Desarrollo de aplicaciones con arquitectura mono-repositorio

Igvir Ramirez V.
Junio, 2022.

Palabras clave: mono-repositorio, control de versiones, monorepo.

Resumen

Los Mono-repos (repositorios monolíticos), son utilizados por grandes empresas, como Google y Facebook, y por proyectos de código abierto de gran alcance como: React, Laravel, Babel o NixOS. Este trabajo realiza un recorrido por los conceptos para dar visión general de la definición, las características de una estructura multi-repositorio, y las características de los mono-repositorios, así como de sus beneficios y desafíos. Con el apoyo de la investigación bibliográfica se plantearon alguna preguntas de investigación para luego realizar una propuesta de lineamientos generales para el desarrollo de aplicaciones con arquitectura mono-repositorio. Los hallazgos son varios pero destacan principalmente en dos áreas. En primer lugar en el área técnica, donde el uso de herramientas y técnicas representa un elemento de decision que considere los beneficios, la relaciones entre los diferentes proyectos y los retos a la hora de elegir arquitectura mono-repositorio. Otra área en la que la investigación encontró hallazgos fue el del entorno organizacional, se encontraron referencias a elementos culturales relevantes en el éxito de la adopción de arquitectura mono-repositorio. Actualmente, se puede observar un aumento en los sistemas de TI complejidad impulsada por la adopción de arquitectura orientada a servicios, micro-servicios y sin-servidor (*serverless* por su nombre en inglés). Por lo tanto, las empresas se benefician de un repositorio único debido a la mejora del conocimiento del equipo. Conocimiento que resulta de la disminución de las barreras entre los diferentes actores que típicamente se presentan en el trabajo interdepartamental, facilita la movilidad de roles entre los desarrolladores y puede influir en la mejora de la calidad del trabajo en equipo. Los desafíos puede ser importantes por lo que se debe considerar la salud del código, la complejidad de la solución y la inversión en tiempo para incorporar herramientas tanto para el desarrollo como para la ejecución.

Introducción

En el desarrollo de software utilizando arquitecturas de micro-servicios, el desarrollo ágil y los despliegues continuos, hay varios elementos necesarios para lograr que el diseño y la construcción respondan a la necesidad de comunicación continua, estandarización y eficiencia organizacional. Uno de estos elementos es la gestión del código fuente que, sin ser el único factor involucrado, puede ser el punto de partida para los beneficios o los problemas de la solución a desarrollar. Cuando una organización asume el reto de construcción de soluciones de *software* suficientemente robustas para escalar a un nivel empresarial, debe tener en consideración estrategias de gestión de todos los recursos y muy especialmente de la gestión del código, dado que es muy fácil dejarse llevar por la tentación de iniciar la construcción y postergar la definición de lineamientos. Incluso es común que se pierdan de vista y que, al no aplicarse desde el inicio, su ausencia genere un efecto de bola de nieve y en consecuencia se vuelva un tema cada vez más grande a medida que el proyecto avanza. Tener un lineamiento inicial, fomentar una disciplina interna, velar por el uso de estas línea de instrucción y hacerlas propias de la construcción, facilita que el proceso se vuelva natural al punto que todos puedan "saltar" al proceso de producción con una estrategia exitosa. Ante esta necesidad, se plantean el estudio de los enfoques mono-repositorio versus el multi-repositorio, conocer su estructura, ventajas y desventajas. Además se proponen lineamientos generales para su adopción basados en la bibliografía disponible y la experiencia de otras organizaciones referentes en la industria.

Conceptos

1. Repositorio de código

Un repositorio de código es el lugar en el que se almacena y se puede realizar la distribución del código fuente de una aplicación de software. En el entorno empresarial, el repositorio de código debe ser un servidor seguro que utiliza sistemas de control de versiones para facilitar la coordinación del equipo de desarrollo. El repositorio debe contener las diferentes versiones de la aplicación, el historial de cambios realizados y los cambios aplicados sobre cada nueva versión. Además, debe permitir poder revertir, de ser necesario, esos cambios. Para cumplir su cometido, el repositorio debe permitir acceso seguro y en paralelo a los diferentes usuarios involucrados en el equipo de desarrollo, bien sea que estos requieran trabajar en la misma o en diferentes versiones.

