Revisión del estado del arte de los lenguajes para especificación de sistemas masivamente distribuidos en la nube

Sergio Andrés Rodríguez Torres, Estudiante de ingeniería de sistemas
Escuela colombiana de ingeniería Julio Garavito

***Resumen*—A lo largo de este trabajo investigativo vamos a abarcar la evolución de la gestión de operaciones de TI así como los lenguajes de esppecificación usados para crear automaticamente infraestructura para sistemas masivamente distribuidos en la nube.**

***Palabras Claves*—Infraestructura, automatización, contenedores, maquinas virtuales, orquestadores**

I. Introduction

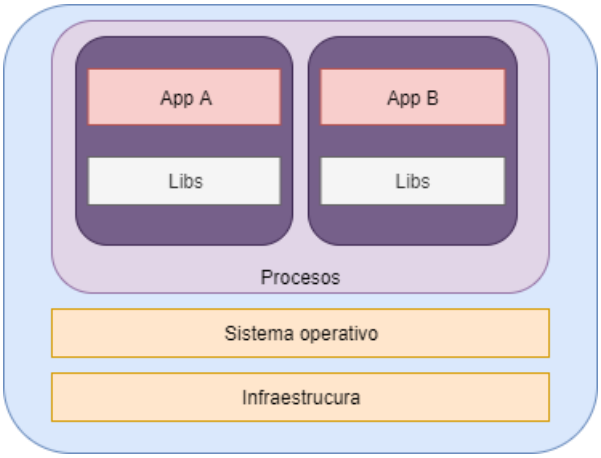
La gestión de operaciones de TI consiste en controlar y monitorear los servicios e infraestructuras de TI. La Gestión de Operaciones de TI lleva a cabo tareas relacionadas con la operación de componentes y aplicaciones de infraestructura.

Esto incluye la programación de trabajos, actividades de soporte y restauración y el mantenimiento rutinario de la infraestructura.

II. Estado del arte

En la infraestructura tradicional, los sistemas tradicionales se pueden dividir en: infraestructura, sistema operativo, y los procesos que corren sobre el Kernel del sistema operativo como lo evidencia la figura 1.

Inicialmente se instalaba el hardware y el software que requiriera el servidor, pero esto genera una arquitectura rígida sin posibilidad de escalar, que requería una inversión inicial elevada en hardware que con el tiempo se iba a hacer obsoleto, adicionalmente mantener hardware requiere de recursos y tiempo que no todas las empresas se lo podían permitir, además se alejaba de los objetivos organizacionales, por lo que surgió la necesidad de tener infraestructura como servicio (IaaS) y de esta forma contratar servicios de cloud computing con terceros y que fueran estos terceros los encargados de mantener el hardware disponible y actualizado.



Los primeros proveedores de este servicio se enfrentaron con la necesidad de poder proveer máquinas de la capacidad adecuada a cada cliente de acuerdo con sus necesidades, no era viable comprar muchas máquinas de capacidades variadas para alquílalas a los clientes y surge la idea de dividir una maquina física de gran capacidad en varias virtuales y así poder hacer gestión de los recursos que se le asignaban a cada una de estas máquinas y tener un hardware que fuera flexible a la necesidad del cliente.

II-A. Virtualización

Empecemos por discutir que es la virtualización, y es que la definición de virtualización ha evolucionado a lo largo de los años, en la década de los 60 y 70 se consideraba virtualización al método de dividir lógicamente los mainframes para permitir la ejecución simultanea de varias aplicaciones, pero cerca a los 90 con la adopción de los sistemas operativos y la reducción de costos de los sistemas x86 se hizo posible la idea de simular un ambiente completo con el sistema operativo, aplicaciones, librerías y procesos separados y lo más importante

que este ambiente virtual fuera indistinguible de uno físico, esta práctica conllevaba un desperdicio de recursos computacionales de solo un 10-15 % aun dividiendo en muchas maquinas, lo que nos permite abstraer el hardware físico convirtiendo un servidor en muchos servidores. [1]

