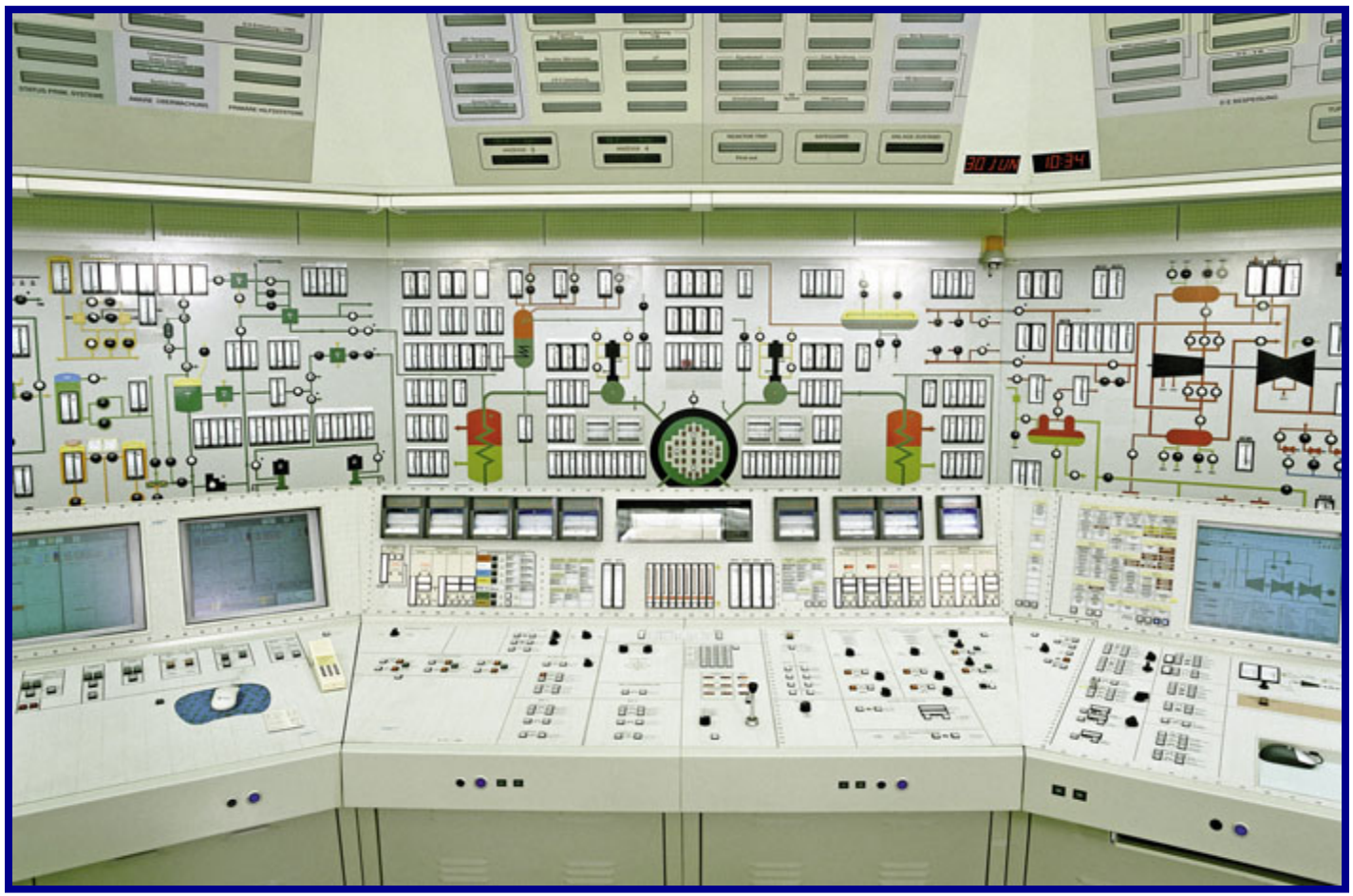
Monitorización y observabilidad



Curso de escalabilidad v2, día 4

Contenido

- Despliegue continuo
- Monitorización
- Observabilidad: granularidad de eventos
- Alertas en producción
- Soporte a producción: guardias

Despliegue continuo

We all test in production.
Some of us have a separate environment to run more tests.

Unknown.

¿Por qué hacer despliegue continuo?

