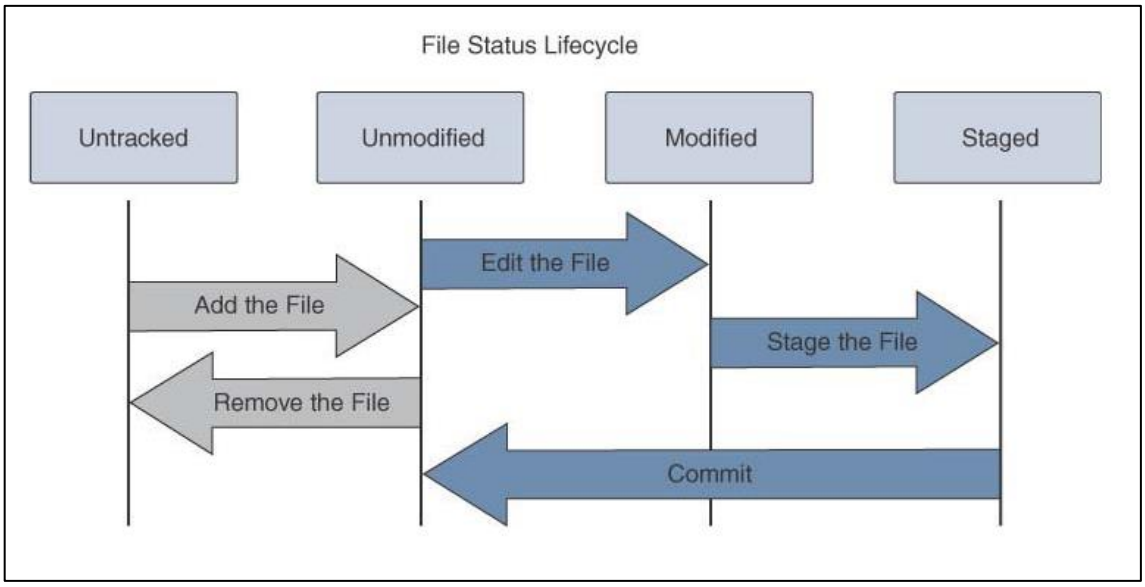


Figura 2.11 Ciclo de vida del estado del archivo Git. [3]

Ventajas:

- Es compatible con diferentes sistemas operativos, incluyendo Windows, macOS, Linux, Solaris y AIX.
- Está licenciado bajo la GPL v2.
- Es capaz de manejar proyectos de diferentes tamaños, desde pequeños hasta grandes, manteniendo la eficiencia y la velocidad.
- Es fácil de aprender y usar, tanto para los nuevos usuarios como para los expertos.
- Ofrece una ventaja competitiva con funciones como cómodos entornos de integración, flujos de trabajo múltiples, commits, los check-in y ramificaciones locales múltiples.
- Puede ser integrado en el flujo de trabajo alojando repositorios en GitHub o Bitbucket para que su equipo pueda ejecutar tareas de manera sencilla. [11]

2.3.3 Gestión de la configuración

La administración de la configuración establece y mantiene la calidad y consistencia de los requisitos, atributos funcionales y propiedades de la infraestructura de TI. Puppet, Ansible y Chef son ejemplos de herramientas de administración de configuración que se usan ampliamente. [4]

2.3.3.1 Puppet

Puppet es una herramienta de gestión de configuración que se utiliza para automatizar la configuración de servidores y dispositivos de red. Puppet se creó en 2005, y es una de las herramientas de automatización más respetadas del mercado actual. Fue creado como un proyecto de código abierto, y todavía lo es hoy, pero también está disponible como una oferta comercial llamada Puppet Enterprise que fue creada por Puppet Labs en 2011. Está escrito en Ruby y define sus instrucciones de automatización en archivos llamados manifiestos de Puppets. [3]

Mientras que Ansible no tiene agentes, Puppet se basa en agentes. Significa que se debe instalar un agente de software en cada dispositivo que se vaya a administrar con Puppet. Este es un inconveniente para Puppet, ya que hay instancias de dispositivos de red en los que los agentes de software de terceros no se pueden instalar fácilmente. [3]

Los dispositivos proxy se pueden utilizar en estas situaciones, pero el proceso es menos que ideal y significa que Puppet tiene una mayor barrera de entrada que otras herramientas de automatización. Puppet tiene una arquitectura de cliente/servidor, donde el cliente es el agente de software que se ejecuta en los dispositivos administrados y el servidor es el servidor principal de Puppet, denominado Puppet Master. [3]

