otras herramientas pragmáticas y conceptos ágiles

microservices

"un enfoque para desarrollar una única aplicación como una suite de pequeños servicios, cada uno ejecutándose en su propio proceso y comunicándose a través de mecanismos ligeros (generalmente, una API de recursos HTTP)"

- se construyen en torno a **funcionalidad**
- se despliegan de manera **independiente**

están construidos en distintas tecnologías y utilizan diferentes tecnologías de almacenamiento

fuente: Fowler

sistemas monolíticos

cliente + backend + base de datos

cualquier cambio en el sistema implica construir y desplegar una nueva versión del backend

múltiples servicios por host

- más sencillo puramente desde el punto de vista de mantenimiento de host
- menos overhead con respecto a la virtualización
- dificultades de monitorización: ¿la CPU de qué servicio?
- consumición de recursos por parte de un único servicio

múltiples servicios por host

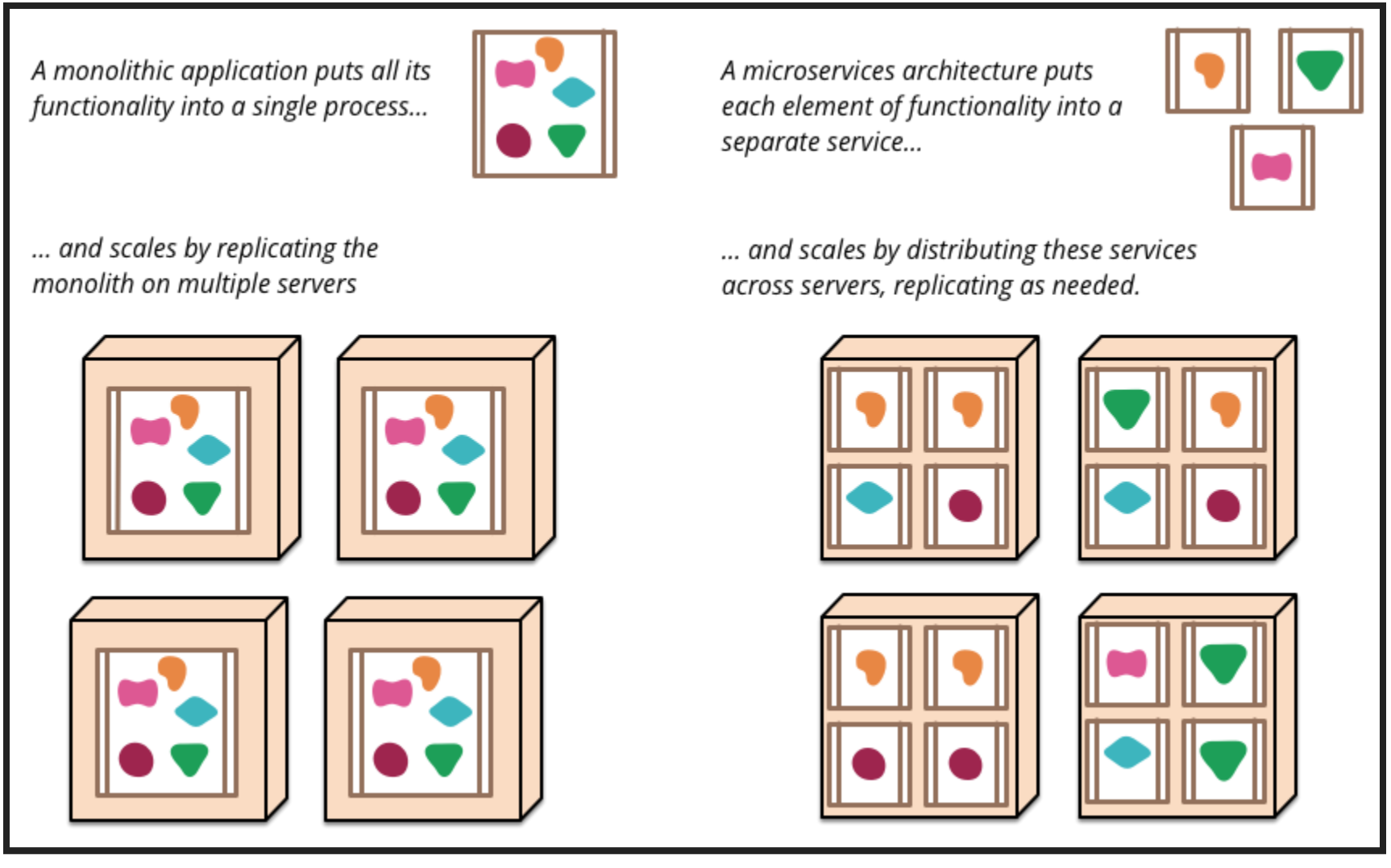
- riesgo de que un despliegue afecte al resto de servicios
- posibilidad de dependencias contradictorias
- inhibe autonomía de equipos: se acaba necesitando gestión centralizada
- imposibilidad de despliegues basados en imágenes o servidores inmutables
- no todos los servicios tienen por qué escalar igual

