Alta disponibilidad en una arquitectura de microservicios para IoT

Manel Mena, Javier Criado, Luis Iribarne, Antonio Corral

Grupo de Informática Aplicada, TIC-211, Universidad de Almería, España
{manel.mena,javi.criado,luis.iribarne,acorral}@ual.es

Resumen. En los últimos años, el uso de dispositivos IoT ha crecido exponencialmente. A la hora de trabajar con ellos, nos encontramos con una serie de problemas difíciles de solucionar. Por un lado, el simple hecho de comunicarse con esos dispositivos puede resultar problemático ya que pueden utilizar diferentes tipos de tecnologías con respecto a la comunicación. Por otro lado, este tipo de dispositivos suele tener como objetivo realizar su función utilizando la menor energía posible, por lo que presentan ciertas limitaciones en cuanto a rendimiento. Tratar de trabajar con estos dispositivos en entornos de alta disponibilidad se vuelve difícil debido a esas restricciones, ya que se convierten en cuellos de botella dentro del entorno de ejecución. En este artículo presentamos la evolución de Digital Dice, una arquitectura de microservicios para el manejo de dispositivos IoT y sistemas ciberfísicos, y las estrategias que hemos utilizado para intentar alcanzar la meta de la alta disponibilidad.

Palabras Clave: IoT · Microservicios · CPS · Alta Disponibilidad

1. Introducción

El creciente uso de dispositivos IoT en todo tipo de dominios, como ciudades inteligentes, edificios inteligentes, Industria 4.0, entre otros dominios, ha provocado el crecimiento de la adopción de este tipo dispositivos, en ocasiones, sin un control formal del proceso. En 2020 ya teníamos más de veinte mil millones de dispositivos [1], y no hay señales de que esta tendencia vaya a detenerse pronto. Uno de los mayores problemas a la hora de trabajar con estos dispositivos es que para su manejo nos encontramos multitud de protocolos diferentes, lo que provoca muchas dificultades a la hora de trabajar con ecosistemas heterogéneos IoT o con sistemas ciberfísicos. Por otro lado, dada la finalidad de la mayoría de dispositivos IoT, donde la eficiencia es esencial, nos encontramos con restricciones de rendimiento importantes a la hora de trabajar con estos dispositivos.

Para solventar estos problemas, creamos el concepto de Digital Dice (DD) [2]. Un Digital Dice es una representación virtual de un dispositivo IoT o un sistema ciberfísico basado en microservicios que establece un patrón de comunicación común, con tecnologías web conocidas por la mayoría de desarrolladores, que permiten una gestión de los mismos abstrayéndonos de los protocolos y las tecnologías que estos dispositivos utilizan. Para realizar su cometido, un DD está

representado como una Thing Description (TD) de la Web of Things (WoT) [3]. La WoT es un paradima fundamentado en un conjunto de estándares y protocolos desarrollados por el World Wide Web Consortium (W3C) para construir ecosistemas IoT de manera flexible, escalable y abierta utilizando tecnologías web como su capa de aplicación. Básicamente nos permite declarar *things* de forma estándar, descubrirlas, interactuar con ellas y monitorizar su estado. La Thing Description es para las *things* similar a lo que OpenAPI es para los servicios RESTful, un lenguaje formal para su definición y documentación.

Elegimos utilizar una arquitectura de microservicios por las ventajas que este tipo de arquitecturas proporciona, como son la escalabilidad, la mantenibilidad, la resiliencia y la flexibilidad que este tipo de arquitecturas proporciona; Ventajas que acompañan de manera muy cercana el concepto de alta disponibilidad o High availability (HA) [4]. Este concepto está normalmente representado por la métrica del *uptime* o porcentaje de tiempo total en que la aplicación funciona de manera nominal. Los dos principios fundamentales del diseño de sistemas de software que pueden ayudarnos a lograr una alta disponibilidad son: (a) La eliminación de puntos únicos de fallo en la medida de lo posible; (b) La capacidad del sistema de responder a errores en tiempo real. Las arquitecturas de microservicios pueden cumplir de forma sencilla con estos principios. En la mayoría de los casos, estas arquitecturas cuentan con la replicación de microservicios como técnica de redundancia para eliminar los *puntos únicos de fallos*, y la inclusión de patrones operativos como el *circuit breaker* [6] nos permite definir en tiempo real el comportamiento cuando los fallos ocurren en nuestro sistema.

