TEORÍA

Este examen consta de 40 cuestiones. En cada caso sólo una de las respuestas es correcta. Para indicar la respuesta basta con rellenar la casilla correspondiente en la hoja de respuestas adjunta. Todas las cuestiones tienen el mismo valor. Si son correctas, aportan 0,25 puntos a la nota obtenida. Si son incorrectas descontarán 1/5 del valor correcto, es decir, -0,05 puntos. Conviene pensar cuidadosamente las respuestas.

La duración de esta parte del examen es de 1 hora 40 minutos.

1. Los sistemas distribuidos...

A	...siempre están formados por un conjunto de agentes concurrentes que puede ejecutarse en un conjunto de ordenadores interconectados.
B	...proporcionan algún nivel de tolerancia a fallos.
C	...permiten acceder a recursos compartidos.
D	...pueden utilizar paso de mensajes como su mecanismo de comunicación entre agentes.
E	Todas las anteriores. Todos los apartados anteriores son ciertos. Se necesitan múltiples agentes. Todos ellos deben ejecutarse a la vez, concurrentemente. Se necesitan varios ordenadores para ello. Eso fuerza a que haya una red que los intercomunique. Los sistemas distribuidos deben proporcionar transparencia de fallos (tolerancia a fallos). De otra manera un fallo en un solo nodo podría provocar que el sistema dejara de funcionar. Los recursos pueden ser compartidos por los agentes. De esa manera se pueden reducir los costes económicos, incrementando el uso efectivo de los recursos hardware adquiridos. Pueden utilizarse dos mecanismos de intercomunicación: intercambio de mensajes y compartición de memoria.
F	Ninguna de las anteriores.

2. Algunas de las áreas de aplicación en computación distribuida son...

A	...las transparencias de ubicación, replicación, migración, persistencia, transaccional, acceso y fallos. La transparencia es un objetivo en todo sistema distribuido. Por tanto, todos los tipos de transparencia que se mencionan en este apartado son ejemplos de objetivos en un sistema distribuido, pero ninguno de ellos es una área de aplicación. Las áreas de aplicación son tipos de aplicaciones informáticas que tengan especial sentido en el campo de los sistemas distribuidos.

B	...WWW, redes de sensores, *Internet of Things*, computación cooperativa, *clusters* altamente disponibles, computación en la nube, etc. Cierto. Estas fueron algunas de las áreas de aplicación de la computación distribuida presentadas en el Tema 1.
C	...productor-consumidor con buffer acotado, problema de los lectores-escritores, problema de los cinco filósofos, etc. Falso. Estos son ejemplos de problemas clásicos de la programación concurrente.
D	...secciones críticas, consenso distribuido, difusión atómica, pertenencia a grupo, consistencia final, *commit* distribuido, transacciones anidadas, etc. Falso. Estos son algunos conceptos generales, problemas o mecanismos que podemos encontrar en el campo de la computación distribuida. Ninguno de ellos llega a ser una aplicación (y mucho menos una área de aplicación).
E	Todas las anteriores.
F	Ninguna de las anteriores.

3. El objetivo científico-técnico de la computación en la nube es...

A	...diseñar algoritmos centralizados. Falso. Los algoritmos centralizados suelen introducir un único punto de fallo (por lo que no suelen ser convenientes para garantizar transparencia de fallos) y "cuellos de botella" (es decir, algo que introduce una pérdida de rendimiento importante). No puede recomendarse su uso generalizado para diseñar e implantar servicios distribuidos escalables. La computación en la nube trata de proporcionar servicios distribuidos escalables y adaptativos. Por tanto, el diseño de algoritmos centralizados debería evitarse.
B	...ganar dinero. Falso. Este no es un ejemplo de objetivo científico-técnico.
C	...desplegar contenedores tolerantes a fallos. Falso. Un contenedor no es una pieza de software que pueda ser desplegada, sino el elemento utilizado para desplegar componentes en él. Esta frase no tiene sentido. Además, el objetivo científico-técnico de la computación en la nube no está restringido a la etapa de despliegue.
D	...convertir la creación y explotación de los servicios software en algo más sencillo y eficaz. Ese es precisamente su objetivo principal. Así se describió en el Tema 1.
E	Todas las anteriores.
F	Ninguna de las anteriores.

4. En el área de la computación en la nube hay varios roles relacionados con el ciclo de vida de un servicio software. Esos roles son...

A	Web, trabajador y MV. Falso. Estos son los tres tipos de roles o componentes en Windows Azure, pero estos roles están relacionados con cómo estructurar un servicio en múltiple componentes. Esta cuestión no pregunta por esa clase de roles sino con aquellas relacionadas con las actividades que se llevan a cabo durante las diferentes etapas del ciclo de vida de un servicio software.

B	Monitorización, análisis, planificación, ejecución y conocimiento (MAPE-K). Falso. Estas han sido las cinco etapas o elementos identificados en la especificación original de IBM para el ciclo de vida en una aproximación de computación autónoma. Ese no es el objetivo de esta cuestión.
C	Usuario, desarrollador, administrador y proveedor. Cierto. Los roles que hemos identificado en el Tema 1 son estos cuatro.
D	SaaS, PaaS e IaaS. Falso. Estos no son roles relacionados con actividades del ciclo de vida sino que son los tres modelos de servicio que usualmente se distinguen en el campo de la computación en la nube.
E	Todas las anteriores.
F	Ninguna de las anteriores.

5. ¿Cuál es la relación entre sistemas concurrentes y sistemas distribuidos?

