plain concepts

REDISCOVER THE MEANING OF

TECHNOLOGY

plain concepts

Dev Day: Más que Código

28.03.2019

Agenda

9:30	Cómo petarlo con Blockchain en 45'
10:15	Derribando la torre de marfil
11:00	CAFÉ Y NETWORKING
11:30	Kubernetes 101
12:15	Desplegar en la nube y no morir en el intento
13:00	Depende ¿de qué depende?

plain concepts

plain concepts

Dev Day:
Más que Código

28.03.2019

Derribando la torre de marfil

Alejandro González y Jesús María Escudero

Developers @ PlainConcepts

Alejandro González

Developer

@glez4lex

Jesús María Escudero

Developer

@jm_escudero

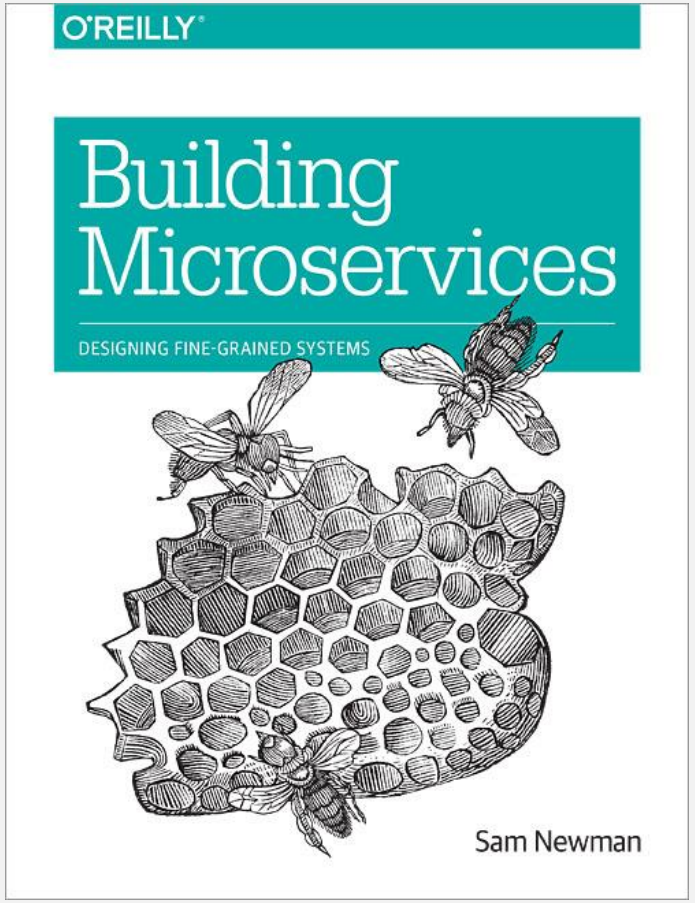
Una advertencia

*Microservicios **no** es una bala de plata*

Qué vamos a ver

El paso de una aplicación monolítica a una aplicación basada en microservicios

- Cómo ir separando nuestra aplicación en microservicios
- Ventajas y opciones que nos ofrece
- Puntos clave a tener en cuenta
- Retos que añade (y algunas posibles soluciones)



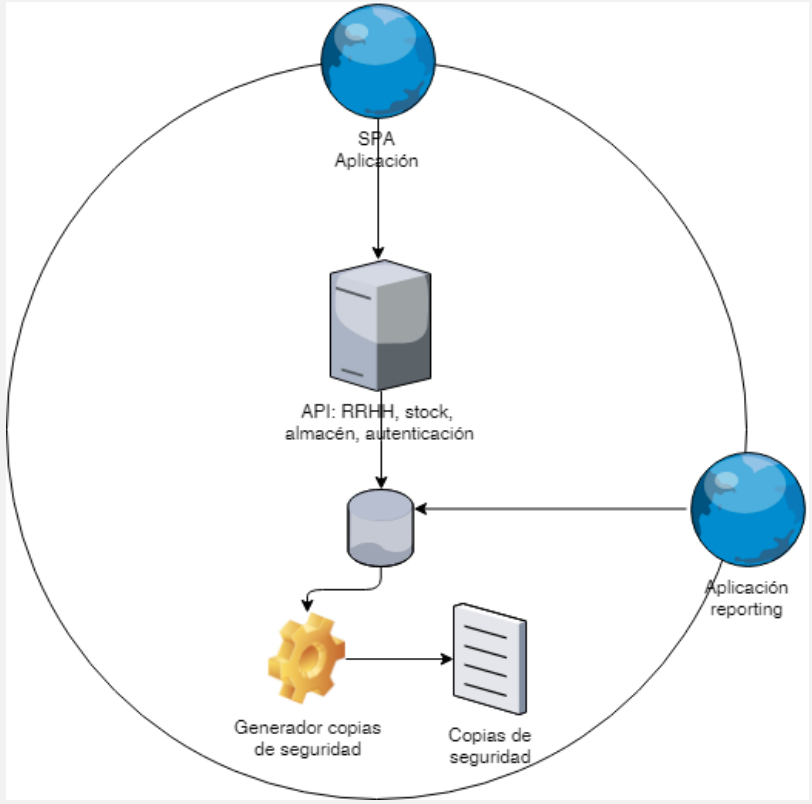
de Sam Newman

El punto de partida

La torre de marfil

Arquitectura inicial:

- Una SPA como aplicación cliente
- Un API REST que expone toda la lógica de: RRHH, stock, operaciones y autenticación de usuarios
- Una aplicación de gestión de las copias de seguridad
- Una aplicación de reporting



El punto de partida

La torre de marfil

Una aplicación de una empresa de logística que se encarga de:

- Gestionar el stock, la operativa de almacén y recursos humanos
- Autenticar a los usuarios que la manejan
- Visualizar reporting para la toma de decisiones estratégicas