II-A1. Hipervisores: El hipervisor también o VMM es una capa de software que se encarga de supervisar y crear los ambientes virtuales, además gestiona los recursos a necesidad de las máquinas virtuales y garantizar ambientes separados para las máquinas virtuales, básicamente es la capa que permite la virtualización, estos se pueden categorizar en dos tipos.[1][2]

II-A1a. Tipo 1: Conocido también como hipervisor nativo ya que este interactúa directamente con el hardware o "metal desnudo", opera a bajo nivel sin necesidad de un sistema operativo, este se caracteriza por tener un bajo sobre costos de recursos, pero también se restringe por el muy limitado hardware compatible con esta técnica y las características que ofrece no son tan flexibles o variadas como lo son en su contraparte tipo 2. Principalmente se encarga de gestionar los recursos como el disco, CPU, memoria y periféricos a los sistemas virtuales. [2]

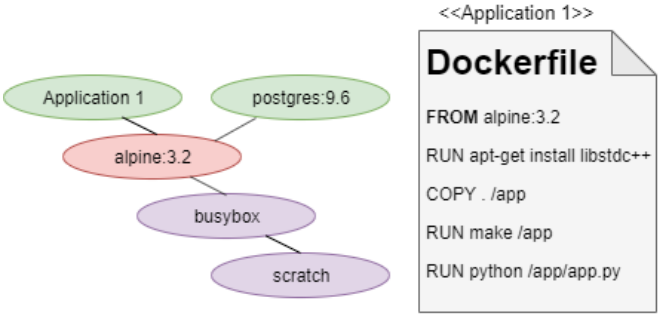
II-A1b. Tipo 2: Este tipo de hipervisor requiere de un sistema operativo completo sobre el cual correr, es decir se instala sobre el sistema operativo, esto permite que haya menos problemas de compatibilidad que dependan del hardware ya que es el sistema operativo el que se encarga de interactuar con el hardware con los controladores que le correspondan, pero esto conlleva un mayor sobre costo de recursos. Este tipo de hipervisores tiene múltiples propósitos como la portabilidad de aplicaciones sobre múltiples plataformas como es el caso de la Java Virtual Machine (JVM). [2]

II-B. Contenedores

Los contenedores son una alternativa de virtualización de aplicaciones y consiste en colocar una capa extra entre el sistema operativo y las aplicaciones, lo que consigue desacoplar las aplicaciones del sistema operativo en el que están corriendo, dando una mayor portabilidad a las aplicaciones y además permitiendo usar múltiples versiones de las aplicaciones en el mismo sistema operativo al tiempo, esto disminuye los problemas de compatibilidad al permitir aplicaciones incompatibles coexistir en el mismo sistema operativo. [3]

II-B1. ¿Porque surgen los contenedores?: En la búsqueda encontrar un mecanismo que permitiera encapsular los procesos que corren en una misma máquina, colocar un sandbox que permitiera aislar por completo cualquier variable del sistema operativo o la infraestructura física, es decir se genera un namespace específico para ese proceso, además nos permite colocar ciertas restricciones sobre lo que pueda exceder el sandbox, como por ejemplo los recursos de los que puede disponer. Esa es la noción más básica de un contenedor. [3]

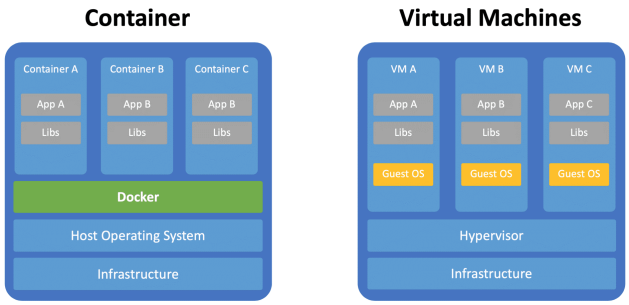
II-B2. Imágenes de un contenedor: El término contenedor también puede usarse para referirse a una imagen de un contenedor, estás imágenes por lo general poseen una jerarquía padre-hijo, esto se debe a que las imágenes se suelen construir unas sobre otras permitiendo reutilizar el estado de una imagen que posee ciertas herramientas y sobre ellas agregar nuevas cosas, esto genera una jerarquía en forma de árbol. [4]



