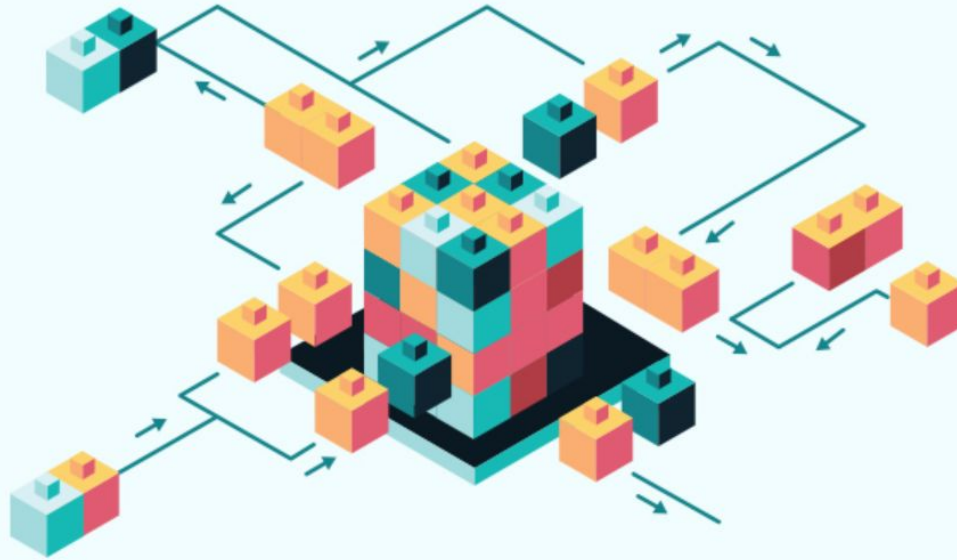


# Microservicios

¿Amigos o enemigos?



# Microservicios

¿Amigos o enemigos?

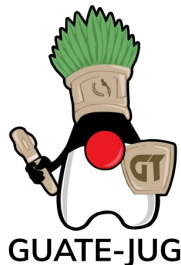


# Jorge Cajas



@cajasmota

- JUG co-leader en GuateJUG
- Desarrollador Java certificado con +8 años de experiencia.
- Ganador del Duke Choice Award en 2016 con GuateJUG
- Parte del comité organizador del JConf Guatemala
- Conferencista desde 2012 en eventos nacionales e internacionales.
- Consultor en MangoChango S.A.



# Encuesta 1

<http://etc.ch/rJQr>

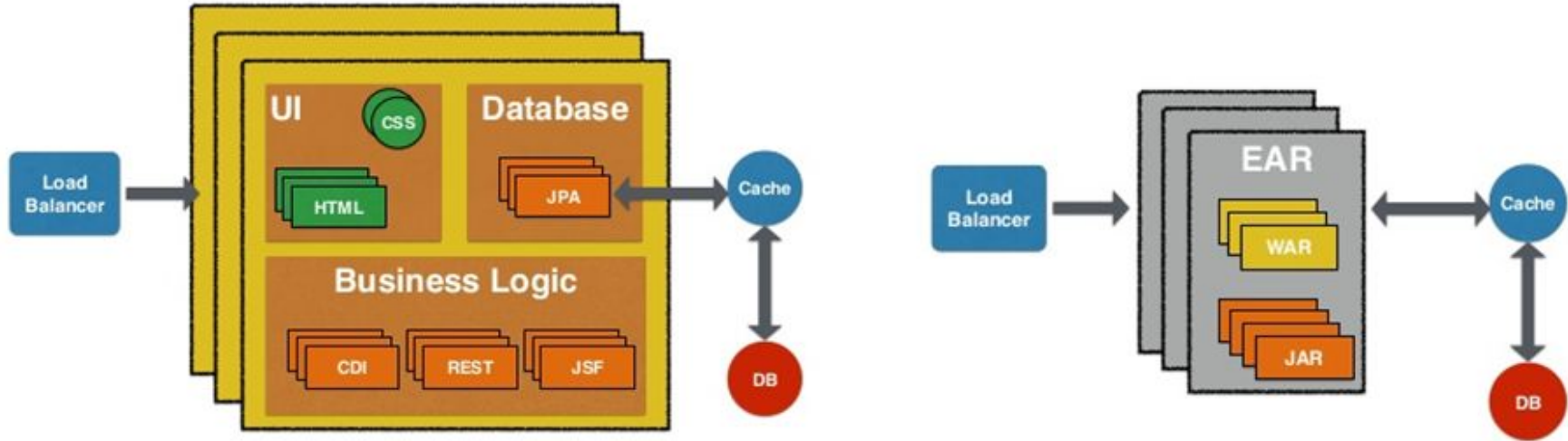
# Contenido

- **Los monolitos son cosas del pasado?**
- **Los microservicios son el futuro?**
- **Migración de monolitos a microservicios**
- **12 Factor apps**
- **Conclusiones**

# ¿Los monolitos son del pasado?



# Arquitectura monolítica



Fáciles de construir, escalar y mantener.



