Universidad de San Carlos de Guatemala
Facultad de ingeniería
Laboratorio de Telecomunicaciones y redes locales
Ing. Danilo Escobar

# Investigación del proyecto final del laboratorio

Sergio Augusto León Urrutia
201700722
sergioleon49@gmail.com

noviembre de 2020

# INTRODUCCION

El laboratorio de Telecomunicaciones y redes locales nos incentiva a la investigación de los siguientes temas: Git, Docker, Netbox, Ansible, YAML y JSON, temas de gran importancia y que actualmente abarcan un segmento de mercado muy importante, debido a su amplia implementación en el mundo laboral, especialmente porque representan tecnologías nuevas, que permiten desarrollar trabajos o proyectos de muy buena calidad y profesionales.

El uso de GIT, para un desarrollador de sistemas o un ingeniero que pretende llevar un control adecuado de sus proyectos es indispensable, ya que es una herramienta que nos permite llevar el control de las versiones de un proyecto.

Docker por otra parte es una tecnología de contenedores, que nos permite tener servicios empaquetados, esto con el fin de poder transportarlos a cualquier sistema operativo, sin preocuparnos por compatibilidades, como la versión de Windows o Linux que tengamos.

Hablar de infraestructura de redes es hablar de caos, sin embargo, Netbox nos permite llevar un control super detallado de una infraestructura de una red, ya sea grande o pequeña, esto sin duda alguna facilita a los administradores de sistemas tener un mejor control de toda la red, además que permite encontrar fallos mas rápido, porque conocen cada nodo de la red y s configuración.

Las configuraciones de las redes tienen a ser cansadas, debido a que en ocasiones tenemos una gran cantidad de dispositivos que configurar, sin embargo, con la herramienta Ansible podemos automatizar todo este proceso, sin necesidad de programas extras mas que Python podemos realizar Playbooks que nos configuraran todos nuestros dispositivos en un abrir y cerrar de ojos.

Por ultimo y no menos importante conocemos los formatos de YAML y JSON, dos formatos estandarizados, muy populares que nos permitirían trabajar con muchas de las herramientas explicadas anteriormente, porque muchas de las instrucciones que realizamos con Ansible por ejemplo están en formato YAML y en Netbox podemos hacer consultas de nuestra infraestructura y esta consulta nos devuelve un texto en formato JSON.

# GIT

Git es un software de control de versiones, fue diseñado con el objetivo de optimizar y llevar el control del flujo de un trabajo, por medio de una especie de línea de tiempo. Este softare fue diseñado por Linus Torvals y el objetivo primordial del software es que los desarrolladores puedan controlar mejor el código fuente de los programas, teniendo herramientas para regresar a versiones anteriores, comparar versiones anteriores o trabajar en conjunto a otras personas en diferentes ramas del proyecto llamadas “Branches”.

Hoy en día Git es una herramienta fundamental cuando hablamos de desarrolladores o personas que trabajan en el área de TICs, porque es una manera muy eficiente de trabajar con los códigos de programación, proyectos de electrónica, tesis, libros, etc.

Es un software multiplataforma que puede usarse en cualquier sistema operativo, por excelencia el sistema operativo Linux es con el que se facilita mas trabajar este software, sin embargo, cada vez esta siendo mejor optimizado para que sea igual a trabajar en Linux, en Windows o en cualquier otro sistema operativo.