Un ejemplo del uso de esta jerarquía es el dockerfile, en el dockerfile se encuentra la especificación de un ambiente para un contenedor en un archivo de texto, la palabra clave From se refiere a que imaginé en árbol de jerarquía se va a usar como base para la definición del ambiente, este ambiente es generado como una nueva imagen. [4]

Los contenedores además permiten la noción de estar manejando imágenes, sin necesidad de que se encuentren instaladas directamente en el sistema operativo, sino que se cargan cada vez que se inicia el conteiner, por lo que podemos tener múltiples versiones de múltiples aplicaciones con librerías y requerimientos de diferentes versiones incluso incompatibles en un mismo sistema operativo si están en contenedores diferentes, como son solo imágenes que no tienen contacto directo con el

sistema operativo nos permiten tener siempre un ambiente limpio en el cual en cualquier momento podemos eliminar o remplazar la imagen que usamos para nuestra aplicación sin necesidad de tener que eliminar o remplazar programas o componentes del sistema operativo. [3][4]



II-B3. Contenedores vs máquinas virtuales: Realmente no hay punto de comparación directa ya que ambas tecnologías buscan solucionar problemas diferentes, inclusive se pueden utilizar las dos en conjunto, las máquinas virtuales buscan hacer una separación virtual de la infraestructura, lo que permite tener todas las herramientas y el escritorio de una máquina tradicional de forma artificial, por otro lado los contenedores buscan una forma de aislar una aplicación o proceso a tal punto que podemos utilizar imágenes y tener ambientes estériles de cualquier afectación de un agente externo como el sistema operativo o la infraestructura y nos permite modificar y hacer control de versiones de forma mucho más sencilla al no haber un proceso de instalación que ensucie el sistema operativo, ofreciendo la facilidad de cambiar una versión con otra, además ofrece solución al problema en el que el código se comporta de una forma en local y de otra forma en producción ya que tanto en local como en producción lo vamos a correr sobre el mismo contenedor por lo que va a ser transparente para la aplicación la máquina o el sistema operativo en la que se ejecute. [5]

II-B4. Docker: Docker es la alternativa más usada para realizar contanerización, esto principalmente a que fue de los primeros en el mercado y en ofrecer un sistema multiplataforma, además de contar con la comunidad más grande y un vasto ecosistema de herramientas y aplicaciones que se encuentran disponibles en Docker hub, la librería de imágenes docker, esta se puede consultar desde docker cli. [6]

II-B5. LXD: Los contenedores de Linux fueron la primera alternativa a la contanerizacion, surgieron en el 2008 y contaban con soporte nativo del sistema operativo, trayendo a la luz el concepto de contenedor, su principal debilidad es que se restringen a el sistema operativo, por lo que no son tan portables como la competencia que ofrece contenedores ligeros multiplataforma. [6]

II-B6. RKT: Es la alternativa de contenedores Linux con CoreOS y busca brindar funcionalidades específicamente orientadas a la seguridad de las aplicaciones. Entre estas destaca KVM (Kernel-based Virtual Machine) que como su nombre lo indica busca aislar por completo las aplicaciones simulando un Kernel diferente para cada contenedor. [6]

II-C. Orquestadores

Con el surgimiento de los contenedores y la ventana que se abrió al permitir un versionamiento de forma muy sencilla en comparación a los métodos tradicionales de instalación de aplicaciones, al disponer de contenedores que podemos colocar dónde y cuántos queramos en una máquina, esto facilitó los requerimientos de infraestructura para hacer crecimiento de los recursos en horizontal, es decir crear más instancias o contenedores de la aplicación y luego repartir la carga, pero para repartir la carga y garantizar una cierta configuración de instancias se requería de un sistema que orqueste las instancias, a dichos sistemas se les conoce como orquestadores de contenedores, estos además nos permiten hacer escalamiento automático en base a la carga que está recibiendo la aplicación. Su aparición revolucionó la industria y hizo mucho más flexibles la infraestructura y la forma en la que se desarrollan aplicaciones. [3][7]