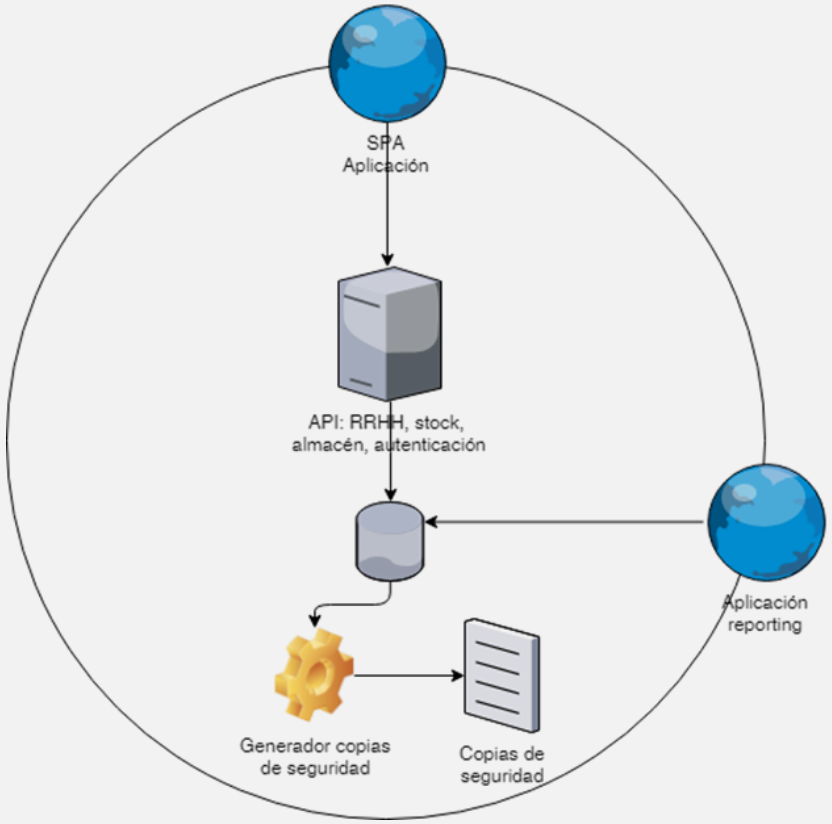
Dificultades que tienen:

- Refactorizar es extremadamente costoso
- Necesitan un stack tecnológico óptimo
- Desplegar nuevas características y parches es lento y bloqueante
- Copias de seguridad eternas debido al tamaño de la base de datos
- Dificultad para integrarse con servicios de terceros (p. ej. autenticación), debido alto grado de acoplamiento
- Dificultad para dimensionar correctamente los recursos de la solución para hacer frente a picos como el Black Friday, sin disparar los costes
- Otros equipos que trabajan en aplicaciones en otros departamentos encuentran complicado integrarse

Servicio de Seguridad

RETO

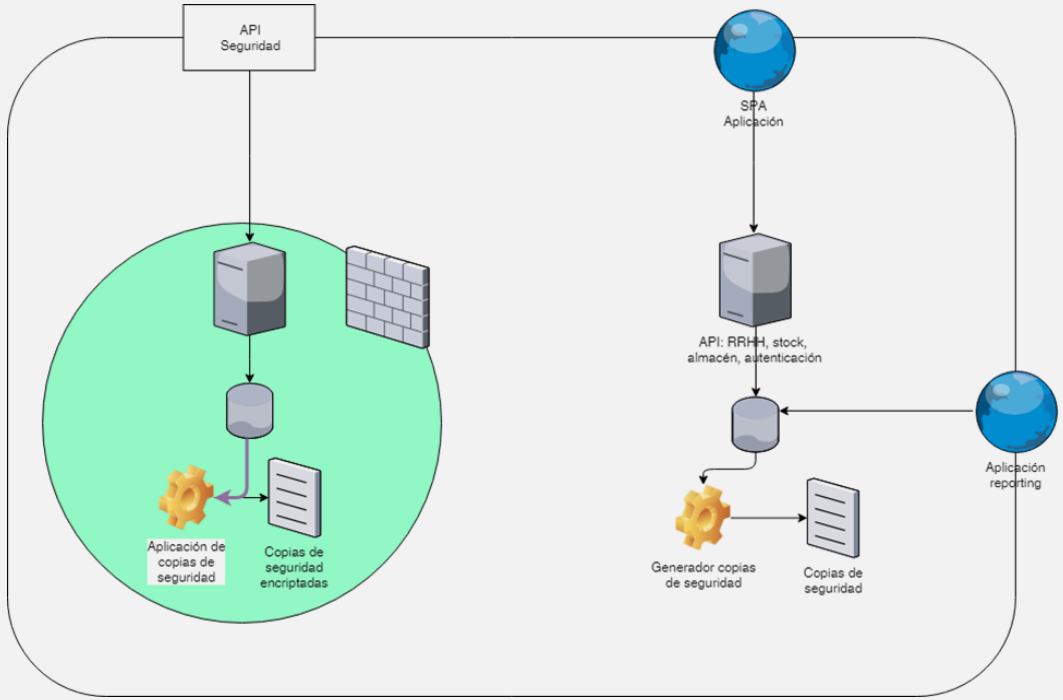
- Poder utilizar autenticación de terceros
- Usar un sistema de autenticación probado y depurado
- Cumplir legislación en cuanto a almacenar información sensible de usuarios
- Mejorar la seguridad de la información de los usuarios
- Poder extender la autenticación de forma más o menos transparente a otras APIs y productos que desarrollen en el futuro



Servicio de Seguridad

SOLUCIÓN

- Creamos un servicio de Single Sign On
- Usaremos Identity Server (probado y testeado)
- Aislamos con infraestructura específica ese servicio: red virtual, firewall, limitar accesos de desarrolladores…
- Separamos la base de datos
- Encriptamos la información sensible de usuarios
- Las copias de seguridad se encriptan automáticamente y se guardan en un sitio especial (con accesos especiales)
- Nuestro punto de entrada de autenticación será este. Si añadimos otros modos de autenticación, nos aislamos de esos cambios