2. Enfoque mono-repositorio

Mono-repositorios, *Mono-repository* o simplemente *mono-repo* como se les conoce también en inglés, es un concepto de arquitectura y una estrategia de gestión [1] del código fuente que consiste en agrupar en un único contenedor de código todos los componentes de la solución. Un Mono-repositorio en lugar de administrar múltiples repositorios, mantiene todas sus partes de código aisladas dentro de un único espacio aunque están virtualmente separadas siguiendo alguna convención definida por el equipo de trabajo. Pero el concepto de aislamiento aquí no debe confundirse, un mono-repositorio es una estrategia de gestión y no implica crear aplicaciones monolíticas. Generalmente es todo lo contrario, el concepto mantiene la correcta separación de dependencias que las arquitecturas de micro-servicios requieren y su uso no compromete otros principios de desarrollo. Una característica importante para un mono-repo es que, si bien contiene múltiples proyectos distintos, estos deben tener entre ellos relaciones bien definidas. El mono-repo no se trata de colocar el código en un solo repositorio, si no hay relaciones bien definidas entre los proyecto, no se denomina mono-repo.

3. Enfoque multi-repositorio

El enfoque multi-repositorio, como su nombre lo indica, utiliza varios repositorios para albergar los múltiples componentes, librerías o servicios de un proyecto desarrollado por una empresa. En su forma más extrema, alojará cada conjunto mínimo de código utilizable o cada funcionalidad independiente (como un micro-servicio) en un repositorio individual.

4. Estructuras de repositorio de código fuente

Estructura	Implementación	Implementada por
Multi-repo	Un repositorio para cada componente o librería de código	Amazon, Netflix , Lyft
Mono-repositorio	Un repositorio para todos los componentes o incluso para toda la compañía	Google, Facebook, Microsoft, Uber, Twitter, React, Angular
Híbrido: Multi-repo manejados como mono-repo	Los cambios se realizan en multiples repositorios pero se gestionan como un mono-repo	Android, Chrome
Híbrido: Mono-repo manejado como multi	Los cambios se realizan en un mono-repo pero luego se dividen en multiples repositorios de solo lectura para construcción o distribución	Symfony, Shopsys

Tabla 1: Estructuras de repositorio de código fuente [7, 16]

5. CI/CD

El término CI/CD proviene de las siglas en inglés: *Continuous integration and Continuous delivery*. Es un método para entregar aplicaciones de manera frecuente aprovechando las ventajas de la automatización en las diferentes etapas de desarrollo de aplicaciones. Los principales conceptos atribuidos a CI/CD son integración continua, entrega continua e implementación continua.

6. Arquitectura de Micro-servicios

La arquitectura de micro-servicios, también conocidos como micro-servicios, son un estilo arquitectónico que estructura una aplicación como una colección de servicios con bajo acoplamiento para implementar capacidades de negocio. La arquitectura de micro-servicios permite la entrega/implementación continua de aplicaciones grandes y complejas. También permite que una organización modernice la base tecnológica de sus aplicaciones.

Ventajas de la arquitectura mono-repositorio

La investigación arrojó que varios autores coinciden en algunas ventajas asociadas al uso del enfoque mono-repositorio:

- Promueve la cultura de trabajo: con la adopción de mono-repositorios, cada individuo en la organización es consciente de los objetivos comerciales y esto lo convierte en un equipo unificado y, por lo tanto, puede contribuir más específicamente hacia las metas y objetivos de la organización.
- Mejora la coordinación y facilita la rotación de roles entre desarrolladores: los desarrolladores pueden ejecutar fácilmente toda la plataforma en su entorno de trabajo y esto les ayuda a comprender todos los servicios, sus relaciones y las interacciones que ocurren cuando funcionan juntos. Esto contribuye no solo al conocimiento sino también a la calidad de la solución, al permitir la detección de errores incluso antes de enviar el código al repositorio central.
- Refactorización simplificada: en entornos de desarrollos ágiles la capacidad de descartar una ruta que no funciona o de reestructurar el código para tomar un camino no anticipado es muy importante. Desde cambiar un nombre o ajustar una dependencia, la refactorización es más simple y puede requerir un único comando para afectar a toda la solución o el area de alcance. Todo esto con un mínimo de sobrecarga asociada al cambio de contexto, ya que todo está ordenado en un solo lugar bajo convenciones que facilitan su comprensión.
- Simplificación del proceso de automatización y pruebas: El enfoque mono-repositorio ofrece también un único lugar para almacenar todas las configuraciones y pruebas. Dado que todo se encuentra dentro de un repositorio, es posible configurar los procesos de CI/CD y el paquete una vez y luego simplemente reutilizar las configuraciones para compilar todos los paquetes antes de publicarlos en forma remota. Lo mismo ocurre con las pruebas unitarias, e2e y de integración: los procesos de CI pueden ser configurados para realizar todas las pruebas asociadas sin tener que lidiar con una configuración adicional.
- Facilidad para reutilizar el código con paquetes compartidos mientras se mantienen aislados. El enfoque mono-repo permite reutilizar paquetes como referencias comunes que se compilan y consumen desde un único punto de entrada mientras que se mantienen aislados los servicios unos de otros.

Desafíos de la arquitectura mono-repositorio

Algunos elementos asociados a la adopción del enfoque mono-repositorio representan una desafío para el entorno empresarial:

- Herramientas no-estándar o personalizadas: gran parte de las herramientas disponibles para la gestión del código fueron creadas a partir de la idea de manejar todo el contenido del repositorio como un único artefacto. La gestión mono-repositorio, sin embargo, requiere la capacidad de diferenciar entre espacios de trabajo. si bien algunas herramientas han mejorado su soporte aun es posible que la gestión de las

herramientas para CI/CD represente una carga de trabajo adicional para el equipo durante la adopción de un enfoque mono-repositorio.
- Ofrece mayores beneficios a organizaciones más grandes: La arquitectura mono-repositorio escala sus beneficios en proporción al tamaño del equipo de desarrollo, así, organizaciones más grandes se verán más beneficiadas con su uso en comparación a aquellas con equipos pequeños.
- Mono-repositorios con herramientas y/o procesos deficientes probablemente se desempeñen peor en áreas de: pruebas, seguimiento de dependencias, reutilización de código y revisión de código se se les compara con enfoques multi-repos con herramientas y/o procesos deficientes. La adopción del enfoque mono-repositorio demandará más disciplina y cumplimiento de procesos y convenciones al equipo de desarrollo y a la organización.

Cultura organizacional y conocimiento

La selección de la arquitectura mono-repositorio no debe asumirse como elemento puramente técnico. El modelo tiene ya varios años aplicándose y varias grandes empresas de software han optado por este enfoque para la organización de su código, incluidas Facebook, Google y Microsoft. Si bien la literatura consultada [7] indica que los estudios sobre el tema no son abundantes y que las referencias más comunes se encuentran en publicaciones y textos en internet [16] y teniendo en cuenta que entre las desventajas del uso del repositorio único radica en el hecho de que "*algunos de los problemas asociados con multi-repositorios aún deben resolverse de alguna manera en un mono-repositorio*"[7], se hace necesario indagar sobre los elementos del entorno organizacional han permitido implementar exitosamente arquitecturas mono-repositorio en grandes empresas. Netflix, un referente en desarrollo de micro-servicios y metodologías ágiles, utiliza un enfoque multi-repositorio esto parece ser coherente con su manifiesto que favorece una cultura de "libertad y responsabilidad" que "faculta a los ingenieros para crear soluciones utilizando cualquier herramienta que sienten que son las más adecuadas para las tareas". Microsoft que no solo adoptó el enfoque de repositorio único, sino que propuso una ampliación de Git para manejar mono-repositorios más grandes [18] es otra de las organizaciones que se ha enfoca en el cambio desde la cultura y mentalidad de crecimiento individual y colectivo. Esta investigación no profundiza en mayor detalle sobre otros aspectos culturales de la organización pero existen evidencias para señalar que una cultura de colaboración abierta es requerida en el entorno organizacional que quiera implementar el enfoque mono-repositorio. Los informes de experiencia de Microsoft, Google y otras grandes empresas confirman que *el conocimiento del equipo es un factor importante en la elección de la estructura del repositorio con un esfuerzo por orientarse hacia un conocimiento holístico del equipo en lugar de un conocimiento colectivo del equipo*. La forma en que nos comunicamos y compartimos información ejerce un impacto directo en conocimiento del equipo, que es un "*mecanismo crítico para facilitar las actividades de innovación dentro del proceso de desarrollo de software*" [7]. Se ha demostrado que el software de alta calidad depende de la colaboración de alta calidad dentro de un equipo, con factores que influyen como confianza, valor compartido y coordinación de la experiencia. [4, 7]