Ahora vamos a ver algunas alternativas:

II-C1. Kubernetes: Es una alternativa open-source para orquestación de contenedores, inicialmente desarrollado por Google, buscaba ofrecer una plataforma que automatice el despliegue, escalamiento y operación de las aplicaciones en contenedores a través de clústeres de hosts, Amazon también ofrece una alterativa para integrar kunernetes con su infraestructura, denominado Amazon Elastic kubernetes, este ofrece la configuración del master y la facilidad para crear los nodos y armar los clústeres, esto es especialmente útil para integrar sistemas

que venían de Kubernetes en infraestructura AWS. [7]

II-C1a. Funcionamiento interno de kubernetes: Un clúster de Kubernetes se compone de varias máquinas o nodos. Cada uno de estos nodos pueden ser de tipo master o minion.

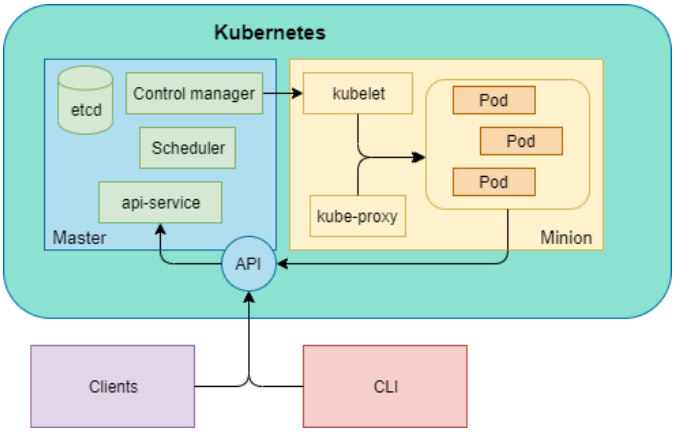
Los nodos master son los encargados de coordinar el clúster, es necesario que exista al menos uno de estos en el clúster, su principal tarea es decidir en que nodo minion se ejecuta cada contenedor, para mantener el estado del clúster en caso de que algún contenedor o nodo falle, también de decidir como escalar en caso de requerirlo, manteniendo el numero deseado de contenedores en todo momento, sus componentes son:

- Etcd: Es una base de datos tipo clave-valor en la cual se almacena la configuración global del clúster
- Api-service: Es el API que exponen los nodos master para comunicarse con los nodos minion y los clientes para realizar configuraciones.
- Scheduler: Es el encargado de determinar en que nodo de ejecuta un contenedor.
- Control-manager: es el encargado de ejecutar los distintos controladores, que se encargan de asegurar que en todo momento se cumple el estado deseado del clúster.

Los nodos minion son los encargados de ejecutar los contenedores desplegados en el clúster y se compone de:

- Kube-proxy: Es el encargado de gestionar las IPs virtuales asignadas a cada contenedor, así como la red virtual.
- Kubalet: Es el componente más importante de un nodo minion, cuya función principal es asegurarse de que todos los contenedores que deben ejecutarse en este nodo se están ejecutando, por lo que contantemente está monitoreando el estado de los contendores.
- Container engine: es el motor de contenedores que ejecuta el nodo minion, pude ser Docker, RKT o otras alternativas, sobre este se crean los Pod que son las unidades atómicas del clúster, estos contienen básicamente los contenedores desplegados. [8]

II-C2. Docker Swarm: Es una la plataforma open-source, nativa para orquestar grupos de contenedores Docker, su principal ventaja es la compatibilidad que tiene con el ecosistema de contenedores Docker, por lo que cualquier software, servicio o herramienta que este diseñada para correr en un contenedor Docker va a funcionar bien con



Docker Swarm, además esta usa el mismo Docker cli.[9][10]