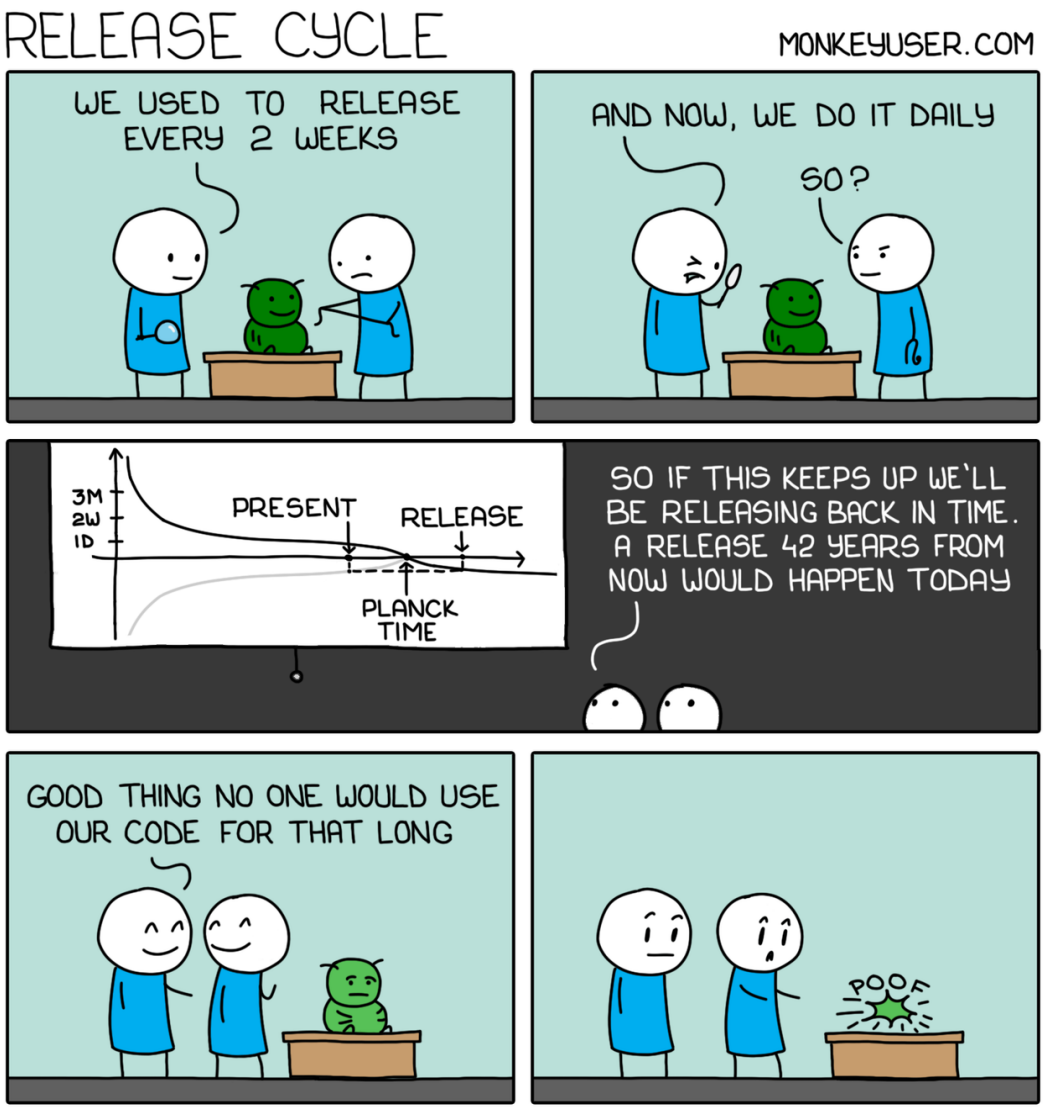
El mayor promotor de **agilidad** conocido

No requiere contenedores (pero ayudan)

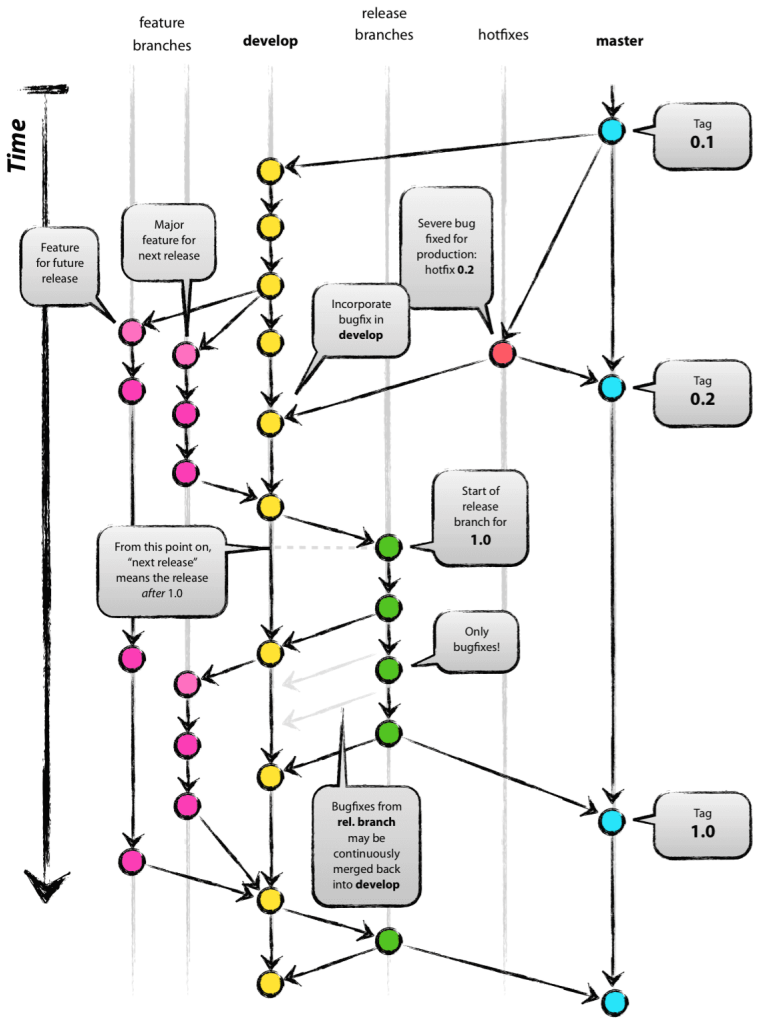
Sí requiere:

- testing
- **monitoring**
- rollback
- disciplina

¿Cómo de continuo?



