Jonathan Anderson

AWS

Certified Solutions Architect

Associate

The ultimate guide for the

SAA-C03

Imagen que contiene lego, juguete

Descripción generada automáticamenteE X A M

# Chapter 6

**Conclusion - Architecture Best Practices**

La plataforma AWS Well-Architected ayuda a los arquitectos de la nube a crear una infraestructura segura, potente, resistente y eficiente para sus aplicaciones y cargas de trabajo. La excelencia operativa, la protección, la confiabilidad, la calidad del rendimiento y la optimización de costos son los cinco pilares en los que se basa AWS. La plataforma AWS Well- Architected proporciona un enfoque cohesivo para que los clientes y socios evalúen las arquitecturas e implementen diseños escalables a lo largo del tiempo.

AWS Well-Architected Framework comenzó como un único documento técnico, pero se ha ampliado para incluir objetivos específicos del dominio, laboratorios prácticos y la herramienta AWS Well-Architected Tool. Disponible de forma gratuita en la consola de administración de AWS, la herramienta AWS WA proporciona un mecanismo para revisar regularmente sus cargas de trabajo, identificar problemas de alto riesgo y registrar sus mejoras.

AWS tiene un ecosistema de cientos de miembros del Programa de socios bien diseñados. Puede contratar a un socio en su área para que lo ayude a analizar y revisar sus solicitudes.

## Differences Between Traditional and Cloud Computing Environments

En un entorno de TI tradicional, una empresa debe entregar capacidad basada en la mejor estimación de tráfico pico (por ejemplo, Black Friday). Esto implica que, durante mucho tiempo, una parte sustancial del poder puede desperdiciarse de manera efectiva.

Esta es la razón por la que Cloud Computing fue creado para su propio uso; puede utilizar la fuerza adicional obtenida de otros (en el caso de AWS-Amazon). Es posible iniciar y detener servidores, bases de datos, almacenamiento, etc. en horas o incluso minutos, según sea necesario.

## Cloud computing with AWS has four main advantages:

Proporciona servicios cuando son necesarios utilizando la infraestructura general de AWS para implementar físicamente sus aplicaciones más cerca de los usuarios (en un modelo tradicional, debe crear sus propios centros de datos donde se encuentran los usuarios); una gran cantidad de servicios administrados sugiere que puede centrarse en la creación de su producto, y la baja complejidad se elimina al poder diseñar AWS para una cantidad específica y tiene la capacidad de realizar un seguimiento de cuánto gasta su organización en cada servicio.

**Principios de diseño Escalabilidad**

## An IT architecture can be scaled in two ways:

Escalado vertical: que implica actualizar desde una fuente. Esto demuestra que puede ocupar más RAM, un procesador más rápido o más capacidad de almacenamiento. Este es un enfoque válido, pero hasta cierto punto, no se puede agregar un número infinito de RAM a un disco.

Escalado horizontal: escalado de varias fuentes. Suena fácil sobre el papel (solo obtén más servidores), pero no todas las aplicaciones y arquitecturas están diseñadas con un enfoque horizontal en mente, lo que lo hace complicado.

## Stateless applications

En aplicaciones sin estado, el escalado horizontal es ideal. Una aplicación sin estado es una aplicación que no requiere el conocimiento de la anterior

solicitud enviada a esa aplicación y no almacena ninguna información sobre la sesión como tal. Si está más familiarizado con la programación funcional, una aplicación sin estado es esencialmente una función pura que ofrece la misma salida con la misma entrada.

Hay dos enfoques clave involucrados en compartir la carga de trabajo entre diferentes dispositivos, que son más fáciles de decir que de hacer:

* + - 1. Modelo de inserción: para dispersar el tráfico entrante a muchas instancias en las que se ejecuta la aplicación, mediante un Elastic Load Balancer (ELB). También puede utilizar Amazon Route 53 (un servicio DNS) para implementar DNS touring, pero esta no es una solución elástica y tiene sus limitaciones.
      2. Modelo de extracción: almacene elementos de acción en una cola (Amazon SQS) e indique a las autoridades que recuperen su propio elemento de acción.

**Stateless Components**

La mayoría de las aplicaciones no son 100% apátridas; almacenan algún tipo de información de estado (por ejemplo, necesitan saber si el usuario ha iniciado sesión para poder seleccionar el contenido específico de ese usuario). Aún puede hacer que algunas de estas arquitecturas no tengan estado al no almacenar nada que persista en el sistema de archivos local más de una solicitud.

Puede usar cookies HTTP para almacenar información de sesión, pero no solo las cookies deben enviarse con cada solicitud, sino que también se pueden falsificar en el lado del cliente.