Es liguero y sencillo, por lo que provee despliegues rápidos y menos consumo de recursos en comparación con otras alternativas de Orquestación como Kubernetes, pero no provee la misma flexibilidad para crear sistemas complejos. [9][10]

II-C3. ECS: Amazon elastic container service (ECS) nos permite gestionar diferentes contenedores EC2 y ofrece una serie de configuraciones generales que nos facilitan la configuración del sistema orquestado, está directamente ligado a los servicios ofrecidos por Amazon y se caracteriza por ser completamente administrado, por lo que no es necesario instalar o configurar nada de la arquitectura necesaria para el funcionamiento del orquestador, cuenta con características que fortalecen la seguridad del contenedor, además puede usarse con AWS Fargate, que es un cómputo sin servidor para contenedores, lo que permite especificar los recursos de cada instancia y determinar si se necesitan más. [11]

II-C3a. Fargate: Amazon Fargate es un motor informático sin servidor que permite eliminar la necesidad de administrar y especificar los recursos, para asignar de forma dinámica la cantidad correcta de cómputo necesario y permite escalar a la capacidad necesaria las instancias de nuestro clúster buscando optimizar los recursos consumidos y pagados en la plataforma. [11][12]

III. Infraestructura como código

Ir un paso más allá de los orquestadores de contenedores es posible, generar una infraestructura a partir de una descripción en código donde se indica las propiedades de la infraestructura, esto se ha vuelto crucial para automatizar la infraestructura

y los flujos de despliegue de que las aplicaciones modernas que suelen escalar horizontalmente aumentando el número de instancias, además el código puede tener varias versiones e ir evolucionando muy rápidamente, esto sucede por lo general cuando se trata de proyectos muy grandes en los que muchas personas están interactuando, este paradigma surge como alternativa para automatizar varios ambientes con comportamientos iguales sobre la infraestructura cómo pueden ser ambientes de desarrollo, pruebas, staging y producción, en un inició esto se resolvía con mantener una rigurosa documentación sobre cómo estaba estructurada la infraestructura y luego se replicaba en cada uno de los ambientes los mismos pasos para crear una infraestructura igual y así es cómo llegamos a la primera aproximación, una aproximación imperativa que básicamente es una serie de comandos descriptivos por lo general asociados a un command line donde línea de línea se van especificando los componentes y sus respectivas configuraciones específicas hasta completar la infraestructura. [13]

Ejemplo:

```
cli create RDS
cli create k8s
cli create vm
cli create vpc
```

Esto nos permite definir detalladamente paso a paso cada uno de los componentes y la configuración de los componentes pero tiene la desventaja de que puede generar scripts demasiado largos y complejos, especialmente con infraestructuras con demasiados componentes y altamente complejas, además de que está aproximación hace que los scripts sean poco flexibles a cambios al ser un proceso imperativo modificar un paso o configuración puede tener dependencias futuras de este paso, por lo que es posible que también tengamos que modificarlas por lo que no es una solución qué escale de forma sencilla. [13]

Las siguientes aproximación es declarativa de cuál permite definir la serie parámetros que debe tener la infraestructura en su estado final y se delega al sistema algunos de los detalles para llegar a esa configuración, obviamente se pierde granularidad en las configuraciones pero nos permiten sistema mucho más sencillo y escalable ya que se modifica algún parámetro valor el sistema que va a interpretar el archivo se va a encargar de solucionar cualquier tipo de dependencia o incompatibilidad para hacernos llegar a ese estado final deseado por lo general se utilizan archivos yml. [13]

Otra ventaja de la aproximación declarativa sobre la imperativa es que en caso de que durante la ejecución de algún comando se genere un error y el error no se maneje de forma adecuada en el script se puede generar la infraestructura de forma incompleta, creando inconsistencias, por otro lado en la declarativa el sistema es el que se encarga de interpretar el archivo y crear la infraestructura, por lo que si surge algún error interno reintentará generar la infraestructura y si no lo logra nos lo informará, buscando garantizar que las infraestructura obtenida corresponda a la especificación provista. [13]

