Serverless Computing

Club de Investigación Tecnológica

Theodore Hope <theodorehope@gmail.com>
San José, Costa Rica

2018.08.22

¿ Qué ?

- Serverless Computing
  - Modelo de computación que elimina la abstracción del “servidor”.
  - Unidad de lógica es una función (de código).
  - Patrón de arquitectura.
  - Aprovisionamiento automático de recursos.
  - Modelo “Pay-as-you-go”, por ejecución de código.

Analogía ...

- RDBMS
  - El query más veloz y eficiente es el que no se ejecuta.

- Serverless
  - El servidor más fácil de administrar es el que no existe.

¿ Cómo llegamos aquí ?

- Evolución de “Dónde corren nuestras aplicaciones”
  - Equipo físico
  - Virtualización (propia o IaaS)
  - PaaS
  - Containers (Docker)
  - Serverless

¿ Cómo se Come ?

- Funciones (bloques de código) se suben (*deploy*) al proveedor.
  - *Deployment* rápido y ágil.
- Eventos se definen y se asocian a la invocación (ejecución) de cada función.
- Código es ejecutado cuando ocurre un evento.
  - Eventos en paralelo → ejecución paralela

Serverless ...

- Las funciones (de código) son las unidades de ejecución y escalamiento.
- En el modelo de s/w no existen máquinas, VMs, etc.
- "Stateless" – ambiente efímero de ejecución.
- Escalamiento automático según ejecución de funciones.
- Cobro muy granular; no hay cobro por tiempo ocioso.
- Arquitectura implícitamente tolerante a fallas.

¿ Qué hay de nuevo ?

- Cambio de paradigmas:
  - Aplicaciones convencionales se abstraen de sus servidores, y sólo queda el código.
  - Arquitectura de software dirigida por eventos.
  - Innovación en modelo de cobro.

¿ Quién y Dónde ?

- AWS Lambda
  - Java, Node.js, Python, C#, Go
- Google Cloud Functions
  - Node.js, Python
- Microsoft Azure Functions
  - C#, F#, Node.js, Java
- IBM Cloud Functions (OpenWhisk)
  - Node.js, Swift, Java, Go, PHP, Python
- Serverless Framework - https://github.com/serverless/serverless

¿ Para cuáles aplicaciones conviene ?

- Cuándo SÍ
  - Apps con tráfico irregular y en ráfaga (“bursty”), altamente paralelizables.
  - APIs para back-end de apps móviles
  - Arquitecturas de microservicios
  - Arquitecturas reactivas (v.g. http req, eventos en colas, etc.)
  - IoT back-end (enorme cantidad de solicitudes pequeñas)

- Cuándo NO
  - Tareas indivisibles de larga duración.
  - Requerimientos muy altos de CPU y/o RAM (v.g. HPC).
  - Requerimientos de tiempo real.
  - Requerimientos de comunicación sincrónica con terceros.

¿ Para quiénes ?

- Para todo el mundo ;-)

- Startups ❤️ porque:
  - Minimiza costo de infraestructura durante desarrollo.
  - Elimina preguntas de *due diligence* sobre infraestructura.

¿ DevOps ➔ NoOps ?

- No hay dimensionamiento de capacidad ni mantenimiento ;-)
  - Tiempo de aprovisionamiento se reduce/elimina.
- Desintermediación entre Devs e Infraestructura.
  - Devs hacen *deployment* y ejecutan código directamente.
- Enfocarse más en:
  - Monitoreo y visiblidad
  - "Graceful degradation" y resiliencia
  - Continuous Integration (CI) / Continuous Deployment (CD)

Ahora bien ... 🤔

- Dependencia de proveedor ("vendor lock-in").
- ¿Disponibilidad de personal capacitado?
- Menos transparencia:
  - Yo controlo menos de mi infraestructura que antes.
- Difícil de observar, trazar, depurar aplicaciones. 😱
- Latencia significativa.
- Tamaño de ejecución limitada.
- Código debe ser *stateless*. (¿Eso es malo?)
- Optimizaciones de sistemas convencionales no existen. (¿Y qué?)

Qué curioso ...

- Hace 10 años:
  - Back-ends monolíticos.
  - Front-ends cientos de páginas (HTML dinámico).

- Hoy:
  - Back-end con muchos microservicios (o funciones).
  - Front-ends monolíticos
    ("SPA – Single Page App" con HTML/JS/CSS/Ajax).

Editorial ...

- Un montón de microservicios fuertemente acoplados equivale a un monolito.
  - Dividir una aplicación en microservicios no garantiza el desacoplamiento de sus componentes.

- La tendencia de descomposición funcional (v.g., en microservicios, funciones, etc.) llegó para quedarse.

There’s no party like a Turing machine party,
because a Turing machine party may or may not stop.

Usos en serio, ¡ de verdad !

- Thompson Reuters
  - 4000 req/sec (25×10^9 req/month)
- Bustle
  - 10^9 req/month (approx 386 req/sec) a 8×10^{10} usuarios
- MapReduce en AWS Lambda & S3
  - 200 GB procesados por $ 0.39 y ningún servidor
- API en AWS
  - 2.1×10^6 API reqs por $ 11. ($ 0.00000523809 / req)
- Expedia
  - 1.2×10^9 Lambda req/month (en 2017)

xkcd.com/2031