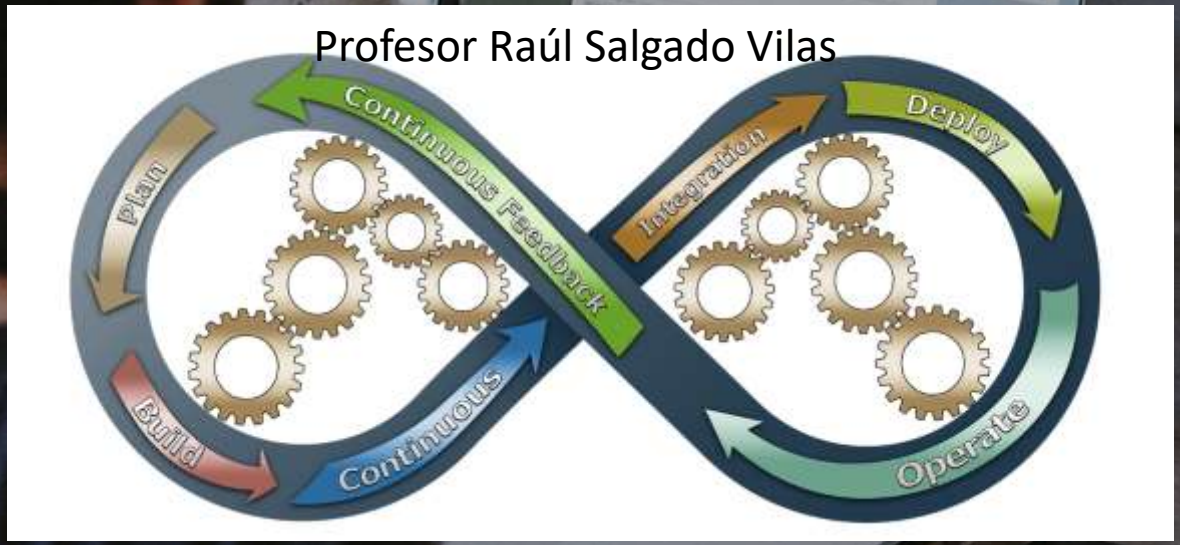
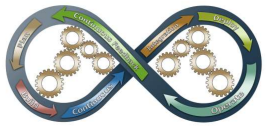
edix
Digital Workers

UF-1 Introducción A DeVOPS

Profesor Raúl Salgado Vilas





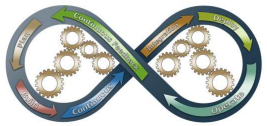
CULTURA DevOPS

- DevOps es una revolución en la manera de trabajar de equipos y organizaciones: no debe entenderse solo en el contexto técnico, sino también en el cultural, una corriente de cambio en la que prima la colaboración desde la gestión hasta desarrollo y operaciones.

- DevOps en la actualidad, tanto a nivel técnico como cultural que se enlaza con la introducción a la cultura Agile de muchas organizaciones IT a día de hoy: el control total sobre los procesos de despliegue de nuevas funcionalidades, desde su concepción hasta su puesta en marcha.

- DevOps no puede entenderse únicamente como un conjunto de tecnologías y mejores prácticas, sino como una revolución cultural. Es una corriente de cambio dentro de las organizaciones en la que prima:

  - ✓ Colaboración e interacción abierta entre los equipos de operaciones y de desarrollo.
  - ✓ Derribar los silos, barreras artificiales creadas en las organizaciones que frenan la colaboración natural.
  - ✓ Ciclos cortos de diseño, implementación y despliegue que permiten anticiparse a los problemas y adaptarse al cambio.
  - ✓ No confundir con la automatización que es la forma natural de reducir costes y optimizar recursos.



CULTURA DevOPS

- DevOps es un término relativamente nuevo para describir lo que también ha sido llamado como ‘administración de sistemas ágiles’, y que se centra en el trabajo y la colaboración de los equipos de Desarrollo y Operaciones.

- En la actualidad, las empresas se mueven a la velocidad de la nube y es por ello por lo que DevOps se ha convertido en un enfoque cada vez más extendido y popular para la entrega de software.

- Los equipos de Desarrollo y Operaciones utilizan esta metodología para construir, probar, implementar y monitorizar las aplicaciones debido a que les permite actuar con velocidad, mantener altos índices de calidad y controlar los cambios con suma rapidez.