Por defecto, los agentes verifican su configuración cada 30 minutos y se aseguran de que exista una coincidencia entre la configuración esperada y la configuración local. Hay agentes de software para hosts Linux y Windows, así como para varios dispositivos de red, incluidos Cisco NX-OS e IOS XR. [3]

Puppet administra los sistemas de manera declarativa, lo que significa que el administrador define el estado en el que debe estar el sistema de destino sin preocuparse de estados previos del sistema. Puppet modela el estado deseado del sistema, aplica ese estado e informa cualquier diferencia entre el estado deseado y el estado actual del sistema con fines de seguimiento. [3]

Para modelar los estados del sistema, Puppet utiliza un lenguaje declarativo basado en recursos denominado lenguaje específico de dominio (DSL) de Puppet. El estado deseado del sistema se define en este lenguaje y Puppet converge la infraestructura al estado deseado. La comunicación entre Puppet Master y los agentes se realiza a través de una conexión SSL encriptada. [3]

Varios componentes conforman la arquitectura Puppet, como se muestra en la Figura 2.12 [3]

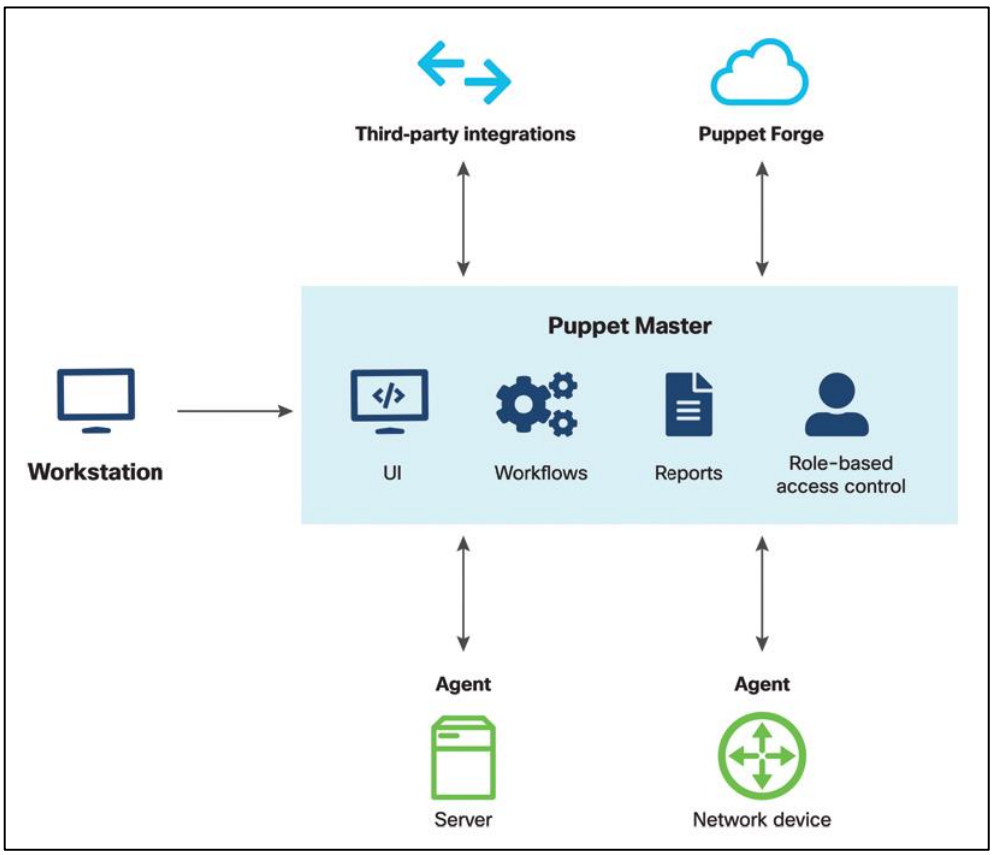


Figura 2.12 Arquitectura de Puppets [3]

El servidor de control central, Puppet Master, proporciona funciones como informes, interfaz de usuario web, seguridad y más. Los agentes de software se ejecutan en el nodo de destino que será monitoreado por Puppet. Los agentes se conectan al Maestro y recuperan la configuración correcta para el dispositivo en el que se están ejecutando. Puppet Forge es un repositorio central basado en la comunidad donde las personas pueden compartir sus manifiestos y módulos de Puppet. [3]