git-flow



Vincent Driessen: A successful Git branching model

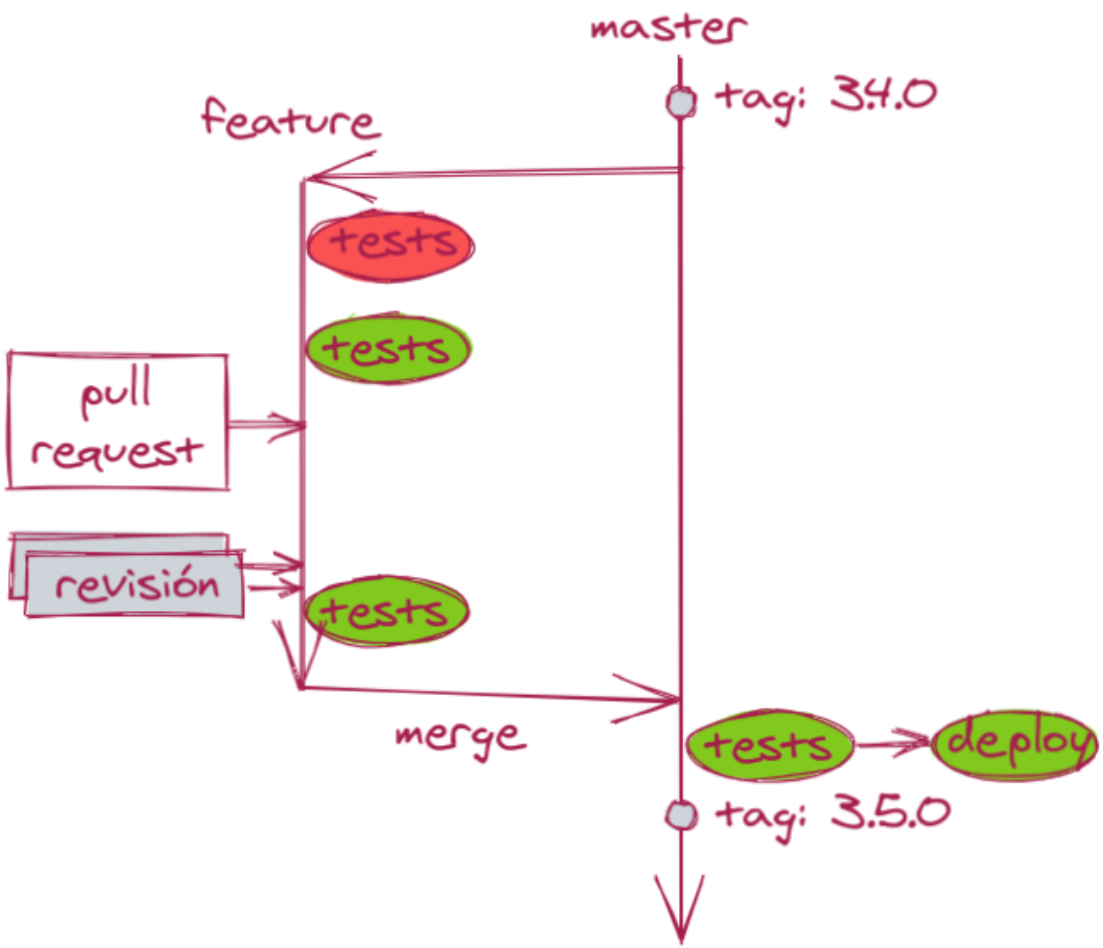
Quemad git-flow

Unas palabras de su creador:

> *If your team is doing continuous delivery of software, I would suggest to adopt a much simpler workflow (like GitHub flow) instead of trying to shoehorn git-flow into your team.*

Vincent Driessen: A successful Git branching model

Metodología TPP (ToPaProd)



Sólo mantenemos **una versión**

Podemos hacer *rollback* a la versión anterior

Compatibilidad hacia atrás extrema

Seguimos SemVer **a rajatabla**

Versión x.y.z

x: versión mayor (**major**)
Rotura de compatibilidad ⇒ cambia x

y: versión menor (**minor**)
Cambio de funcionalidad ⇒ cambia y

z: versión parche (**patch**)
Arreglo de bugs ⇒ cambia z

Ejercicio: cambios compatibles

Nuestra API publica cuatro llamadas:

```
createUser({email, password}) → {userId, email, password}
modifyUser(userId, {email?, password?}) → modified
login(email, password) → {userId, token}
deleteUser(userId) → deleted
```

Queremos añadir el campo **opcional** `username` a `user`

¿Cómo hacemos para no romper la compatibilidad?

Ejercicio +

Ahora queremos hacer que el campo `username` sea **obligatorio**

¿Cómo podemos hacerlo de forma compatible?

¿Cómo afecta a nuestros cuatro servicios?