# ¿Los microservicios son el futuro?

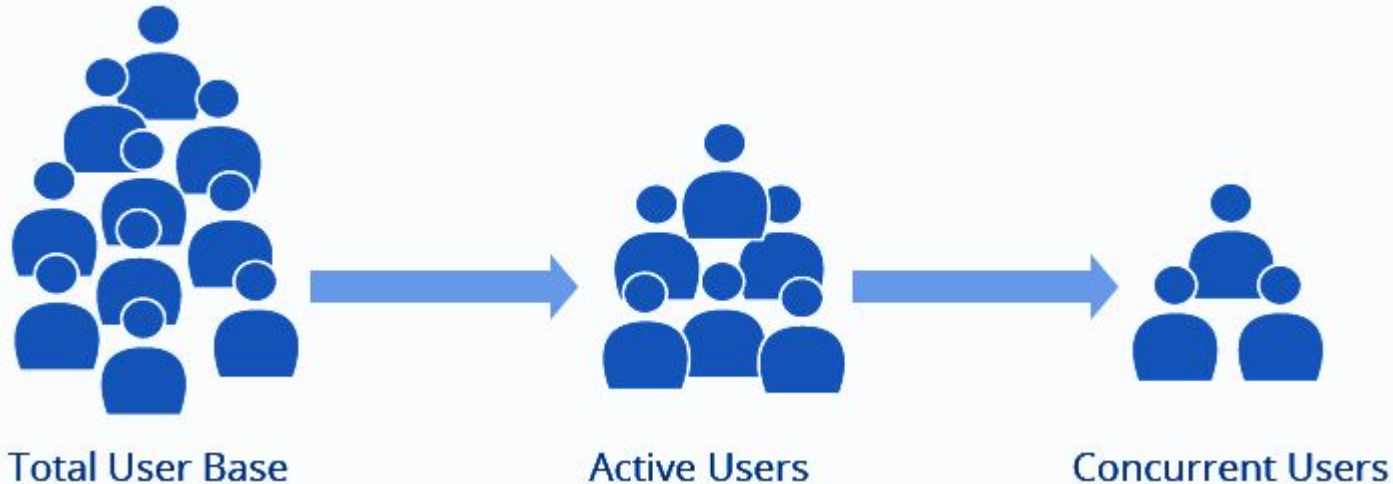




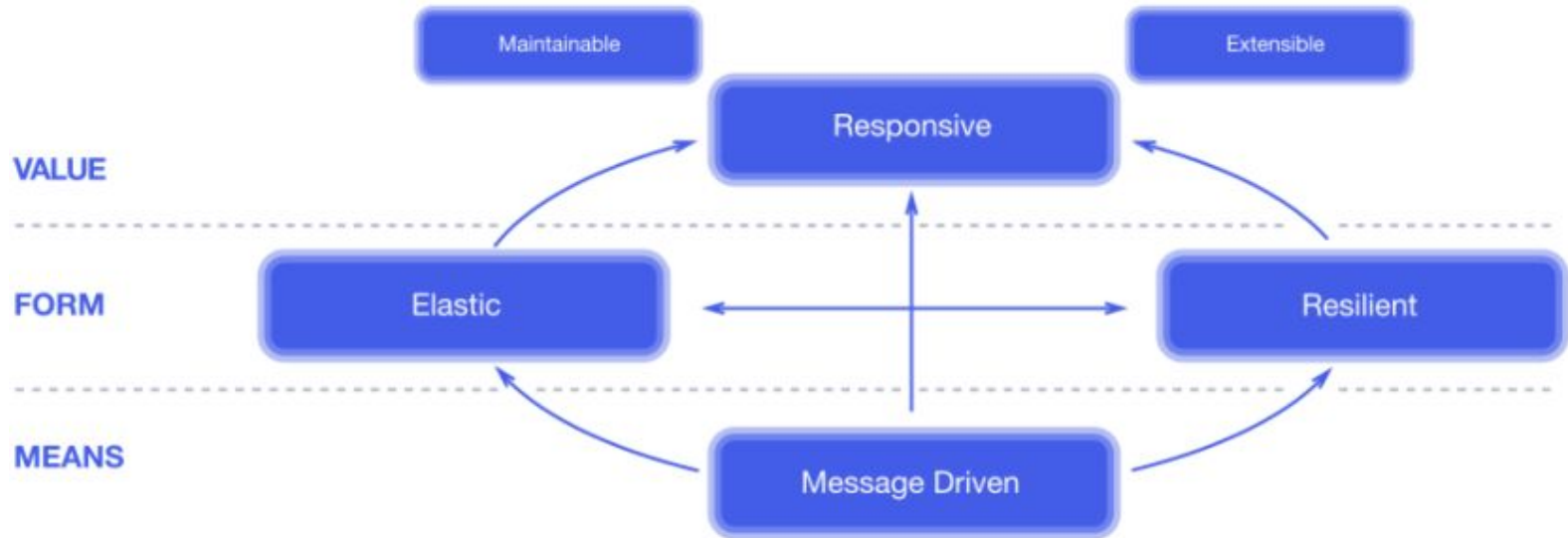
# Encuesta 2

<http://etc.ch/z7BW>

