Christian David Lopez Morcillo
Sistemas distribuidos
Tercer Examen.

1. Realice un cuadro comparativo donde cite desventajas y ventajas con respecto al aprovisionamiento con máquinas virtuales y el aprovisionamiento con contenedores virtuales.

R/**Definiciones:**

Una máquina virtual: es un software perfectamente aislado que puede ejecutar sus propios sistemas operativos y aplicaciones como si fuera un ordenador físico. Una máquina virtual se comporta exactamente igual que lo hace un ordenador físico y contiene sus propios CPU, RAM, disco duro y tarjetas de interfaz de red (NIC) virtuales (es decir, basados en software).

Un contenedor: máquinas virtuales mucho más portables y menos exigentes a nivel recursos de cómputo que las máquinas virtuales convencionales. El contenedor opera como un proceso para el sistema operativo y almacena la aplicación que queremos ejecutar y todas sus dependencias. Esta aplicación solamente tiene visibilidad sobre el sistema de ficheros virtual del contenedor y utiliza indirectamente el kernel del sistema operativo principal para ejecutarse.

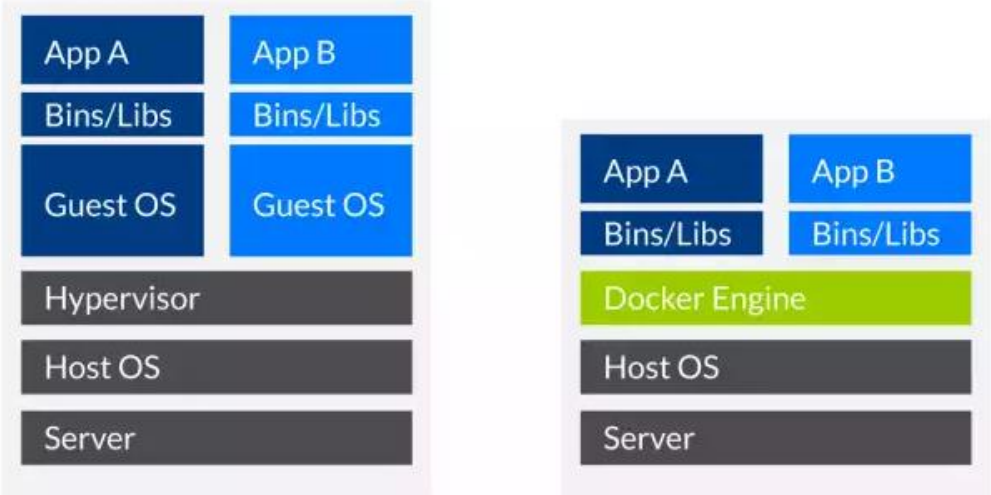

Figure 1 Aquinas virtuales vs contenedores

Comparación:

En el siguiente cuadro comparativo se ilustra los conceptos de máquinas y contenedores virtuales. En las columnas se discrimina por tipo de virtualización y en las filas por ventajas y desventajas. En cada celda aparecerá las características que diferencian positiva o negativamente un tipo de virtualización contra el otro.

	Máquinas Virtuales	**Contenedores Virtuales.**
Ventajas	*Son más seguros, puesto que funcionan como maquinas físicas separadas. Además, se les puede hacer hardening por separado y adecuado a la necesidad de cada sistema. *Es una tecnología más madura, con mayor soporte y más difundida. *Permiten el multi-ejecución de servicios. Muy útil en los escenarios en que se desea virtualizar varias aplicaciones que deben comunicarse dentro de un mismo servidor. *Al emular maquinas físicas individuales, permiten que el S.O. guest sea independiente del S.O. host.	*Son más portables, puesto que su rápido despliegue y poco consumo de recursos permite migrarlos entre servidores, nube y computadores con el menor impacto posible. *Son más rápidos de aprovisionar. El tiempo promedio para crear y lanzar un contendor está en el orden de los segundos. *Consumen menos recursos del sistema, representa una menor carga puesto que comparten un solo núcleo y bibliotecas de aplicaciones. *Su instalación es más rápida y sencilla. *Se usan especialmente para la ejecución de tareas individuales de forma modular. Ideales para ejecución de una aplicación a la vez. *Son más económicos puesto que aprovechan mejor el hardware.
Desvent.	*Son menos portables. Dependiendo de la VM, el tamaño de los archivos que componen la instancia, puede llegar a dificultar o impedir su migración. *Son más lentos aprovisionando. El tiempo de aprovisionamiento promedio puede variar entre el orden de los segundos y los minutos. *Son más costosos dado al consumo mayor de recursos de hardware. Esto se traduce en mayor gasto de infraestructura y de energía. *Su instalación requiere de más tiempo y es más dificultosa.	*En cuanto a la seguridad son más vulnerables, puesto que un ataque puede llegar a afectar muchos contenedores. Es necesario incluso el uso de software adicional para reforzar la seguridad. *Al ser una tecnología más reciente, hay menos soporte y están menos difundidos. *Al utilizar las librerías y binarios del S.O. host, el S.O. guest depende de S.O. host.