servidor compartido

idea: facilidad de mantenimiento, herramientas de monitorización por servicio, reducción de overhead de JVM

- restricción en la elección técnológica: puede limitar opciones para implementar el servicio, así como de automatización y mantenimiento de sistemas
- in-memory session state: problemas de escalado
- mayor dificultad de análisis de uso de recursos y threads
- coste de licencias y de características no usadas

alternativas a considerar: servicios auto-contenidos



fuente: Fowler

servicios como componentes

- componentes: unidad de software independientemente reemplazable y actualizable
- biblioteca: componentes enlazados en un programa y llamados usando llamadas a funciones en memoria
- servicios: componentes fuera del proceso que se comunican con mecanismos como peticiones a servicios web o RPC

utilizar servicios como componentes permite poder desplegarlos de manera independiente

un servicio por host

monitorización y gestión de contingencias independiente

deja de existir un único punto de fallo para todo el sistema global

escalado y "securización" independientes

posibilidad de despliegues basados en imágenes y servidores inmutables

¿coste de licencias?: Tomcat

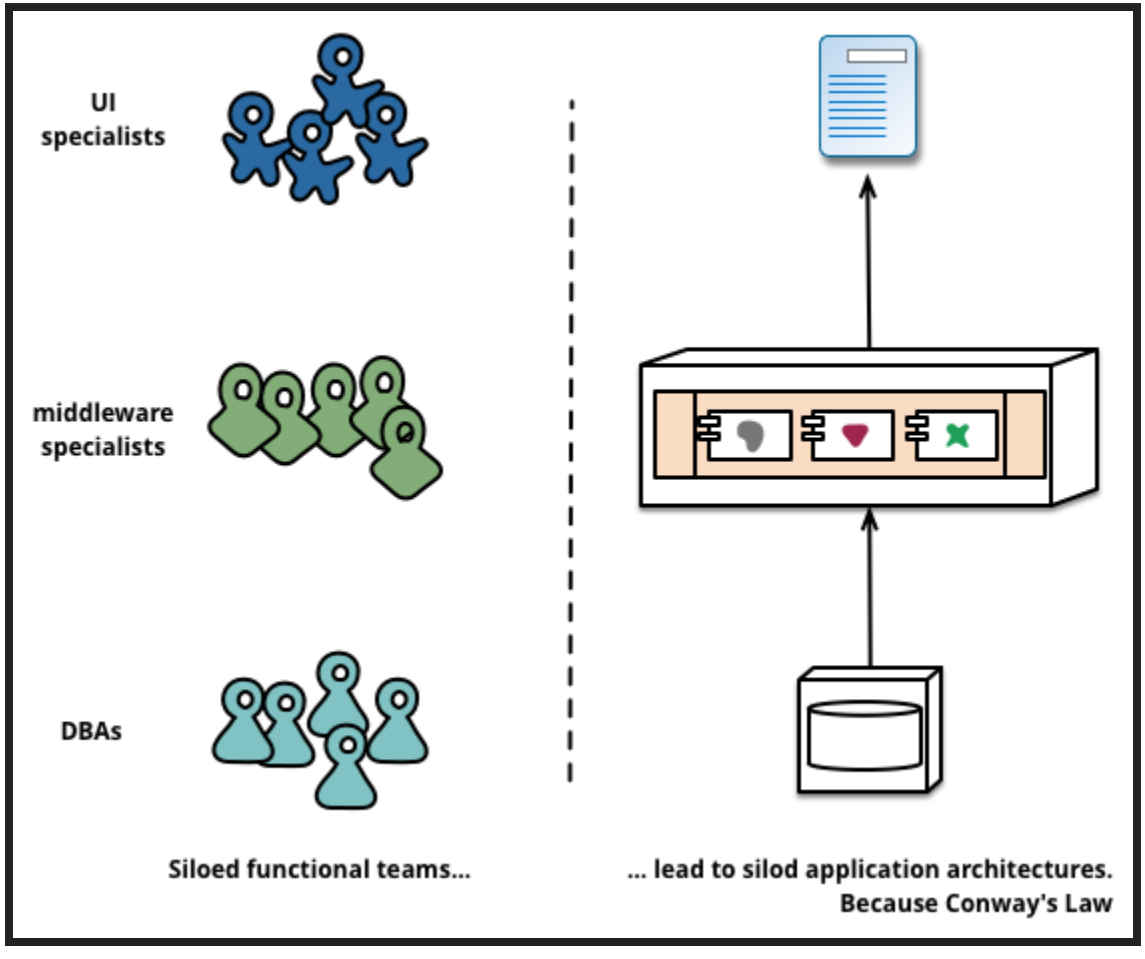
más elementos a mantener...