Para un administrador de sistema o administrador de red que usa Puppet como una solución de gestión de configuración, el primer paso es definir, usando Puppet DSL. Estas definiciones se capturan en archivos de texto llamados manifiestos de Puppet. Algunos ejemplos de estados deseados en esta etapa podrían ser la existencia de una determinada VLAN en todos los conmutadores de la red o tal vez solo un subconjunto de dispositivos, cómo y qué interfaces deben configurarse en los enrutadores de borde y qué servidores NTP usar para la sincronización de tiempo. [3]

Una vez que se define el estado deseado, Puppet ofrece la opción de simular los cambios necesarios para alcanzar ese estado y ver qué sucedería si se aplicaran los cambios. Esto da al administrador la oportunidad de ejecutar los manifiestos de Puppet y tomar nota de qué cambios se aplicarían realmente a los dispositivos. [3]

Si los cambios a realizar para alcanzar el estado deseado están en línea con las expectativas, los manifiestos se pueden ejecutar y los cambios se pueden aplicar. Los agentes de Puppet informan al Maestro con el estado de las tareas que se están ejecutando. [3]

El agente de software de Puppet que se ejecuta en los dispositivos administrados es el elemento de aplicación de la solución. El otro componente que generalmente se instala en el dispositivo administrado es “facter”, que tiene la función de recopilar datos sobre el dispositivo administrado y enviarlos al servidor de control maestro. “facter” informa al nodo de control hechos como el sistema operativo, la configuración del hardware y el número y tipo de interfaces. [3]

Puppet Master analiza los hechos, los compara con su base de datos y crea un catálogo que describe cómo se debe configurar el dispositivo administrado. A continuación, el catálogo específico del dispositivo se envía de vuelta al dispositivo. El dispositivo recibe el catálogo y compara la política con su estado actual, y si son diferentes, aplica los cambios especificados en el catálogo. Si la nueva política y el estado actual del dispositivo coinciden, no se realiza ningún cambio. Se envía un informe al Maestro después de que el agente realiza estas funciones. [3]

Los manifiestos de Puppet son archivos de texto estándar que contienen el código DSL de Puppet y tienen la extensión .pp. Contienen configuración declarativa y son descriptivos y fáciles de leer. La Figura 2.13, muestra una declaración de un puerto utilizado para garantizar que la interfaz Ethernet 1/3 esté en modo de capa 2 o de conmutación: [3]

```
# Configuring the interface using Puppet
cisco_interface { "Ethernet1/3" :
    switchport_mode     =>  enabled,
}
```

Figura 2.13 Configuración de interfaz utilizando Puppet [3]

Puppet Forge es un repositorio basado en la nube que contiene manifiestos y módulos aportados por la comunidad. Los módulos de Puppet son colecciones de archivos y directorios que contienen manifiestos de Puppet. Cuando descarga módulos de Puppet Forge, con cada módulo obtiene un grupo de subdirectorios que contienen todos los componentes necesarios para especificar el estado deseado. [3]

Ventajas:

- Puppet es una herramienta basada en modelos que requieren un uso limitado de la programación.
- Usa un lenguaje declarativo propio para describir la configuración del sistema.
- Ayuda a minimizar los errores humanos y permite al equipo a escalar la infraestructura a través del uso de código y la automatización sin agentes.
- La versión comercial de Puppet ofrece informes inmediatos, administración de nodos, orquestación, soporte de productos y control de acceso. [11]

2.3.3.2 Chef

Chef es otra solución popular de administración de configuración de código abierto que es similar a Puppet. Está escrito en Ruby, utiliza un modelo declarativo, está basado en agentes y hace referencia a sus instrucciones de automatización como recetas y libros de cocina. [3]

Una estación de trabajo Chef es un host o una computadora que se utiliza para administrar la red. Pueden existir una o más estaciones de trabajo en un entorno, según el tamaño y la complejidad. Una estación de trabajo Chef contiene todas las herramientas necesarias para desarrollar y probar las tareas de automatización de la infraestructura que se capturan en documentos llamados recetas. [3]

Una estación de trabajo Chef tiene una instancia de Chef llamado Infra Client y puede ejecutar herramientas de línea de comandos como *chef* y *knife*, así como herramientas de prueba como *Test Kitchen*, *ChefSpec* y *Cookstyle*. Una computadora de estación de trabajo también incluye *chef-repo*, el depósito central en el que se crean, prueban y mantienen recetas y libros de cocina. [3]

La Figura 2.14, representa la Arquitectura de Chef. [3]