Ejercicio +

Nos aseguran desde negocio que no se van a borrar usuarios

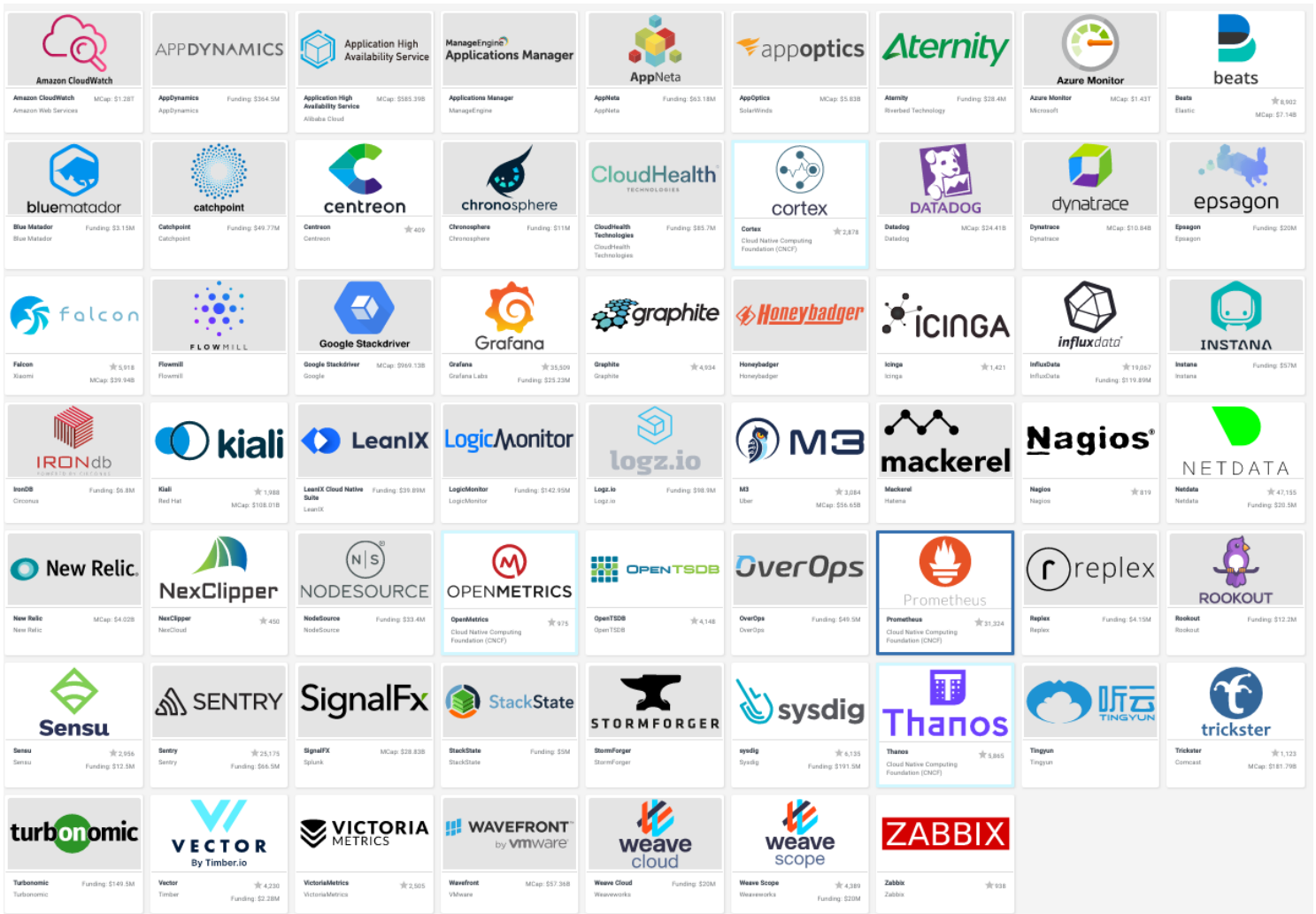
¿Qué hacemos con `deleteUser()`?

Diseña una estrategia de versionado para eliminarlo

Awesome!