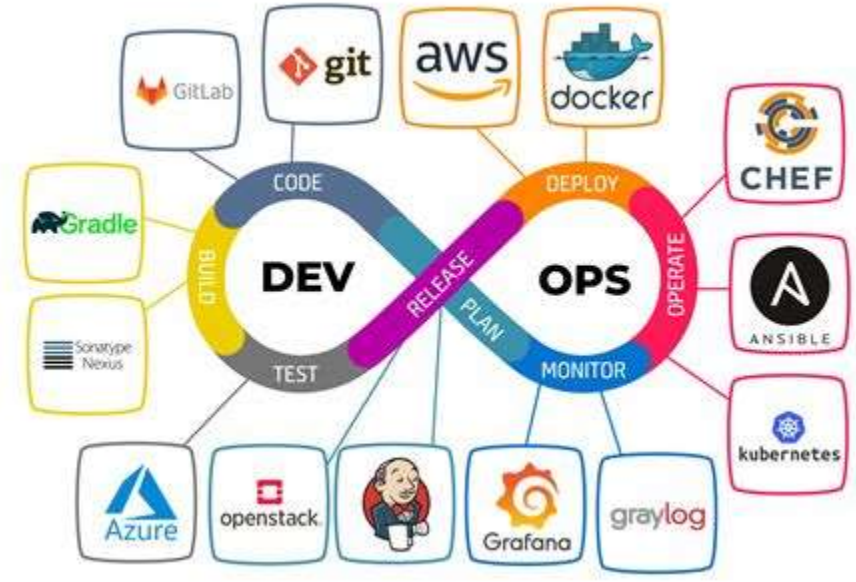
- DevOps es esencial para cualquier empresa que aspire a ser ágil y que pretenda ser capaz de responder rápidamente a las demandas del mercado. Por lo tanto, DevOps es un enfoque hacia la entrega de software y promueve una colaboración más estrecha entre las líneas de negocio, desarrollo y operaciones, al tiempo que elimina las barreras entre las partes interesadas y los clientes.

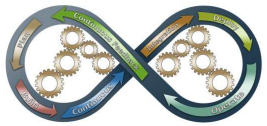
CULTURA DevOPS

- En general, realizar un cambio en el Business as Usual es difícil y si se hace viene siempre impulsado por una necesidad de negocio.

- Se trabajo en DeVOPS para evitar largos plazos de entrega del software hasta su paso a producción, evitando los sistemas de trabajo tradicionales:

  - ✓ Impiden a las empresas brindar servicios de vanguardia.
  - ✓ Impiden a los equipos IT el trabajar de forma ágil.
  - ✓ Lastran la experiencia del cliente.





CULTURA DevOPS

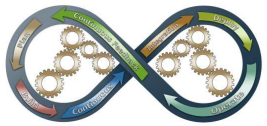
- Metodologías ágiles: Aparece un conjunto de **metodologías ágiles**, en comparación con la ingeniería de software tradicional. Son un conjunto de técnicas de desarrollo dedicadas principalmente a los sistemas complejos y al desarrollo de productos con características dinámicas, no deterministas y no lineales, donde las estimaciones precisas, planes estables y predicciones son, a menudo, difíciles de conseguir en etapas tempranas.

- Estas técnicas son una aplicación directa del manifiesto Agile, firmado por expertos del desarrollo de software hace veinte años:

  - ✓ Individuos e interacciones sobre procesos y herramientas.
  - ✓ Software funcionando sobre documentación extensiva.
  - ✓ Colaboración con el cliente sobre negociación contractual.
  - ✓ Respuesta ante el cambio sobre seguir un plan.

- Las principales metodologías ágiles son las siguientes:

  - ✓ Adatptative Softwareware Development (ASD).
  - ✓ Crystal Clear.
  - ✓ Feature Driven Development (FDD).
  - ✓ Kanban.
  - ✓ Extreme Programming (XP).
  - ✓ Scrum (más usada).



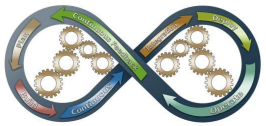
CULTURA DevOPS

- Testing: La segunda parte de esta lección estará centrada en cómo dotar al código de validaciones que garanticen que hace lo que tiene que hacer y que lo seguirá haciendo a medida que evolucione.

- No solo esto, sino que el uso de tests en nuestro código y aplicación, en las condiciones adecuadas (rápidos, consistentes, fiables), es lo que posibilita una de las características más deseables de toda la cultura DevOps: la velocidad.