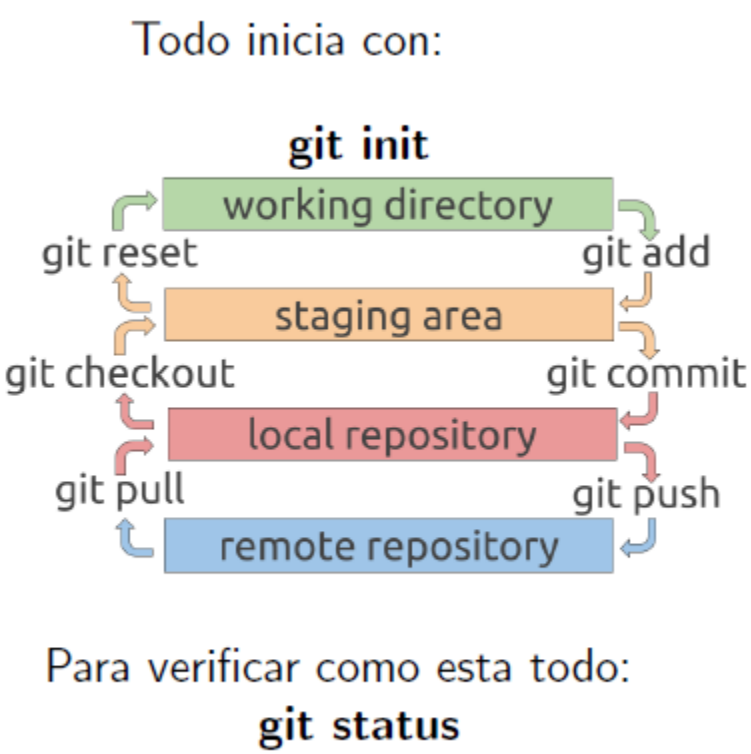
## ¿Qué se necesita para usar git?

Para usar git solo se necesita una computadora y una cuenta en algún servicio de repositorios, como Github.com bitbucket.com entre otros. Al ser un sofware open sourse la mayoría de estos servicios de almacenamiento de repositorios son gratuitos, seguramente existen algunos de pago que quizás tendrán métodos de seguridad mejores, para evitar robo de proyecto, sin embargo, desconozco de este tema, así que lo dejare por el momento como una idea.

## Conceptos importantes:

- **Repositorio**: Un respositorio es un lugar donde se almacena toda la información de nuestro proyecto, es allí donde vemos las versiones, los colaboradores, las branches, los commit y el estado actual de nuestro proyecto. En otras palabras el repositorio es básicamente nuestro proyecto.
- **Commit**: Un commit es un “screenshot” del estado actual de nuestro proyecto.
- **Branch**: Es una rama del proyecto, en un proyecto pueden haber varios colaboradores trabajando en difentes partes del programa por aparte, esto se hace haciendo una Branch para cada desarollador, esto hace que cada uno trabaje por su propia cuenta y que sus cambios no afecten a los de los demás.
- **Working directory**: donde estoy trabajando.
- **Staging area**: Agrego archivos y preparo.

## Flujo de trabajo con Git:




Para trabajar con git es importante seguir el flujo de trabajo anterior:

El primer paso es el **git init**, este paso indica que en nuestro proyecto vamos a empezar a trabajar con git, esto permite poder utilizar todos los comandos de git en nuestro proyecto.

Seguido de eso (Habiendo creado o editado archivos de nuestro proyecto) vendría el usar un **git add**, esto es para agregar los archivos de nuestro proyecto al staging área, esto es decir que agregamos esos archivos o cambios a un repositorio local, es decir que solo esta almacenado en nuestra computadora, específicamente en la carpeta en la que estamos trabajando con git.

Una vez tenemos archivos en el staging área podemos usar un **git commit**, esto haría una especia de captura de el estado actual de nuestro proyecto, al hacer esto podemos crear un comentario, que de información de este estado del proyecto, por ejemplo “Aquí el programa funciona”.

Y para finalizar se realiza un **git push**, esto es para guardar el estado actual del staging área (Repositorio local), en un repositorio remoto, al cual se puede acceder desde otra computadora, desde el navegador web, etc.

Por ultimo y esto es algo que se puede hacer en cualquier punto de nuestro flujo de trabajo es un **git status** y este nos muestra información importante del estado actual

de nuestro proyecto, por ejemplo los cambios en los archivos, si hay archivos para hacer commit, etc.

## INSTALAR GIT

**Windows**:

Para instalar git en Windows solo es de ir al siguiente link y descargar el instalador para nuestro sistema operativo, ya sea de 32 o 64 bits:

https://git-scm.com/download/win

Una vez descargado lo ejecutamos como administrador y le damos siguiente a todo y si en algún momento nos aparece una opción que diga algo así como “Add path” asegurarnos de marcarla.