Monitorización

Un amplio ecosistema



Fuente

Nagios, Netdata

Nagios: monitorización a bajo nivel

Netdata: Infográfico

Netdata: Demo

Cuatro señales de oro

Latencia: tiempo en responder

Tráfico: peticiones por segundo

Errores: tasa de fallos en el sistema

Saturación: si el sistema está cerca del límite

Fuente: SRE Book

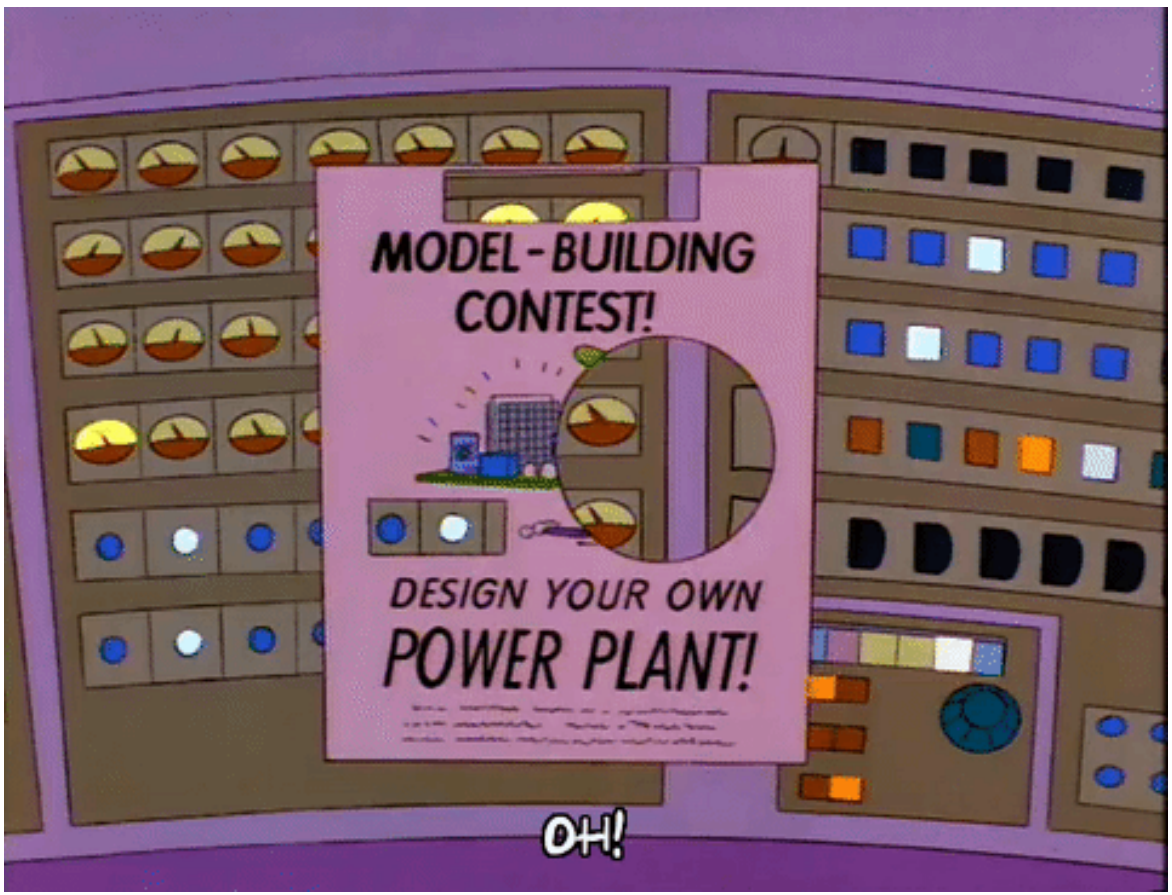
Monitorización reactiva o proactiva

Reactiva:
Almacenamiento de datos
Revisión post-incidente

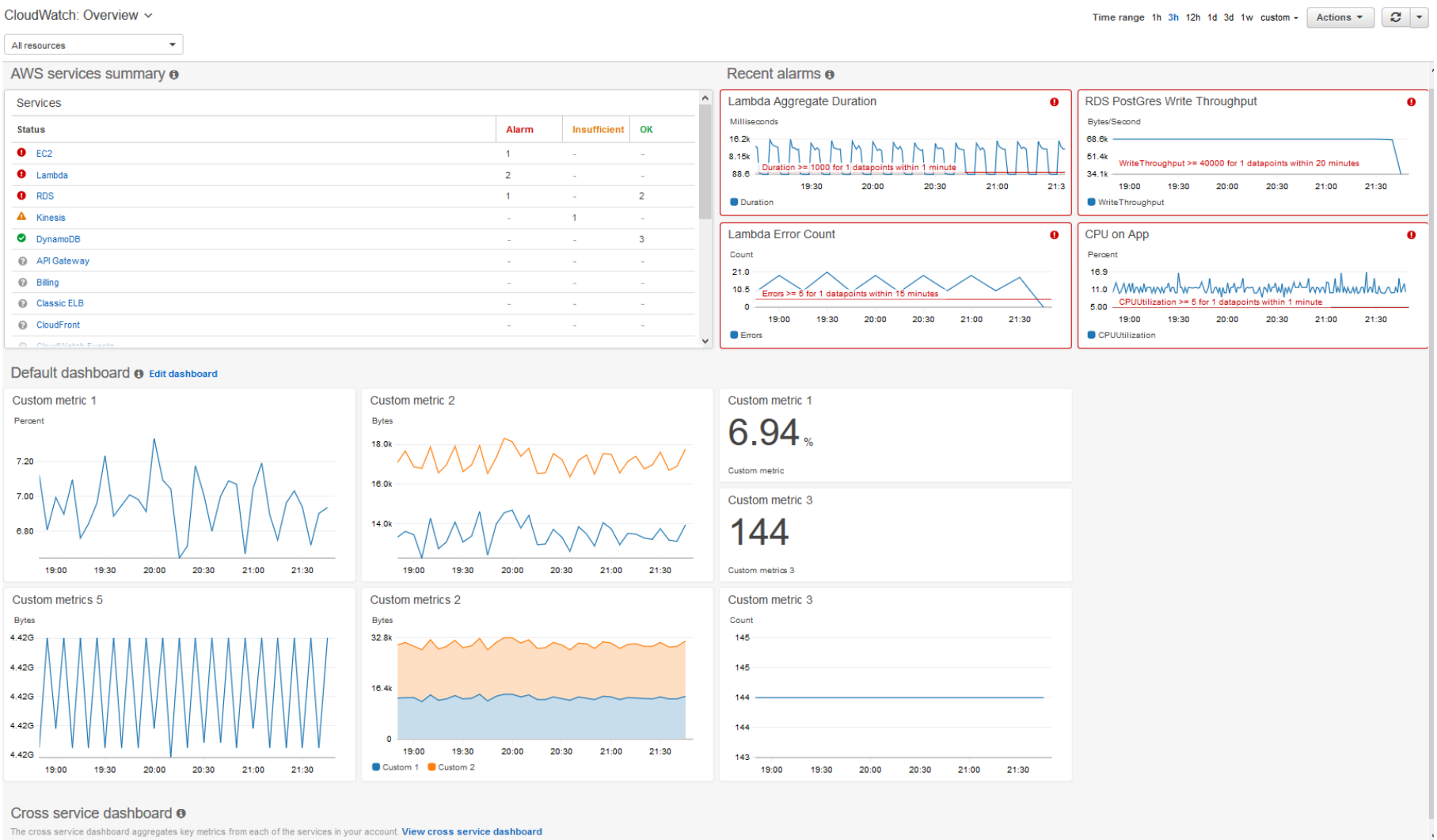
Proactiva:
Los datos peligrosos nos deberían llegar
Alertas

El dashboard

¿Cuadro de mandos?
¿Panel de control?