A	Todo sistema distribuido es también un sistema concurrente. Cierto. Los sistemas distribuidos están formados por agentes que colaboran para alcanzar un objetivo común y que son ejecutados en múltiples ordenadores independientes que facilitan la imagen de un sistema único y coherente. Si hay múltiples agentes que colaboran entre sí, entonces también tenemos un sistema concurrente. Por tanto, tener un sistema distribuido también implica tener un sistema concurrente.
B	Los sistemas concurrentes no son sistemas distribuidos. Aunque algunos sistemas concurrentes puedan estar compuestos por agentes que se ejecuten en un solo ordenador (y esto implica un sistema que no es distribuido), un subconjunto de los sistemas concurrentes está formado por todos los sistemas distribuidos. Por ello, esta afirmación no es cierta en todos los casos. Es decir, hay sistemas concurrentes que son sistemas distribuidos.
C	Los sistemas distribuidos no son sistemas concurrentes. Falso. Todos los sistemas distribuidos son también sistemas concurrentes.
D	Todo sistema concurrente es también un sistema distribuido. Falso. Hay múltiples ejemplos de sistemas concurrentes que no necesitan ejecutarse en múltiples ordenadores. Por ejemplo, un proceso Java que implante el problema de los cinco filósofos sería un ejemplo de este tipo. Al poderse implantar mediante un único proceso (por ejemplo, empleando un hilo de ejecución para cada filósofo), no resulta posible hablar de un sistema distribuido.
E	Todas las anteriores.
F	Ninguna de las anteriores.

6. En un sistema distribuido, sus agentes pueden interactuar…

A	…utilizando un mecanismo de intercambio de mensajes. Cierto. Todos los ejemplos que hemos utilizado en el seminario sobre ZeroMQ ilustran este hecho.

B	…siguiendo una aproximación cliente-servidor. Cierto. Esa es una de las posibles aproximaciones. Las aplicaciones que utilicen ROI, RPC o el patrón de comunicación REQ/REP de ZeroMQ siguen esta aproximación.
C	…siguiendo una aproximación *peer-to-peer*. Cierto. Un ejemplo lo proporcionan las aplicaciones de intercambio de ficheros. Bajo esta aproximación los agentes no pueden ser clasificados como clientes ni como servidores.
D	…compartiendo memoria. Cierto. Además del intercambio de mensajes, los agentes de un sistema distribuido pueden comunicarse mediante memoria compartida. Tanto un proceso multi-hilo como las diferentes réplicas de un mismo componente mantienen objetos de memoria que son utilizados concurrentemente por múltiples hilos o procesos.
E	Todas las anteriores.
F	Ninguna de las anteriores.

7. En un sistema distribuido…

A	…cada uno de sus agentes tiene un estado privado y no interactúa con otros agentes. Falso. Si no interactuaran con el resto de agentes, los procesos serían entidades independientes y aisladas. En ese caso sería un proceso completamente independiente y no formaría parte de un sistema distribuido con cierto objetivo común para todos sus agentes.
B	…los agentes pueden tener su propio estado, pero colaboran para alcanzar un objetivo global. Cierto. Cada agente puede tener su propio estado, aunque ese estado no necesita ser "privado" (sus elementos pueden estar compartidos con otros agentes). Además, esos agentes deben colaborar. Eso es clave en los agentes de un sistema concurrente o distribuido. De otra manera, serían procesos aislados.
C	…los agentes son independientes y no comparten recursos. Falso. Ya se ha explicado en los dos apartados anteriores.
D	…la concurrencia es origen de muchos problemas. Por tanto, los sistemas distribuidos modernos no son concurrentes. Falso. La concurrencia es una propiedad inherente en los sistemas distribuidos; es decir, los sistemas distribuidos deben ser concurrentes.
E	Todas las anteriores.
F	Ninguna de las anteriores.

8. Las interacciones *peer-to-peer*…

A	…no se utilizan en sistemas distribuidos. Falso. Es una de las aproximaciones para la interacción en sistemas distribuidos. La otra es la interacción cliente-servidor.

B	…asumen que los agentes están interesados en alguna clase de recurso y cuando un agente obtiene uno de esos recursos, lo distribuye a otros agentes. Cierto. Ese es el núcleo de esta aproximación. Una vez el agente obtiene un recurso (aunque sea solo una parte, como en los sistemas de intercambio de ficheros) puede ofrecerlo a otros agentes. Por tanto, no hay ninguna distinción entre rol cliente y rol servidor. Todos los agentes son capaces de comportarse de ambas maneras: como fuente y como destino de las transferencias de recursos.
C	…distinguen claramente entre agentes clientes y agentes servidores. Falso. De manera general a los agentes de un sistema peer-to-peer se les llama "servents". Esa es la contracción de *SERVer* y *cliENT*.
D	…son un tipo de interacción fuertemente centralizada. Falso. Normalmente los sistemas en los que se utilice la interacción peer-to-peer están descentralizados.
E	Todas las anteriores.
F	Ninguna de las anteriores.

9. La WWW…

A	…es un ejemplo de aplicación distribuida que sigue un modelo de interacción *peer-to-peer*. Falso. Esta área de aplicaciones distribuidas suele utilizar una interacción cliente-servidor.
B	…usa los navegadores web como un tipo específico de agente servidor. Falso. Los navegadores web suelen participar como agentes clientes en su interacción con otros agentes.
C	…es un tipo de área de aplicación de los sistemas distribuidos donde se transfieren documentos entre servidores y clientes. Cierto. En el caso más general, se transfieren documentos (por ejemplo, el contenido de las páginas web estáticas) entre servidores y clientes.
D	…no utiliza un modelo de interacción cliente-servidor. Falso. Suele utilizarse precisamente ese.
E	Todas las anteriores.
F	Ninguna de las anteriores.