Digital Dice toma su nombre del concepto de Digital Twin [5], dado que ambos son representaciones virtuales de dispositivos o sistemas físicos. Sin embargo, mientras que Digital Twin se enfoca en virtualizar dispositivos para realizar pruebas y simulaciones sin influir en los procesos comerciales, Digital Dice se enfoca en el control y la gestión de estos dispositivos. La palabra Dice surge de las diferentes facetas (o microservicios) que forman parte de la implementación de un único componente en nuestra arquitectura.

2. Propuesta

En esta sección se describen las diferentes estrategias utilizadas para proporcionar a DD alta disponibilidad (HA) y cómo estas estrategias nos ayudan a mejorar el rendimiento general de un sistema.

Estrategias de comunicación con dispositivo físico. `Reflection` (microservicio) como el único punto de conexión con el dispositivo físico es nuestra principal apuesta para evitar que el dispositivo se colapse.

Como hemos dicho anteriormente, muchos dispositivos IoT tienen limitaciones en cuanto a rendimiento ya que suelen estar gobernados por microcontroladores, que están diseñados sobre todo para priorizar el ahorro energético. Además, en algunos entornos IoT, como las instalaciones basadas en bus KNX o Modbus, hay limitaciones en el número de conexiones que podemos establecer. Cuando

muchos usuarios solicitan datos simultáneamente, el bus puede colapsar, produciéndose una pérdida en el envío de mensajes e interrumpiendo la comunicación.

El uso del `Reflection`, así como el uso de la base de datos como sistema intermedio, hacen que las operaciones de lectura no tengan que llegar al dispositivo físico, lo que ya evita un gran número de peticiones a dichos dispositivos. Además de eso, tener un único punto de conexión con el dispositivo garantiza que las operaciones de escritura se realicen por orden de llegada.

Estrategias de replicación de microservicios. En cuanto a las estrategias de replicación de microservicios, primero debemos entender que los microservicios que forman parte del DD se basan en contenedores tipo *docker*. Gracias al uso de contenedores podemos aislar cada uno de los entornos de ejecución y trabajo de los microservicios de las máquinas de despliegue. El uso de contenedores ofrece la gran ventaja de poder desplegar nuestra solución en cualquier máquina, tanto en máquinas *bare-metal* como en infraestructura *cloud*.

Como cliente para contenedores usamos Kubernetes. A diferencia de algunas de sus alternativas, este sistema cuenta con una serie de servicios pre-instalados que evitan tener que instalar nuevos artefactos necesarios para que Digital Dice funcione correctamente. Por otro lado, Kubernetes nos proporciona una serie de ventajas, como por ejemplo, generación automática de subredes para aislar los microservicios de cada uno de nuestros DD, permitir la comunicación entre los diferentes contenedores utilizando el nombre del contenedor como DNS interno, o trabajar con una serie de herramientas que nos permiten hacer un seguimiento de métricas de cada contenedor, como Prometheus [7] o Grafana [8].

Kubernetes permite replicar contenedores de forma automática, utilizando una serie de métricas como CPU, memoria u otras métricas personalizadas.

Estrategias de comunicación entre microservicios. Cuando se habla de estrategias de comunicación en DD, tenemos que dividirlas en dos puntos de vista diferentes: cómo se comunica el DD con el usuario final y qué sucede con la comunicación entre los microservicios.