AWS Cloudwatch



Ejercicio: diseña tu dashboard

Entra en AWS Cloudwatch

Crea un dashboard

Añade la CPU de tus servidores

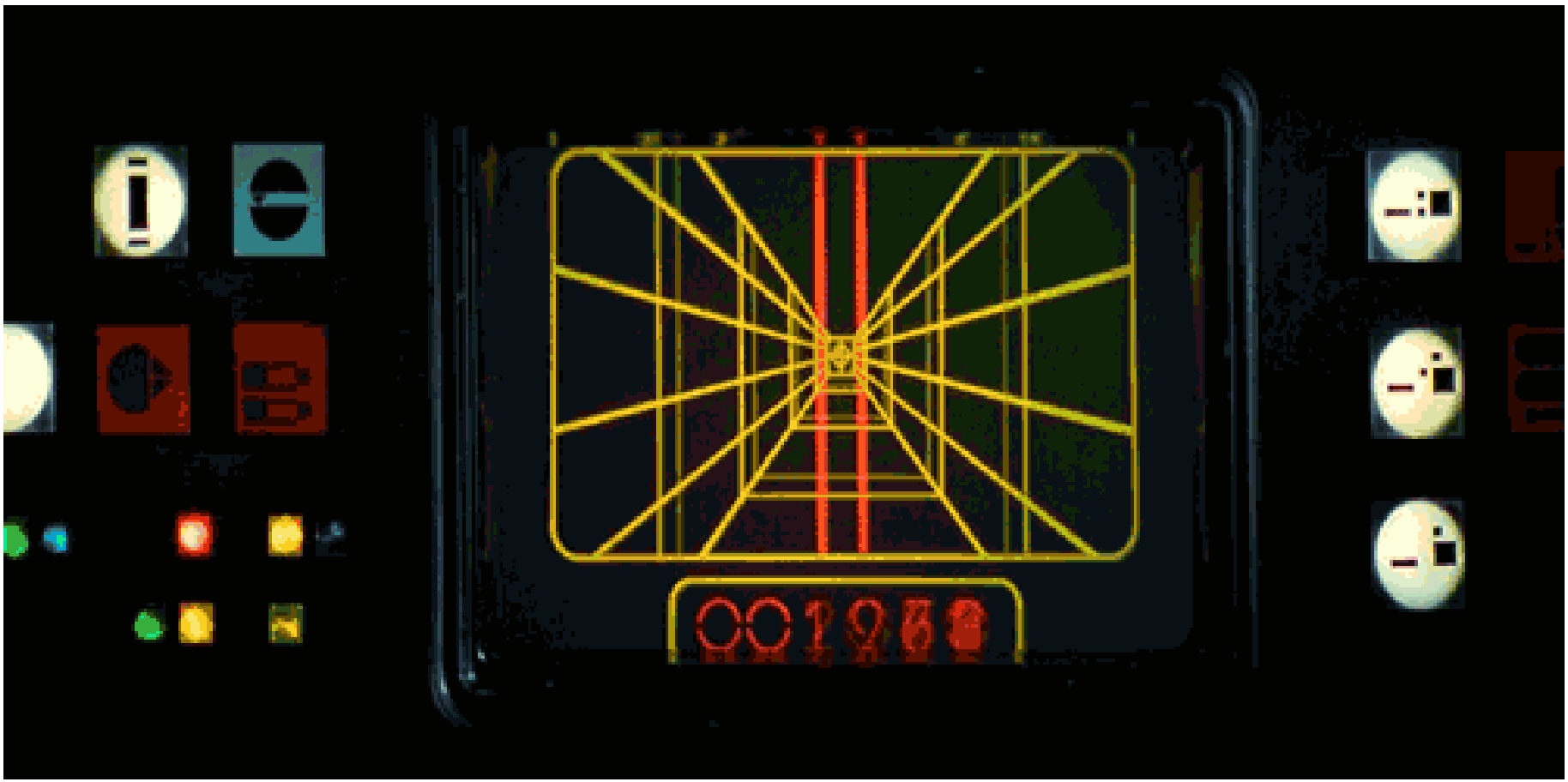
Añade la red de tus servidores

Ejercicio +

Asegúrate de que las métricas se muestran
con resolución de un minuto

Muestra ahora el percentil 99 de CPU medido por horas
durante una semana

Dashboarded!



Prometheus

Base de datos temporal (*time-based*)