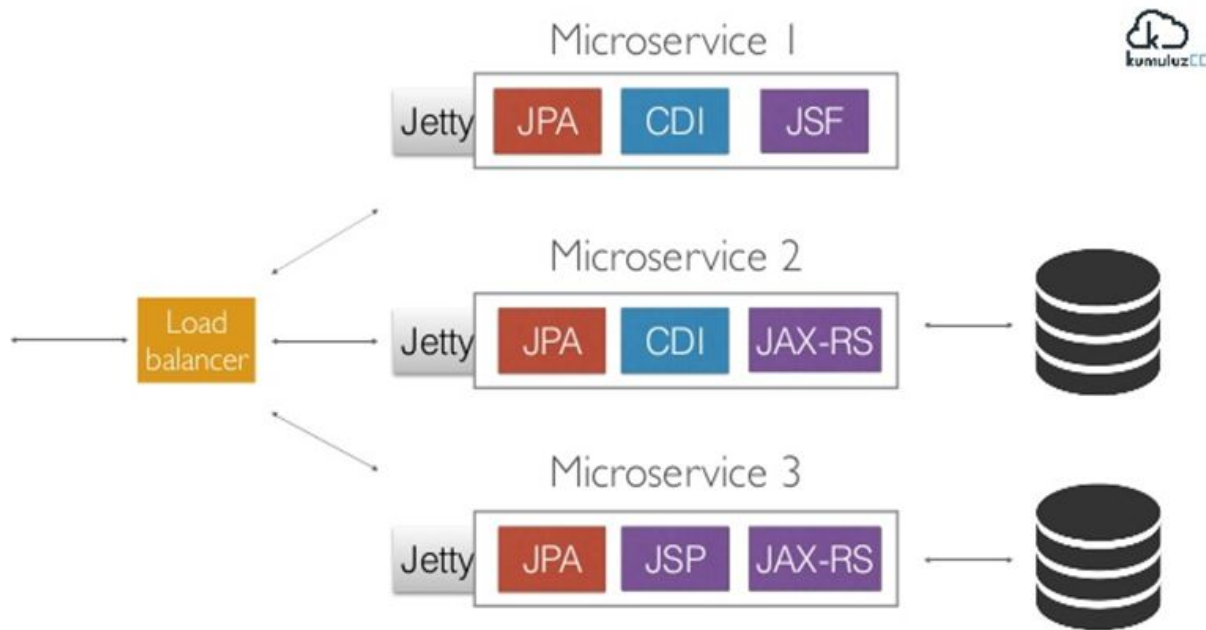
# Usuarios concurrentes



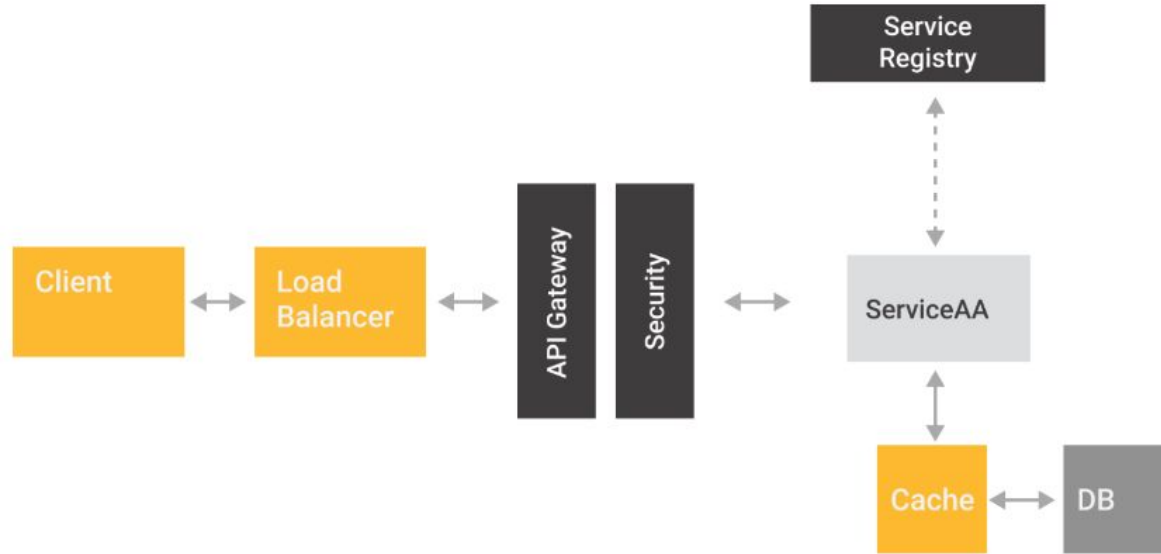
# ¿Porque microservicios?