2. Realice un cuadro comparativo donde cite las diferencias entre vagrant y docker. El cuadro comparativo debe dar respuesta a los casos en que es útil emplear cada tecnología.

Definiciones:

Docker: es una tecnología de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de Virtualización a nivel de sistema operativo. Utiliza características de aislamiento de recursos del kernel de Linux, tales como cgroups y espacios de nombres (namespaces) para permitir que "contenedores" independientes se ejecuten dentro de una sola instancia de Linux.

Vagrant: es una herramienta para la creación y configuración de entornos de desarrollo virtualizados. Proporciona la creación y gestión sencilla de entornos de trabajo "portables" y replicables que funcionan sobre tecnologías de virtualización conocidas, ofreciendo además un modo de trabajo claro para poder transportar dichos entornos

Comparación:

Primero que todo hay que dejar en claro que vagrant y docker son tecnologías diferentes y no son excluyentes entre sí. Es decir, es posible utilizar Vagrant para crear un entorno capaz de ejecutar Docker dentro de éste y así desplegar una aplicación. Sin embargo, a continuación, se describen las diferencias que pueden encontrase.

Característica	Docker	Vagrant
Tipo de virtualización:	Contenedores	Máquina virtual
Nivel de aislamiento:	Débil	Muy alto
Tiempo de creación:	<10 min	>10 min
Tamaño del despliegue:	Al menos 100MB	Al menos 1GB
Tiempo de arranque:	Segundos	Minutos
Impacto en el sistema:	Muy bajo	Alto
Garantiza recursos en el S.O.:	No	Sí
Cuántos se pueden	>50	<10

albergar a la vez:		
Principal ventaja:	Rápido, ligero, fácil de aprender	Fácil de gestionar.
Casos de uso:	*En un ambiete de pruebas para levantar rápidamente una aplicación. *En un ambiente de producción para desplegar una aplicación o micro-servicio. * Facilita el testeo de nuevas versiones de librerías o tecnologías nuevas sin comprometer la máquina anfitrión.	*En un ambiente de pruebas para emular el entorno del servidor de un cliente, con ip, ram, cpu, disco, etc. *En un ambiente de producción para desplegar varias aplicaciones que necesitan comunicarse internamente dentro del mismo servidor sin hacer uso de la red.

Ambas herramientas son excelentes para entornos de desarrollo y de pruebas y para pasar a producción entornos completos despreocupándonos por las posibles diferencias o la falta de bibliotecas o servicios necesarios.

3. ¿Cuál es la relación entre la tecnología de contenedores virtuales y microservicios?

R/ La relación que existe entre contenedores virtuales y microservicios, es la modularidad y portabilidad que ambos nos ofrecen para garantizar escalabilidad en las aplicaciones. Esto porque ambas tecnologías tienen como objetivo solucionar las problemáticas que las arquitecturas monolíticas nos presentan.

La arquitectura de sistema basada en Microservicios: se basan en la división de las aplicaciones por módulos (Microservicios) autocontenidos, expuestos vía API REST, y que ofrecen una funcionalidad siguiendo la regla de mínimo acoplamiento y máxima cohesión.

Esta división en Microservicios permite la evolución de cada servicio/funcionalidad por separado, siempre que mantenga sus interfaces de acceso, y más importante, permite la ejecución y escalado horizontal de dichos módulos, permitiendo a la aplicación crecer y decrecer en función de la demanda o necesidades de negocio.

Los contendores virtuales: son entornos de "virtualización ligera". Los sistemas de contenedores, como por ejemplo el conocido Docker, utilizan una funcionalidad de Linux conocida como LXC (LinuX Containers). Esta funcionalidad ejecuta procesos de forma completamente aislada de otros y dentro de un mismo sistema Linux.