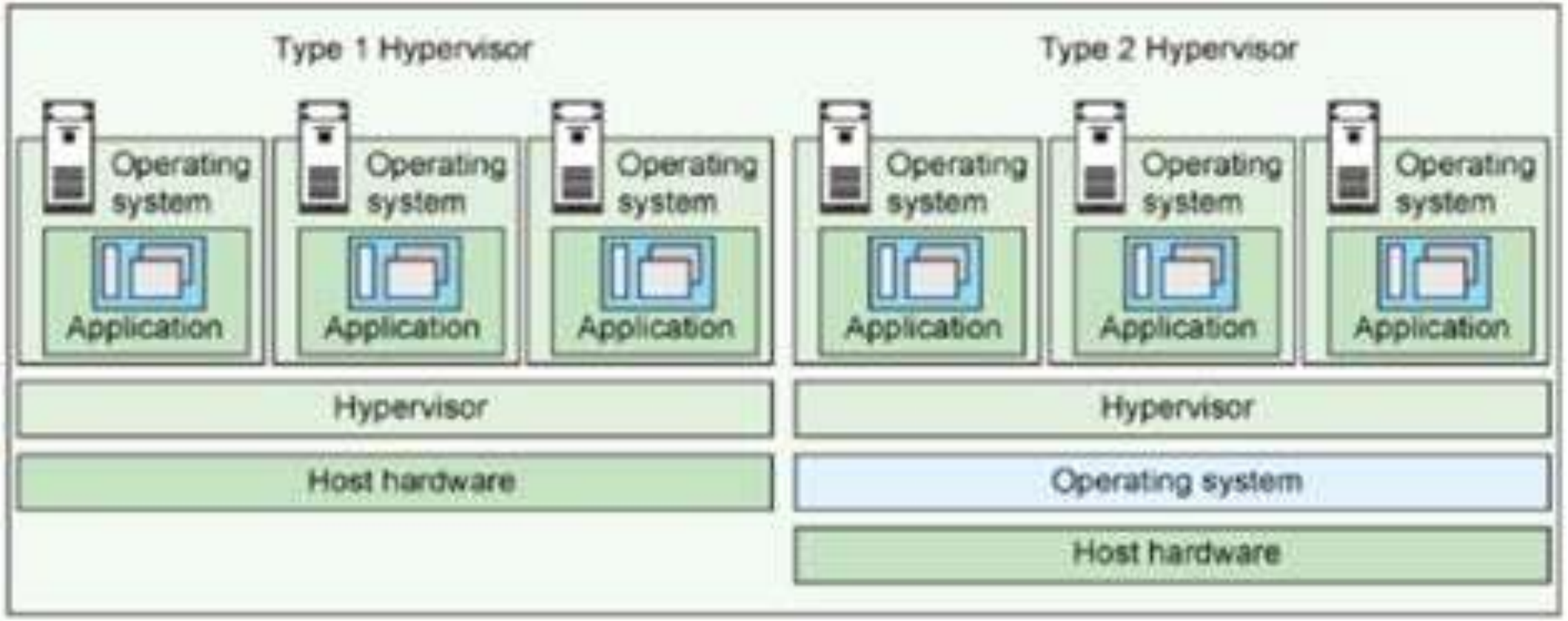
- Ciclos rápidos de desarrollo garantizan que se tenga feedback inmediato de posibles errores, lo que permite arreglarlos y seguir iterando hasta llegar a resolver el problema que nos habíamos propuesto.

- Usaremos técnicas de Desarrollo orientado por tests o TDD (por las siglas en inglés de Test-Driven Development). TDD es un patrón de diseño en el que, a grandes rasgos, los tests se codifican antes que el código fuente que los haría pasar: merece la pena dedicar tiempo a repasar este patrón y compararlo con las alternativas de desarrollo usuales.



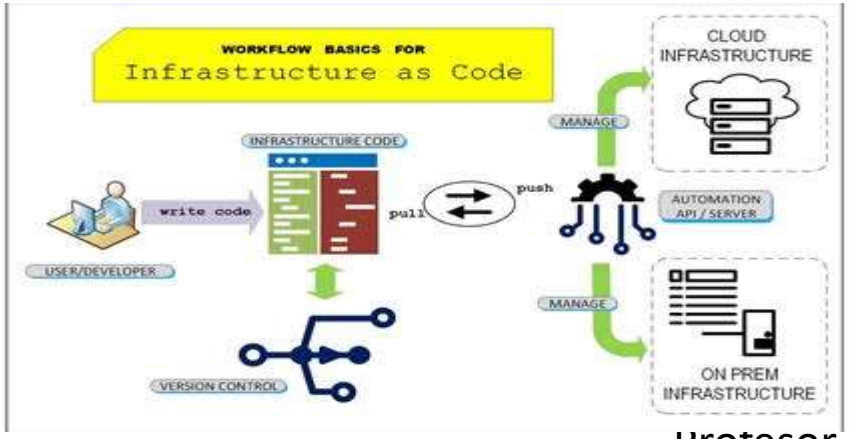
- **Virtualización, Cloud y Contenedores:** Lo primero es definir un ‘entorno tradicional’, aunque realmente es la base de toda la infraestructura de operaciones de los sistemas de todo el mundo: los sistemas físicos.
- Un sistema hardware físico va desde una Raspberry a un portátil o a un servidor dentro de un data center. Es un ordenador en el concepto más habitual de la palabra, aunque se suele reducir la descripción de un sistema por los recursos que ofrece:
  - ✓ CPU.
  - ✓ Memoria.
  - ✓ Almacenamiento.
- **Virtualización** es un paso más allá a la hora de utilizar los recursos físicos es el de la virtualización: aprovechar los recursos disponibles en un servidor o en una red de ordenadores, o incluso en un data center, para generar ‘máquinas virtuales’ que los aíslen y generen un entorno de ejecución o procesamiento.
- El software de virtualización (denominado hipervisor) crea una máquina virtual que imita todo el entorno de hardware. El sistema operativo que está cargado en una máquina virtual es un producto estándar no modificado. Cuando realiza llamadas para recursos del sistema, el software de emulación de hardware captura la llamada del sistema y la redirige para que pueda gestionar estructuras de datos proporcionadas por el hipervisor. Es el propio hipervisor el que realiza las llamadas al hardware fisico real, subyacente a toda la aglomeración de software.