# Arquitectura Microservicios

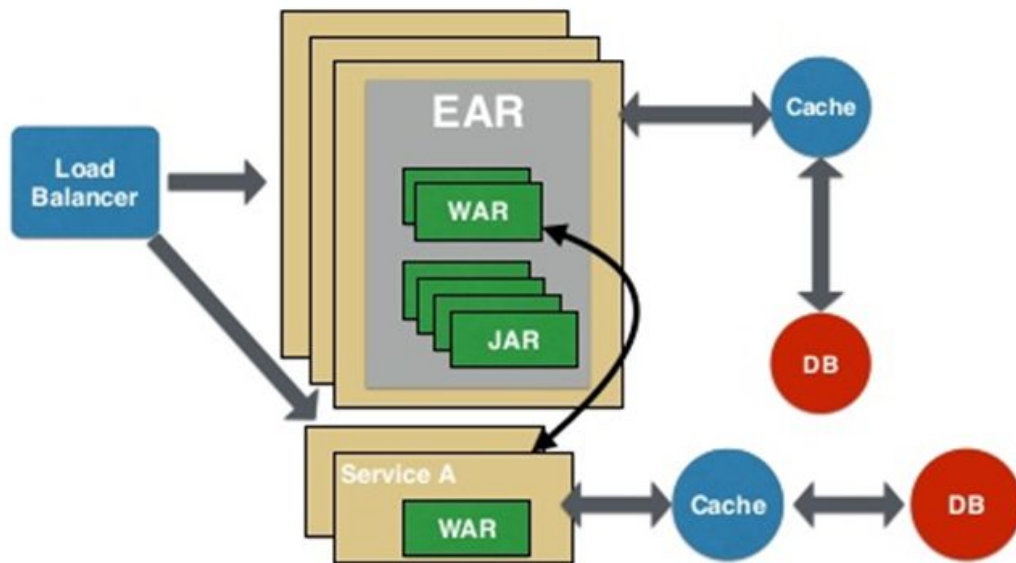


# Arquitectura Microservicios



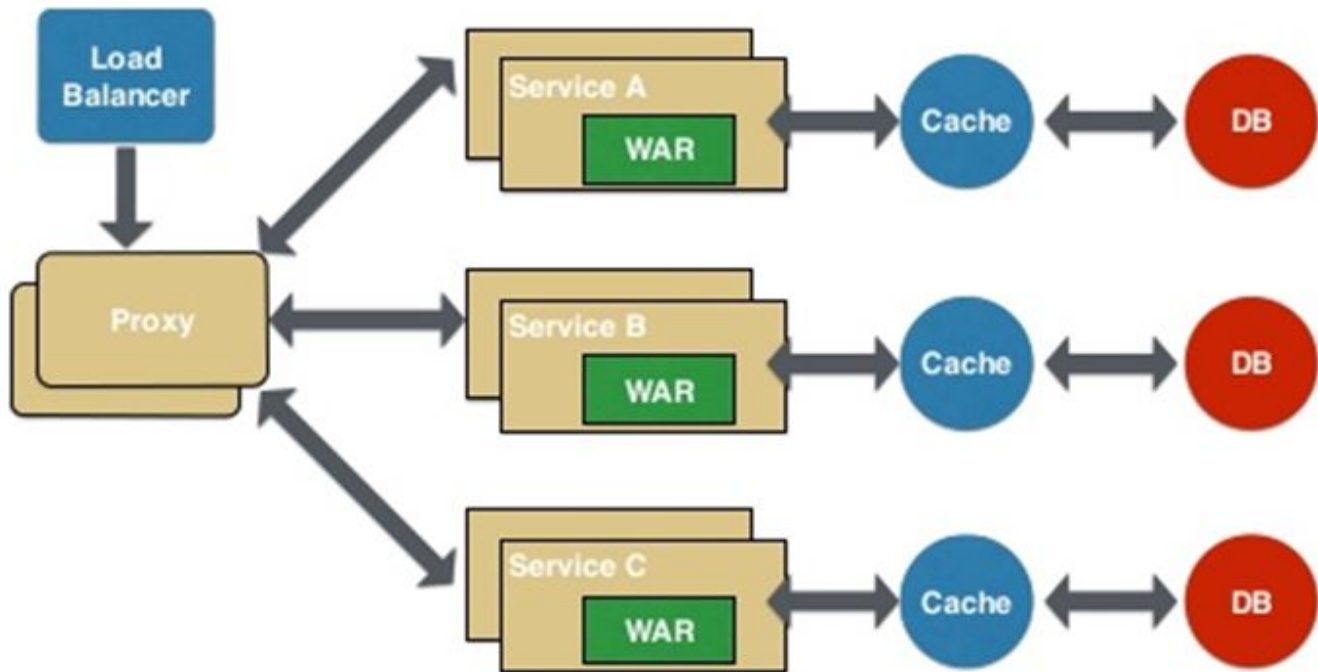
Es por mucho más compleja de construir, desplegar y mantener