10. Sobre los modelos de servicio en la computación en la nube:

A	IaaS: Proporciona aplicaciones generales como su servicio. Un ejemplo es Google Docs / Google Drive. Falso. Esa es la definición y un ejemplo de modelo SaaS.

B	SaaS: Automatiza el despliegue y la elasticidad de las aplicaciones. Un ejemplo es Windows Azure. Falso. Esa es la definición y un ejemplo de modelo PaaS.
C	PaaS: Proporciona una infraestructura virtual como su servicio, donde los componentes pueden ser desplegados de una manera no automatizada. Ejemplo: Amazon EC2. Falso. Esa es la definición y un ejemplo de modelo IaaS.
D	IaaS utiliza los servicios proporcionados por SaaS, que a su vez utiliza los servicios del PaaS. Así se define una arquitectura de tres niveles. Falso. SaaS se ubica sobre PaaS y este sobre IaaS.
E	Todas las anteriores.
F	Ninguna de las anteriores.

11. Las propiedades que se exigen a un sistema distribuido son...

A	Control centralizado. No. El control centralizado difícilmente podrá escalar. La escalabilidad es un objetivo de los sistemas distribuidos.
B	Actualizaciones diarias del software. No. No suele ser necesario actualizar con esa frecuencia. Como en cualquier otra área de la computación, las actualizaciones se necesitan para corregir errores de programación o para extender la funcionalidad de las aplicaciones.
C	Un grado extremadamente alto de concurrencia en cada agente implantado. No necesariamente. Los sistemas distribuidos, por definición, son concurrentes pero esa concurrencia puede ser obtenida ejecutando múltiples hilos o procesos. Es decir, no se obliga a que cada agente sea un proceso multi-hilo. Podemos implantar la concurrencia mediante varios procesos con un solo hilo de ejecución cada uno. Por ejemplo, en Node.js sucederá eso.
D	Estar programados en Node.js. No. Se puede utilizar cualquier lenguaje de programación que resulte útil para implantar la funcionalidad de cada componente. Java, C y C++ son ejemplos típicos. La distribución no depende para nada del lenguaje de programación utilizado.
E	Todas las anteriores.
F	Ninguna de las anteriores.

12. Algunos de los problemas fundamentales (y sus soluciones) en la computación distribuida son...

A	Coordinación de componentes (vía paso de mensajes, diseñando algoritmos que requieran un intercambio mínimo de mensajes).

B	Gestión de fallos (utilizando replicación, detectores de fallos y mecanismos de recuperación).
C	Persistencia de estado (vía *commit* distribuido, replicación y almacenamiento persistente).
D	Consistencia de estado (utilizando replicación y protocolos de consistencia).
E	Todas las anteriores. El tema 2 presenta la coordinación, la gestión de fallos, la persistencia de estado y la consistencia de estado como cuatro problemas fundamentales en la computación distribuida. Algunas de sus posibles soluciones han sido incluidas en cada uno de los apartados anteriores, entre paréntesis.
F	Ninguna de las anteriores.

13. El modelo de sistema distribuido presentado en el Tema 2...

A	...considera detalles de bajo nivel sobre el comportamiento del sistema. Así garantiza un resultado más preciso en la etapa de diseño del software. Falso. Los detalles de bajo nivel no se consideran en los modelos. Un modelo elimina esos detalles y presenta una imagen de un determinado sistema o mecanismo empleando un nivel de abstracción alto.
B	...asume que todos los agentes son multi-hilo. Falso. El modelo asume que los agentes son procesos secuenciales, representados mediante autómatas. La ejecución de una acción se modela como una transición entre estados. En cada autómata solo puede haber una única secuencia de transiciones; es decir, representa un único hilo de ejecución.
C	...siempre asume procesos síncronos y comunicación síncrona. Falso. Por ejemplo, la comunicación puede ser asíncrona. En ese Tema 2 se definieron diferentes niveles de sincronía, considerando múltiples aspectos: comunicación, relojes, canales, procesos… De hecho, se sugería que la comunicación debía ser asíncrona para mejorar la escalabilidad.
D	...representa la ejecución de procesos como una secuencia de acciones o eventos interrumpibles. Falso. Las acciones deben ser atómicas, es decir, no pueden ser interrumpidas una vez iniciadas. Los términos evento y acción no son sinónimos. Un evento se comporta como una guarda o precondición para la ejecución de una acción. Esto implica que la acción no será iniciada hasta que ocurra el evento correspondiente. No tiene sentido hablar de “eventos interrumpibles”.
E	Todas las anteriores.
F	Ninguna de las anteriores.

14. Cuando comparamos servidores asíncronos con servidores multi-hilo...

A	Los servidores asíncronos implantan de manera trivial las acciones atómicas definidas en el modelo de sistema distribuido propuesto en el Tema 2. Verdadero. Los servidores asíncronos suelen tener un solo hilo de ejecución y usan eventos como guardas o precondiciones para el inicio de la ejecución de una acción. Esas acciones no pueden interrumpirse una vez iniciadas. Por tanto, se ejecutan atómicamente.