organización según el negocio

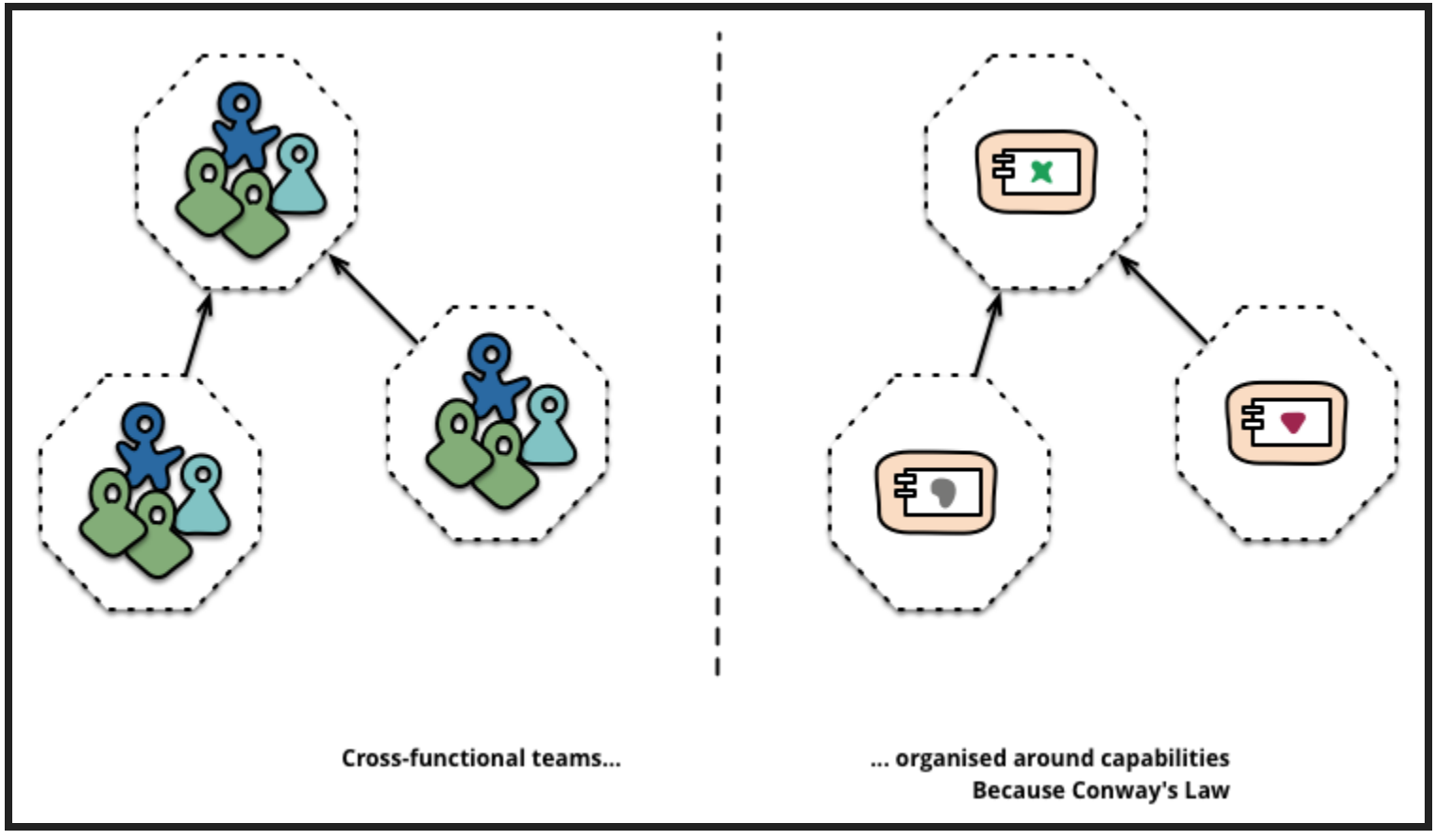
equipos multi-funcionales con las habilidades necesarias para el desarrollo, la experiencia del usuario, la gestión del proyecto, las bases de datos...