Almacena métricas

Se alimenta desde agentes

- Agente en host
- Agente centralizado

Agentes a medida

Grafana

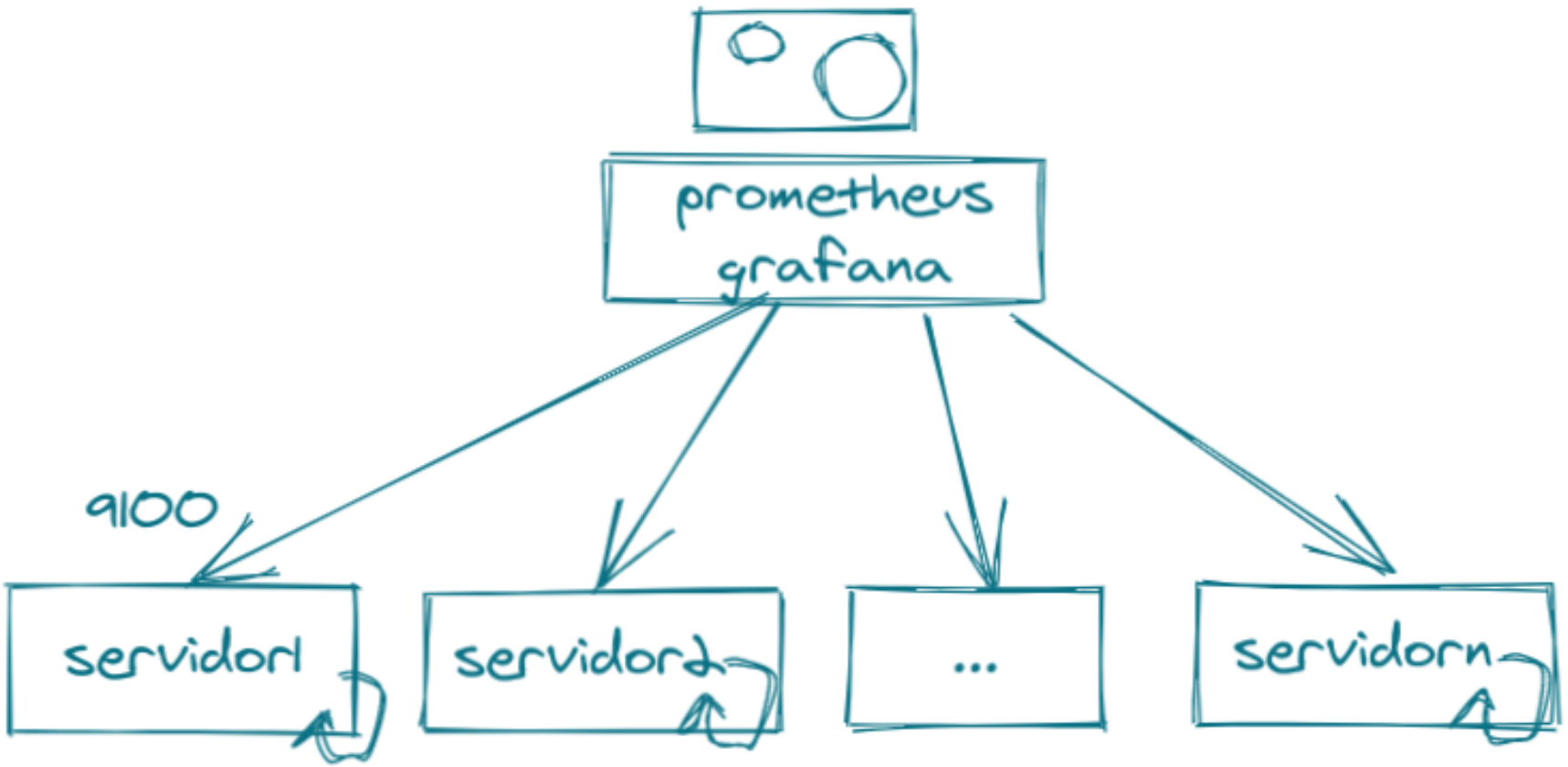
Herramienta de visualización de Prometheus

Dashboards

Notificaciones y alertas

Demo

Prometheus + agentes



Ejercicio: Agente de Prometheus

Añade agente de Prometheus al host

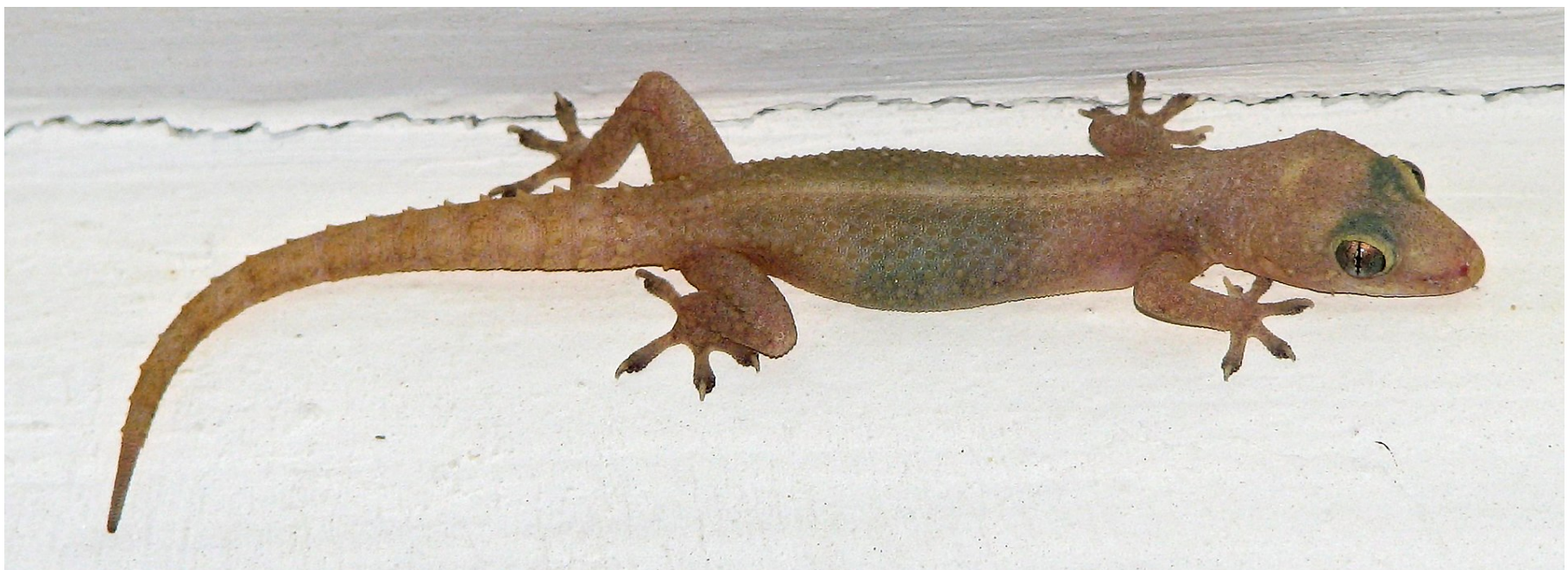
Abre puerto 9100 en el *Security Group*

Pasa la IP a Alfredo por Slack

□

Time traveled!

Observabilidad



Los tres pilares

Logs: eventos en ficheros

Métricas: mediciones de magnitudes

Trazas: unen sistemas diferentes

Fuente: Distributed Systems Observability

¡Abajo los pilares!