**Linux**:

Abrimos una terminal y escribimos el siguiente comando:

sudo apt install git-all

Con esto tenemos instalado git en Linux

**Mac**:

Basta con abrir una terminal como en Linux y escribir los siguientes comandos:

brew install git

brew install git-gui

## EJEMPLOS DE GIT

**Antes de comenzar:**

Es importante que tengamos creada una cuenta en un servicio de repositorios remotos, como github, bitbucket, gitlab, etc. Una vez creada nuestra cuenta podemos realizar el siguiente paso, que es configurar a git para que se enlace con nuestra cuenta de repositorios remotos, para esto solo es de abrir una terminar en cualquier sistema operativo y escribir los siguientes comandos:

git config −−global user.name ”Juan Caminante”

git config −−global user.email johnyw@example.com

En el primer comando podemos poner nuestro nombre y el el segundo es importante poner el correo utilizado en nuestra cuenta de github, bitbucket o la que sea que estemos utilizando.

# Creamos un directorio y un archivo

mkdir helloworld

cd helloworld

echo "Hello World" > README

# Inicializamos el repositorio

git init

Initialized empty Git repository in /home/dave/helloworld/.git/

git add README

git commit -d "First commit"

git remote add origin git@gitlab.micronautas.com:testing.git

git push -u origin master

## COMANDOS COMUNES DE GIT

**git help**: Muestra una lista con los comandos más utilizados en GIT.

**git init**: Podemos ejecutar ese comando para crear localmente un repositorio con GIT y así utilizar todo el funcionamiento que GIT ofrece. Basta con estar ubicados dentro de la carpeta donde tenemos nuestro proyecto y ejecutar el comando. Cuando agreguemos archivos y un commit, se va a crear el branch master por defecto.

**git add + path**: Agrega al repositorio los archivos que indiquemos.

**git add -A**: Agrega al repositorio TODOS los archivos y carpetas que estén en nuestro proyecto, los cuales GIT no está siguiendo.

**git commit -m "mensaje" + archivos:** Hace commit a los archivos que indiquemos, de esta manera quedan guardados nuestras modificaciones.

**git commit -am "mensaje"**: Hace commit de los archivos que han sido modificados y GIT los está siguiendo

**git checkout -b NombreDeBranch**: Crea un nuevo branch y automaticamente GIT se cambia al branch creado, clonando el branch desde donde ejecutamos el comando.

**git Branch**: Nos muestra una lista de los branches que existen en nuestro repositorio.

**git checkout NombreDeBranch**: Sirve para moverse entre branches, en este caso vamos al branch que indicamos en el comando.

**git merge NombreDeBranch**: Hace un merge entre dos branches, en este caso la dirección del merge sería entre el branch que indiquemos en el comando, y el branch donde estémos ubicados.

**git status**: Nos indica el estado del repositorio, por ejemplo cuales están modificados, cuales no están siendo seguidos por GIT, entre otras características.

**git clone URL/name.git NombreProyecto**: Clona un proyecto de git en la carpeta NombreProyecto.

**git push origin NombreDeBranch**: Luego de que hicimos un git commit, si estamos trabajando remotamente, este comando va a subir los archivos al repositorio remoto, específicamente al branch que indiquemos.

**git pull origin NombreDeBranch**: Hace una actualización en nuestro branch local, desde un branch remoto que indicamos en el comando

# DOCKER

La idea detrás de Docker es crear contenedores ligeros y portables para las aplicaciones software que puedan ejecutarse en cualquier máquina con Docker instalado, independientemente del sistema operativo que la máquina tenga por debajo, facilitando así también los despliegues.

La pregunta ahora es ¿que es un contenedor?, la analogía de contenedor nace de los clásicos contenedores que van en los barcos de un país a otro, unas cajas gigandes que contienen un conjunto de paquetes de diferentes tipos, pero ¿Qué tiene que ver esto con software?, lo que pasa es que un contenedor es eso, un conjunto de aplicaciones contenidas en un bloque cerrado, siendo mas específicos, hay aplicaciones que para ejecutarse necesitan un conjunto de aplicaciones secundarias que hacen que esta funcione, por ejemplo puede que necesite tener instalado Java, Python, Maven, etc. Sin embargo en ocasiones es complicado tener que instalar todos estos requerimientos en nuestra computadora solo para ejecutar una aplicación, entonces se usa un contenedor, que es un paquete que contiene la aplicación en cuestión y todo lo que necesita para funcionar y al tener todo esto empaquetado en un contenedor podríamos llevarlo de un lugar a otro, porque no importaría la computadora que estemos usando, ósea si la computadora tiene instalados estos requerimientos, porque estos ya están dentro del contenedor y nos evita preocuparnos por las actualizaciones de los paquetes y todo lo relacionado a esto.