B	"Dirigido por eventos" es un sinónimo para "multi-hilo". Falso. "Dirigido por eventos" es un sinónimo para "servidor asíncrono".
C	Los servidores asíncronos se bloquean cuando hay concurrencia mientras los servidores multi-hilo realizan accesos concurrentes a recursos sin bloquearse. Falso. Cada servidor asíncrono suele tener un solo hilo. Por tanto, no pueden bloquearse debido a interferencias con otros hilos. Por el contrario, en un servidor multi-hilo sí que necesitaremos bloquear a los hilos cuando intenten acceder a algún recurso compartido con el resto de hilos. Eso ocurre con las variables compartidas, por ejemplo, para implantar secciones críticas con la protección adecuada para evitar inconsistencias en sus modificaciones.
D	JavaScript es un ejemplo de lenguaje de programación específicamente diseñado para implantar servidores multi-hilo. Falso. Es un ejemplo de lenguaje de programación que puede implantar servidores asíncronos.
E	Todas las anteriores.
F	Ninguna de las anteriores.

15. Estas son algunas propiedades que debemos exigir a los sistemas distribuidos…

A	Tolerancia a fallos.
B	Alta disponibilidad.
C	Seguridad.
D	Escalabilidad.
E	Todas las anteriores. Cierto. Estas cuatro propiedades han sido listadas en la introducción del Tema 2 como propiedades a exigir en todo sistema distribuido.
F	Ninguna de las anteriores.

16. La consistencia de estado significa que…

A	El estado gestionado por un componente sólo puede tener una única instancia en el sistema; p. ej., está almacenado en una base de datos centralizada. Falso. La consistencia de estado es un sinónimo para "consistencia entre réplicas". Ese término solo tiene sentido cuando haya múltiples copias de cada elemento o dato. Este apartado afirma justo lo contrario. Asume que hay una sola instancia en todo el sistema. En ese caso, no habrá réplicas. Por tanto, no podríamos hablar sobre la consistencia de estado o sobre el modelo de consistencia utilizado en ese sistema.

B	Todas las variables globales tendrían que ser accedidas en exclusión mutua para evitar condiciones de carrera. Falso. Véase la justificación del apartado anterior. Las "condiciones de carrera" también podrán generar inconsistencias, pero esas inconsistencias no están relacionadas con la divergencia entre las distintas réplicas de un dato en un sistema distribuido. Bajo un modelo de programación con servidores asíncronos (con un solo hilo de ejecución cada uno) no será necesario preocuparse por las "condiciones de carrera". La exclusión mutua en el acceso a las variables de un determinado proceso estará garantizada sin necesidad de tomar ninguna precaución adicional.
C	Cuando un componente está replicado hay un conjunto de invariantes que limitan el grado de divergencia entre las réplicas de cada uno de sus datos. Cierto. Ese es el significado utilizado en el Tema 2 para este concepto.
D	Cuando un componente está replicado, o bien todas sus réplicas están activas y funcionan correctamente o todas fallan y no pueden continuar. Falso. No tiene excesivo sentido obligar a que todas las réplicas de un servicio paren a la vez. La replicación se utiliza para conseguir la continuidad de los servicios. Cuando una réplica falle, las restantes deben conseguir que el servicio siga estando disponible (es decir, que el servicio no falle).
E	Todas las anteriores.
F	Ninguna de las anteriores.

17. La persistencia de estado significa que…

A	Una aplicación distribuida no puede tener datos volátiles. Todos los datos deberían residir en ficheros (en el disco duro) o en bases de datos. Falso. Cuando se garantice la persistencia de estado, aseguraremos la durabilidad de los datos. Sin embargo, esto no prohíbe que utilicemos otras variables temporales. Por ejemplo, para mantener algunos datos que puedan calcularse a partir de los que se guarden de manera persistente. Esto puede ser interesante si esos datos "derivados" deben ser accedidos con frecuencia. En lugar de calcularlos en cada petición, los mantendremos en algunas variables, en memoria principal.

B	El acceso a cualquier dato tendría que realizarse siempre bajo la protección de una transacción distribuida. Falso. El término "transacción distribuida" se refiere a un conjunto de subtransacciones ejecutadas cada una en un nodo distinto. Todas las subtransacciones colaboran en una secuencia global de operaciones sobre la que se aseguran las propiedades de atomicidad y aislamiento. No es obligatorio el uso de transacciones distribuidas para acceder a datos almacenados en memoria secundaria. En muchos casos bastará con ejecutar una transacción local o, simplemente, un acceso aislado al dato de interés.
ℂ	Cuando se aplique un cambio sobre un dato persistente, su durabilidad está garantizada. Cierto. Este es el significado utilizado en el Tema 2 para referirnos a la "persistencia de estado".
D	Cada dispositivo de almacenamiento secundario que sea utilizado por una aplicación distribuida estará replicado. Falso. La replicación no tiene por qué aplicarse sobre los dispositivos de almacenamiento secundario para garantizar la durabilidad de la información. Si existiera esa obligación, todos los discos utilizados en un sistema distribuido deberían ser discos RAID (*Redudant Array of Inexpensive Disks*). No es la solución común. Aparte, la persistencia de estado no tiene por qué depender del almacenamiento secundario. Algunos SGBD de altas prestaciones mantienen los datos en memoria principal, utilizando múltiples ordenadores con mucha memoria RAM. En ese caso sí que resulta necesaria la replicación para asegurar la durabilidad. VoltDB es un ejemplo de este tipo.
E	Todas las anteriores.
F	Ninguna de las anteriores.

18. En el modelo de sistema distribuido del Tema 2…

A	Los eventos internos se refieren a acciones aplicadas por la lógica del agente. Por ejemplo, para recibir un mensaje. Falso. Los eventos internos se han definido así, pero la recepción de un mensaje no es un evento interno. Es un evento externo. Está relacionado con una acción que ha sido iniciada por otro agente.