Otra ventaja de infraestructura como código es que también nos permite hacer versionamiento sobre la infraestructura, incluso hacer pruebas sobre el estado en el que está y cómo se va mejorando poco a poco este versionamiento, es aconsejable que sea no mutable, así se manejen versiones estáticas de la infraestructura que no dependen de cambios sino de la imagen de una versión anterior. [13]

III-A. Scripts de automatización en IaaC

Los scripts usados para la automatización de infraestructura buscan crear un conjunto de instrucciones y procesos repetibles, que abarcan desde configuraciones para herramientas de software, frameworks, hasta operar herramientas administradoras de la infraestructura, como levantamiento de servicios y orquestadores de contenedores, incluso es posible automatizar tareas básicas de mantenimiento como encender, pagar y reiniciar las maquinas físicas o automatizar el despliegue de una infraestructura completa, pudiendo encender máquinas de forma remota, crear máquinas virtuales o contenedores y sistemas complejos a partir de imágenes, también es posible automatizar la creación y configuración sistemas desde cero, configurando cosas básicas como la conexión de red que incluye puertos disponibles, IPs, máscaras y demás, por lo que es posible no solo configurar maquinas sino todo tipo de dispositivos como routers y hardware personalizado, permitiendo automatizar otros aspectos de la infraestructura de nuestros sistemas como las conexiones de red VPC y de más. Por supuesto la instalación de aplicaciones y configuración de estas está incluida, es común usar

estos scripts de automatización en conjunto con los más tradicionales scripts bash, por lo que podemos automatizar prácticamente todo y de este modo se busca reducir la interacción de las personas en tareas repetitivas. [14]

Se usan lenguajes de programación para describir dichos procesos, estos tiene acceso a la consola del sistema por lo que pueden interactuar con el sistema a tan bajo nivel como sea necesario, los lenguajes interpretados son los más populares para esta tarea por su simpleza y portabilidad, los más famosos para este propósito son Python y Ruby, pero también se usan lenguajes más robustos para la construcción de frameworks que luego ayudan a la automatización y coordinación de estos scripts, algunos de los frameworks más usados son Puppet y Ansible.

III-B. Inconvenientes y ventajas de IaaC

IaaC no es perfecto, requiriere de una curva de aprendizaje y maduración para ser efectivo, esto implica la inversión de recursos en investigación y desarrollo de herramientas para lograr la automatización de la infraestructura a través de código, por lo que puede no ser adecuado para todas las empresas, especialmente para aquellas que suelen permanecer estáticas y no tiene un ritmo acelerado de despliegues o cambios en la infraestructura. Por otro lado, es ideal para las empresas con un ritmo acelerado de despliegues diarios, esto puede ser muy común si hablamos de infraestructura de microservicios, en este caso se hace casi que un requisito, porque estaremos creciendo constantemente de forma horizontal, creando nuevos microservicios y sacando versiones nuevas de forma regular.

III-B1. Seguridad: La seguridad de la infraestructura depende principalmente de la calidad de los scripts de automatización, pueden ser favorables si se configuran todas las configuraciones necesarias, se actualiza de forma regular las aplicaciones y servicios, dejando pocas brechas a vulnerabilidades de seguridad, pero por lo general se configuran sistemas complejos, lo que dificulta esta tarea y si lo sumamos con el uso de malas prácticas de programación segura en el desarrollo de estos scripts terminamos con sistemas vulnerables y con vulnerabilidades replicadas en múltiples puntos del sistema, que al solo revisarse una vez al realizar el script quedan en el olvido. [16]

Las vulnerabilidades que más comúnmente podemos encontrar están relacionadas a configuración incorrecta de seguridad, OWASP define esta vulnerabilidad como cualquier configuración errática que se pueda presentar en cualquier capa de la aplicación, incluidos los servicios de red, plataforma, servidor web, servidor de aplicaciones, base de datos, marcos, código personalizado y máquinas virtuales, contenedores o almacenamiento preinstalados y van desde configuraciones incorrectas, el uso de cuentas o configuraciones predeterminadas, servicios innecesarios, opciones heredadas, etc [15] Un estudio realizado por Akond Rahman, Chris Parnin, y Laurie Williams evidencia a mayor detalle lo comunes que son estas vulnerabilidades en scripts de automatización, durante el estudio se estudiaron siete vulnerabilidades:

- Credenciales de administrador por defecto
- Contraseñas vacías
- Valores secretos explícitos
- Dirección IP invalida
- Cometarios sospechosos
- Uso de HTTP sin TLS
- Uso de algoritmos de inscripción débiles

El observador usado en el estudio encontró que para las compañías GitHub, Mozilla, Openstack, y Wikimedia, respectivamente el 29.3 %, 17.9 %, 32.9 %, y 26.7 % de todos los scripts contenía al menos una de las vulnerabilidades estudiadas. [16]

III-C. Sistemas de gestión IaaC

III-C1. Ansible: Red Hat Ansible Automation Platform es el software de Red Hat para automatizar la configuración, la administración y despliegue de sistemas, opera principalmente a través del uso de Ansible Playbooks, que es básicamente un conjunto de instrucciones en lenguaje de especificación YAML usado para definir tareas a automatizar. [17]

III-C2. Puppet: Puppet ofrece dos productos para gestionar infraestructura o servicios, Open Source Puppet y Puppet Enterprise, siendo el segundo una solución más robusta de pago, para gestionar infraestructura usa un lenguaje de dominio especifico (DSL) para expresar las tareas a automatizar, además ofrece herramientas para orquestación visualización y control del sistema. [18]

III-C3. AWS CLI: Es la consola de control de los servicios de Amazon Web Services, provee una interfaz que permite administrar desde bases de

datos, orquestadores, sistemas de enrutamiento, manejo de red, almacenamiento y múltiples servicios de comunicación entre otras cosas, técnicamente no está en la misma categoría que los otros sistemas de gestión, ya que no es flexible y por si solo ni es capaz de hacer tareas automatizadas de forma específica, pero si abstrae la lógica de la infraestructura en una interfaz sencilla, no requiere de administración del usuario. Los servicios de AWS de pueden automatizar con otros sistemas de gestión como Ansible o Puppet, por lo que si lo requerimos podemos mezclar los beneficios de ambas herramientas, trayendo la flexibilidad de los otros sistemas de gestión y manteniendo la sencillez de los servicios AWS. [19]

III-C3a. Posibilidades y limitaciones de AWS CLI: La consola de Amazon provee múltiples funciones para controlar la infraestructura en Amazon Web Services, esta se restringe exclusivamente a manipular elementos de AWS, pero nos permite tener total control de los elementos, pudiendo crear, modificar y configurar prácticamente todos los servicios de AWS y nos permite una granularidad en la configuración igual a la que podemos tener desde la consola Web, pero en una consola de comandos que está pensada para los desarrolladores, para que puedan crear todo tipo de script que automatice un flujo de despliegue de infraestructura sobre AWS, permitiendo por ejemplo a través de varios scripts y sistemas de gestión de IaaC automatizar la creación de un ECS con los recursos especificados para cada instancia, con las configuraciones de red adecuadas, extraer el código de la aplicación que se planea levantar de algún repositorio, generar un ejecutable compilando el código, generar una imagen Docker de la aplicación con el ejecutable, todas las librarías y dependencias necesarias, desplegar la imagen en los EC2 del ECS, montar un sistemas de autoscaling que mantenga estable nuestra en buen estado nuestra ampliación y monitorearla con cloudWatch, para saber el consumo de recursos y la estabilidad del servicio. [19][20]

IV. Conclusión

Las diversas tecnologías que han surgido y evolucionado a lo largo de los años se van amoldando cada vez más a los nuevos requerimientos de la industria, como lo es el rápido ciclo de desarrollo producto de las nuevas prácticas de desarrollo, como metodologías agiles y nuevas arquitecturas de infraestructuras de TI como SOA y en concreto la más reciente arquitectura de microservicios, esto nos hace requerir nuevos mecanismos para hacer más eficiente el ciclo de despliegue, esto se logra a través de nuevas herramientas que facilitan la configuración y gestión de infraestructura flexible, acompañado de automatización de tareas que busca reducir la interacción humana en tareas repetitivas. Además, responde a el crecimiento volátil de la industria permitiendo sistemas complejos masivamente distribuidos y auto gestionados desde la nube.