Convenciones, lineamientos y recomendaciones

La adopción de un enfoque mono-repositorio puede ser una alternativa válida en el desarrollo de aplicaciones modernas. Considerando las ventajas y desafías encontrados por este estudio, es pertinente adoptar convenciones de desarrollo asi como para la organización de los artefactos asociados. Seguir algunos lineamientos facilita la toma de decisiones del equipo sobre este y otros temas de organización, evitando así retrasos en tareas básicas con el objetivo de concentrar más tiempo y energía en la innovación, generación de conocimiento y productividad.

Convenciones

Una convención es un conjunto de estándares, reglas, normas o también criterios que son de aceptación general para el equipo de desarrollo. Su definición es muy importante para mantener un criterio de decisión a lo largo del ciclo de desarrollo.

- Defina la estrategia de *Branching* a utilizar.

> *Una estrategia de branching es la estrategia que adoptan los equipos de desarrollo de software al escribir, fusionar e implementar código cuando utilizan un sistema de control de versiones.*

La definición es esencialmente un conjunto de reglas que los desarrolladores pueden seguir cómo convención para interactuar utilizando con una base de código compartida. Existen varias estrategias que se han vuelto de uso común, entre ellas están: *Gitflow* y *Trunk-based development*.

- Defina una convención de nombres para el desarrollo.

  > *No usar las convenciones de nombre apropiadas genera confusión y complica al equipo de mantenimiento del código.*

  La convención de nombres puede agrupar cuales ramas estarán disponibles en su repositorio de forma permanente y cuales serán temporales. Su convención de nombres debe ser simple y directa. Para las ramas temporales, una convención de nombre que incluya el identificador único de la tarea o la incidencia que atiende facilita el seguimiento. Así para la atención del bug identificado con el número 999 podría usar un nombre de rama :

  ```
  bug/ID-999
  ```

  En algunos casos la convención de nombre de un mono-repositorio podría incluir el nombre del autor usando un formato similar al del ejemplo {autor}/{tipo/{id}:

  ```
  igvir/feature/ID-777
  ```

Lineamientos

Los lineamientos planteados en este estudio describen las etapas, fases, pautas y formatos necesarios para desarrollar el desarrollo de software con arquitectura mono-repositorio:

- Todo proyecto de desarrollo debe definir un proceso de Gestión de Versiones (*Release Management*). La gestión de versiones es el sistema que le permite controlar el ciclo de vida del desarrollo de software, desde la planificación hasta las pruebas y luego la versión. El proceso de *Release Management* debe establecer lineamientos detallados para cada etapa del proceso, sus transiciones y recomendar el o los sistemas y herramientas de gestión de código fuente donde se alojará el repositorio de código. Las etapas del ciclo de vida del desarrollo de software comúnmente definidas son: Plan, Construcción (*Build*), Pruebas (*Test*), Despliegue (*Deploy*) y Revisión (*Review*). Cuando se completan todas la etapas se tiene una versión.
- El equipo de desarrollo deberá seguir una convención de nombre para cada etapa del ciclo de vida de desarrollo. Desde la convención para los identificadores de tareas e incidencias, pasando luego al uso de nombres de las ramas en cada etapa y los números de versión. Las ramas permanentes serán siempre puntos de partida o de llegada pero mantendrán su nombre en los distintos proyectos que conformen el mono-repositorio. se deben evitar nombres de rama muy largos, prefiera dos o tres elementos significativos. En mono-repositorios se debe utilizar el formato: {app}/{tipo}/{ID}

  ```
  website-app/feature/ID-456
  ```

- El desarrollo debe ser realizado en una rama *feature*. La idea principal es encapsular el trabajo en un espacio separado hasta que se vuelva estable. Esta separación facilita que varios desarrolladores trabajen en una tarea en particular sin alterar la base de código principal. También significa que la rama principal nunca

debe contener código que no funcione, lo cual es una gran ventaja para los entornos de integración continua.