# De monolitos a microservicios

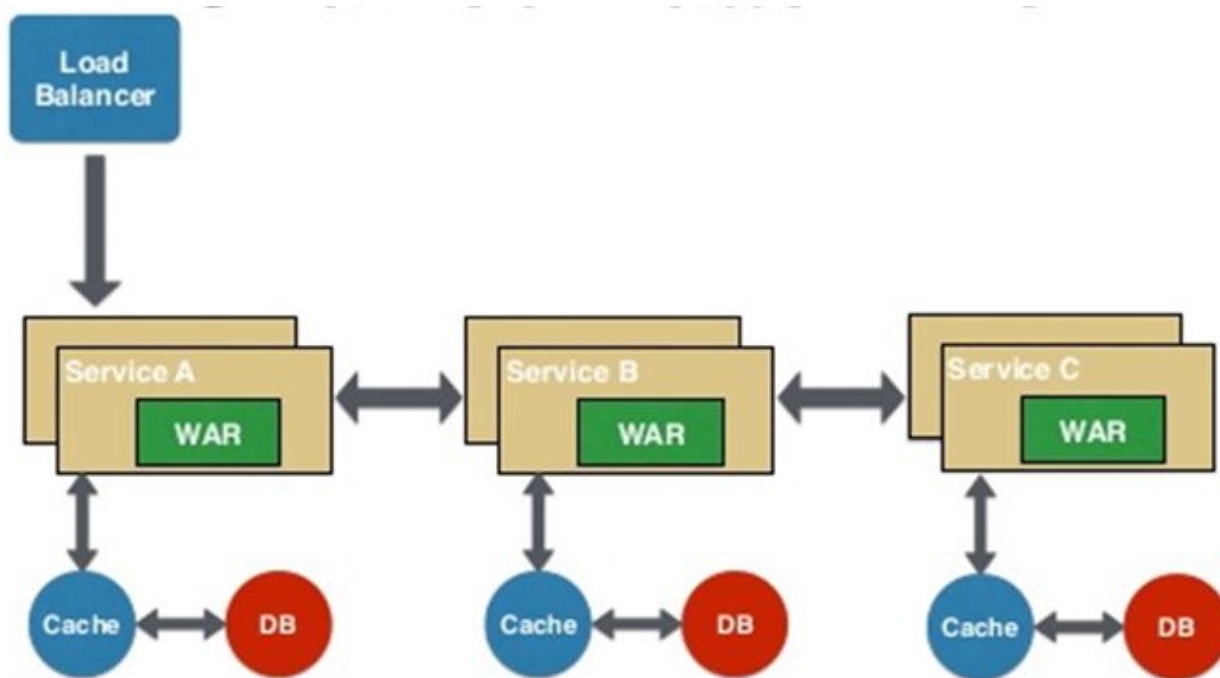




# De monolitos a microservicios

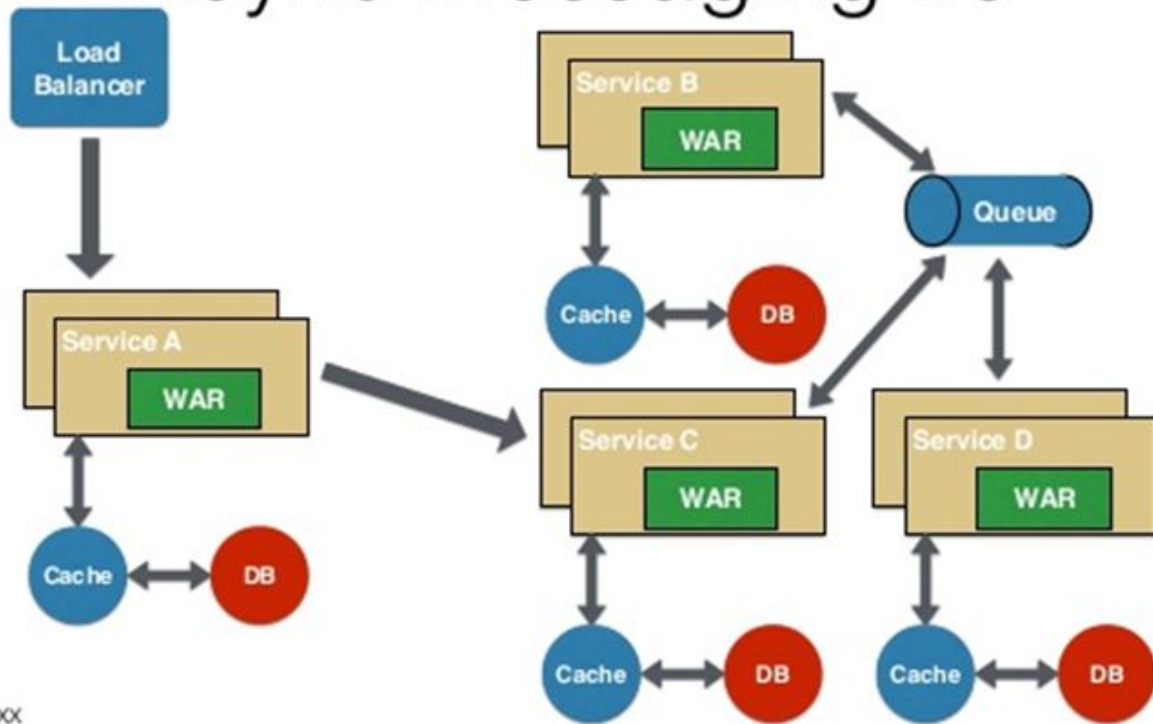


# De monolitos a microservicios

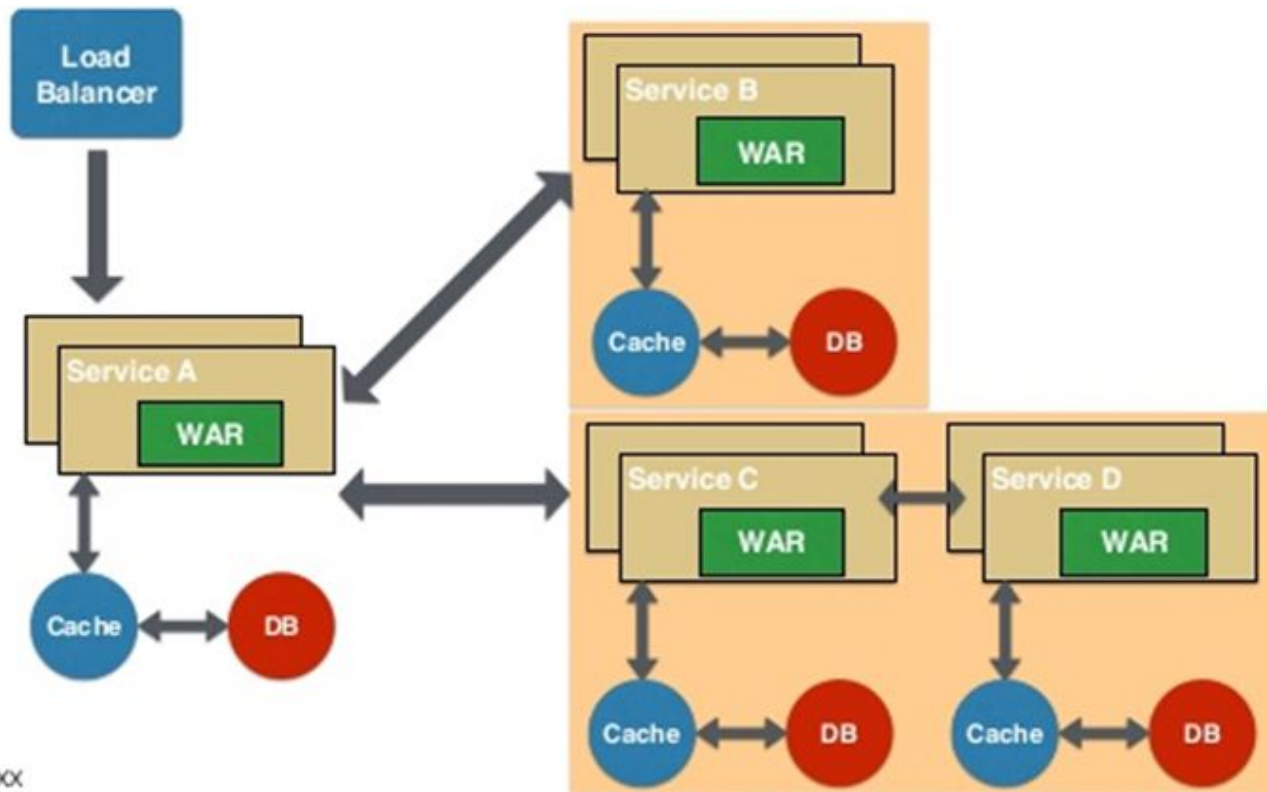


# De monolitos a microservicios

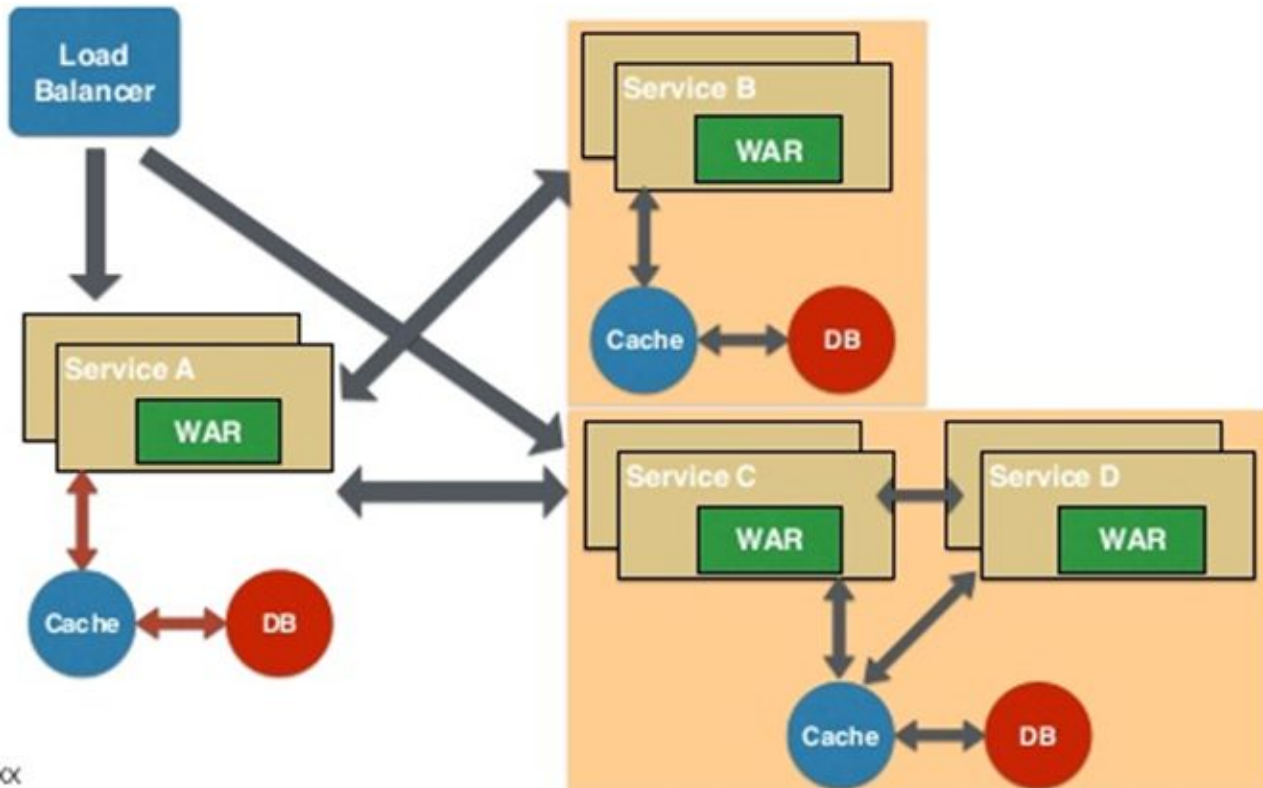
## Async Messaging #5



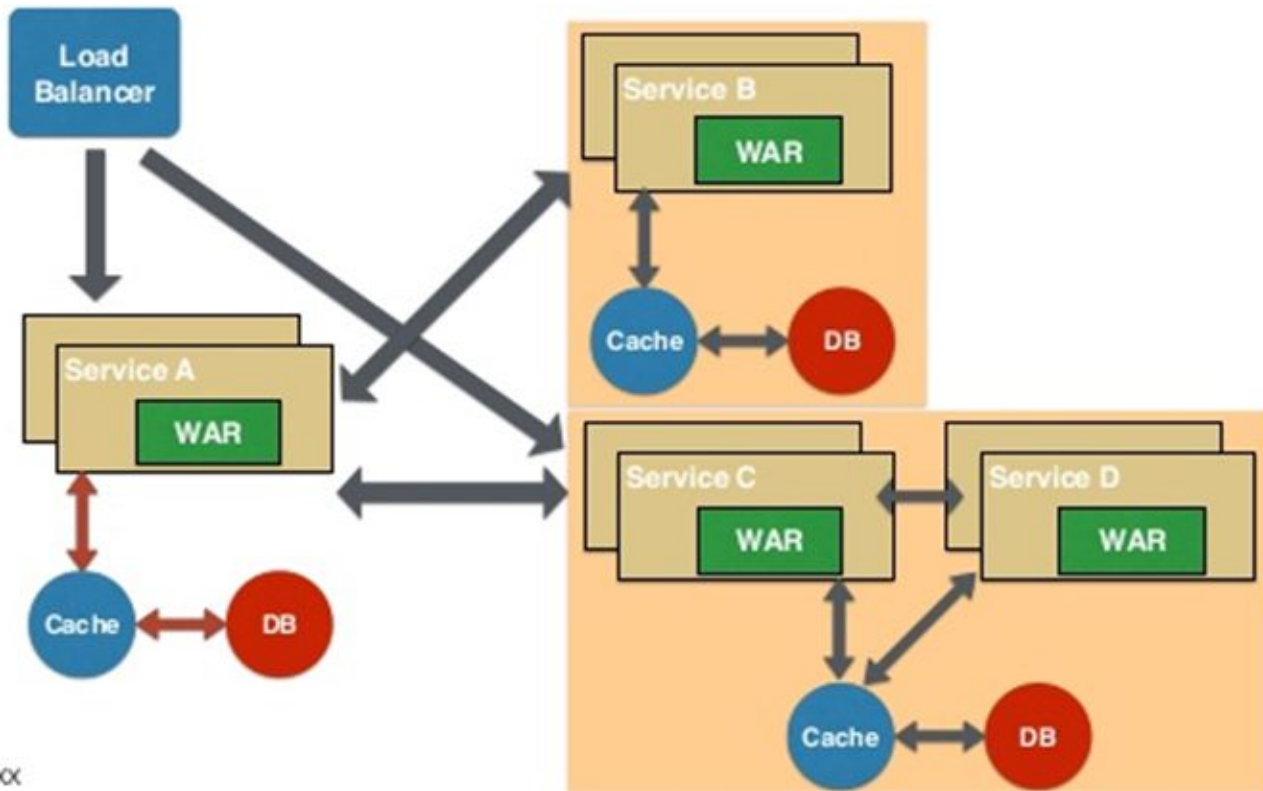
# De monolitos a microservicios



# De monolitos a microservicios



# De monolitos a microservicios





# 12 Factores para microservicios

<b>1. CODEBASE</b> One codebase tracked in SCM, many deploy	<b>2. DEPENDENCIES</b> Explicitly declare isolate dependencies	<b>3. CONFIGURATION</b> Store config in the environment
<b>4. BACKING SERVICES</b> Treat backing services as attached resources	<b>5. BUILD, RELEASE, RUN</b> Strictly separate build and run stages	<b>6. PROCESSES</b> Execute app as stateless processes
<b>7. PORT BINDING</b> Export services via port binding	<b>8. CONCURRENCY</b> Scale out via the process model	<b>9. DISPOSABILITY</b> Maximize robustness & graceful shutdown
<b>10. DEV/ PROD PARITY</b> Keep dev, staging, prod as similar as possible	<b>11. LOGS</b> Treat logs as event stream	<b>12. ADMIN PROCESSES</b> Run admin / mgmt tasks as one-off processes