Además, los sistemas de contenedores ofrecen portabilidad de aplicaciones "containerizadas". Es decir, la posibilidad de portar el stack de ejecución de un servicio (ej. Linux + Tomcat + Java + Microservicio), de tal forma que con desplegar el contenedor mediante un comando en otra máquina se puede desplegar por completo en otro.

Relación entre contenedores y microservicios: Por tanto, vemos que los contenedores son una gran herramienta complementaria para las arquitecturas basadas en microservicios porque los contenedores permiten modularizar y portabilizar a nivel de virtualización todo el entorno de ejecución de un servicio, lo que es ideal para el despliegue de un microservicio porque esto nos garantiza poder escalar rápidamente módulos de una aplicación sin preocuparse por afectar los demás módulos.

Además, esta relación nos permite que los desarrolladores pueden implementar de forma independiente cada servicio, lo que ayuda con el aislamiento de fallos. Si un componente es problemático en una aplicación monolítica, esas cuestiones pueden traer abajo todo el sistema.

4. ¿Por qué los microservicios podrían requerir una aproximación DevOps?

R/

Definiciones:

DevOps es un acrónimo inglés de development (desarrollo) y operaciones (operaciones), que se refiere a una cultura o movimiento que se centra en la comunicación, colaboración e integración entre desarrolladores de software y los profesionales de operaciones en las tecnologías de la información (IT). DevOps es una respuesta a la interdependencia del desarrollo de software y las operaciones IT. Su objetivo es ayudar a una organización a producir productos y servicios software rápidamente.

Relación:

Por tener muchas partes modulares, los microservicios están en un nivel de complejidad distinto a la arquitectura monolítica. Los sistemas de microservicios son distribuidos por naturaleza y como tal, son difíciles de construir y mantener. Esto hace que se requiera una gran habilidad en DevOps para desplegar y mantener una aplicación de microservicios en producción.

Los DevOps nos permiten poder lidiar con el ciclo de vida de una aplicación por microservircios, puesto que nos ayudan a automatizar las pruebas, el despliegue y el mantenimiento de dichas aplicaciones, tareas que si no son automatizadas serían muy difíciles de hacer en un ambiente de producción empresarial.

Además, dado que los microservicios son desarrollados por excelencia a través de metodologías de desarrollo ágil. Se requiere de devops para el desarrollo de infraestructuras que cambien tan al mismo nivel que las metodologías agiles.

5. ¿Por qué la arquitectura de una aplicación orientada a microservicios requiere herramientas de integración continua?

R/

Definiciones:

Integración continua: Práctica de desarrollo software donde los miembros del equipo integran su trabajo frecuentemente, al menos una vez al día. Cada integración se verifica con un build

automático (que incluye la ejecución de pruebas) para detectar errores de integración tan pronto como sea posible.

Relación:

Una arquitectura orientada a microservicios requiere herramientas de integración continua porque esta permite garantizar calidad en el proceso de desarrollo de los microservicios, calidad en el producto final, eficiencia del tiempo de los desarrolladores y agilizar el proceso de entrega de software a los usuarios.

- Se garantiza la calidad del proceso de desarrollo porque en integración continua las etapas, por las que el software pasa, se hacen conocidas por todos los que intervienen en el desarrollo.
- Se garantiza la calidad del microservicio porque se detentan errores lo más pronto posible en las fases de desarrollo de software. Además, se realiza análisis continuo del código para verificar su eficiencia y robustez.
- Se garantiza eficiencia del tiempo porque los desarrolladores pueden enfocarse en escribir código y solucionar bugs, en vez de realizar tareas repetitivas y desgastantes como lo son las pruebas.

Además, las herramientas de integración continua nos permiten automatizar el proceso de desarrollo de microservicios a través de la definición de un pipeline o conjunto de etapas automatizadas por las cuales el software pasa, de tal forma que se puede agilizar la entrega o liberación de nuevas funcionalidades.

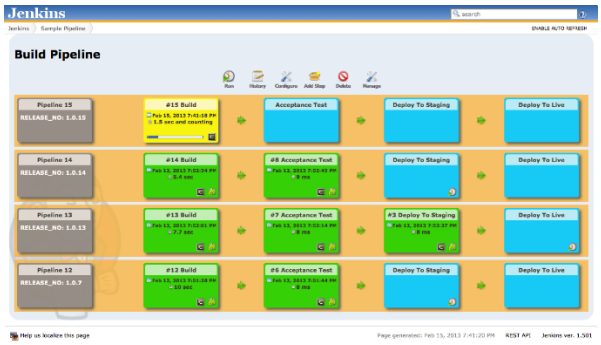

Figure 2 Pipeline en Jenkis

Por otro lado, las herramientas que nos provee la integración continua nos permiten definir qué hacer cuando el código generado falla en alguna etapa del pipeline definido y también nos permite definir criterios claros para saber cuándo el software pude pasar de un entorno a otro.