B	Tanto los eventos internos como los externos generan transiciones de estado. Cierto. Los eventos, de cualquier tipo, generan transiciones de estado ya que inician la ejecución de una acción.
C	La ejecución de un agente se modela como una única secuencia de eventos. Tanto la concurrencia como los agentes multi-hilo no pueden representarse. Falso. La primera frase es cierta, pero la segunda oración es falsa. Aunque los agentes multi-hilo no están soportados por este modelo, eso no implica que se renuncie a la concurrencia. Un sistema distribuido está compuesto por múltiples agentes. A pesar de que cada uno sea un agente secuencial, el escenario global es concurrente. Los agentes compartirán recursos. Colaborarán para alcanzar un objetivo común. Habrá interacción entre ellos. Con ello, estarán componiendo un sistema concurrente. Por tanto, la concurrencia está representada y es admitida por este modelo de sistema.
D	Ya que todos los sistemas distribuidos deben ser transparentes ante fallos, este modelo asume que los fallos nunca ocurren. Falso. La especificación dada en este modelo asume que los fallos podrán ocurrir. El tipo de fallo asumido es la parada de los procesos.
E	Todas las anteriores.
F	Ninguna de las anteriores.

19. La comunicación en el modelo de sistema sencillo del Tema 2…

A	…asume que los eventos internos definen una relación "precede localmente" de orden total en cada agente. Cierto. Esta es la condición FIFO incluida en la relación causal.
B	…asume que los eventos externos definen una relación "causa directamente" donde un evento de salida es la causa de un evento de entrada. Cierto. El evento de salida más común es el envío de un mensaje. El evento de entrada típico es la recepción de un mensaje. La transmisión de mensajes está modelada por la condición "causa directamente" de la relación causal.
C	El cierre transitivo de las relaciones "causa directamente" y "precede localmente" define la relación de comunicación "causal". Cierto. Lamport definió la relación "happens before" (o relación causal) como el cierre transitivo de dos condiciones elementales (secuencia interna de eventos o "precede localmente" y eventos relacionados por la transmisión de un mensaje o "causa directamente"). Esas dos condiciones reciben en el Tema 2 los nombres mencionados en este apartado.
D	La relación de comunicación "causal" permite identificar a los eventos no relacionados como "concurrentes". Cierto. La relación causal establece dependencias entre los eventos. Cuando un par de eventos no está relacionado (es decir, no se cumple "a → b" ni "b → a"), entonces diremos que esos dos eventos son concurrentes (esto es, "a || b").
E	Todas las anteriores.
F	Ninguna de las anteriores.

20. Para especificar programas en el modelo de sistema sencillo del Tema 2…

A	El modelo asume guardas atómicas, protegidas por acciones. Falso. El modelo asume acciones atómicas protegidas por guardas.

B	Las acciones atómicas son una fuente potencial de errores. Por ello, están implantadas como bloques de código interrumpibles en todos los lenguajes de programación. Falso. Las acciones atómicas evitan las "condiciones de carrera". Por tanto, son una herramienta que evita muchos errores a la hora de ejecutar un programa. Esa atomicidad debe ser cuidadosamente mantenida cuanto nuestro modelo de sistema pase a ser implantado sobre un sistema real. El lenguaje de programación utilizado debe facilitar herramientas para mantener esa atomicidad. Eso implica que cada fragmento de código que implante una acción debe ser lógicamente no interrumpible.
C	Las guardas son una fuente potencial de condiciones de carrera. Por eso no se utilizan en los lenguajes de programación multi-hilo. Falso. Las guardas y las acciones atómicas evitan las "condiciones de carrera". Esto ya se ha comentado en el apartado anterior. Cuando el lenguaje de programación soporte múltiples hilos de ejecución, las acciones atómicas deben traducirse como operaciones de un monitor y las guardas serían las condiciones que permiten acceder a (o continuar con la ejecución de) esas operaciones. Esta aproximación ya fue estudiada en CSD.
D	El modelo asume acciones atómicas protegidas por condiciones (también llamadas guardas). Cierto. Esto ya se ha indicado en el primer apartado y se ha explicado en el apartado C.
E	Todas las anteriores.
F	Ninguna de las anteriores.

21. El middleware es una capa de software que...

A	...está colocado entre el hardware y el sistema operativo. Falso. Está ubicado sobre el sistema operativo, facilitando cierto grado de transparencia de distribución a las aplicaciones desarrolladas sobre él.
B	...garantiza transparencia de fallos para los componentes de las aplicaciones distribuidas. Falso. Se han presentado diferentes tipos de middleware en el Tema 3. La mayoría no gestionan los fallos ni soportan directamente la replicación de componentes. Por tanto, son incapaces de garantizar la transparencia de fallos. Por ejemplo, la RPC (llamada a procedimiento remoto) es un ejemplo de middleware. Facilita transparencia de ubicación, pero no replica automáticamente los servicios invocados. Por tanto, cuando un servidor falle utilizando RPC, su cliente obtendrá una excepción como resultado de ese fallo.
C	...utiliza contenedores para desplegar servicios distribuidos. Falso. Un middleware no tiene por qué preocuparse del despliegue de componentes. RPC y RMI son dos ejemplos de middleware que no lo hacen.
D	...está implementado en JavaScript. Falso. Los middleware facilitan interoperabilidad. El lenguaje de programación utilizado para programar sus elementos no tiene por qué ser JavaScript.
E	Todas las anteriores.
F	Ninguna de las anteriores.