## ¿Qué beneficio tiene?

Docker es una herramienta diseñada para beneficiar tanto a desarrolladores, testers, como administradores de sistemas, en relación con las máquinas, a los entornos en sí donde se ejecutan las aplicaciones software, los procesos de despliegue, etc.

En el caso de los desarrolladores, el uso de Docker hace que puedan centrarse en desarrollar su código sin preocuparse de si dicho código funcionará en la máquina en la que se ejecutará.

*Por ejemplo, sin utilizar Docker un posible escenario podría ser el siguiente (hay otras formas de solucionar este escenario, pero por poner un ejemplo claro):*

*– Pepe tiene en su ordenador instalado Java 8, y está programando una funcionalidad específica de la aplicación con algo que solo está disponible en esa versión de Java.*

*– José tiene instalado en su máquina Java 7, porque está en otro proyecto trabajando sobre otro código, pero Pepe quiere que José ejecute el código de su aplicación en su máquina. O José se instala Java 8, o la aplicación en su máquina fallará.*

*Este escenario desaparece con Docker. Para ejecutar la aplicación, Pepe se crea un contenedor de Docker con la aplicación, la versión 8 de Java y el resto de recursos necesarios, y se lo pasa a José.*

*José, teniendo Docker instalado en su ordenador, puede ejecutar la aplicación a través del contenedor, sin tener que instalar nada más.*

*Fuente: https://www.javiergarzas.com/2015/07/que-es-docker-sencillo.html*

Por eso Docker también es muy bueno para el testing, para tener entornos de pruebas. Por un lado, es muy sencillo crear y borrar un contenedor, además de que son muy ligeros, por lo que podemos ejecutar varios contenedores en una misma máquina (donde dicho contenedor tendría el entorno de nuestra aplicación: base de datos, servidor, librerías…). Por otro, un mismo contenedor funcionará en cualquier máquina Linux: un portátil, el ordenador de tu casa, máquinas alojadas en Amazon, tu propio servidor…

Esto además beneficia a la parte de sistemas, ya como los contenedores son más ligeros que las máquinas virtuales, se reduce el número de máquinas necesarias para tener un entorno.

Y lo que es mejor, Docker es open source.

**¿Cuál es la diferencia de un contenedor y una maquia virtual?**

Aunque ambas se parecen no son iguales, porque un contenedor de Docker es mucho más ligero que una máquina virtual, además que una maquina virtual necesita un sistema operativo para funcionar y el contenedor de Docker no, porque utiliza el sistema operativo en el que esta instalado Docker, entonces Docker solo utiliza los recursos más básicos de la computadora, como la RAM, Disco duro, etc, elementos que sin importar el sistema operativo no cambian y todos aquellos elementos que complicarían la ejecución del contenedor en diferentes sistemas operativos ya van incluidos en el contenedor, así que no importan esos problemas.

## INSTALAR DOCKER

**Windows**:

Aunque anteriormente Docker solo funcionaba en Linux, hoy en día es posible utilizarlo en Windows, sin embargo es necesario tener versiones Pro o Enterprise.

Lo descargamos del siguiente link y una vez descargado lo instalamos como cualquier programa de Windows:

https://hub.docker.com/editions/community/docker-ce-desktop-windows

Si se quieren hacer pruebas para ver el funcionamiento de Docker en Windows se puede guiar de este turorial que es bastante sencillo:

https://enmilocalfunciona.io/instalando-y-probando-docker-en-windows-10/

**Linux**:

Para instalar Docker en Linux basta con seguir los siguientes comandos desde una terminal.

sudo apt update

sudo apt upgrade

sudo apt-get install  curl apt-transport-https ca-certificates software-properties-common

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add –

sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"

sudo apt update

apt-cache policy docker-ce

sudo apt install docker-ce

Una vez realizados todos esos pasos tendríamos instalado Docker, específicamente para Ubuntu 18.04