evitar la Ley de Conway:

"cualquier organización que diseña un sistema (definido de forma amplia) producirá un diseño cuya estructura es una copia de la estructura de comunicación de la organización"



fuente: Fowler



fuente: Fowler

productos, no proyectos

modelo de proyecto: se entrega el software cuando se considera completo, delegando la responsabilidad al equipo de mantenimiento. El equipo de desarrollo se desmantela.

modelo de producto: el equipo de desarrollo asume plena responsabilidad también cuando el software está en producción. Esto les lleva a un contacto diario con el software y sus usuarios.

el software se convierte en una relación viva

comunicación sencilla

frente a los ESB, que incluyen sofisticados sistemas de enrutado, "orquestación", transformación, reglas de negocio, etc...

microservicios desacoplados y coherentes, usando HTTP

- end-point inteligente
- "tuberías tontas"

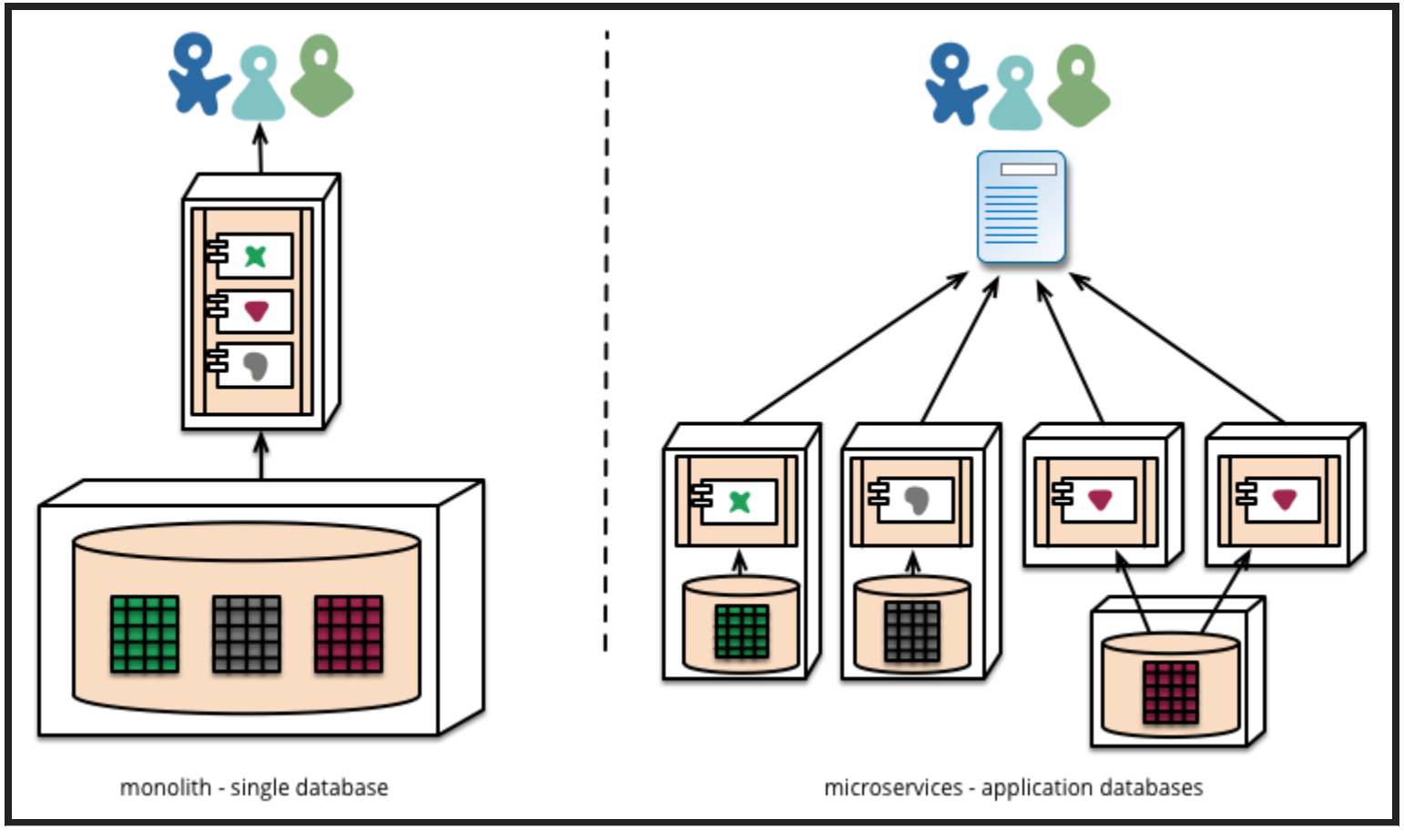
gobernanza descentralizada

para cada problema, su solución: **no todo es un clavo y no todo es un martillo**

en vez de un catálogo centralizado de estándares en una tecnología única, un **catálogo de herramientas** útiles para que otros desarrolladores puedan resolver problemas similares

ejemplos: Netflix, Amazon...

gestión de datos descentralizada



fuente: Fowler

otras

- entrega continua, integración continua
- ejecución automática de tests y despliegue
- diseño tolerante a fallos
- diseño evolutivo

¿un servidor para cada microservicio?

no necesariamente...

usar servidores empotrados permite una mayor automatización, despliegues más rápidos y menor esfuerzo de mantenimiento

spring boot, dropwizard:

ejecución de un jar sin servidor de aplicaciones

contras

impacto en **rendimiento** debido al overhead HTTP, de red y de serialización

heterogeneidad e **interacciones complejas**

gestión de dependencias

necesidad de mantener un registro de servicios

mayor **complejidad de mantenimiento**: paliable con gestión de la configuración

imágenes como artefactos