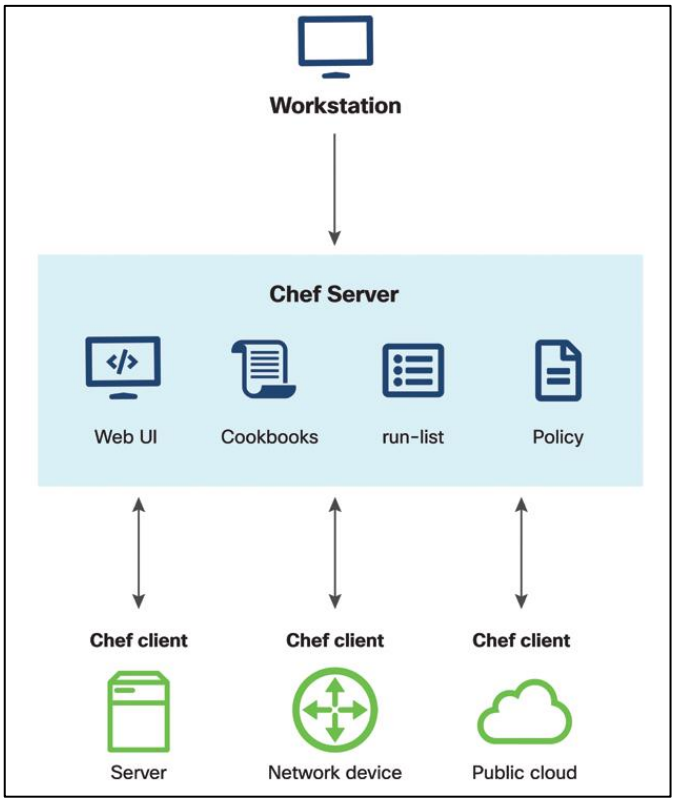


Figura 2.14 Arquitectura de Chef [3]

Los libros de cocina de Chef contienen recetas, atributos, bibliotecas, archivos, plantillas, pruebas, recursos personalizados y metadatos. *chef-repo* debe administrarse con un sistema de control de versiones como Git. Las recetas se crean en Ruby y la mayoría de ellas contienen patrones de configuración simples que se aplican a través del cliente Chef. Los libros de cocina se cargan desde la estación de trabajo a Chef Infra Server. [3]

Chef Infra Server actúa como el centro principal de toda la información de configuración. Chef Infra Client, que se instala en cada dispositivo administrado, se conecta a Chef Infra Server para recuperar los datos de configuración que se aplicarán en el dispositivo cliente administrado. Después de que finaliza cada ejecución de Chef Infra Client, los datos de la ejecución se cargan en Chef Infra Server para fines históricos y de resolución de problemas. [3]

El trabajo real de configuración del dispositivo administrado se realiza en la medida de lo posible a través de Chef Infra Client en el dispositivo administrado; descargar estas tareas de Infra Server hace que la solución Chef sea más escalable. Infra Server también indexa todos los datos de la infraestructura, incluidos los entornos, los nodos y los roles, dejándolos disponibles para la búsqueda. La consola de administración de Chef es una

interfaz basada en web a través de la cual los usuarios pueden administrar nodos, libros de cocina y recetas, políticas, roles. [3]

Los libros de recetas son los elementos fundamentales para la configuración y distribución de políticas con Chef. Un libro de cocina define un escenario y contiene todo lo necesario para respaldar ese escenario: recetas que especifican los recursos que se usarán, plantillas, valores de atributos, pruebas, metadatos, distribuciones de archivos y recursos y bibliotecas personalizados. Un gran conjunto de recursos para admitir los requisitos de automatización de infraestructura más comunes viene integrado con Chef Infra Client. Como Chef está escrito en Ruby, se pueden crear fácilmente recursos y capacidades adicionales, si es necesario. [3]

Un nodo de Chef es cualquier dispositivo que tenga instalado el software Chef Infra Client, lo que significa que es administrado por Chef Infra. Chef Infra admite una gran variedad de nodos, incluidos servidores virtuales y físicos; nodos basados en la nube que se ejecutan en nubes públicas y privadas; dispositivos de red de proveedores como Cisco, Arista, F5 y otros; y entornos de contenedores. [3]

Las funciones principales de Chef Infra Client son registrar el nodo y autenticarse en Chef Infra Server utilizando pares de algoritmos de claves públicas RSA (*Rivest, Shamir y Adleman*), sincronizar libros de recetas, configurar el nodo para que coincida con el estado deseado especificado en los libros de recetas e informar a Infra Server con informes de estado. Chef tiene una herramienta integrada en Infra Client llamada *Ohai* que se utiliza para recopilar información del sistema, como el sistema operativo, la red, la memoria, el disco, la CPU y otros datos; Ohai es similar a facter en Puppet. [3]