Servicio de Seguridad

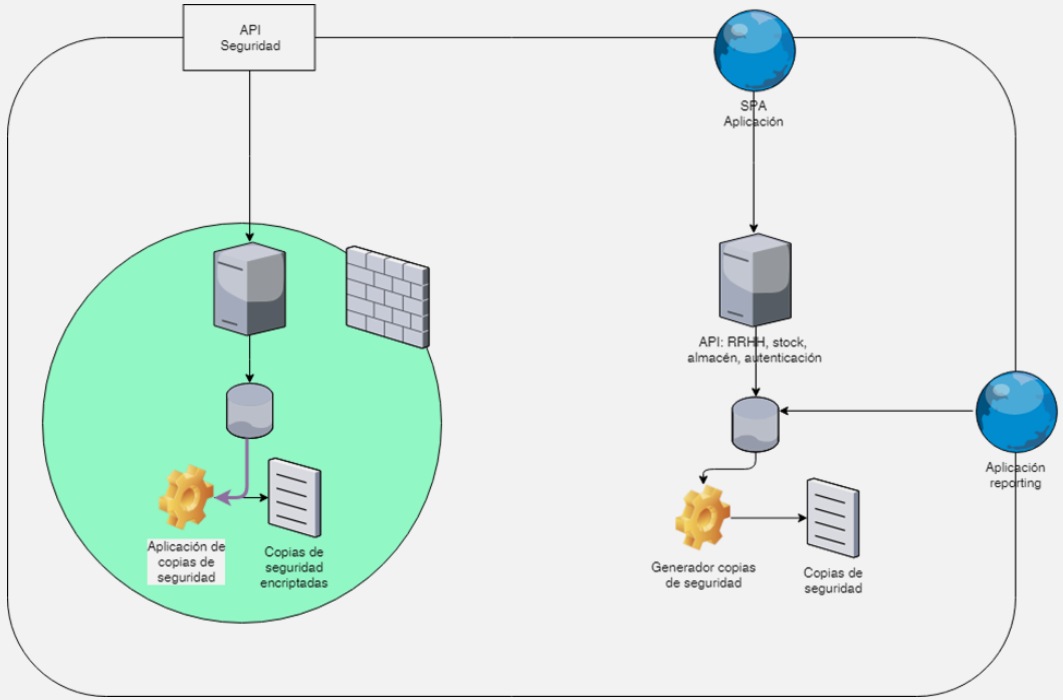
A TENER EN CUENTA, ADVERTENCIAS, NOTAS

- Hemos añadido complejidad de infraestructura
- Hemos separado la base de datos
- Podemos usar tipos distintos de bases de datos
- Las copias de seguridad se harán más rápidas
- Si la aplicación quiere datos de los usuarios, tendrá que comunicarse con este servicio para obtenerla → hay que definir cómo se van a comunicar los servicios entre sí

Servicio de Business Intelligence

RETO

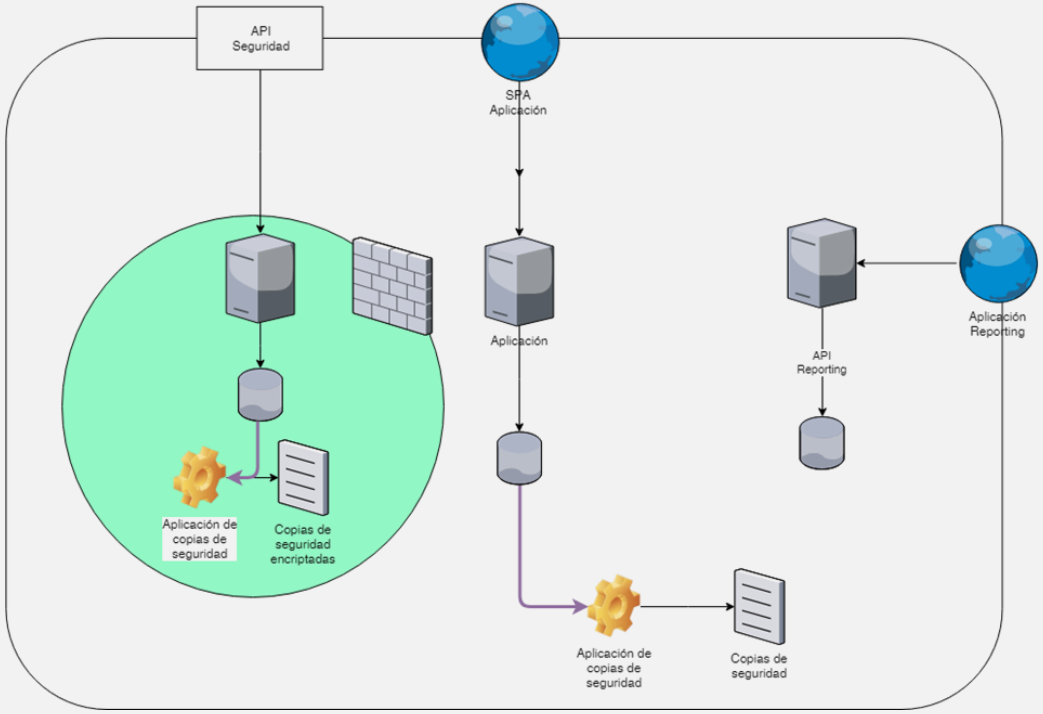
- El modelo de BBDD y la tecnología limitan las posibilidades de reporting
- La aplicación de reporting se tiene que integrar a nivel de BBDD, convirtiendo la misma en un API compartida y generando un grado altísimo de acoplamiento
- El equipo que desarrolla la solución tiene menos experiencia en esta área del negocio que el equipo que desarrolla la aplicación de reporting
- El flujo de trabajo de la solución y sus tiempos influyen fuertemente en la capacidad de evolucionar el reporting



Servicio de Business Intelligence

SOLUCIÓN

- Desarrollar un nuevo microservicio de Reporting que expondrá un API pública que compartirán ambas soluciones
- Incorporar un producto de BBDD óptimo para el caso: NoSQL
- Transferir la responsabilidad de este microservicio al equipo de reporting
- Dividir el despliegue, de forma que el nuevo microservicio tendrá su propio pipeline