- **La emulación o imitación de hardware** también es conocida como virtualización de metal desnudo (del inglés **bare metal virtualization**), para simbolizar el hecho de que ningún software se encuentra entre el hipervisor y el 'metal' del servidor. Como hemos mencionado, el hipervisor intercepta las llamadas del sistema desde la máquina virtual huésped y coordina el acceso al hardware subyacente directamente.

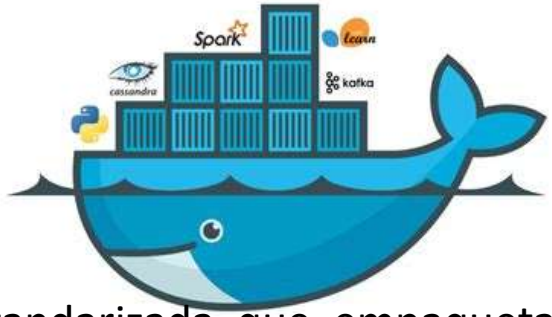


- **Cloud Computing:** Todos estos recursos de virtualización o ejecución en contenedores tienen algo en común: tienen que ‘correr’ en algún sito y suelen ser nubes públicas, privadas o híbridas. Se ha de garantizar:
  - ✓ Disponibilidad de servicio.
  - ✓ Capacidad extra de recursos de forma puntual.
  - ✓ Posibilidad de pedir más recursos a demanda.
  - ✓ Tolerancia a fallos.
- En cualquier entorno, para poder escalar necesitamos poder tener más recursos disponibles: Si mi aplicación tene problemas de memoria (o necesita más CPU para hacer un procesamiento), se añaden más recursos a la máquina → scale-up. Si una máquina no es capaz de atender todas las peticiones de red entrantes, tengo que añadir más máquinas → scale-out.
- Para resolver estos problemas (y muchos otros) están las clouds: proveen recursos y servicios ‘en la nube’ para eliminar los problemas de:
  - ✓ Poseer infraestructura.
  - ✓ Configurarla.
  - ✓ Mantenerla.
  - ✓ Actualizarla o mejorarla.

- **Infraestructura como código (IaC):** Hemos visto que podemos definir, de forma precisa, el entorno de ejecución de nuestro software.
- Además, con el uso de Clouds o Virtualización, podemos también disponer de servicios añadidos, como pueden ser:
  - ✓ Almacenamiento elástico.
  - ✓ Bases de datos.
  - ✓ Servicios de logs.
  - ✓ Redundancia.
- Existe una tecnología que nos permite **escribir** nuestra infraestructura en lenguajes declarativos y, de este modo, guardarlos en un control de versiones junto con nuestra aplicación (o en otro repositorio). Está tecnología recibe el nombre de *infraestructura como código.*
- Existen varias alterna vas que debemos conocer:
  - ✓ **Específicas**: relacionadas con las clouds públicas, como Cloudforma on para AWS o Azure Resource Manager templates.
  - ✓ **Genéricas**: como es el caso de Terraform.

