No podemos decir que hay una definición formal del estilo arquitectónico de microservicios, pero podemos intentar describir lo que vemos como características comunes para las arquitecturas que se ajustan a la etiqueta. Al igual que con cualquier definición que describa características comunes, no todas las arquitecturas de microservicio tienen todas las características, pero esperamos que la mayoría de las arquitecturas de microservicios muestren la mayoría de las características. Si bien los autores hemos sido miembros activos de esta comunidad bastante flexible, nuestra intención es intentar una descripción de lo que vemos en nuestro propio trabajo y en esfuerzos similares por equipos que conocemos. En particular, no estamos estableciendo alguna definición para cumplir.

Los microservicios se rigen por los siguientes principios:

* Encapsulación
* Automatización
* Dominio Céntrico
* Descentralizado
* Independiente
* A prueba de fallos
* Observable

Cuando se busca dividir una gran aplicación en partes, a menudo la administración se enfoca en la capa de tecnología, lo que lleva a equipos de interfaz de usuario, equipos lógicos del lado del servidor y equipos de bases de datos. Cuando los equipos están separados en esta línea, incluso los cambios simples pueden llevar a un proyecto entre equipos a requerir un mayor tiempo y aprobación presupuestaria. Un equipo inteligente optimizará esto y se reagrupará por el menor de los dos males.

Los microservicios hacen sostener un enorme potencial para cambiar las reglas de juego de aplicaciones empresariales. Las aplicaciones basadas en microservicios le permiten distribuir el trabajo entre múltiples grupos de tal forma que cada grupo pueda trabajar en secciones de aplicaciones individuales sin imponer trabajo adicional en las demás. Las arquitecturas de microservicio también le permiten descomponer una aplicación en servicios de ejecución independiente. Puede actualizar microservicios individuales más fácilmente y colocar la actualización resultante en producción sin la necesidad de un largo trabajo de integración entre todos sus equipos de desarrollo diferentes.

Dado que los servicios pueden fallar en cualquier momento, es importante poder detectar los fallos rápidamente y, de ser posible, restaurar el servicio automáticamente. Las aplicaciones de microservicio ponen mucho énfasis en el monitoreo en tiempo real de la aplicación, verificando tanto los elementos arquitectónicos (cuántas solicitudes por segundo obtiene la base de datos) como las métricas relevantes para el negocio (como cuántas órdenes por minuto se reciben).

En cualquier esfuerzo de integración, el éxito depende de qué tan bien encaja el software en los componentes. Es difícil determinar exactamente dónde deberían estar los límites de los componentes. El diseño evolutivo reconoce las dificultades para hacer que los límites sean correctos y, por lo tanto, la importancia de que sea fácil refactorizarlos. Pero cuando sus componentes son servicios con comunicaciones remotas, entonces la refactorización es mucho más difícil que con las bibliotecas en proceso. El cambio de código es difícil a través de los límites del servicio, los cambios de interfaz deben coordinarse entre los participantes, deben agregarse capas de compatibilidad con versiones anteriores y las pruebas son más complicadas.

Otro problema es si los componentes no se componen de forma limpia, entonces lo único que se está haciendo es cambiar la complejidad desde el interior de un componente a las conexiones entre los componentes. Esto no solo cambia la complejidad, sino que lo mueve a un lugar menos explícito y más difícil de controlar. Es fácil pensar que las cosas son mejores cuando se mira el interior de un componente pequeño y simple, mientras se pierden conexiones desordenadas entre los servicios.