Al igual que Puppet Forge, Chef Supermarket es una ubicación central mantenida por la comunidad donde se crean y comparten libros de cocina entre los miembros de la comunidad. [3]

La Figura 2.15 representa a un libro de recetas de Chef de muestra utilizado para configurar una interfaz en un conmutador Cisco Nexus: [3]

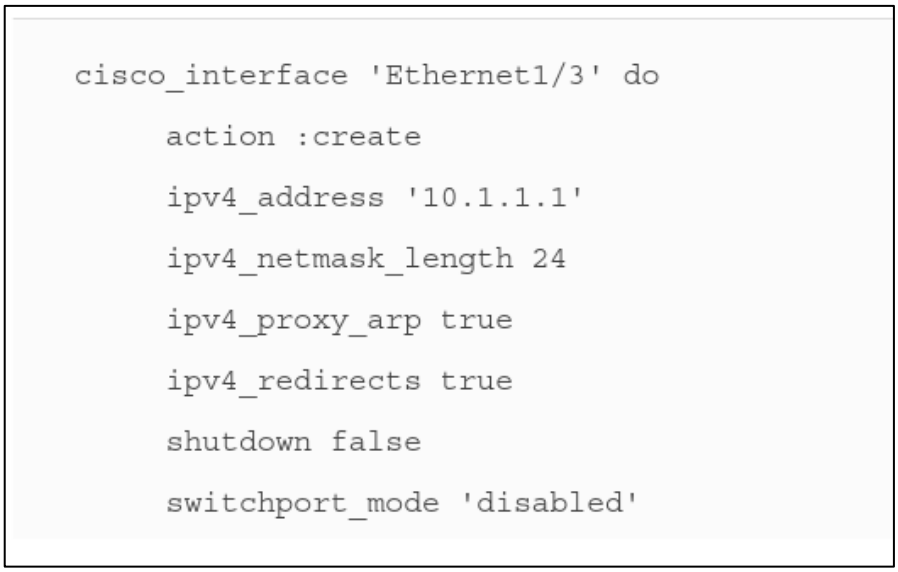


Figura 2.15 Libro de recetas de Chef en una interfaz de un conmutador Cisco [3]

Ventajas:

- Chef está desarrollado en Erlang y Ruby y usa un Lenguaje Específico de Dominio (DSL) que es puro Ruby para la gestión del sistema.
- Les permite a los usuarios ajustarse a los requisitos de empresas cambiantes con rapidez, consistencia y escalabilidad.
- Administra varios ambientes de nube y centros de datos.
- Asegura alta disponibilidad en el servidor.
- Se puede ejecutar en modo servidor/cliente, o como una herramienta de gestión independiente.
- Es compatible con múltiples plataformas, como Windows, macOS, Ubuntu, Solaris, FreeBSD, RHEL/CentOS, AIX, Fedora y Debian. [11]

2.3.4 Infraestructura virtual

Las infraestructuras virtuales son servicios basados en la nube que ofrecen infraestructura o plataforma como servicio (PaaS), como Amazon Web Services (AWS) o Microsoft Azure. Cuando se utilizan junto con herramientas de automatización, las infraestructuras virtuales admiten DevOps, al permitir a los administradores del sistema probar el nuevo código automáticamente sin interacción humana. [4]

2.3.5 Contenedores

Los contenedores de Linux son componentes de virtualización que aíslan ciertas cargas de trabajo o aplicaciones del sistema host durante el proceso de desarrollo. Docker, Kubernetes, ElasticBox y CoreOs son ejemplos de proveedores y herramientas que sirven para este propósito. [4]

2.3.5.1 Kubernetes

Kubernetes es un sistema de código abierto para orquestación, administración y escalado de software basado en contenedores. [11]

Su lenguaje de programación es Go, fue creado por Google, lanzado en el año 2014 y ahora su mantenimiento lo realiza la *Cloud Native Computing Foundation*. [11]

Es conocido como una de las mejores herramientas de automatización de DevOps para aplicaciones empaquetadas con contenedores. Kubernetes permite a los desarrolladores empaquetar y aislar clústeres de contenedores de manera lógica para un fácil despliegue en múltiples máquinas. [11]

Puede utilizarse en una arquitectura de nodo maestro-esclavo para automatizar la programación y el despliegue de contenedores mantener la conectividad de los nodos. Con Kubernetes, se puede crear contenedores Docker de manera automática y asignarlos de acuerdo con las demandas y las necesidades de escalamiento. [11]

Ventajas:

- Kubernetes lleva a cabo la implementación y retracción gradual de cambios en el software y/o su configuración, mientras monitorea su estado de salud.
- Proporciona direcciones IPs y nombres DNS a los Pods (unidad más pequeña implementado en Kubernetes), así como balanceo de la carga entre ellos.
- Ofrece una amplia variedad de opciones de almacenamiento, incluyendo nubes locales y públicas, así como soluciones de almacenamiento en red como NFS, Gluster, etc.
- Ofrece un escalado horizontal a través de la interfaz de usuario, comandos basados en el uso de la CPU.
- Cuenta con características de auto reparación que permiten reiniciar contenedores fallidos, redistribuidos y reemplazados cuando un nodo muere, y eliminar contenedores que no respondan a las pruebas de salud. [11]

3 RESULTADOS, CONCLUSIONES Y RECOMENDACIONES

En base al estudio de la Automatización de Redes y de las DevOps, en este capítulo se realiza una comparación de las soluciones de Automatización de Redes que se encuentran descritas en el capítulo 2, también se describen comparaciones entre las herramientas más importantes que utilizan las DevOps en la implementación de Automatización de Redes.

3.1 RESULTADOS

3.1.1 Comparación entre las herramientas Python y Ansible

Se realiza una comparación de las herramientas de Automatización de Redes basadas en scripts como son Phyton y Ansible. Las comparaciones no son tan alejadas unas de las otras, debido a que Ansible está basado en Phyton. Ansible con el tiempo fue adquirido por RedHat y actualmente es una de las soluciones de gestión de redes más populares del mercado. En la Tabla 3.1 se indica la comparación entre estos dos tipos de soluciones de Automatización de Redes:

Tabla 3.1 Comparación entre Phyton y Ansible

Python	Ansible
Más utilizado para el desarrollo de aplicaciones, análisis de datos, inteligencia artificial, etc.	Más utilizado para la administración y orquestación de redes (específicamente para la automatización de tareas de TI).
Es de código abierto, tiene una comunidad muy grande y activa.	Está basado en Python, por lo tanto, también es de código abierto, pero existe una versión de pago llamada Ansible Tower.
Es un lenguaje de programación completo y generalista. Es uno de los lenguajes de programación más utilizados en el mundo.	Es de fácil aprendizaje, no necesita de otro software remoto (no tiene agente).
Es todo un entorno de programación.	Es un módulo integrado a Pyhton,
Su enfoque es el desarrollo del software	Su enfoque es gestionar nodos remotos, tareas de infraestructura y configuración de servidores.

3.1.2 Comparación entre las herramientas Jenkins y Docker

Jenkins y Docker también son consideradas como herramientas de DevOps para dar soluciones a la Automatización de las Redes. Al ser servidores de compilación, se puede detallar diferencias entre estas dos soluciones. Evidentemente Docker es mucho más robusto puesto que está ambientado a servicios de virtualización de contenedores. En la Tabla 3.2 se indica la comparación entre dos tipos de servidores de compilación:

Tabla 3.2 Comparación entre Jenkins y Docker

Jenkins	Docker
Sirve para supervisión de tareas repetidas como construcción y prueba de software.	Su mayor uso es en contenedores de software que permite empaquetar y desplegar aplicaciones en entorno aislado.
Está basado en Java.	Su lenguaje de programación es Go
Compatible con varias herramientas de CI/CD.	Compatible con otras herramientas como CircleCI y GitHub.
Se ejecuta en varias plataformas como una aplicación en un servidor o una máquina virtual	También es multiplataforma pero su mayor uso es en máquinas virtuales que tengan Docker.
Es de código abierto.	Es de código abierto, pero también ofrece soluciones comerciales.

3.1.3 Comparación entre las herramientas Puppet y Chef

Al igual que Ansible, Puppet y Chef son herramientas de gestión de redes muy utilizados en DevOps, aunque no son muy utilizados sirven de mucho para configuración de servidores en una infraestructura de TI. Con el tiempo han sido desplazados por herramientas de gestión más robustas como Ansible o Cisco ACI pero tienen una comunidad de desarrolladores importante que lo sigue usando y repotenciando. La Tabla 3.3 indica la comparación entre Puppet y Chef:

Tabla 3.3 Comparación entre Puppet y Cheff

Puppet	Chef
Es de código abierto y también comercial	Es de código abierto similar a Puppet
Su lenguaje de programación es C++ Ruby y Clojure.	Su lenguaje de programación es Ruby.
Se basa en agentes en cada dispositivo con el inconveniente de ser difícil de modificar cuando se ingrese un nuevo agente de diferente software.	Se basa en agentes, pero estos se adaptan más fácil a cambios con mayor rapidez que Puppet.
Su arquitectura es maestro/ esclavo.	Su arquitectura es cliente (Chef) /Servidor (Chef Infra Server).
Los agentes son instalados en dispositivos host así como de red.	Un nodo Chef también es compatible con dispositivos host y de red.
Utiliza lenguaje declarativo.	Utiliza un lenguaje especifico de dominio.