Si se quiere más información de la instalación y ejecutar unos comandos de prueba se puede seguir el siguiente tutorial:

https://www.hostinger.es/tutoriales/como-instalar-y-usar-docker-en-ubuntu/

## COMANDOS BASICOS DE DOCKER

**docker images**: para ver la lista de imágenes descargadas.

**docker ps**: para ver la lista de contenedores en ejecución.

**docker ps -a:** para ver la lista de todos los contenedores ejecutados.

**docker rm:** para borrar contenedores en nuestro equipo.

**docker rmi:** para borrar imagenes en nuestro equipo.

**docker search**: para buscar imágenes en el Docker Hub.

**docker pull**: para descargar imágenes desde Docker Hub.

# NETBOX

NetBox es un completo software para gestionar toda la infraestructura de nuestro hogar o de nuestra empresa. Además, debido a sus opciones también es posible utilizarlo en un data center para saber en todo momento cómo está la red local configurada. Esta herramienta ha sido creada por el equipo de administradores de redes de DigitalOcean, el popular servicio de Cloud Computing.

NetBox fue desarrollado por los administradores de redes y enfocado específicamente a las necesidades de la red y a la infraestructura telemática de dicho datacenter, pero ahora han decidido hacer público este grandísimo trabajo. NetBox funciona como una aplicación web basada en el framework Django y utiliza una base de datos PostgreSQL para almacenar toda la información necesaria.

La importancia de Netbox es precisamente tener bien documentada nuestra infraestructura, porque tenemos muchos parámetros de configuración que nos permiten especificar cada dispositivo de nuestra red, configurarle desde la marca y el numero de serie, hasta en lugar físico en el que se encuentra, además de asignarle valores que este tiene dentro de su configuración, como su dirección IP y otros valores.

Netbox, para el caso del proyecto final del laboratorio de telecomunicaciones y redes locales se ejecutará en un contenedor de Docker, la ventaja de tener un contenedor de netbox es su portabilidad porque podemos tener una copia de nuestra infraestructura sin importar en donde estemos, especialmente cuando hablamos de infraestructuras muy grandes donde es imposible acceder a los dispositivos físicamente.

Netbox también nos permite el uso de APIs con las cuales podremos extraer información de nuestra infraestructura, esto puede ser muy útil porque si tenemos una red muy grande sería muy difícil buscar entre toda la conexión de netbox lo que estamos buscando, entonces utilizando una aplicación como Postman, nos permitiría extraer esa información que necesitamos y poder hacer cambios o lo que sea que necesitemos en ella.

En netbox no se utilizan comandos, especialmente porque tiene una interfaz grafica muy amigable para el usuario y la interacción es muy sencilla y vaya que tiene que ser así, porque configurar toda esa información desde un CLI seria un dolor de cabeza.

# API

API significa interfaz de programación de aplicaciones. Las API permiten que sus productos y servicios se comuniquen con otros, sin necesidad de saber cómo están implementados. Esto simplifica el desarrollo de las aplicaciones y permite ahorrar tiempo y dinero.

Las API le otorgan flexibilidad; simplifican el diseño, la administración y el uso de las aplicaciones, y proporcionan oportunidades de innovación, lo cual es ideal al momento de diseñar herramientas y productos nuevos (o de gestionar los actuales).

A veces, las API se consideran como contratos, con documentación que representa un acuerdo entre las partes: si una de las partes envía una solicitud remota con cierta estructura en particular, esa misma estructura determinará cómo responderá el software de la otra parte.

Las APIs están en todos lados, especialmente cuando hablamos de aplicaciones que son ejecutadas en diferentes sistemas operativos o en diferentes marcas, un ejemplo de esto es el sistema Android, Android es un sistema operativo propietario de Google que se los vende a muchas empresas, como Samsung, LG, Xiaomi, etc. El problema de Android es que cada fabricante utiliza diferentes procesadores, diferentes cámaras, bocinas, etc, esto tiene como consecuencia que el sistema tiene que ser “Modular”, para cualquier marca, y es aquí donde entran las APIs, porque permiten a los desarrolladores utilizarlas y en base a ellas editar el sistema operativo y controlar cada cosa a su manera.

