Microsoft: Teoría y aplicación

La charla dedicada a los Microservicios es presidida por el profesional Lucas Krause, quien ha venido trabajando en este rubro por 10 años, así como de consultor por otros 15.

El expositor empieza su presentación hablando sobre la existencia de un problema único por cada empresa y haciendo mención de un libro que si bien no es de carácter tecnológico, abarca el tema general de arquitectura relativo al tema en cuestión.

Dentro del tema de Arquitectura, nos menciona que los patrones son parte importante del desarrollo, y que estos son relativos a la comunicación. En otras palabras, mediante los patrones se busca entablar una discusión de alto nivel que permita el adecuado intercambio de ideas. Del mismo modo, se busca llegar a una estandarización para lograr que todos los involucrados estén de acuerdo en una misma forma para llegar a alcanzar los objetivos que se hubieran planteado.

Habiendo introducido el tema, se realiza una breve explicación de la Arquitectura monolítica refiriéndose a la misma como grandes sistemas o aplicaciones que hacen mucho pero son difíciles de mantener, debido a que estas pueden crecer mucho de manera vertical, pero en caso de necesitar un crecimiento de manera horizontal el desarrollo se vuelve mucho más tedioso y duplicado. Por otro lado, los microservicios rompen con las funciones de negocio, basándose en contextos limitados y servicios; siendo amigable a la migración, a pesar de que este cambio se realiza de manera bastante lenta. También es agnóstico a la tecnología, permitiendo el uso de diversas tecnologías para la realización de un mismo gran sistema. Micro servicios es lo que debería de haber sido SOA, pues comparten algunas similitudes, destacándose la primera por su peso ligero.

Para indagar un poco más en el tema principal, se exponen los principios necesarios para una correcta implementación de micro servicios, los cuales son: Encapsulamiento, Automatización, Dominio centralizado, independiente, fail-safe y Observable. En relación a esto, la escalabilidad de los microservicios se basa en dividir el proceso en actividades pequeñas, de acuerdo a sus similaridades; entonces se encuentran tres dimensiones para el crecimiento del software: Y—descomposición funcional, X – duplicación horizontal y Z – particionamiento de data (compartir data).

Si bien existe la necesidad de seguir un estándar para un mejor desarrollo, debería ser suficiente seguir las filosofías de micro servicios para lograr la correcta implementación de un proyecto de este estilo, esta son: hacer las cosas bien,el núcleo(unix) y romper en pequeñas piezas para poder reutilizarlas a necesidad. También, se afirma que la clave para el éxito es la mantenibilidad, ayudándose de los principios SOLID Y DRY; de manera que se pueda expandir sin muchas complicaciones y siendo más rápido y eficiente.

Como complemento, se menciona algunos cambios que se están llevando a cabo hoy en día (sea change), en donde se mencionan Time is ripe, automatización, containers, dev-ops, abstracción de alto nivel. Del mismo modo, se mencionan patrones y aproximaciones muy utilizadas en este contexto de los micro servicios como son: CQRS, event Sourcing, API Gateway/proxy , Orchestrated API (TCP server).

Cambiando de tema,para aproximarnos al éxito se debe entender el negocio y su estructura, verificar que las tecnologías encajan de manera adecuada, y tener presente el tamaño de la aplicación, centrándose no solo en la parte tecnológica, sino también en un enfoque más holístico. Entonces se puede llegar a una definición de éxito basándose en necesidades cubiertas, sin defectos, escalable, seguro,robusto ,fácil de manejar, a tiempo,etc.

Se debe tomar en cuenta que para entender el servicio se deberá dividir los problemas utilizando use case ,historias,etc; tener en cuenta la usabilidad/interacción, al igual que la economía API. En el lado de la estructura de la organización podemos encontrar equipos por servicios, los patrones soap y dry como ayuda al desarrollo y comway´s, law, solucionado con scrum. Del mismo modo, el tamaño del equipo no debe tener más de 10 a 12 personas, dependiendo de la organización, de manera que se logre una alta confianza y una buena capacidad de comunicación.

Por el lado tecnológico encontramos hoy en día tecnologías de Lenguajes de eventos inmutables como: erlang, elix y scala; base de datos de grafos como orientdb y neo4j; Search optimized indexes como lucene y solr; y finalmente lenguajes de programación como Java, .net, nodejs, GO, Rust y Julian. A partir de todas estas tecnologías se debe escoger cuál se adecua más a tu negocio, pues si bien esta arquitectura te permite combinar diferentes tecnologías, no necesariamente todas las combinaciones serán las ideales para la empresa. Otro punto a tener en cuenta son las estrategias de particionamiento, para dividir el proceso en los microservicios adecuados, esto se puede lograr mediante particionamiento por verbo, sustantivo, single responsability o por las partes que cambian juntas.

Beneficios en el uso de este tipo de arquitectura están relacionados con la performance que se logra en ejecución y desarrollo, la capacidad de cubrir las expectativas del cliente, los lenguajes ubicuos y la posibilidad de un rápido despliegue, fácil testeo y bajos costos de escalabilidad. De manera opuesta encontramos desafíos de su utilización como la mayor complejidad para desarrollo y mantenimiento; la dificultad del testeo del sistema y la distribución de transacciones, así como el desafío de administrar tan gran sistema en varios pequeños equipos y el aumento inminente de memoria, lo que genera una necesidad de herramientas para dicho fin. Encontramos también problemas en la organización, pues el trabajo por equipos que manejan distintos tipos de lenguajes, puede crear una división entre los trabajadores de la empresa y romper con la cultura interna de la misma; siendo tal vez el caso de que la organización no es lo suficientemente madura para trabajar con esta decisión de arquitectura.

Por último ya en la misma aplicación de la arquitectura en una empresa existe la posibilidad de aplicar cierta tecnologías como CQRS para lo que hay que tener en cuenta patrones, reads de writes y un buen punto de inicio para separando código para poder optimizarlo por separado. Encontramos también event sourcing que permite almacenar eventos en lugar de cosas y hacer replay de versiones anteriores; DDD/Bounded contexts que se relaciona a la posibilidad de usar distintos lenguajes en el proceso conjunto y la API Gateway que permite la comunicación de las diferentes plataformas a la lógica desarrollada. Para el problema del espacio en memoria, la elección de una BD NoSql, inmutables y estáticas, puede marcar la diferencia de acuerdo a las necesidades que presente el proyecto.

Finalmente, se concluye la charla haciendo mención de la seguridad que se logra con este tipo de arquitectura, pues es seguro por naturaleza y una ronda de preguntas.