3.1.4 Comparación entre las herramientas Ansible y NetBrain

Ansible y NetBrain son dos herramientas diferentes utilizadas en el ámbito de las TI. En Tabla 3.4 se muestra una comparación entre ambas herramientas.

Tabla 3.4 Comparación entre Ansible y Netbrain

Ansible	Netbrain
Herramienta de automatización y de configuración de sistemas que permite gestionar servidores y dispositivos en una infraestructura de TI	Plataforma de automatización de red que permite a los equipos de TI visualizar, automatizar redes complejas.
Utiliza un lenguaje de programación simple y fácil de usar para para describir la configuración deseada del sistema.	Mayor variedad de características, como la capacidad de generar mapas visuales de red en tiempo real y la automatización de tareas repetitivas
Su modelo es cliente-servidor donde el cliente envía solicitudes al servidor.	Su modelo se integra a la red con herramientas y plataformas de TI existentes.

3.1.5 Comparación entre el desarrollo sin CI y con CI

Bajo las premisas de DevOps, se han realizado varios esfuerzos para mejorar los procesos de desarrollo de software a fin de aumentar la velocidad, la confiabilidad y la precisión. Todos los procesos de desarrollo de software se pueden automatizar mediante la canalización de integración y entrega continuas (CI/CD)

La integración continua es una práctica de desarrollo de software en la que los desarrolladores envían su código a un repositorio central que forma parte de un sistema de control de versiones cada hora, día o semana o con alguna otra frecuencia acordada dentro del equipo. A medida que los desarrolladores envían su código al repositorio central, es importante verificar que el código se integre bien con la base de código completa.

A medida que el código se compromete a un repositorio central, en la mayoría de los casos, a un sistema de control de versiones de estilo Git, se activa la fase de prueba automatizada de la canalización. Los disparadores comunes en esta etapa incluyen integraciones automáticas con herramientas como Cisco Webex Teams para notificaciones inmediatas sobre el estado de las pruebas. En caso de que una prueba falle, se notifica al desarrollador y puede corregir los problemas que causaron la falla de la prueba muy temprano en el proceso de desarrollo.

La Tabla 3.5 compara los entornos de desarrollo que han implementado canalizaciones de CI y los que no lo han implementado.

Tabla 3.5 Entornos de desarrollo con y sin CI

Desarrollo sin CI	Desarrollo con CI
Mayor número de errores no detectados hasta que el código se integre con el resto del sistema.	Menos errores se van detectando y se resuelven de manera temprana antes que el código se integre al resto del sistema.
Pruebas insuficientes.	Compilaciones, pruebas, documentación e informes automatizados.
Pocos lanzamientos al año.	Múltiples lanzamientos por día.
Falta de pruebas de integración.	Servidores de compilación y prueba dedicados.
Retrasos en el proyecto.	Confirmaciones frecuentes.
Menos características en los proyectos culminados.	Más características en los proyectos finalizados.
Inestabilidad.	Estabilidad.

3.2 CONCLUSIONES

- Para conseguir resultados en DevOps es necesario mejorar el rendimiento en los flujos, la retroalimentación y en la experimentación y aprendizaje continuos, esto se consigue haciendo visible el trabajo, limitando el trabajo en curso, reduciendo el tamaño de los grupos, identificando errores, evaluando continuamente limitaciones y eliminando las dificultades en el trabajo diario durante todo el desarrollo y puesta a prueba de un sistema Automatizado de Redes.

- Mediante la retroalimentación en DevOps se evidenció que esta vía ayuda a las líneas de producción a aprender de sus propios errores y una vez aprendido del error es menos probable que vuelva a suceder, de esta forma se aumenta la eficiencia en producción y se requiere de menos personal en operaciones.

- La experimentación y aprendizaje continuo en DevOps es el método más utilizado puesto que siempre existen las fases de prueba y detección de errores en todas las etapas, además de asumir los riesgos, esta fase siempre va a estar encargada de dar solución en un tiempo determinado, siendo la vía más eficiente de las tres prácticas que se describieron en este documento.

- La automatización basada en scripts es el tipo de automatización más usado por los desarrolladores, consta de un amplio repositorio en línea como GitHub y en todo el mundo existen colaboradores en Phyton, Chef, Puppet y Ansible. Al obtener estos repositorios compartidos, un desarrollador de software en conjunto con los operadores tiene respaldo que le facilita la implementación de un sistema automatizado.