Recomendaciones

Las recomendaciones incluidas en este estudio son propuestas para el desarrollo de aplicaciones con especial énfasis en aquella que opten por el uso de un enfoque mono-repositorio total o parcial. Dada la naturaleza única de cada proyecto estas deben ser analizadas y adaptadas al entorno cuando se necesario. Se invita a la comunidad a ampliar esta investigación y su alcance corregir algunos aspectos, emprender mejoras o incluir nuevos elementos de interés para el desarrollo de software.

- Se recomienda el usp de *GitFlow* como estrategia de *Branching* para equipos en formación o de nivel intermedio. Equipos de desarrollo avanzados y con altos niveles de automatización de tareas y un modelo de CI/CD maduro podrían pasar a *Trunk-based development* en coordinación con los involucrados del proyecto.

- Se recomienda utilizar nombres estándar para las ramas permanentes. Como convención de nombre para las ramas de desarrollo se recomienda el nombre o el prefijo *dev* y esta debe utilizarse como la rama principal de desarrollo salvo en los casos en los que use la estrategia *trunk development* en los que esta rama no es requerida. La idea de la rama de desarrollo es agrupar los cambios en ella y restringir que los desarrolladores realicen cambios directamente en la rama maestra. Los cambios en la rama de desarrollo se revisan y, después de las pruebas, se fusionan. Los detalles del flujo deben definirse en la estrategia de *branching*.

- La rama maestra *Main* o *Master* es la rama predeterminada del repositorio de control de versiones. Debe ser estable en todo momento y no permitirá realizar un registro directo sobre ella. Esta rama solo puede fusionarlo después de la revisión del código y contar con una aprobación. Todos los miembros del equipo son responsables de mantener el maestro estable y actualizado.

- Algunas estrategias optan por una tercera rama permanente para aseguramiento de la calidad o *QA*. QA o rama de prueba, contiene todo el código para las pruebas y las pruebas automatizadas correspondientes a todos los cambios implementados. Antes de que cualquier cambio pase al entorno de producción, debe someterse a pruebas de calidad para obtener una base de código estable. El uso de esta rama permanente uso es recomendado para equipos medianos y grandes para la automatización de las pruebas. En otros casos el código de prueba puede acompañar a la rama de desarrollo y las ramas temporales.

- Algunas ramas son temporales y cumplen funciones especificas por lo que sus nombres ayudan a entender su objetivo:

    - *Release*: Se utiliza para identificar el código que será liberado en conjunto en una versión. Aunque es temporal puede extenderse su retención más tiempo que otras ramas temporales según se defina en la estrategia de *Branching*.
    - *Bug Fix*: Para atender incidentes sobre código estable
    - *Hot Fix*: Para atender incidentes sobre la etapa de revisión
    - *Feature Branches*: Para desarrollo de tareas concretas
    - *Experimental Branches*: Para explorar funciones
    - *WIP Branches*: Para trabajo temporal que puede demorar un poco más en completarse.