Referencias

[1] Graziano, C. D. (2011). A performance analysis of Xen and KVM hypervisors for hosting the Xen Worlds Project. Obtenido de Ministerio de Educación y Ciencia: http://recursostic.educacion.es/observatorio/web/

[2] Popek, Gerald J.; Goldberg, Robert P. (1974). ”Formal requirements for virtualizable third generation architectures”. Communications of the ACM. 17 (7): 412–421. Obtenido de ACM: https://dl.acm.org/doi/10.1145/361011.361073

[3] Katie Lane. ”Kubernetes vs. Docker: What Does It Really Mean?”. Obtenido de sumologic: https://www.sumologic.com/blog/kubernetes-vs-docker/

[4] Ben Corrie. ”What is a Container?”, May 16, 2017. Obtenido de youtube - VMware Cloud Native Apps https://www.youtube.com/watch?v=EnJ7qX9fkcU

[5] Roderick Bauer. ”What’s the Diff: VMs vs Containers”, June 28, 2018. Obtenido de backblaze: https://www.backblaze.com/blog/vm-vs-containers/

[6] ”Docker Alternatives”, Obtenido de Aqua container security: https://www.aquasec.com/wiki/display/containers/Docker+Alternatives+-+Rkt%2C+LXD%2C+OpenVZ%2C+Linux+VServer%2C+Windows+Containers

[7] Steve Tegeler. May 15, 2017. ”What is Kubernetes?”. Obtenido de vmware: https://www.vmware.com/topics/glossary/content/kubernetes

[8] Kubernetes.io. (16 de 04 de 2019). ”Kubernetes architecture”. Obtenido de kubernetes.io: https://kubernetes.io/es/docs/concepts/architecture/

[9] Savaram Ravindra. 3 Sep, 2018, ”Kubernetes vs. Docker Swarm: What’s the Difference?”. Obtenido de thenewstack: https://thenewstack.io/kubernetes-vs-docker-swarm-whats-the-difference/

[10] Harsh Binani. October 3rd, 2018, ”Kubernetes vs Docker Swarm — A Comprehensive Comparison”. Obtenido de hackernoon: https://hackernoon.com/kubernetes-vs-docker-swarm-a-comprehensive-comparison-73058543771e

[11] AWS Amazon. .Amazon Elastic Container Service”. Obtenido de Amazon: https://aws.amazon.com/es/ecs/

[12] AWS Amazon. (s.f.). Obtenido de Amazon: https://aws.amazon.com/fargate/

[13] Sai Vennam. ”What is Infrastructure as Code?”, Aug 15, 2019. Obtenido de youtube - IBM Cloud: https://www.youtube.com/watch?v=zWw2wuiKd5o

[14] Yaser Jararweha,, Mahmoud Al-Ayyouba, Ala’ Darabseha, Elhadj Benkhelifa b, Mladen Voukc, Andy Rindos d. ”Software defined cloud: Survey, system and evaluation”, 18 November, 2015. Optenido de journal homepage: www.elsevier.com/locate/fgcs

[15] OWASP. (2017). owasp.org. Obtenido de https://www.owasp.org/index.php/Top_10-2017_A6-Security_Misconfiguration
[16] Rahman, A., Parnin, C., Williams, L. (2019). The Seven Sins: Security Smells in Infrastructure as Code Scripts. ICSE.
[17] RedHad. Obtenido de Ansible: https://www.ansible.com/overview/it-automation
[18] Puppet. Obtenido de puppet: https://puppet.com
[19] AWS Amazon. (s.f.). Obtenido de Amazon: https://aws.amazon.com/es/cli/
[20] AWS Amazon. (s.f.). Obtenido de Amazon: https://docs.aws.amazon.com/cli/latest/reference/