22. Algunas características del middleware son...

A	Proporciona una API estándar.
B	Utiliza protocolos de interacción estándar.
C	Proporciona servicios de interés general.
D	Garantiza la interoperabilidad de componentes desplegados sobre distintas plataformas.
E	Todas las anteriores. Todas estas características se han listado y explicado en la guía de estudio del Tema 3. Esas características fueron descritas y propuestas en el artículo escrito por Phil Bernstein en el que se analizó qué debía entenderse como middleware y se clasificaron los diferentes tipos de middleware existentes.
F	Ninguna de las anteriores.

23. Los sistemas de objetos distribuidos...

A	...necesitan un middleware para gestionar la invocación a objetos remotos. Cierto. Se utiliza un *Object Request Broker (ORB)* para este fin. Es un tipo particular de middleware.
B	...son inherentemente menos escalables que los sistemas distribuidos basados en sistemas de mensajería. Cierto. Usan una interacción cliente-servidor. Esto implica un modelo de comunicación sincrónico (similar al REQ/REP de ZeroMQ) que bloquea al cliente hasta que una respuesta es enviada por el servidor. Toda fuente de bloqueos debería ser evitada para desarrollar sistemas distribuidos escalables.
C	...tienen un mayor acoplamiento que los sistemas distribuidos no orientados a objetos. Cierto. Los sistemas de objetos distribuidos utilizan referencias a objeto para acceder a los objetos remotos. En muchos casos, esto permite que se transfiera una buena cantidad de información entre diferentes componentes del sistema en cada invocación. En esa situación se necesitarían mensajes grandes y se reduciría el rendimiento del sistema. Una alta tasa de mensajes enviados y recibidos entre componentes que interactúen entre sí sería un indicativo de un acoplamiento alto. Eso debe evitarse en los sistemas escalables.
D	...ofrecen, normalmente, transparencia de ubicación. Sí. El uso de *proxies* en los clientes proporciona transparencia de ubicación. Los procesos clientes no saben en qué ordenador reside el objeto que se está invocando.
E	Todas las anteriores.
F	Ninguna de las anteriores.

24. Los sistemas de mensajería...

A	...son persistentes cuando el emisor permanece bloqueado esperando alguna

	respuesta del receptor. Falso. Son sincrónicos en ese caso.
B	...son no persistentes cuando la comunicación está dirigida por un agente gestor (*broker*). Falso. Los gestores (*brokers*) solo se necesitan en la comunicación persistente.
C	...pueden ser persistentes y estar basados en gestor (*broker-based*). Cierto. Esta es una combinación válida. El estándar AMQP (*Advanced Message Queueing Protocol*) utiliza esa aproximación.
D	...pueden ser sincrónicos y no persistentes. ZeroMQ es un ejemplo de este tipo. Falso. Esa combinación es válida y llega a ser utilizada en algunos casos. Por ejemplo, RPC es un ejemplo bien conocido. Sin embargo, ZeroMQ no utiliza esa combinación: es débilmente persistente (en lugar de "no persistente"), sin gestor (*brokerless*) y asincrónico.
E	Todas las anteriores.
F	Ninguna de las anteriores.

25. Los estándares…

A	…facilitan la interoperabilidad. Cierto. Este es uno de sus principales objetivos.
B	…no pueden utilizarse en sistemas distribuidos. Falso. Los middleware implantan estándares y se utilizan en los sistemas distribuidos.
C	…garantizan la transparencia de fallos. Falso. No todos los estándares son específicos de los sistemas distribuidos. Sin utilizar componentes replicados no se podrán gestionar algunos escenarios de fallo. Por tanto, incluso los estándares utilizados en sistemas distribuidos podrán en algunos casos no gestionar adecuadamente los fallos.
D	…mejoran el rendimiento. Falso. Muchos estándares facilitan soluciones eficientes a ciertos problemas, pero puede haber otras soluciones no estándar con mejor rendimiento. Por tanto, no todos los estándares facilitan la solución óptima desde el punto de vista del rendimiento.
E	Todas las anteriores.
F	Ninguna de las anteriores.

26. Desde el punto de vista de un programador, cuando se sigue un estándar…

A	…los programas son fáciles de escribir, pues los elementos utilizados ofrecen una menor complejidad.

B	…el resultado final es más fiable, pues el estándar define claramente cómo hacer las cosas.
C	…el código tiene un mantenimiento sencillo pues, aunque los estándares se actualicen, sus cambios suelen mantener la interoperabilidad.
D	…los programas son fáciles de escribir, pues los estándares están basados en conceptos claros y bien definidos.
E	Todas las anteriores. Cierto. Todos estos aspectos fueron considerados cuando se describieron los estándares en el Tema 3. La estandarización y la interoperabilidad son dos de los principales objetivos de los *middleware*.
F	Ninguna de las anteriores.

27. Dos aproximaciones de invocación a métodos remotos en el área de servicios web son…

A	SOAP y REST. Cierto. Estas han sido las dos aproximaciones presentadas con ese objetivo en el Tema 3. La guía de estudio explicaba detalladamente la segunda. SOAP se describirá en otras asignaturas.
B	ZeroMQ y nanomsg. Falso. Son ejemplos de middleware de comunicaciones basados en mensajes que implantan comunicación asincrónica.
C	RPC y RMI. Falso. Son mecanismos de invocación remota utilizados en sistemas orientados a objetos o basados en procesos. Ninguno de los dos ha sido específicamente diseñado para ser utilizado en servicios web.
D	Cliente-servidor y *peer-to-peer*. Falso. Son las dos aproximaciones genéricas de interacción entre agentes en la computación distribuida. La interacción peer-to-peer no está basada en invocación de métodos u operaciones.
E	Todas las anteriores.
F	Ninguna de las anteriores.