Servicio de Business Intelligence

PUNTOS CLAVE

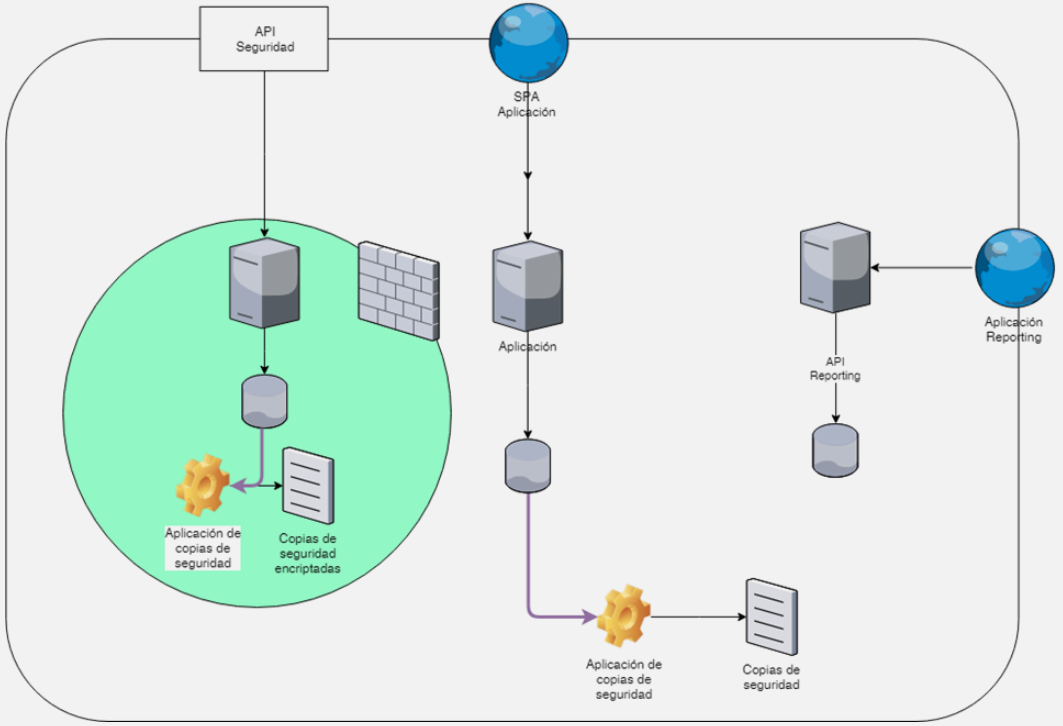
- La integración a nivel de API requiere una buena comunicación entre ambos equipos y seguir algunas buenas prácticas como implementar documentación a nivel de API (con Swagger p. ej.) o versionado semántico

- La incorporación de datos para alimentar al API de reporting se puede resolver mediante el patrón de bombeo de datos

Rompiendo el negocio

RETO

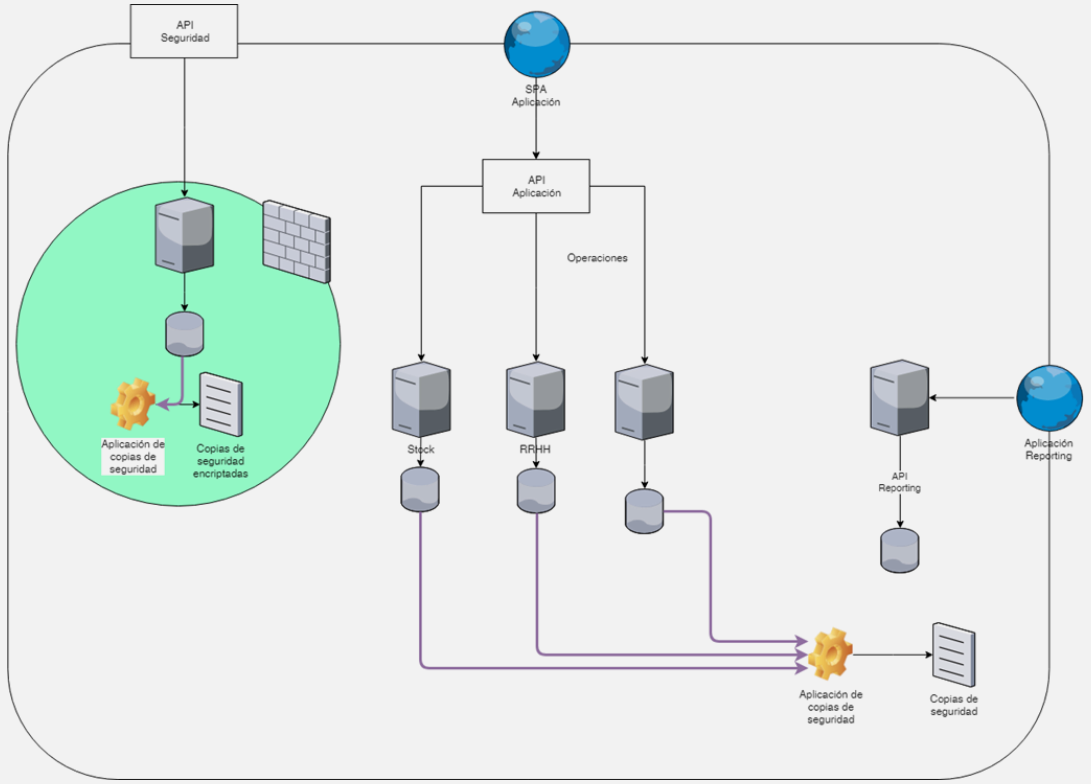
- Recursos humanos tiene un equipo de desarrollo propio que mantiene una aplicación Java que tendría que integrarse también
- Seguimos teniendo problemas de dimensionado cuando vienen picos de trabajo y los costes siguen siendo mejorables
- Tenemos bastantes problemas de concurrencia y de rendimiento en operaciones aparentemente sencillas



Rompiendo el negocio

SOLUCIÓN

- Desarrollar un nuevo microservicio de RRHH
- Desarrollar un nuevo microservicio de Operaciones
- Transferir la responsabilidad de RRHH al equipo dedicado
- Actualizar el stack: el nuevo servicio de RRHH se desarrollará en Java
- Actualizar el stack de Operaciones: Go sobre una base de datos de grafos



Rompiendo el negocio

PUNTOS CLAVE

- La integración entre los nuevos servicios se hará a través de un bus de eventos, de forma que reducimos el acoplamiento al mínimo, pero: el código de los clientes que deberán interpretar los eventos deberá ser altamente tolerante para evitar los problemas de integración y deberemos emplear buenas prácticas para facilitar la misma.

- La integración con la(s) aplicación cliente se hará a través de un API REST específica que orquestará la composición de las operaciones contra los microservicios requeridos (empleando tecnologías como GraphQL.)