abstracción del servicio en una máquina virtual

servidores inmutables

aseguramos que nunca se hacen cambios en servidores en ejecución

cualquier cambio se realiza a través de una pipeline para crear una nueva VM

!!hasta podemos deshabilitar el acceso SSH!!

fuente: Building microservices, Sam Newman

devops

---

participación conjunta de equipos de operaciones (sistemas) y desarrollo en todo el ciclo de vida del servicio, desde su concepción y diseño, pasando por el proceso de desarrollo y hasta el mantenimiento del mismo

comparten los **mismos valores** que el manifiesto ágil

el "horror" de sistemas

esta aplicación no se desplegará en nuestra infraestructura porque... {está obsoleta, no tenemos capacidad suficiente, no damos soporte a esa versión}

---

la arquitectura de esta aplicación no cumple con nuestro modelo de {red, almacenamiento, seguridad}

principios

- cultura, simplicidad, automatización, mejora continua

infrastructure as code

escribir código para gestionar configuración y provisión automática de infraestructura y despliegues

no se trata sólo de scripts

- control de versiones, testing, patrones de diseño...

herramientas

- las propias del desarrollo ágil: scrum, kanban
- propias de operaciones: virtualización, cloud
- de despliegue: jenkins, travis
- de gestión de configuración: puppet, chef, ansible
- de uso de contenedores: docker, vagrant, AWS, openstack

puppet

un gestor de la configuración

¿para qué?

para definir el estado de nuestra infraestructura de sistemas y automáticamente forzar dicho estado

automatización del proceso de entrega de software

- provisionamiento de servidores físicos y virtuales
- orquestación
- reporting

puppet

pros

libera **tiempo** en los proyectos

asegura la consistencia, fiabilidad y estabilidad del entorno

facilita una **colaboración y comunicación** estrecha entre administradores de sistemas y desarrolladores

ayuda a **entregar valor**

puppet

contras

hay que conocer su lenguaje de configuración (programación)