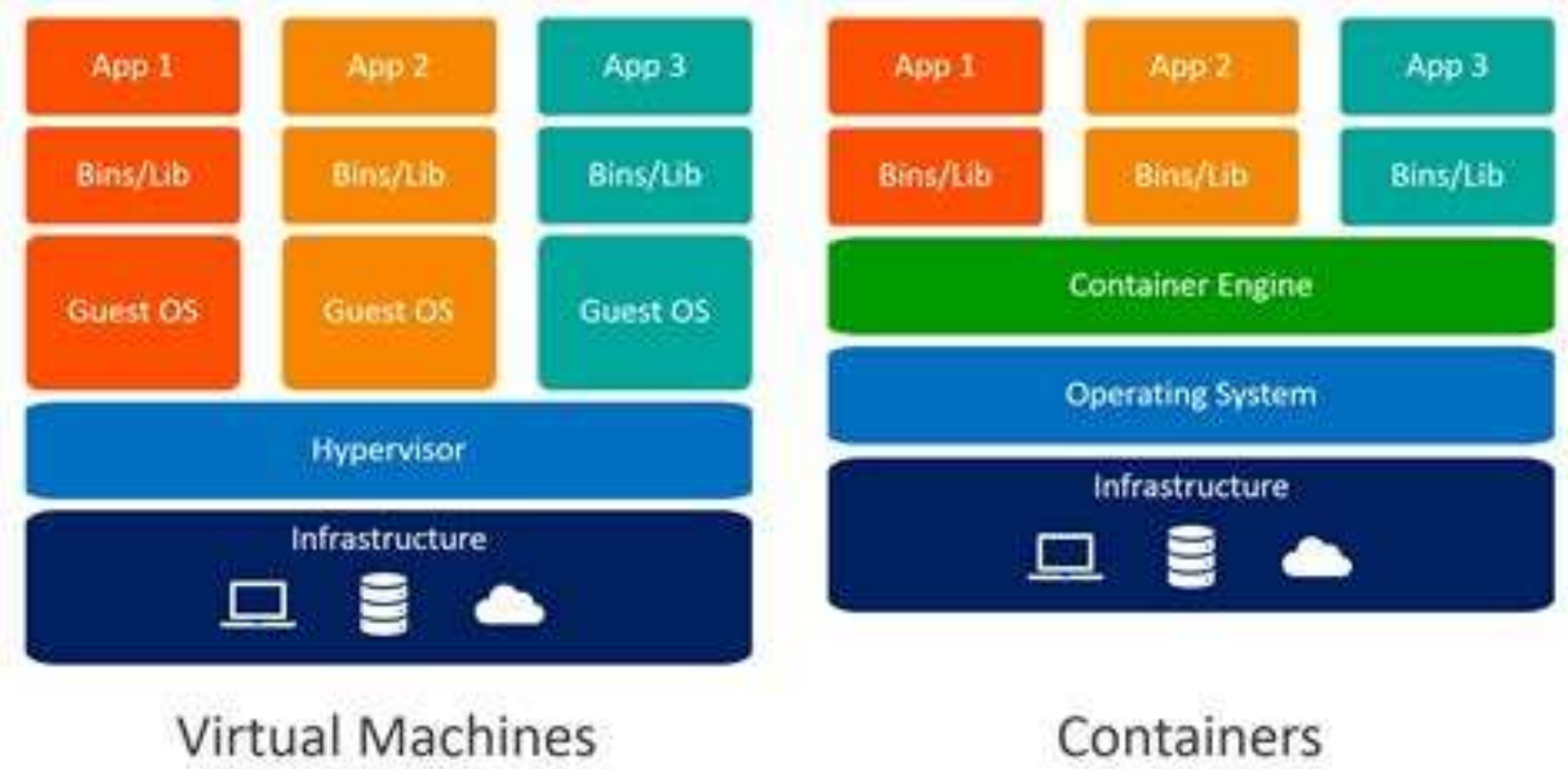


- **Docker:** Docker representa otra **tecnología de virtualización**, en este caso, **aplicada al sistema operativo**: es una plataforma de código abierto para desarrollar, distribuir y desplegar aplicaciones en entornos aislados y seguros denominados *contenedores*.



- Un contenedor es una unidad de software estandarizada que empaqueta el código de una aplicación y todas sus dependencias para permitir su ejecución de manera rápida y consistente, independientemente del entorno en el que se ejecute.
- Los contenedores tienen la característica de ser livianos, ya que no necesitan la carga adicional de un hipervisor como ocurre con las máquinas virtuales, sino que se ejecutan directamente dentro del núcleo de la máquina host. Esto significa que con un
- determinado hardware podremos ejecutar mayor número de contenedores que si estuviéramos utilizando varias máquinas virtuales.
- La plataforma Docker nos permite empaquetar nuestras aplicaciones con todo lo necesario para su ejecución (librerías, código, herramientas, configuraciones), eliminando así dependencias del sistema operativo de la máquina host y facilitando un despliegue muy rápido de nuestras aplicaciones.