Como ya hemos mencionado antes, DD utiliza el estándar propuesto por WoT. Una de las premisas de Digital Dice es facilitar su uso por parte de cualquier desarrollador web o móvil, por lo que desde los inicios de DD se decidió que los usuarios finales deberían poder interactuar con ellos a través de una API. Sin embargo, el uso de una API en el campo del IoT trae consigo una serie de limitaciones inherente a la tecnología cuando hablamos de datos en tiempo real, por lo que al final DD utiliza lo que podríamos denominar un patrón de conexión híbrido. En primer lugar, esa API por definición es *stateless*, lo que implica que la consulta se realiza y responde en el momento. En segundo lugar, usamos eventos enviados por el servidor o Server Sent Events (SSE), un protocolo *stateful*, que implica que el usuario esté suscrito a un método hasta que se cierra la conexión. En DD, podemos suscribirnos a cualquier interacción de lectura a través de SSE, lo que significa que podemos suscribirnos a la propiedad de un dispositivo a través de una solicitud HTTP y luego recibir los cambios de la propiedad

en tiempo real; todo sin que el usuario final tenga que aprender una tecnología fuera de su dominio. El uso de SSE tuvo como consecuencia la necesidad de solucionar la siguiente limitación: Con las conexiones HTTP/1, un servicio sólo puede mantener abiertas cinco conexiones con cada usuario. HTTP/2 resuelve este problema eliminando esa limitación.

Para solventar los problemas derivados de la replicación de los microservicios, DD utiliza *service mesh* o malla de servicios, cuyos exponentes tecnológicos son *envoy*, *istio* o *linkerd* [9]. A diferencia de otros sistemas que también pueden gestionar la comunicación entre microservicios, la *service mesh* es una capa integrada en la aplicación, que es visible y registra si la interacción entre las diferentes partes de la aplicación es correcta. Facilita la optimización de las comunicaciones y evita tiempos de inactividad a medida que crece la aplicación. Aparte de la monitorización, también aplica una serie de patrones recomendados en arquitecturas de microservicios como el *circuit-breaker*.

3. Conclusiones

En este artículo hemos presentado las diferentes actuaciones realizadas para dotar la arquitectura presente en DD de alta disponibilidad. Estas estrategias están basadas en la comunicación con los dispositivos físicos, la replicación de microservicios y la comunicación entre microservicios y con el usuario final.

Como trabajo futuro pretendemos establecer un metamodelo que nos permita establecer un lenguaje para generación de Digital Dice, teniendo en cuenta no solo la funcionalidad de los dispositivos representados, sino también la configuración de replicación de la arquitectura de DD que los soportan.

Agradecimientos: Proyecto PAIDI UrbanITA (P20_00809), Sistema Andaluz del Conocimiento, Junta de Andalucía. Beca FPU17/02010 (Manel Mena).

Referencias

1. Al-Sarawi, S., Anbar, M., Abdullah, R., Al-Hawari, A. B.: Internet of things market analysis forecasts, 2020–2030. In WorldS4, pp. 449–453, 2020.
2. Mena, M., Criado, J., Iribarne, L., Corral, A.: Assembling the web of things and microservices for the management of cyber-physical systems. Journal of Universal Computer Science 27 (7), pp. 734–754, 2021.
3. W3C: Web of Things. https://www.w3.org/WoT/. Acc.: 05-05-2022.
4. Piedad, F., Hawkins, M.: High availability: design, techniques, and processes. Prentice Hall Professional, 2001.
5. Tao, F., Zhang, H., Liu, A., Nee, A. Y.: Digital twin in industry: State-of-the-art, IEEE Transactions on Industrial Informatics 15 (4) pp. 2405–2415, 2018.
6. Richardson, C.: Microservices patterns: with ex. in Java. Simon and Schuster, 2018.
7. Turnbull, J.: Monitoring with Prometheus. Turnbull Press, 2018.
8. Chakraborty, M., Kundan, A. P.: Grafana. In Monitoring Cloud-Native Applications, pp. 187–240, 2021.
9. Khatri, A., Khatri, V.: Mastering Service Mesh: Enhance, secure, and observe cloud-native applications with Istio, Linkerd, and Consul. Packt Pub. Ltd, 2020.