# ¿12 Factores para monolitos?

<b>1. CODEBASE</b> One codebase tracked in SCM, many deploy	<b>2. DEPENDENCIES</b> Explicitly declare isolate dependencies	<b>3. CONFIGURATION</b> Store config in the environment
<b>4. BACKING SERVICES</b> Treat backing services as attached resources	<b>5. BUILD, RELEASE, RUN</b> Strictly separate build and run stages	<b>6. PROCESSES</b> Execute app as stateless processes
<b>7. PORT BINDING</b> Export services via port binding	<b>8. CONCURRENCY</b> Scale out via the process model	<b>9. DISPOSABILITY</b> Maximize robustness & graceful shutdown
<b>10. DEV/ PROD PARITY</b> Keep dev, staging, prod as similar as possible	<b>11. LOGS</b> Treat logs as event stream	<b>12. ADMIN PROCESSES</b> Run admin / mgmt tasks as one-off processes

# Observabilidad



# Observabilidad



Prometheus



Grafana

# Conclusiones

## Ventajas

- **Modularidad**
- **Escalabilidad**
- **Versatilidad**
- **Test-eabilidad**
- **Agilidad**

## Desventajas

- **Complejidad**
- **Necesidad de un equipo multidisciplinario**
- **Despliegue / Gestión / Monitoreo**
- **Test-eabilidad**
- **Costo**

# Conclusiones

## Cuando sí...

- Alto (muy alto) flujo de usuarios
- Luego de un profundo análisis
- Se tiene un equipo multidisciplinario

## Cuando no...

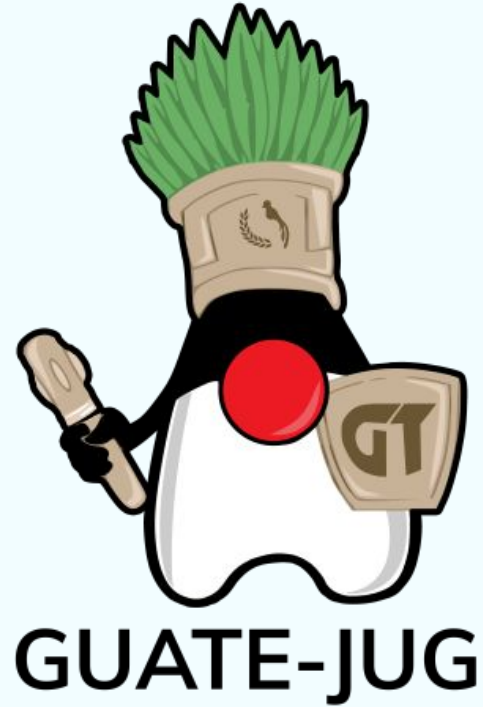
- Pocos usuarios.
- Porque está de moda y quiero ser cool.
- Solo se tienen un par de patojos chispudos.



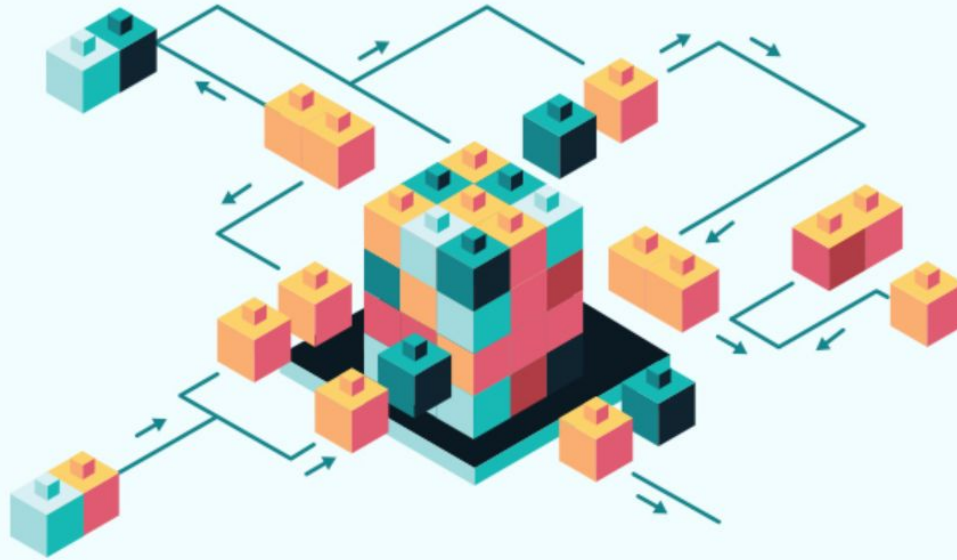
# Bonus



+



# ¿Preguntas?



# Gracias