Por último, la integración continua nos facilita el control de versiones para dar seguimiento a los cambios realizados a los microservicios y a establecer estrategias de roll back cuando el código falla.

6. Es aconsejable el uso de contenedores dentro de contenedores para el caso de integración continua? Justifique su respuesta.

No se desaprueba el uso de contenedores dentro de contenedores, especialmente en ambientes de prueba, sin embargo, existen diversos problemas que pueden ocurrir al usar contenedores dentro de contenedores.

Uno de estos problemas es el almacenamiento o uso de volúmenes. Este problema se da porque el contenedor host no es capaz de abastecer al contenedor guest de volúmenes dado los formatos de sistema de archivos que son usados en los contenedores virtuales.

Exceptuando estos casos, es posible desplegar contenedores dentro de contenedores durante procesos de integración continua para la automatización de pruebas.

7. ¿Cómo se realizan las pruebas automáticas en una arquitectura de microservicios?

R/

Las arquitecturas basadas en microservicios son arquitecturas que hacen modérales las funciones del sistema informático. Esto se hace con el objetivo de poder escalar individualmente cada funcionalidad según la demanda. Dada la gran cantidad de microservicios que un sistema puede tener, se hace necesario la automatización de pruebas que sería imposible de otra manera.

A continuación, se explica los tipos de pruebas automáticas que se realizan a los microservicios:

1. Pruebas unitarias

El alcance de las pruebas de unidad es interno al servicio. En términos de volumen de pruebas, son los más grandes en número. Las pruebas unitarias deberían idealmente ser automatizadas, dependiendo del lenguaje de desarrollo y el marco dentro del servicio.

2. Prueba de contrato

Las pruebas de contrato deben tratar cada servicio como una caja negra y todos los servicios deben ser llamados de forma independiente y sus respuestas deben ser verificadas. Cualquier dependencia del servicio debe ser stubs que permitan que el servicio funcione, pero no interactúen con ningún otro servicio. Esto ayuda a evitar cualquier comportamiento complicado que puede ser causado por llamadas externas y convertir el enfoque en realizar las pruebas en un solo servicio.

3. Pruebas de integración

Debe realizarse la verificación de los servicios que se han sometido a pruebas individuales. Esta parte crítica de las pruebas de arquitectura de microservicios depende del buen funcionamiento de las comunicaciones entre servicios. Las llamadas de servicio deben hacerse con integración a servicios externos, incluyendo casos de error y éxito. Por lo tanto, las pruebas de integración validan que el sistema funciona de forma integrada y que las dependencias entre los servicios están presentes como se esperaba.

4. Pruebas de extremo a extremo

Las pruebas de extremo a extremo verifican que todo el flujo de procesos funcione correctamente, incluyendo toda la integración de servicios y bases de datos. Las pruebas exhaustivas de las operaciones que afectan a múltiples servicios garantizan que el sistema funciona en conjunto y satisface todos los requisitos.

5. UI / Pruebas Funcionales

La prueba de interfaz de usuario es la prueba de mayor orden ya que prueba el sistema como un usuario final lo utilizaría. La prueba de este nivel debe sentirse como un usuario tratando de interactuar con el sistema. Todas las bases de datos, interfaces, servicios internos y de terceros deben trabajar juntos sin problemas para producir los resultados esperados.

8. ¿Qué beneficios trae para una datacenter emplear la tecnología de microservicios? (incluya en su respuesta un aspecto relacionado con la migración en vivo)

R/

Utilizar microservicios en datacenters tiene muchos beneficios para los usuarios y los desarrolladores, algunas de estas son:

- Permiten que las aplicaciones sean pequeños servicios compuestos de múltiples contenedores que corren sobre servidores físicos. Esto evita el despliegue de grandes aplicaciones monolíticas sobre los datacenters, eliminando así gran parte del proceso de programación y entrega de software.

- Reducen el número de máquinas físicas en los datacenters. Algunas de estas máquinas, en el futuro, solo serán maquinas con solo el software mínimo necesario para levantar contenedores. Esto reduce costos y espacio.

- Hacen que el tiempo de despliegue de las aplicaciones en un datacenter sea menor y con herramientas para la gestión y monitoreo de contenedores, como Kubernetes y Mesos, se facilite el aprovisionamiento de servicios.