En resumen, las API le permiten habilitar el acceso a sus recursos y, al mismo tiempo, mantener la seguridad y el control. Cómo habilitar el acceso y a quiénes depende de usted. Seguridad de las API. Para conectarse a las API y crear aplicaciones que utilicen los datos o las funciones que estas ofrecen, se puede utilizar una plataforma de integración distribuida que conecte todos los elementos, incluidos los sistemas heredados.

Un ejemplo de una API podría ser el siguiente: si estás creando una aplicación que es una tienda online, no necesitarás crear desde cero un sistema de pagos u otro para verificar si hay stock disponible de un producto. Podrás utilizar la API de un servicio de pago ya existente, por ejemplo PayPal, y pedirle a tu distribuidor una API que te permita saber el stock que ellos tienen.

# YAML

YAML es un formato de serialización, que los humanos pueden entender, si significado en español es “No es otro lenguaje de marcado”, es llamado así porque un lenguaje de marcado puro podría ser XML, y YAML no lo es, un un código semi marcado.

YAML es utilizado mucho para configuraciones en Linux, es ejecutado línea por línea y es importante tomar en cuenta la identacion, la cual es realizada con espacios en blanco, no se utiliza la tecla tabulación como se utiliza en Python.

Fue desarrollado en el año 2001 y desde entonces se utiliza muchísimo en el mundo del desarrollo o de los administradores de sistemas.

YAML fue creado bajo la creencia de que todos los datos pueden ser representados adecuadamente como combinaciones de listas, hashes (mapeos) y datos escalares (valores simples). La sintaxis es relativamente sencilla y fue diseñada teniendo en cuenta que fuera muy legible pero que a la vez fuese fácilmente mapeable a los tipos de datos más comunes en la mayoría de los lenguajes de alto nivel. Además, YAML utiliza una notación basada en la sangría y/o un conjunto de caracteres Sigil distintos de los que se usan en XML, haciendo que sea fácil componer ambos lenguajes.

- Los contenidos en YAML se describen utilizando el conjunto de caracteres imprimibles de Unicode, bien en UTF-8 o UTF-16.
- La estructura del documento se denota indentando con espacios en blanco; sin embargo no se permite el uso de caracteres de tabulación para sangrar.
- Los miembros de las listas se denotan encabezados por un guion ( - ) con un miembro por cada línea, o bien entre corchetes ( [ ] ) y separados por coma espacio ( , ).
- Los vectores asociativos se representan usando los dos puntos seguidos por un espacio. en la forma "clave: valor", bien uno por línea o entre llaves ( { } ) y separados por coma seguida de espacio ( , ).
- Un valor de un vector asociativo viene precedido por un signo de interrogación ( ? ), lo que permite que se construyan claves complejas sin ambigüedad.
- Los valores sencillos (o escalares) por lo general aparecen sin entrecomillar, pero pueden incluirse entre comillas dobles ( " ), o comillas simples ( ' ).
- En las comillas dobles, los caracteres especiales se pueden representar con secuencias de escape similares a las del lenguaje de programación C, que comienzan con una barra invertida ( \ ).
- Se pueden incluir múltiples documentos dentro de un único flujo, separándolos por tres guiones ( --- ); los tres puntos ( ... ) indican el fin de un documento dentro de un flujo.

- Los nodos repetidos se pueden denotar con un ampersand ( & ) y ser referidos posteriormente usando el asterisco ( * )
- Los comentarios vienen encabezados por la almohadilla ( # ) y continúan hasta el final de la línea.
- Los nodos pueden etiquetarse con un tipo o etiqueta utilizando el signo de exclamación ( ! ) seguido de una cadena que puede ser expandida en una URL.
- Los documentos YAML pueden ser precedidos por directivas compuestas por un signo de porcentaje ( % ) seguidos de un nombre y parámetros delimitados por espacios.. Hay definidas dos directivas en YAML 1.1:
- La directiva %YAML se utiliza para identificar la versión de YAML en un documento dado.
- La directiva %TAG se utiliza como atajo para los prefijos de URIs. Estos atajos pueden ser usados en las etiquetas de tipos de nodos.