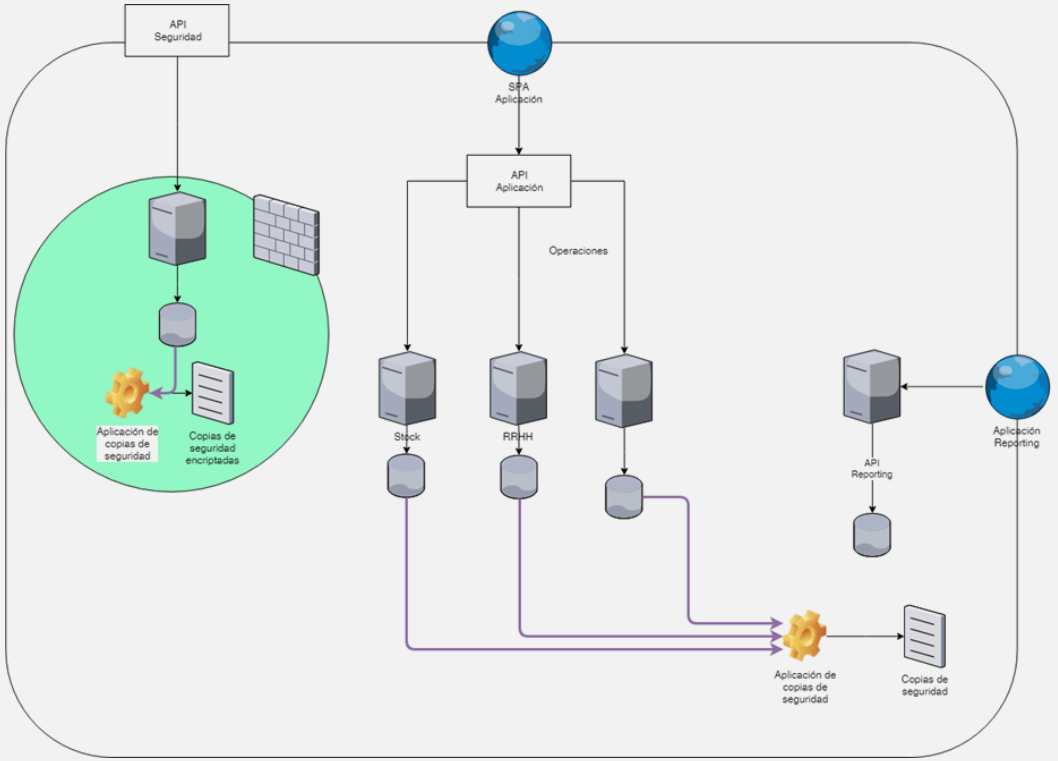
- El despliegue se vuelve aún más complejo.

- Gestionar el logging a su vez también.

Exponiendo nuestras API

RETO

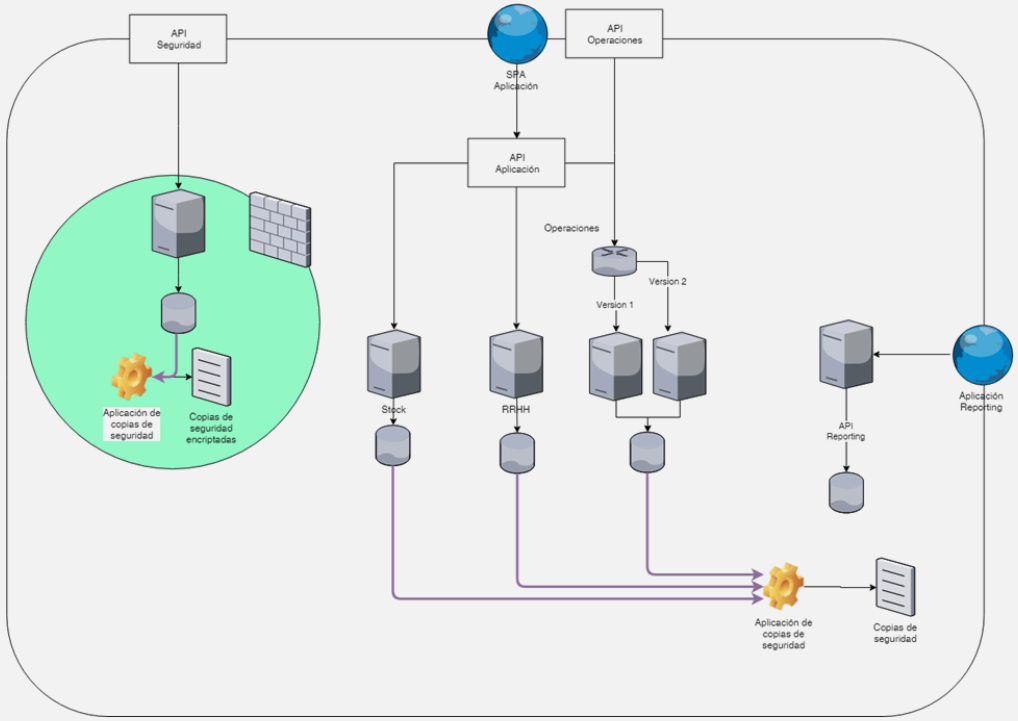
- Queremos que los proveedores puedan acceder a nuestra API de operaciones
- No queremos que exponer esa API nos impida evolucionar nuestra aplicación
- No queremos que nuestros cambios afecten a los proveedores
- Queremos tener una documentación actualizada de nuestra API



Exponiendo nuestras API

SOLUCIÓN

- Vamos a exponer al exterior algunos endpoints del API de operaciones
- Vamos a generar documentación en Swagger para que los proveedores sepan siempre cómo usar la última versión
- Añadiremos un API Gateway para que redirija las versiones al servicio que corresponda, para aislar al proveedor de lo que hay por detrás (y también a nosotros mismos, que somos consumidores)