- Por otro lado, los microservicios y contenedores hacen que se utilice mucho mejor el hardware disponible lo que resultara en mayor eficiencia en el consumo de energía de los datacenters.

- También, la utilización de microservicios acompañada de tecnologías como contenedores virtuales, hace que las aplicaciones distribuidas den mejor respuesta **al facilitarse la migración de servicios en caliente a los lugares físicos donde son más usados**, utilizando la modularidad de los microservicios y la portabilidad de los contenedores. Esto tiene impacto en la latencia que los usuarios perciben.

- Además, a través de la automatización y la infraestructura como código, las aplicaciones ahora tengan la capacidad de controlar sus tecnologías de infraestructura subyacentes.

Bibliografía:

Punto 1:

[1] 2017. [Online]. Available: http://www.lomasnuevo.net/cloud/maquinas-virtuales-vs-contenedores/. [Accessed: 21- May- 2017]
[2] 2017. [Online]. Available: https://guiadev.com/docker-vs-maquinas-virtuales-mejor/[Accessed: 21- May- 2017]
[3] 2017. [Online]. Available: http://www.techweek.es/virtualizacion/tech-labs/1003109005901/ventajas-desventajas-virtualizacion.1.html [Accessed: 21- May- 2017]
[4] 2017. [Online]. Available: https://www.1and1.es/digitalguide/servidores/know-how/docker-container-las-ventajas-de-los-contenedores-web/. [Accessed: 21- May- 2017]

Punto 2:

[5] 2017. [Online]. Available: https://es.wikipedia.org/wiki/Docker_(software)[Accessed: 21- May- 2017]
[6] 2017. [Online]. Available: https://es.wikipedia.org/wiki/Vagrant_(software)/. [Accessed: 21- May- 2017]
[7] 2017. [Online]. Available: https://www.campusmvp.es/recursos/post/Docker-vs-Vagrant-diferencias-y-similitudes-y-cuando-usar-cada-uno.aspx [Accessed: 21- May- 2017]
[8] 2017. [Online]. Available: http://www.miguelvilata.com/blog/docker-vs-vagrant-en-la-gestin-de-entornos-de-desarrollo. [Accessed: 21- May- 2017]

Punto 3:

[9] 2017. [Online]. Available: https://innovacionactiva.ieci.es/2015/02/10/nuevas-tendencias-contenedores-y-microservicios/. [Accessed: 21- May- 2017]
[10] 2017. [Online]. Available: https://unpocodejava.wordpress.com/2015/12/14/arquitectura-basada-en-microservicios-parte-3/. [Accessed: 21- May- 2017]
[11] 2017. [Online]. Available: http://searchdatacenter.techtarget.com/es/consejo/Construir-un-entorno-DevOps-con-microservicios-y-contenedores. [Accessed: 21- May- 2017]

Punto 4:

[12] 2017. [Online]. Available: http://wp-alvaromonsalve.rhcloud.com/2016/07/05/herramientas-devops-una-arquitectura-microservicios/. [Accessed: 21- May- 2017]

Punto 5:

[13] 2017. [Online]. http://www.spsolutions.com.mx/integracion-continua-parte-1/. [Accessed: 21- May- 2017]
[14] 2017. [Online]. Available:http://www.javiergarzas.com/2014/08/implantar-integracion-continua.html. [Accessed: 21- May- 2017]

Punto 6:

[15] 2017. [Online]. Available: http://www.cigniti.com/blog/5-approaches-for-automating-microservices-testing/. [Accessed: 21- May- 2017]

Punto 7:

[16] 2017. [Online]. Available: https://martinfowler.com/articles/microservice-testing/#testing-contract-diagram. [Accessed: 21- May- 2017]
[17] 2017. [Online]. Available: https://testdetective.com/microservices-testing/. [Accessed: 21- May- 2017]

Punto 8:

[18] 2017. [Online]. Available: http://www.datacenterknowledge.com/archives/2017/02/23/app-architecture-revolution-microservices-containers-automation/. [Accessed: 21- May- 2017]
[19] 2017. [Online]. Available: http://gestaltit.com/favorites/tom/the-potential-of-microservices-within-the-data-center/. [Accessed: 21- May- 2017]
[20] 2017. [Online]. Available: http://www.datacenterdynamics.com/content-tracks/servers-storage/microservices-will-reshape-your-facility/97322.fullarticle. [Accessed: 21- May- 2017]