- La solución NetBrain para Cisco ACI es una de las más compleja en industria, su objetivo es automatizar redes mediante software, en el desarrollo de este trabajo se evidenció que esta solución es una de las más robustas del mercado puesto que es escalable y ágil y siempre está en constante desarrollo gracias a su integración con Cisco.

- Los servidores de compilación como Jenkins y Docker son mucha utilidad puesto que sus repositorios se manejan en sistemas operativos virtuales y en el caso de Docker tiene un uso mucho más amplo como es el uso de contenedores como

propósito final. De esta forma todo se encuentra en servidores virtuales y de fácil acceso a la nube.

- Las herramientas de Puppet y Chef son útiles para un entorno de aprendizaje, aunque su uso en gestión de redes está un poco opacado por Ansible, no hay duda de que estas dos herramientas son esenciales para poner a prueba un entorno de Automatización de Redes. No hay que descartar que la comunidad de desarrolladores utiliza mucho el lenguaje Ruby y por lo tanto estas dos soluciones siempre tendrán repositorios y una comunidad activa.

3.3 RECOMENDACIONES

- Para futuros proyectos se debe tener un conocimiento básico en lenguajes de programación como Phyton o Ruby y también del uso de plataformas para automatización como Ansible o Cisco ACI. De esta manera será más sencillo el entendimiento acerca del desarrollo de software y de la implementación de sistemas para gestión y automatización.
- Para automatizar una red empresarial, se debe realizar un estudio técnico previo para realizar una buena migración hacia una red automatizada. El beneficio a corto plazo no se va a notar, pero si se va a evidenciar a largo plazo cuando la gestión sea más sencilla.
- Se debe tomar en cuenta las actualizaciones de soluciones que ofrecen empresas como Microsoft, Amazon y también de fabricantes como Cisco, Juniper o Redhat. Estos son solo ejemplos de grandes empresas que trabajan en conjunto integrando la parte de desarrollo de software y de la implementación operativa pues cada vez mas se necesita integrar la ingeniería de datos con los equipos de DevOps

.

4 REFERENCIAS BIBLIOGRÁFICAS

[1] K. Abuelenain, J. Doyle, A. Karneliuk y V. Jain, «Network Programmability and Automation Fundamentals,» Cisco, 2021, pp. 1-5.

[2] J. Edelman, S. S. Lowe y M. Oswalt, «Network Programmability and Automation,» United States of America, O´Reilly Media, Inc, 2018, pp. 46.47,54-57.

[3] C. Jackson, J. Gooley y A. Iliesiu, «Cisco Certified Devnet Associate,» Cisco, 2021, pp. 700-723,810-813,829-836.

[4] CISCO, «¿Qué es DevOps?,» 20 Diciembre 2022. [En línea]. Available: https://www.cisco.com/c/en/us/solutions/cloud/what-is-devops.html.

[5] G. Kim, J. Humble, P. Debois y J. Willis, «The DevOps Handbook,» de *How to Create World-Class Agility, Reliability, & Security in Thecnology Organizations*, Portland, IT Revolution, 2016, pp. 44, 54, 63.

[6] C. Weekly, «ComputerWeekly.es Guia Esencial: Autoamtización de las Redes,» 06 enero 2023. [En línea]. Available: https://www.computerweekly.com/es/definicion/Automatizacion-de-redes.

[7] CISCO, «¿Qué es la Automatizacion de Red? Soluciones relacionadas para la automatización de la red,» 21 Octubre 2022. [En línea]. Available: https://www.cisco.com/c/es_mx/solutions/automation/network-automation.html.

[8] Zoostock, «Zoostock Active Network Automation,» enero 2023. [En línea]. Available: www.zoostock.com. [Último acceso: 12 Octubre 2022].

[9] Zoostock, «Zoostock Solucion Netbrain para Cisco ACI,» 06 enero 2023. [En línea]. Available: www.zoostock.com. [Último acceso: 15 noviembre 2022].

[10] A. Ratan, «Practical Network Automation,» Birmingham, Packt Publishing Ltd, 2017, pp. 7-10.

[11] kinsta, «Explora las 30 Mejores Herramientas de DevOps a Tener en Cuenta en 2022,» 11 Abril 2022. [En línea]. Available: https://kinsta.com/es/blog/herramientas-devops/.

[12] V. Farcic, «The DevOps 2.0 Toolkit Automating the Continuous Deployment Pipeline with Containerized Microservices,» 2016, pp. 195-198.