- En enfoques mono-repositorios se recomienda utiliza una convención de nombres que responda al propósito y la relación de los proyectos que se agrupan. Por ejemplo, si el mono-repo agrupa componentes separados de una misma aplicación como por ejemplo el desarrollo web y móvil la convención recomendada es incluir un prefijo o un sufijo en cada nombre de rama:

```
web-app/main
web-app/dev

ios-app/main
android-app/main
```

En el caso de proyectos mono-repositorios que agrupan micro-servicios se recomienda adoptar una convención de nombre que identifique cada servicio, siempre utilizando un nombre corto pero especifico o un código para cada servicio:

```
svc-1/main
svc-1/dev

svc-2/main
```

Cuando estos servicios hacen parte de un API se recomienda usar la rama principal (*main*) y desarrollo (*dev*) para agrupar las versiones de dicho API:

```
main  <-- Contiene la versión estable del API
dev   <-- Agrupa versiones candidatas de los servicios

release/1.0 <-- Agrupa los servicios dell API v1.0

api/svc-1/main
api/svc-1/dev

api/svc-2/main
```

Conclusiones

Esta investigación ha descrito el uso de arquitectura mono-repositorio en el desarrollo de aplicaciones, especialmente aquellas que se enfocan en el desarrollo de micro-servicios. Se identificaron sus ventajas y desafíos y se definieron lineamientos y recomendaciones junto con los argumentos a favor del uso de este enfoque. Se destaca el elemento cultural de la organización como un factor importante en el éxito así como la disciplina del equipo en la adopción del modelo por lo que se plantean lineamientos y recomendaciones que servirán de base a equipos que adopten el modelo. Una relación más directa sobre el impacto en la calidad o la productividad derivaba de la adopción del modelo requeriría investigación más detallada. Como todo modelo arquitectónico el uso de mono-repositorios es una alternativa válida dentro del portafolio de opciones que buscan agilizar y mejorar el crecimiento del equipo en el marco de desarrollos ágiles de de calidad empresarial.

Agradecimientos

El autor desea agradecer el confiable servicio de Github por facilitar la elaboración de este trabajo. Al equipo de Delivery y Software Factory de GBM Corp. por su contribución a este trabajo. El autor también quisiera expresar su agradecimiento a Emile Perron por permitir el uso de su fotografía vía Unsplash.

Referencias

1. Wikipedia. 2022. Monorepo. Retrieved Jun 2, 2022 from https://en.wikipedia.org/wiki/Monorepo

2. Monorepo's for Microservices Architecture
3. Lerna.js https://lerna.js.org/
4. How to structure microservices in your repository
5. From Monolith to Monorepo
6. GBM Release Management para Software, IN-COT-005, Mar 2022
7. Brousse, N., 2019, The Issue of Monorepo and Polyrepo In Large Enterprises, Retrieved Jun 2, 2022 from https://dl.acm.org/doi/10.1145/3328433.3328435
8. Monorepo, Manyrepo, Metarepo. 2017
9. Automatically detect changes and initiate different CodePipeline pipelines for a monorepo in CodeCommit
10. Savkin V. Misconceptions about Monorepos: Monorepo != Monolith. 2019
11. Monorepo tools https://monorepo.tools/
12. Monorepo vs Multi-Repo: Pros and Cons of Code Repository Strategies, 2022
13. Monorepo, Manyrepo, Metarepo
14. Jaspan C., Jorde M, Knight A., Sadowski C, Smith E., Winter C. and Murphy-Hill E. 2018. Advantages and Disadvantages of a Monolithic Repository: A Case Study at Google. In Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP'18). ACM, New York, NY, USA, 225–234. https://doi.org/10.1145/3183519.3183550
15. Rachel Potvin and Josh Levenberg. 2016. Why Google Stores Billions of Lines of Code in a Single Repository. Commun. ACM 59, 7 (June 2016), 78–87. https://doi.org/10.1145/2854146
16. Brito G., Terra R., Valente M., 2018, Monorepos: A Multivocal Literature Review CoRR abs/1810.09477 (2018). http://arxiv.org/abs/1810.09477
17. Harrys B. 2017. The largest Git repo on the planet. Retrieved Jun 7, 2022 from https://blogs.msdn.microsoft.com/bharry/2017/05/24/the-largest-git-repo-on-the-planet/
18. Hastings R. 2009. Netflix Culture: Freedom & Responsibility. Retrieved Jun 7, 2022 from https://www.slideshare.net/reed2001/culture-1798664/2-NetflixCultureFreedomResponsibility2
19. Haddad R., What Are the Best Git Branching Strategies. Mar 2022. Retrieved Jun 7, 2022 from https://www.flagship.io/git-branching-strategies/