28. El estilo arquitectónico REST…

A	…usa HTTP como “transporte”.
B	…usa sólo cuatro métodos “básicos”: GET, PUT, POST y DELETE.
C	…usa su método GET para acciones de solo lectura.
D	Toma el estilo arquitectónico cliente-servidor como base y promueve el uso de servidores “*stateless*” (para gestionar fácilmente los fallos).
E	Todas las anteriores. Esas son las principales características del estilo arquitectónico REST. Fueron descritas en detalle en la guía de estudio del Tema 3.
F	Ninguna de las anteriores.

29. Algunos ejemplos de “otro middleware” son…

A	gedit. Falso. Es un editor de texto convencional de los entornos X-Window. Como tal, es solo un ejemplo de aplicación de nivel de usuario. Tampoco asume que

	deba ser ejecutada en un entorno distribuido.
B	OAuth. Cierto. Es un ejemplo de *middleware* relacionado con seguridad.
C	Linux. Falso. Es un ejemplo de sistema operativo.
D	MS-DOS. Falso. Es un ejemplo de sistema operativo.
E	Todas las anteriores.
F	Ninguna de las anteriores.

30. El middleware de nombrado…

A	…garantiza transparencia de fallos. Falso. El servicio de nombres no participa de manera directa en ningún subsistema de gestión de fallos. Aunque asumamos que colabore en un subsistema de ese tipo, no será su componente principal. Por tanto, es incapaz de asegurar transparencia de fallos por sí mismo.
B	…proporciona transparencia de ubicación. Cierto. Es un componente clave para asegurar transparencia de ubicación. Con su ayuda los clientes pueden referirse a agentes externos (servidores) utilizando un nombre. Esos nombres son traducidos a direcciones por el middleware de nombrado. Así, la dirección real no es conocida por los clientes. Eso proporciona transparencia de ubicación. Tanto DNS como el servicio de nombres de los sistemas CORBA son ejemplos de este tipo de servicios.
C	…implementa servidores sin estado (*stateless*). Falso. Que los servicios sean *stateless* o *stateful* no depende de la utilización de un middleware de nombrado.
D	…mejora la escalabilidad del sistema. Falso. La resolución de nombres introduce un paso adicional (que normalmente requerirá comunicación con una entidad remota) a la hora de acceder a un objeto, servicio o proceso referenciado mediante un nombre. Por ello, aunque los servicios de nombres son convenientes al facilitar transparencia de ubicación, sus resultados deben mantenerse en cachés para no perjudicar excesivamente al rendimiento y a la escalabilidad del sistema.
E	Todas las anteriores.
F	Ninguna de las anteriores.

31. Un *Acuerdo de Nivel de Servicio* (o SLA) es…

A	…un acuerdo entre proveedores de servicio y clientes de servicio.
B	…una especificación de características de servicio (p. ej., funcionalidad, tiempo de respuesta, rendimiento, disponibilidad…) y sus niveles a garantizar.
C	…uno de los aspectos a considerar para decidir el número de instancias de cada componente durante el despliegue de un servicio.
D	…algo a considerar en sistemas PaaS para rellenar el plan de despliegue y el plan de escalado de un servicio determinado.
E	Todas las anteriores. Todos los apartados anteriores son ciertos. Esa información se facilitó en las diferentes secciones del Tema 4. El SLA es un acuerdo establecido entre proveedores y clientes de un servicio. En un escenario ideal, ese acuerdo debe especificar la calidad de servicio que ofrecerá el proveedor así como la máxima carga introducida por cada cliente. Esa información se utiliza durante la etapa de despliegue para gestionar los aspectos descritos en los apartados C y D.
F	Ninguna de las anteriores.

32. En el Tema 4, un servicio es…

A	…una aplicación distribuida que ha sido desplegada y permanece activa. Cierto. Esta ha sido la definición informal proporcionada en la introducción del Tema 4.
B	…un conjunto de *scripts* independientes con un plan de despliegue. Falso. En el caso general un servicio está formado por múltiples componentes que colaboran para proporcionar un objetivo común. No pueden ser independientes, sino estar estrechamente relacionados. Nada obliga a que se implanten como *scripts*.
C	…una futura aplicación distribuida que todavía está en sus etapas de análisis o diseño. Falso. Un servicio es una aplicación distribuida que ya está en ejecución; de otro modo no podría utilizarse. Por ello, ya ha superado las etapas de análisis y diseño en su ciclo de vida.
D	…un programa Node.js que es ejecutado por un único usuario. Falso. En tal caso ese programa no necesitaría un entorno distribuido. En el contexto del Tema 4, el término "servicio" se refiere a servicios software **distribuidos**.
E	Todas las anteriores.
F	Ninguna de las anteriores.

33. Estas son algunas tareas a considerar cuando una aplicación distribuida está siendo desplegada…

A	Decidir cuántas instancias de cada componente tendrían que ser ejecutadas y dónde.
B	Decidir qué servicios dependientes tendrían que ser utilizados por esta aplicación distribuida.
C	Decidir el orden en que cada uno de sus componentes debería ser iniciado.
D	Contactar con el sistema operativo o contenedor en cada anfitrión para que inicie sus componentes.
E	Todas las anteriores. Cierto. Todas esas tareas son ejemplos de pasos que deben ser tenidos en cuenta.
F	Ninguna de las anteriores.

34. La administración del ciclo de vida de un servicio está estrechamente relacionada con el despliegue. Algunas de sus tareas son…

A	Actualización de componentes.