# JSON

JSON, cuyo nombre corresponde a las siglas JavaScript Object Notation o Notación de Objetos de JavaScript, es un formato ligero de intercambio de datos, que resulta sencillo de leer y escribir para los programadores y simple de interpretar y generar para las máquinas.

JSON es un formato de texto completamente independiente de lenguaje, pero utiliza convenciones que son ampliamente conocidos por los programadores, entre ellos:

- C
- C++
- C#
- Java
- JavaScript
- Perl
- Python
- Entre otros

Dichas propiedades hacen de JSON un formato de intercambio de datos ideal para usar con API REST o AJAX. A menudo se usa en lugar de XML, debido a su estructura ligera y compacta.

Muchos lenguajes de programación proporcionan métodos para analizar una cadena de texto con este formato en un objeto nativo y viceversa.

Según la descripción de Stack Overflow, JSON “define seis tipos de valores: nulo, números, cadenas, booleanos, matrices y objetos”.

Pese a su nombre, no es necesariamente parte de JavaScript, de hecho, es un estándar basado en texto plano para el intercambio de datos, por lo que se usa en muchos sistemas que requieren mostrar o enviar información para ser interpretada por otros sistemas.

Una de las características de JSON, al ser un formato que es independiente de cualquier lenguaje de programación, es que los servicios que comparten información por este método no necesitan hablar el mismo idioma, es decir, el emisor puede ser Java y el receptor Python, pues cada uno tiene su propia librería para codificar y decodificar cadenas en este formato.

Podemos concluir entonces en que JSON es un formato común para ‘serializar’ y ‘deserializar’ objetos en la mayoría de los idiomas.

# ANSIBLE

Ansible es una herramienta de automatización, sirve para configurar y administrar ordenadores, utiliza un sistema de Multi Nodos, esto quiere decir que permite desplegar configuraciones de servidores o de servidores por lotes.

La ventaja de Ansible es que no requiere de programas para su funcionamiento, solo Python 2.4 o superior, con esto puede manejar cadenas de tipo JSON y su salida puede ser escrita en cualquier lenguaje, pero nativamente utiliza YAML para escribir las características reusables del sistema.

**Características:**

Aprovisionamiento: Con Ansible se pueden aprovisionar las últimas plataformas en la nube, host virtualizados e hipervisores, dispositivos de red y servidores físicos.

Gestión de la configuración: Establece y mantiene el rendimiento del producto, al registrar y actualizar la información que describe el software y hardware de una empresa. Esta información generalmente incluye las versiones y actualizaciones que se han aplicado a los paquetes de software instalados y las ubicaciones y direcciones de red de los dispositivos de hardware.

Despliegue de aplicaciones: Cuando se define la aplicación con Ansible y se maneja su despliegue con Ansible Tower es posible llevar un control de todo el ciclo de vida de una aplicación. Desde desarrollo hasta producción.

Seguridad y Cumplimiento: Ansible permite definir la seguridad en los sistemas de forma sencilla. Utilizando la sintaxis de un Playbook es posible definir reglas de firewall,gestión de usuarios y grupos y políticas de seguridad personalizadas en los sistemas que se estén gestionando y además posees un gran número de módulos que ayudan en la labor.

Orquestación: Ansible se usa para orquestar los despliegues de OpenStack por ejemplo. Compañías como Rackspace, CSC, HP, Cisco e IBM confían en Ansible para mantener sus nubes OpenStack disponibles de manera simple y segura.

**Playbook:**

Los Playbooks describen configuraciones, despliegue, y orquestación en Ansible.14 El formato del Playbook es YAML.15 Cada Playbook asocia un grupo de hosts a un conjunto de roles. Cada rol está representado por llamadas a lo que Ansible define como Tareas.

Básicamente una tarea no es más que una llamada a un módulo de Ansible.

Al componer un Playbook con múltiples "jugadas", es posible orquestar despliegues de múltiples máquinas, ejecutando ciertos pasos en todas las máquinas del grupo de servidores web, otros pasos en el grupo de servidores de base de datos, y luego más comandos en los servidores web, etc.