Exponiendo nuestras API

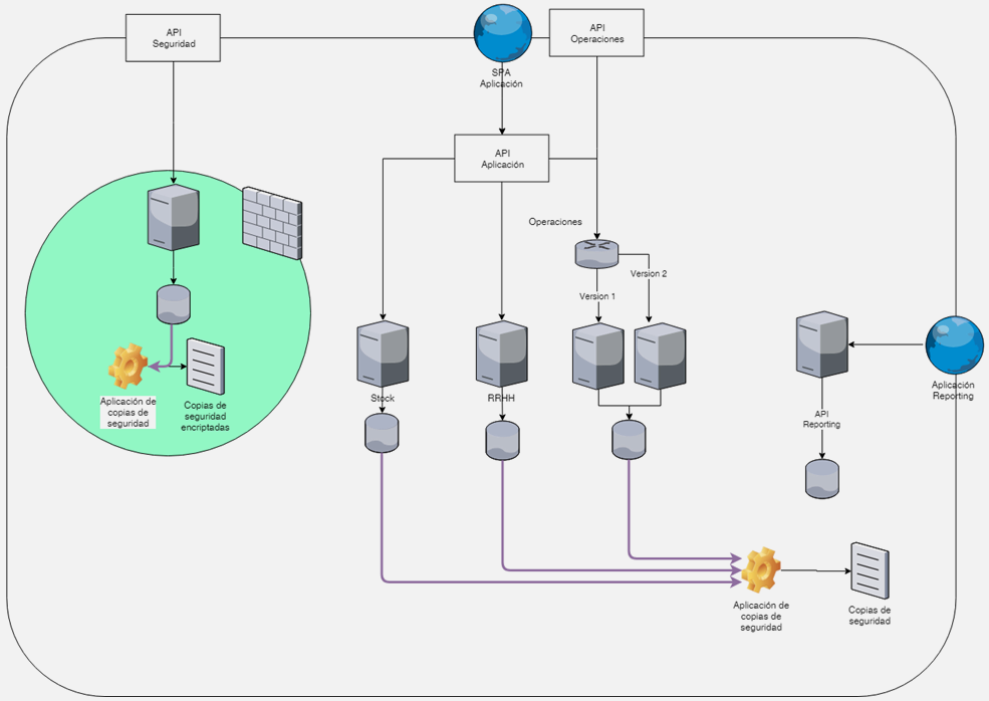
A TENER EN CUENTA, ADVERTENCIAS, NOTAS

- Hay que securizar el acceso (con el servicio de seguridad).
- Hay que generar API Keys para los proveedores.
- Se complica la gestión de los servicios porque tenemos que gestionar distintas versiones de un mismo servicio.
- Herramienta automática para que genere una documentación estándar. En .NET, por ejemplo, Swashbuckle para Swagger.
- Hay que monitorizar el uso de cada versión, porque podemos tener que deprecarlas cuando ya no se usen.

Escalando nuestra aplicación

RETO

- Queremos que nuestra aplicación escale con lógica.
- Queremos escalar los servicios que lo necesiten cuando lo necesiten (Navidad, Back Friday…)



Escalando nuestra aplicación

SOLUCIONES

- Desplegar cada servicio en una máquina distinta
- Vamos a desplegar en la nube (Azure, AWS, Google Cloud…), que nos permite aumentar/reducir las características de cada una de esas máquinas (CPU, memoria…) y poder pagar solamente por lo que gastemos
- Vamos a monitorizar los recursos que consume cada servicio, para poder anticipar cuándo necesitaremos más o menos recursos en esas máquinas
- Vamos a mirar a ver si alguna funcionalidad podría aislarse incluso más (envío de correos electrónicos a proveedores) y se pudiera usar como si fuera serverless (Azure Functions, AWS Lambdas…), porque en estos casos solamente pagaremos por uso de esa función.
- En los momentos en los que ya no podamos escalar dentro de la misma máquina, levantaremos varias instancias de los servicios en máquinas distintas. Necesitaremos un API Gateway para centralizar el flujo.
- Vamos a cachear algunas peticiones que son muy comunes y no cambian mucho (por ejemplo, el listado de estanterías de un almacén)

Escalando nuestra aplicación

A TENER EN CUENTA, ADVERTENCIAS, NOTAS

- Hay que aprovisionar cada una de las máquinas con lo que los servicios necesitan.
- Lo ideal es automatizar el aumento/la disminución de recursos/máquinas
- Hay que orquestar la comunicación entre los servicios (hay que tener cuidado cuando las IP de cada servicio puedan cambiar o pueda haber varias instancias del mismo servicio)
- Necesito estadísticas de uso y de estado de las máquinas
- Si cacheamos información, hay que tener mucho cuidado y planificar qué se cachea (y en qué formato) y cuando se regenera esa información.

TOOLING

- Packer
- Ansible
- Docker
- Kubernetes
- Azure Container Registry
- Nginx
- Redis