problemas de escalado con decenas de miles de nodos

rápida evolución

puppet

cómo funciona

definir

simular

obligar

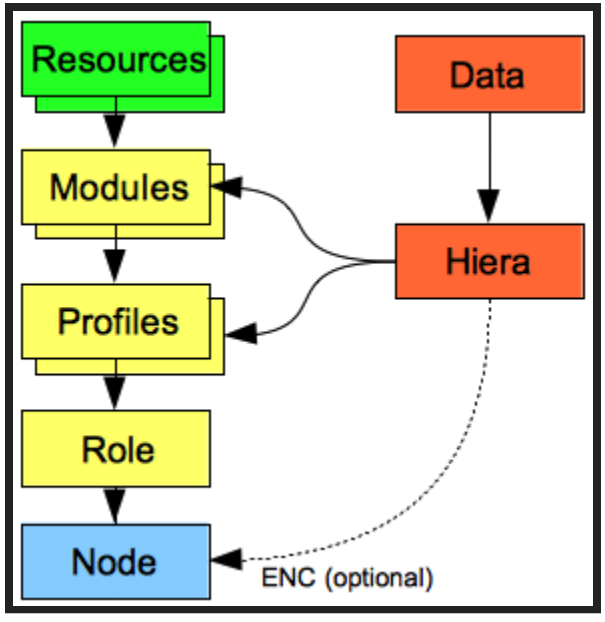
informar

puppet

¿quién lo usa?

puppet

patrón rol-perfil-módulo



puppet

en el ICANE...

patrón rol/perfil

módulos de la forja, propios y de terceros (git) gestionados por `librarian-puppet`

`/etc/puppet/` en su propio repositorio git

clasificación de nodos a partir de hechos (sin ENC)

separación entre lógica (módulos) y datos en yaml (hiera)

puppet

role

```
class role::appserver inherits role {
  notify {"Applying role::appserver": }

  contain profile::rhel
  contain profile::jdk
  contain profile::apache_tomcat

  Class['profile::rhel'] -> Class['profile::jdk']
   -> Class['profile::apache_tomcat']
}
```

puppet

profile

```
class profile::rhel {
  notify {"Applying profile::rhel": }
  if str2bool("$firstdeploy") {
    class {'profile::remove_groups':
      stage => first,
    }
    contain profile::epel_install
    contain profile::packages_install
    Class['profile::epel_install'] ->
      Class['profile::packages_install']
    file {"/etc/facter/facts.d/firstdeploy.yaml":
      ensure => present,
      content => "---\nfirstdeploy: \"false\"\n"
    }
   }
  contain profile::inittab_install
}
```

puppet

datos

```
    ---
groups_dev_group:
    gcm9998:
        ensure: present
        gid: 1003
    sas9999:
        ensure: present
        gid: 1004

users_dev:
    gcm9998:
        ensure: present
        uid: 1003
        gid: 1003
        password: '$6$N.L0QnbP$96GcSdfc.CLR46DPwsmMladmCMpdpj4MgabgBkzis
        shell: '/bin/zsh'
        groups:
          - tomcat6
```

algunos datos...

DB on demand (CERN)