Guardar un ID de sesión específico en una cookie HTTP y almacenar datos de usuario más completos en el servidor sería un mejor enfoque. Se genera una arquitectura con estado almacenando los datos en el servidor, por lo tanto, la solución principal es almacenar estos datos en una base de datos (un DynamoDB es una buena opción).

Cambie los datos a un marco de nivel de almacenamiento como Amazon S3 o Amazon EFS si tiene que lidiar con archivos masivos, lo que le ayudará a evitar componentes con estado.

**Stateful Components**

Hay aplicaciones diseñadas específicamente para ejecutarse en una sola máquina (por ejemplo, juegos multijugador en tiempo real que requieren una latencia extremadamente baja). Si está desarrollando una aplicación de este tipo, no desea distribuir el tráfico horizontalmente a ninguna instancia.

Un enfoque recomendado para el tráfico HTTP/HTTPS es utilizar la característica de sesión persistente del balanceador de carga de aplicaciones para enlazar la sesión de un usuario a una instancia específica. Esto asegura que, mientras sigan jugando, no se muevan a otra máquina física. En otros casos, es posible que desee implementar el equilibrio de carga del lado del cliente. Esto introduce una complejidad adicional, pero a veces puede ser necesario.

**Disposable items instead of fixed servers**

Puede comprar servidores físicos en un entorno de hardware convencional, instalarlos en un centro de datos y colocarlos allí manualmente para actualizar paquetes, etc.

Con AWS, en última instancia, trata los servidores y todos los demás componentes como recursos temporales, proporcionándolos solo cuando es necesario.

Para entregar diferentes fuentes idénticas (o muy similares) rápidamente, puede usar:

* + - 1. Bootstrapping: un script que se configura al aprovisionar, por ejemplo, una instancia EC2. Se ejecuta tan pronto como inicia la instancia y es posible especificar detalles de configuración que varían entre diferentes entornos (por ejemplo, ensayo y producción).
      2. Imágenes Gold: algunos recursos de AWS (EC2, RDS y Elastic Block Storage EBS) se pueden crear a partir de una imagen Gold, que crea una instantánea de un recurso en un momento dado. Este enfoque es generalmente más rápido que arrancar una instancia y le permite arrancar de forma rápida y fiable recursos adicionales esencialmente "clonando" una instancia.
      3. Es posible configurar una instancia y guardar su configuración para crear una AMI (Amazon Machine Image) - a continuación, puede iniciar tantas instancias como desee utilizando eso. En AWS, varias AMI están abiertas, por lo que antes de crear sus propias AMI, compruebe si su caso se ha resuelto.
      4. Los contenedores también son una opción: Docker (con Amazon ECS - Elastic Container Service) es compatible, al igual que Kubernetes (con Amazon EKS).
      5. También hay un modelo híbrido en el que algunas partes de la configuración se capturan en una imagen dorada, mientras que otras se configuran dinámicamente a través de scripts de inicio. AWS Elastic Beanstalk sigue este patrón: le permite utilizar AMI para iniciar un nuevo servicio y realizar acciones de arranque a través de archivos.ebextensions, así como el entorno Variables.

**Infrastructure as Code**

La infraestructura puede (y debe) almacenarse como código y control de versiones. Las plantillas de AWS CloudFormation le permiten crear, administrar y actualizar recursos de forma ordenada y predecible. Esto simplifica la reutilización de arquitecturas, así como la extensión de arquitecturas existentes, lo que es muy útil para grandes organizaciones.

**Automate**

Tradicionalmente, alguien tiene que responder manualmente a los incidentes para aumentar la capacidad de almacenamiento, implementar servidores adicionales, etc. Con AWS, es posible automatizar estos pasos.

## Examples:

Recuperación automática para Amazon EC2: puede crear una alarma de CloudWatch que rastree y recupere una instancia EC2 automáticamente en caso de que algo salga mal. Es mucho mejor que reiniciar manualmente la máquina.

Escalado automático: puede escalar el número de instancias EC2, la capacidad de DynamoDB, ECS y EKS según sea necesario en función de la capacidad deseada. De esta manera, no está ejecutando más turnos de los necesarios (desperdiciando dinero) o menos turnos de los necesarios (lo que no se mantendrá al día con el tráfico si hay prisa). Es mejor programar el lanzamiento del grupo de escalado automático por un tiempo, para que no tenga que esperar a que AWS "note" más tráfico.

Usar alarmas y eventos: por ejemplo, cree una alarma de CloudWatch que envíe un mensaje SNS cuando una determinada métrica supere un determinado umbral. A continuación, estos mensajes ocupan una función lambda, envían un mensaje a la cola de SQS o envían una solicitud al extremo HTTP o HTTPS para que pueda responder al evento en contexto.