Integración entre servicios

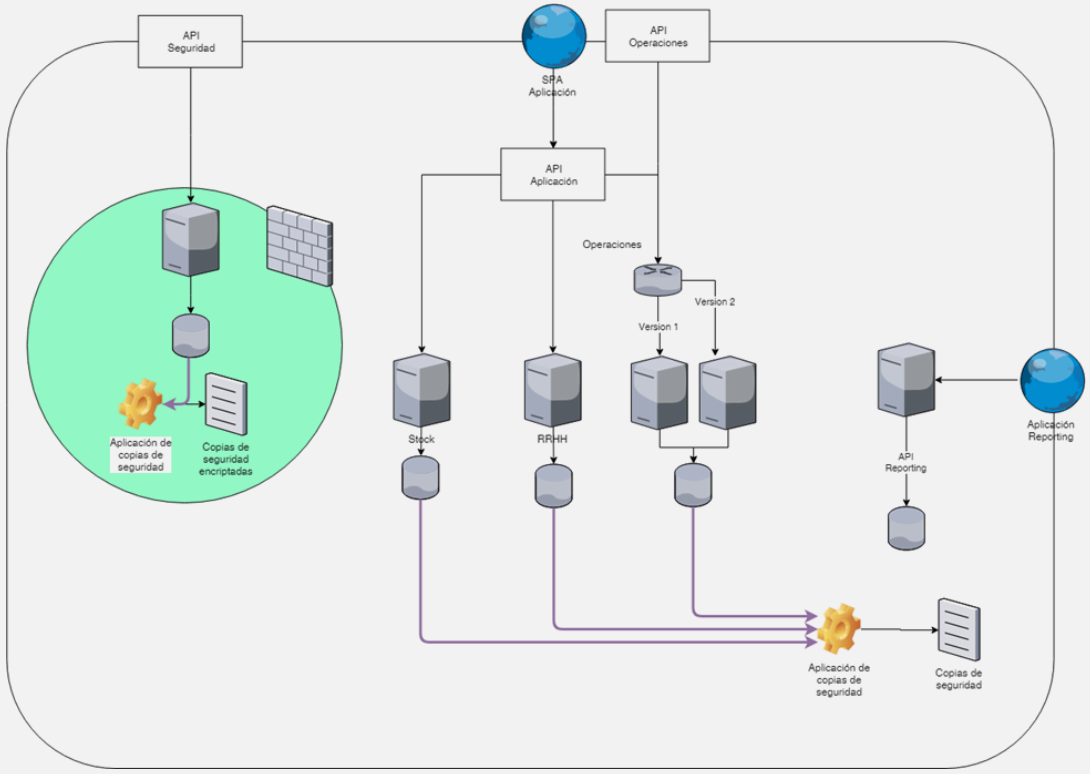
OPCIONES

	A favor	En contra
Cola de eventos	• Puedo añadir fácilmente nuevos servicios que se suscriban a esos eventos • El productor no sabe quién le consume	• Mantener transacciones puede ser duro
Llamadas directas (REST API, RPC…)	• Fácil para gestionar transacciones	• Aún así, hay que desarrollar mecanismos de rollback
Data-pump	• Más fácil de todos	• No es real-time
Orquestación	• Es más sencilla que la cola de eventos o las llamadas directas	• Alto grado de acoplamiento

Integración continua/despliegue continuo

RETO

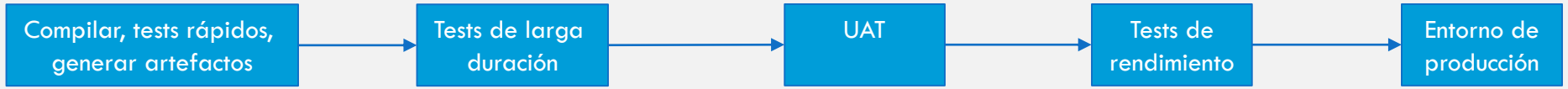
- Queremos garantizar que el código de desarrollo funciona
- Queremos certificar lo antes posible que los últimos cambios de un servicio no rompen la integración con el resto de los servicios
- El despliegue debe ser individual para cada servicio y lo más rápido posible



Integración continua/despliegue continuo

SOLUCIONES

- Cada servicio es un proyecto independiente.
- Cada servicio tiene, como poco un pipeline de build, donde: comprobamos el código de manera estática, compilamos (si aplica), ejecutamos tests y generamos un artefacto **potencialmente desplegable**.
- Creamos varios entornos, por donde va a ir pasando el artefacto, hasta que eventualmente pase a producción



Integración continua/despliegue continuo

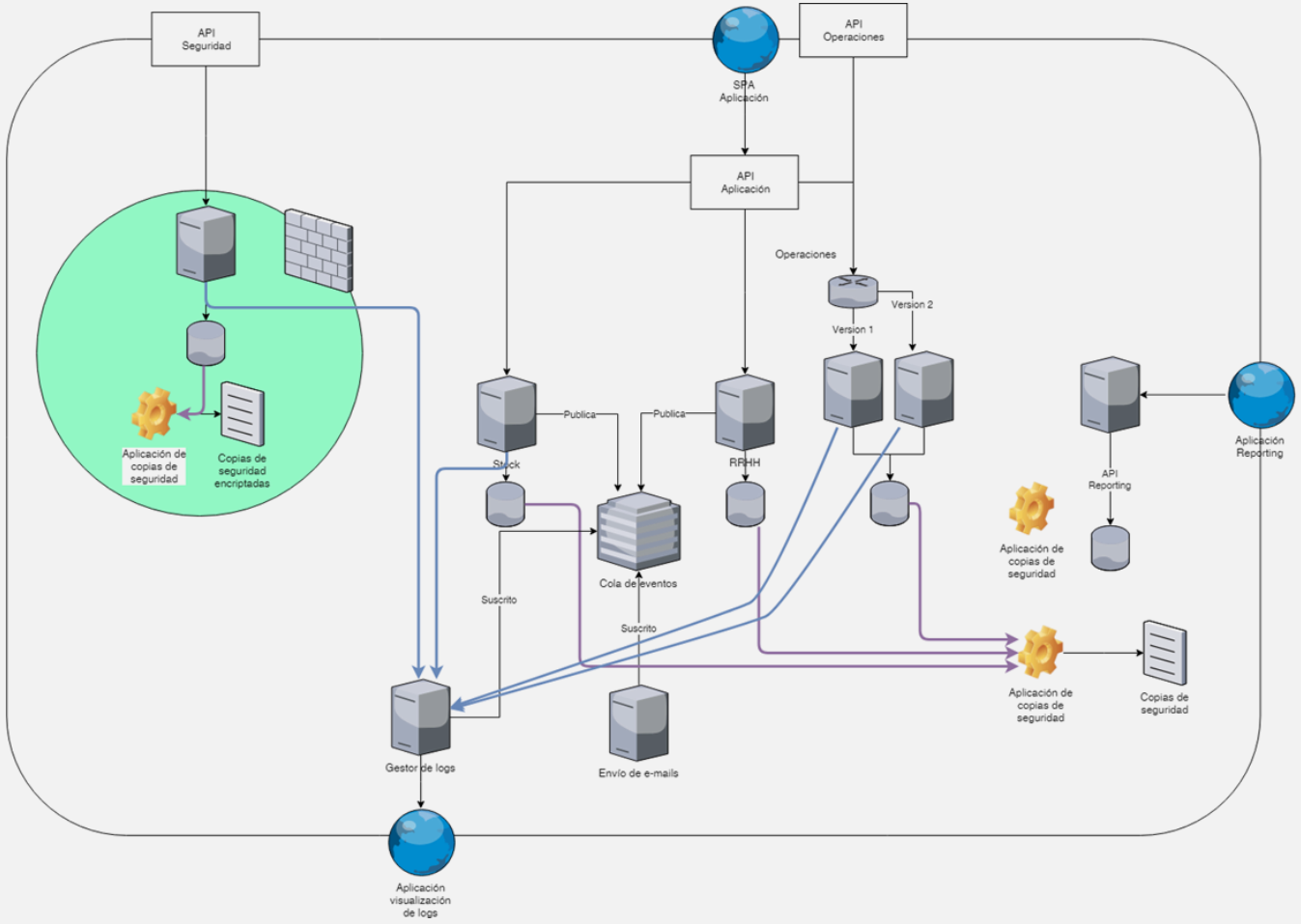
A TENER EN CUENTA, ADVERTENCIAS, NOTAS

- Cualquier **push** generará un artefacto que es candidato a estar en producción
- Mantener el pipeline de build en verde es prioritario
- Aunque sean distintos, hay que configurar entornos (desarrollo, build, testing, UAT, producción) lo más similares posible
- Esos entornos deben ser creables de manera automática (esos entornos deben ser versionables)
- Debo evitar cambiar esos entornos de manera manual
- Al tener un proyecto para cada servicio, se complica un poco el desarrollo (los desarrolladores tendrán que tener arrancados varios servicios). Podríamos usar contenedores para los servicios que no estemos implementando.