B	Cambios de configuración.
C	Detección y recuperación de fallos en los componentes.
D	Decisiones de escalado, dependiendo de la carga actualmente soportada.
E	Todas las anteriores. Cierto. De nuevo, todas esas tareas forman parte de la gestión del ciclo de vida de un servicio. Fueron descritas dentro del Tema 4.
F	Ninguna de las anteriores.

35. Algunos problemas que surgen cuando una aplicación tradicional es desplegada en un ordenador de sobremesa son…

A	Resolución de dependencias del software; es decir, encontrar las bibliotecas apropiadas de las que dependa la aplicación.
B	Dar valores apropiados a las variables de entorno utilizadas por la aplicación, si las hubiere.
C	Configurar adecuadamente la aplicación (p. ej., vía registro en Windows, archivos de configuración en Linux, ficheros en `/Library` en Mac OS, etc.)
D	Averiguar si los requisitos de instalación de la aplicación están soportados por el estado actual del ordenador anfitrión y su sistema operativo.
E	Todas las anteriores. Cierto. Todas las afirmaciones anteriores proporcionan ejemplos de los aspectos a considerar al desplegar una aplicación en un ordenador convencional. El apartado C suele ser gestionado por el programa instalador y resulta transparente para el usuario.
F	Ninguna de las anteriores.

36. Algunos de los elementos en un descriptor de despliegue son…

A	Middleware de nombrado. Falso. No tiene sentido incluir un middleware (el que sea) en un descriptor de despliegue.

B	Orden a utilizar por el usuario (p. ej., docker). Falso. El descriptor de despliegue dirige las tareas de ese despliegue. La orden a utilizar por el usuario se explicó en el contexto del Seminario 6, al describir los contenedores. No forma parte del descriptor de despliegue.
C	Plan de despliegue. Cierto. Las principales partes de un descriptor de despliegue son un plan de despliegue, las plantillas de configuración de componentes ya rellenadas y las descripciones de dependencias.
D	Dockerfile. Falso. El "Dockerfile" es un fichero de configuración específico, necesario cuando se utilicen contenedores *docker*. Se utiliza para proporcionar la secuencia de instrucciones necesaria para construir una imagen *docker*. No forma parte de los descriptores de despliegue "normales".
E	Todas las anteriores.
F	Ninguna de las anteriores.

37. Un componente necesita los elementos siguientes para ser desplegado…

A	Su programa (o BLOB).
B	Una plantilla de configuración rellenada.
C	Una descripción de todas sus dependencias.
D	Una especificación de su "endpoint".
E	Todas las anteriores. Cierto. Todos estos elementos se necesitan para desplegar un componente. Los apartados B, C y D son partes habituales en los descriptores de despliegue. Sin el apartado A no se podría desplegar nada. Es precisamente el elemento a desplegar.
F	Ninguna de las anteriores.

38. La inyección de dependencias…

A	…desacopla el código del componente de cualquier implementación concreta de dependencias, y está soportada en los entornos de contenedores. Cierto. Esta es la definición de inyección de dependencias utilizada en el Tema 4.

B	…requiere el uso de variables de entorno para resolver dependencias. Falso. Cuando se utilice la inyección de dependencias, éstas se resolverán "incrustando" los módulos apropiados en el componente durante la fase de despliegue. Con ese tipo de resolución de dependencias, las variables de entorno puede que no se necesiten.
C	…requiere el uso de archivos de configuración para resolver dependencias. Falso. Ocurre lo mismo que en el caso anterior.
D	…soluciona todas las dependencias estáticamente, es decir, durante la implementación. Falso. La inyección de dependencias es un mecanismo que permite un enlace tardío entre componentes, durante la fase de despliegue generalmente. Decidir ese enlace mientras se escribe el programa, con un mecanismo estático, es poco aconsejable. Si hubiera algún cambio en los componentes a utilizar, eso nos obligaría a no poder resolver las dependencias, con lo que no se podrían desplegar los componentes bajo ese tipo de resolución.
E	Todas las anteriores.
F	Ninguna de las anteriores.

39. En el modelo de servicio IaaS…

A	…el despliegue está completamente automatizado por el proveedor. Falso. Eso solo ocurriría en el modelo de servicio PaaS y todavía no ha alcanzado ese grado de madurez.
B	…varias decisiones de despliegue iniciales no están automatizadas: cantidad de instancias por componente, tipo de MV requerido por cada componente… Verdadero. Esas decisiones deben ser tomadas por el cliente (que es quien intenta desplegar una aplicación) bajo ese modelo de servicio. Obsérvese que ese cliente no es el usuario final de las aplicaciones, sino su desarrollador.
C	…las decisiones de despliegue relacionadas con el ciclo de vida están automatizadas; p. ej., qué niveles de carga disparan acciones de escalado, cómo se actualiza el software de un componente… Falso. Ocurre lo mismo que en el apartado A.
D	…ningún soporte para el despliegue es gestionado por el proveedor. Falso. Un soporte mínimo sí que se llega a facilitar. Por ejemplo, para desplegar una imagen sobre una máquina virtual.
E	Todas las anteriores.
F	Ninguna de las anteriores.

40. En Windows Azure, algunos aspectos de su soporte de despliegue son…

A	Hay un plan básico de actualización de servicios, a pesar de que no soporta servicios con estado.
B	Los componentes se llaman “roles”.
C	Hay una administración básica de dominios de fallo que mejora la disponibilidad de servicio.
D	No hay ningún plan de secuenciación de despliegue.
E	Todas las anteriores. Cierto. Todos estos aspectos fueron incluidos en la descripción de Windows Azure proporcionada en la última sección del Tema 4.
F	Ninguna de las anteriores.