Logs: demasiado volumen

Métricas: poca cardinalidad

Trazado: mucho *overhead*

Fuente: Three Pillars with Zero Answers: Rethinking Observability

El camino a la observabilidad

1. Arbitrarily-wide structured raw events
2. Persisting context thru the execution path
3. Without indexes or schemas
4. High-cardinality, high-dimensionality
5. Ordered dimensions for traceability
6. Client-side dynamic sampling
7. Exploratory visual interface
8. In close to real-time

Fuente: So You Want To Build An Observability Tool...

Alertas basadas en observables

No alertan por una métrica

Alertan por una combinación compleja
Basadas en SLOs

Ejemplo: presupuesto de errores se acerca al 90% del objetivo

Ejercicio: usando honeycomb.io

Hacer las demos de play.honeycomb.io

Integrar honeycomb en tu código

Repasar el bubble up

¡Eso, eso, eso!

Observability-driven Development

Abraza el fracaso

Instrumenta mientras caminas

Cierra el bucle

Charity Majors

Alertas en producción

Métricas peligrosas

Demasiada carga

Deterioro del servicio

Peligro para el negocio

Requiere acción

Otras alertas

Observables peligrosos

Basadas en SLOs

Riesgo para negocio

Alertas de negocio

Fatiga de alertas

you should only page on high level end-to-end alerts, the ones which traverse the code paths that make you money and correspond to user pain

Charity Majors: Love (and Alerting) in the Time of Cholera (and Observability)

Tres condiciones

La alerta debe indicar qué está fallando

Debe corresponderse con un problema de usuario

Si una alerta no requiere acción inmediata, ¡bórrala!

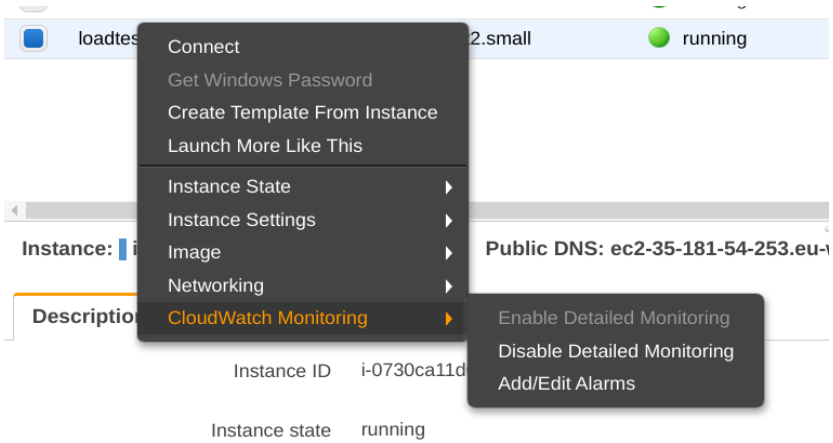
Dos canales

Alertas **inmediatas**
Problema urgente
Busca, llamada de teléfono

Avisos **diferidos**
Problema importante
Se arregla en horas de trabajo

Ejercicio: alerta en Cloudwatch

Asegúrate de que tienes *detailed monitoring* activo en EC2:



Vuelve a AWS Cloudwatch

Añade una alerta si la CPU sube por encima del 90% durante dos minutos

□

Ejercicio +

Cuenta las CPUs de tu servidor:

```
$ cat /proc/cpuinfo
```

Ahora lanza una prueba de estrés:

```
$ sudo apt install stress
$ stress --cpu 1 --timeout 180s
```

Verifica que la CPU sube al 100% en EC2
Verifica tu dashboard en CloudWatch

Comprueba que te llegue la alerta

Triggered!

Soporte a producción

Guardias

What stands between so many decent engineers and greatness is their near illiteracy with production.

Charity Majors

Urgencias

Se **revierten** los cambios

Se arregla o **parchea** el problema

Se deja el desarrollo para horas de trabajo

Deberían ser sólo **incógnitas desconocidas**

Your instincts are usually noble -- you want to do a good job! you care about your users and their experience! you have no other obvious solution! -- but you end up plastering paging alerts on every thing that can or may fail.
You pay for this in lifeblood.

Charity Majors

Arreglo automático

El sistema debería ser **robusto**

Debe ser capaz de responder a errores

Debe **revertir cambios automáticamente**

Ejemplo: **despliegue canario**

Ejercicio: despliegue canario

Revisas tus caídas del último año

¡Casi todas se deben a despliegues fallidos!

¿Cómo podemos reducir el impacto de las caídas?

Ejercicio +

Diseña una estrategia para desplegar sin riesgo

Despliegue parcial (canario)

Rollback automático

Robust!

Arreglo definitivo

Cualquier alerta debería aparecer sólo una vez

No hagas nada nuevo hasta que esté arreglado

Las únicas excusas:

- Un problema realmente esporádico
- Una incógnita desconocida (e insondable)

Ejercicio: Descriptores de fichero

Unix usa descriptores de fichero para los ficheros abiertos

Los descriptores son un recurso finito

¿Sabes predecir cuándo se van a acabar los tuyos?

□

Ejercicio +

Crea un servidor que tarde en responder 10 segundos

Lanza pruebas de carga con concurrencia 100

Intenta que se bloquee el servidor

¿Cuántas peticiones en vuelo tienes?

Ejercicio +

Comprueba cuántos descriptores de fichero tienes

Por usuario y en el sistema

Aumenta los descriptores de usuario y sistema

Verifica que el servidor responda a más peticiones

To infinity and beyond!

Bibliografía

Charity Majors: Yes, I Test in Production (And So Do You)

Google SRE Book: Release Engineering

Google SRE Book: Monitoring Distributed Systems

Charity Majors: So You Want To Build An Observability Tool...

Applying cardiac alarm management techniques to your on-call