- **Docker vs virtual machine:**



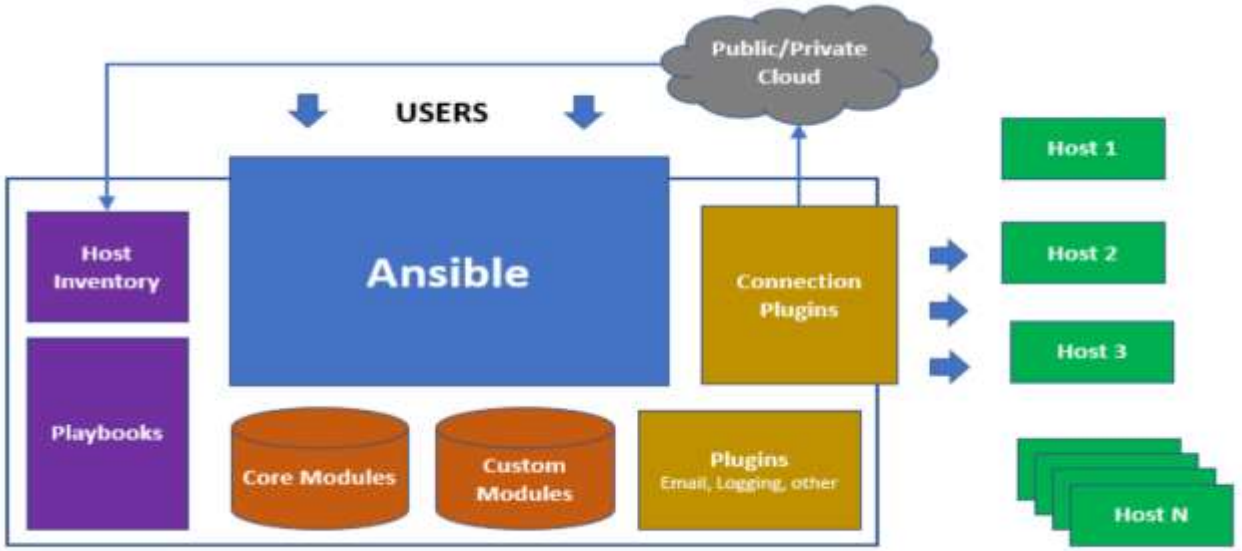
- **Kubernetes:** Kubernetes es una herramienta de código abierto que se utiliza para el despliegue y la gestión de contenedores. No se trata de una plataforma de contenedores, sino que ofrece una gestión de aplicaciones multicontenedor. Esta herramienta fue desarrollada inicialmente por Google basándose en soluciones propias con las que la compañía había estado trabajando internamente durante años. La primera versión se presentó en el año 2014 y, desde entonces, ha ido evolucionando continuamente con nuevas funcionalidades.

- Entre las principales características y capacidades de Kubernetes podemos encontrar:
  - ✓ El empaquetado automático de contenedores.
  - ✓ El descubrimiento de servicios y balanceo de carga.
  - ✓ El almacenamiento desacoplado.
  - ✓ La autoreparación.
  - ✓ La gestión de la configuración de aplicaciones.
  - ✓ La ejecución por lotes y tareas programadas.
  - ✓ El escalado horizontal.
  - ✓ El despliegue de actualizaciones automatizadas y sin cortes de servicio.
  - ✓ Los rollbacks de despliegue a versiones previas.
  - ✓ La gestión de los recursos del clúster.

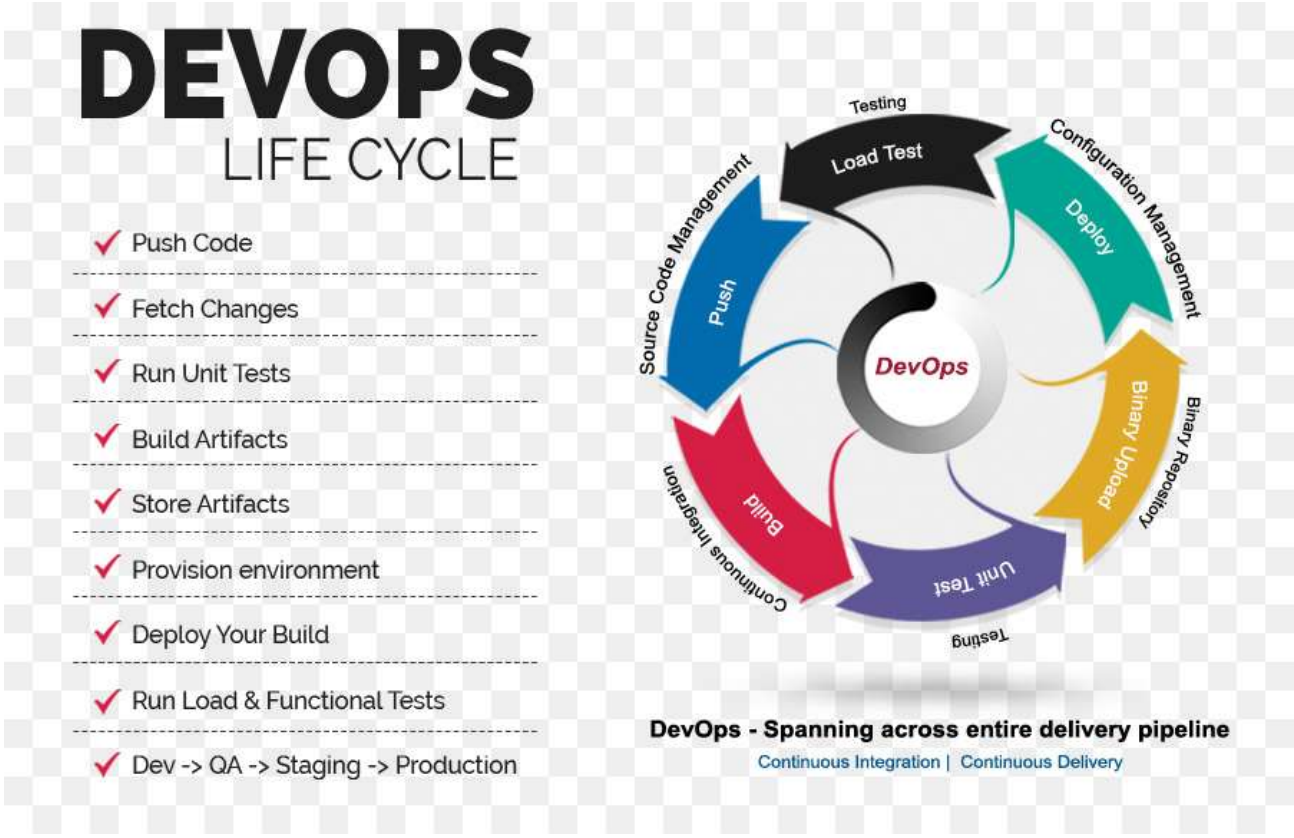
- **Ansible:** Orquestación de sistemas: Ansible automatiza todo lo posible el proceso de instalación, configuración y securización de los sistemas. Ya tengamos una única máquina física o una flota de virtuales:

  - ✓ Configurar un sistema.
  - ✓ Desplegar paquetes
  - ✓ Orquestar tareas avanzadas

- Basada en ficheros en formato YAML en los que se especifica qué tareas deben realizarse, en qué orden y con qué parámetros (servidores en los que aplicar los cambios, variables de entorno que inyectar, parametría genérica, etc).



- **Herramientas de gestión de ciclo de vida:** El ciclo de vida está compuesto por una serie de fases, claramente definidas y diferenciadas, que son utilizados por ingenieros y desarrolladores de sistemas para planificar, diseñar, construir, probar y entregar sistemas de información. El objetivo no es otro que producir sistemas de alta calidad que cumplan o excedan las expectativas del cliente, siempre dentro de las restricciones de tiempo y coste. Si bien existen muchas y muy diferentes maneras de hacerlo, aquellas en las que nos centraremos como DevOps son las que nos permitan hacerlo de manera ágil (por Agile).



- **Sistemas de control de versiones: git es e**l eje principal en la gestión de software es su almacenamiento versionado. Y, en esto, la herramienta por excelencia es git, con el permiso de otras herramientas que ni siquiera vale la pena mencionar; es una herramienta open source y Git es un sistema de control de versiones distribuido en el que el código fuente se almacena en un servidor central organizado en repositorios, pero cada usuario es libre de almacenar una copia local y subirla a donde desee.
- Este servidor central almacena una o más revisiones del código, que popularmente conocemos como branches o ramas. Los usuarios se 'traen' (pull) los cambios de esas ramas a sus estaciones de trabajo y hacen lo que tengan que hacer.Cuando tengo cambios que creo que deben llegar a los demás usuarios, los añadiría (commit) a esa rama y después los 'empujaría' (push) al servidor central. Si todo va bien y nadie más ha modificado esa rama, el código estará disponible para que cualquiera pueda volver a hacer pull y tener mi aportación disponible

- **Versionado y modelos de ramas:** Ahora ya sabemos dónde vamos a guardar nuestro código fuente, sabemos que podemos tener almacenadas versiones del mismo y estamos en condiciones de entregar una nueva funcionalidad. En todo proyecto Git hay una rama principal: main/master y está rama es en la que se supone que se debe encontrar siempre la fuente de verdad de nuestro código fuente.
- Cuando los despliegues de versiones no son constantes/relativamente frecuentes, lo habitual es que exista una rama llamada develop en la que se trabajará durante un tiempo, para proceder a 'lanzarla' (release) y marcarla con un número de versión.
- Adicionalmente, se suelen usar una serie de ramas con una nomenclatura específica:
  - ✓ feature/ - Cuando se trabaja en una nueva característica del software (feature) durante una release.
  - ✓ hotfix/ - Cuando se arregla un error presente en producción (rama main).
  - ✓ bugfix/ - Cuando se arregla un error introducido en una release, antes de que esta llegue a main (y, por tanto, a producción).
- Versionado semántico, con el que, a grandes rasgos, cada versión de nuestro software estará en el formato x.y.z:
  - ✓ x (major) - Suele marcar una familia o línea de versiones concreta, su incremento no garantiza la retrocompatibilidad con la anterior.
  - ✓ y (minor) - Dentro de una línea de versiones, marcan incrementos de funcionalidad que son retrocompatibles entre sí.
  - ✓ z (patch) - Dentro de una versión concreta, marcan incrementos sin funcionalidades relevantes, usualmente correcciones de errores.