TOOLING

- Terraform
- Docker
- Jenkins
- Azure DevOps

Foto Final


API Seguridad
SPA Aplicación
API Operaciones
API Aplicación
Operaciones
Version 1
Version 2
Aplicación de copias de seguridad
Copias de seguridad encriptadas
Publica
Publica
Stock
RRHH
Cola de eventos
Suscrito
Suscrito
Gestor de logs
Envío de e-mails
Aplicación visualización de logs
API Reporting
Aplicación Reporting
Aplicación de copias de seguridad
Aplicación de copias de seguridad
Copias de seguridad

Todavía hay más

- Gestión centralizada de logs
- Backup de base de datos
- Tests
- Gestión de la caché
- …

Alejandro González

Developer

@glez4lex

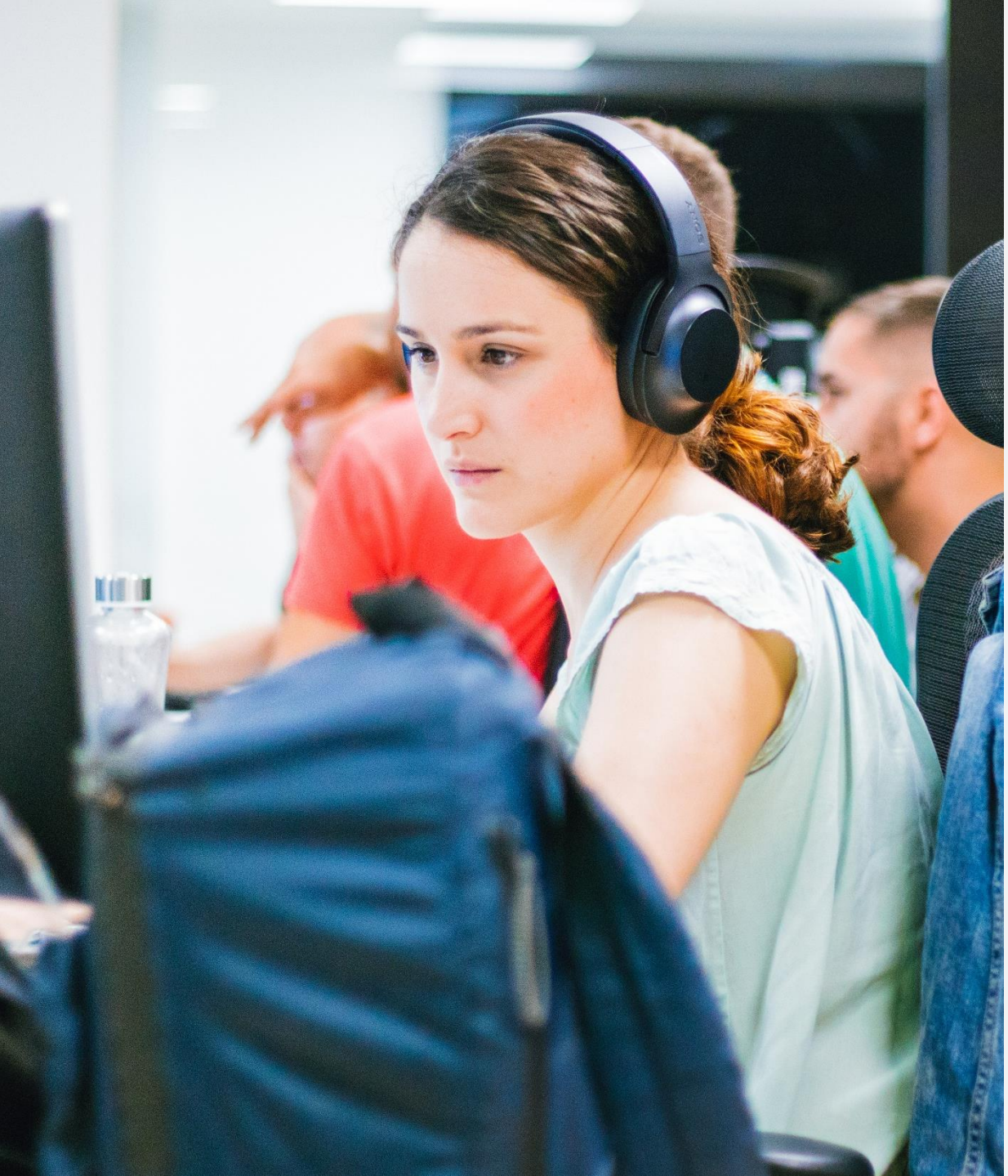
aggutierrez@plainconcepts.com

Jesús María Escudero

Developer

@jm_escudero

jmescudero@plainconcepts.com



¡ESTAMOS CRECIENDO!

Si quieres unirte a nuestro equipo echa un vistazo a las ofertas de trabajo que tenemos publicadas actualmente en nuestro **LinkedIn** o **página web.**

Great Place To Work.®

Best Workplaces™

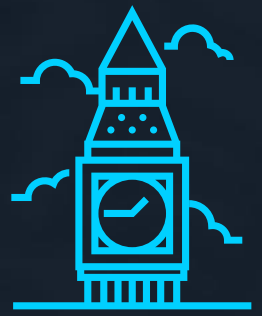
ESPAÑA 2019

plain concepts

www.plainconcepts.com

BARCELONA

SEVILLA



LONDON

BILBAO

DUBÁI

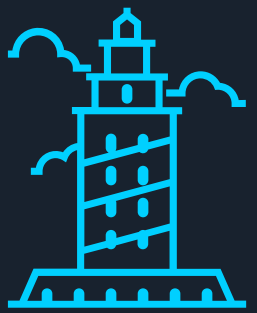
SEATTLE

MADRID

LEÓN

AMSTERDAM

FRANKFURT



CORUÑA

plain concepts

¡MUCHAS GRACIAS!

www.plainconcepts.com

@plainconcepts