**Lista de tareas:**

Cada jugada contiene una lista de tareas. Las tareas son ejecutadas en orden, de una en una, contra cada máquina que encaja con el patrón del host, para luego seguir con la próxima tarea. Es importante entender que, dentro de una jugada, todos los hosts van a obtener las mismas directivas de la tarea. Es el objetivo de un Playbook el mapear un grupo de host a tareas.

Al correr una jugada, que corre de arriba hacia abajo, los host donde fallen las tareas son sacados de la rotación de las jugadas restantes. Si las cosas fallan, simplemente hay que corregir el Playbook y ejecutar nuevamente.

El objetivo de cada tarea es ejecutar un módulo, con parámetros muy específicos. Variables, como ya se mencionó, pueden ser usadas como argumentos de los módulos.

Los módulos son idempotentes, lo que significa que si se los ejecuta de vuelta, solo van a generar los cambios en el sistema que sean necesarios para llegar al estado deseado. Esto da seguridad para ejecutar el mismo Playbook varias veces. No va a cambiar nada salvo que sea necesario hacerlo.

Cada tarea debe tener un nombre, que está incluido en la salida de la ejecución del Playbook. Esta es una salida para humanos, por lo cual es deseable tener una buena descripción de cada paso de la tarea.

Las tareas pueden declararse usando el formato antiguo action : module options, pero es recomendable usar el formato más convencional module : options. Este es el formato recomendando en la documentación, pero se pueden encontrar Playbook con el formato antiguo.

**Conceptos importantes:**

- **Facts**: Información útil de los clientes.
- **Inventario**: Incluye información, estática o dinámica, de los clientes administrados y su información. En ocasiones al inventario, también se le conoce como hostfile.
- **Módulos**: Son las librerías que se utilizan para controlar elementos como ficheros, servicios paquetes o comandos. Estos se copian al nodo cliente para que ejecute la tarea indicada.
- **Nodo**: Objeto a administrar, ya sea un servidor, un router y otros elementos.
- **Play**: Listas de tareas a realizar en los clientes especificados en el Playbook.
- **Playbook**: Se encarga de definir todas las tareas que debemos realizar sobre un conjunto de hosts clientes. Esta lista de tareas se guarda en un archivo con formato yaml.
- **Roles**: Es una agrupación de tareas, ficheros y plantillas, que pueden ser reutilizados.
- **Tareas**: Definición de una acción a realizar.
- **Nodo de control**: El nodo de control es cualquier máquina donde esté instalado Ansible.
- **Nodo gestionado**: Cualquier máquina o dispositivo gestionado con Ansible. Los nodos gestionados también se conocen como hosts. En los nodos gestionados no está instalado Ansible.

## CONCLUSIONES

1. El uso de Git para los proyectos de programación es casi indispensable, especialmente ahora (Noviembre de 2020), por el confinamiento del coronavirus es importante llevar un control de nuestros proyectos en grupo de manera remota, con Git esto se facilita de gran manera y no implica un costo adicional.

2. Docker es una herramienta muy compleja, cuyo uso tiene una curva de aprendizaje bastante empinada, sin embargo, es una herramienta muy útil en lo que respecta a movilidad de aplicaciones o de servicios.

3. Netbox y Ansible son dos herramientas que pueden ir de la mano en todo momento, porque una nos permite automatizar nuestra red y la otra nos permite tener un control de ella, tener estas dos herramientas son una pareja excelente.

## RECOMENDACIONES

1. Los estudiantes de Ingeniería electrónica de la Universidad de San Carlos de Guatemala deberían implementar Git en sus proyectos, porque nos permite tener control de todo nuestro proceso y nos ayuda a trabajar en conjunto con nuestros compañeros de manera remota, especialmente en los Laboratorios porque los proyectos se trabajan con tres personas.

2. La facultad de Ingeniería debería brindar cursos gratuitos para los estudiantes y que puedan aprender a utilizar Docker, porque es una herramienta muy moderna, que permitiría a la Facultad crecer y darle a los egresados herramientas que sin duda les servirían en el mundo laboral.

3. La escuela de Mecánica eléctrica debería considerar agregar cursos en los que se ponga mas importancia a la automatización, en general es una rama muy interesante para los estudiantes de Ingeniería electrónica, porque el campo es muy interesante y sin duda serian temas muy interesantes para los estudiantes.