- **Integración Continua y Despliegue Continuo (CICD)**

✓ Saber cuándo deben empezar a construir un ejecutable.

✓ Saber si un ejecutable cumple los mínimos exigibles para promocionar a un entorno diferente (pruebas, QA, producción…).

✓ Sabe enviar un correo al responsable de una nueva característica, por qué rompe tests de integración, así como detener el proceso de release.

✓ Saber si debe hacer rollback de una release porque está fallando algo. Son muchas las ventajas de tener sistemas de integración continua, llamados así porque su labor es la de escuchar eventos en un VCS y, en caso de encontrar cambios en una o más ramas, empezar a ejecutar una serie de pasos:

- Tests unitarios y de integración.
- Análisis de seguridad y calidad de código.
- Empaquetado.

✓ Publicación de resultado en un almacén. En el caso de que, además, la herramienta sea capaz de desplegar automáticamente a entornos productivos y monitorizar el resultado para dar marcha atrás en caso de errores, estaríamos hablando de sistemas de despliegue continuo.

- **Jenkins** Jenkins es una herramienta open source de automatización que usaremos para aprender los básicos de los sistemas de CICD. Entre sus características más interesantes, destacaremos:
  - ✓ Extensiva comunidad, al tratarse de un proyecto open source con muchísima actividad y formar parte de la lista de proyectos de CD Foundation.
  - ✓ Extensible vía plugins, con más de 1500 ya disponibles y la posibilidad de crear uno nuevo para nuestras necesidades particulares.
  - ✓ Las pipelines se programan en lenguaje Groovy, por lo que son una parte más de nuestro proyecto y, como tal, se guardan en control de versiones.
  - ✓ Por último, se pueden importar/exportar pipelines para comparar funcionalidades con otros usuarios, aumentando de forma dramática la escalabilidad de los proyectos

- **Despliegues y monitorización** Está será la sección en la que veremos las diferentes maneras de desplegar nuestro software: qué maneras existen para que pueda empezar a hacer aquello para lo que lo hemos programado, ya sea en entornos de desarrollo, de calidad (QA), productivos…
- Hay muchos tipos de aplicaciones posibles y cada una de ellas tendrá su propia manera de ser desplegada:
  - ✓ Aplicaciones accesibles vía endpoints HTTP.
  - ✓ Algoritmos de procesamiento de datos.
  - ✓ Modificaciones a una base de datos.
  - ✓ Apps móviles. Nos vamos a centrar ahora en un tipo concreto, bastante común en la industria y que merece la pena ser estudiado con detalle: las aplicaciones web, o aplicaciones que son accesibles mediante endpoints HTTP. A grandes rasgos, lo que veremos será:
- Cómo configurar el despliegue y diferentes técnicas de despliegue disponibles en Cloud:
  - ✓ Tradicional deploy.
  - ✓ Canary.
  - ✓ Blue-Green.
  - ✓ Cómo funciona un balanceador de carga, las redes y la seguridad.
  - ✓ Cómo monitorizar un servicio en marcha. Logging, Tracing y recolección de métricas.

- **Logging, Tracing y recolección de métricas:**Estas técnicas son formas complementarias de obtener datos de lo que está ocurriendo en servicios que están ejecutándose en nuestra infraestructura:
  - ✓ Logging se refiere a la emisión de trazas que nuestra aplicación escribe durante su operación. Estas trazas suelen ser escritas durante la fase de implementación con el objetivo de poder saber qué datos llegan a diferentes secciones del código o tener un detalle de los mensajes de error.
  - ✓ Tracing es una técnica algo más avanzada y compleja en la que, mediante la propagación de una serie de identificadores a lo largo del camino de ejecución del código de nuestra aplicación, podemos trazar el camino que sigue una petición cualquiera, saber por qué métodos ha pasado, cuánto tempo ha estado en cada uno, etc. Es una técnica muy potente pero que necesita que el framework en cuestión lo provea.
- La recolección de métricas es conceptualmente similar al logging, pero lo que se almacena son datos cualitativos de la ejecución: tiempo en ejecutar ciertas funciones o cálculos, número de veces que se llama a un endpoint… Todos estos datos tenen algo en común y es que pueden guardarse en ficheros de sistema para su posterior explotación o pueden enviarse a diferentes servicios de recolección y proceso de datos, que se encargarán de su almacenamiento, clasificación y visualización.
- Ejemplo de estos sistemas son: Logstash, Fluentd, DataDog, Prometheus, Kibana, Elastic Search, (AWS) CloudTrail, CloudWatch