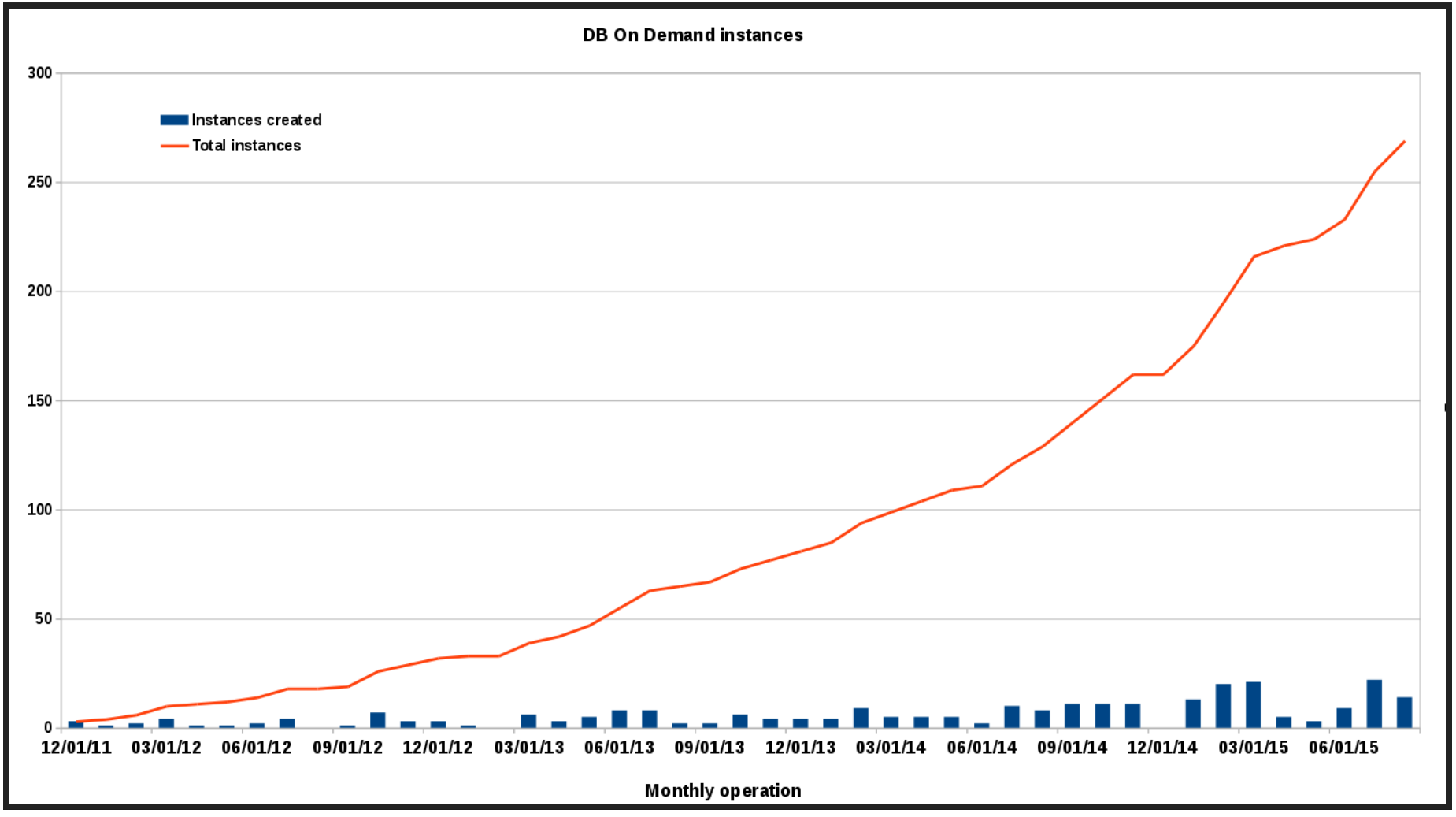
255 instancias de DB (8 Oracle 11g, 30 PostgreSQL, resto MySQL)

~ 10 TB de datos

3 técnicos FTE

2 bugs de MySQL en 4 años

provisión automática + 1-click backup, restore y upgrade



cortesía de Ignacio Coterillo Coz

vagrant

creación y configuración de entornos de desarrollo ligeros, reproducibles y portables

un wrapper sobre virtualbox o vmware

un archivo de configuración define la máquina, el software a instalar y la forma de acceso

adios a la frase "en mi máquina funciona"

vagrant

```
# -*- mode: ruby -*-
# vi: set ft=ruby :
# Entorno de pruebas Liferay
Vagrant.configure(2) do |config|
  config.vm.box = "chef/centos-6.5"
  config.vm.network "forwarded_port", guest: 80, host: 8000
  config.vm.network "forwarded_port", guest:3306, host: 3306
  config.vm.network "forwarded_port", guest: 9019, host: 9019
  config.vm.network "private_network", ip: "192.168.33.10",
              :mac => "00505699451C"
  config.vm.provider "virtualbox" do |vb|
    vb.memory = "4096"
  end
end
```

vagrant

repositorios de plantillas estándar: vagrantbox

¿cómo funciona?

```
vagrant init
vagrant up
vagrant ssh
```

el poder del texto plano

se puede almacenar en un VCS, como si fuera código fuente

de hecho, **es código fuente**: a partir de md se generan artefactos (PDF)

existen servidores de contenido estático basados en markdown:

- jekyll para blogs y websites estáticos
- gollum para wikis basadas en VCS

podemos ser muy productivos con texto plano

markdown

propósito: fácil de escribir y fácil de leer

texto plano

los signos de puntuación significan lo que parecen

Ejemplo: esta presentación

para generar un PDF maquetado con latex a partir de markdown:

pandoc

plantUML

propósito: definir diagramas UML en texto plano

los artefactos (PNG) se generan automáticamente

varios tipos de diagramas: clase, secuencia, estado, caso de uso...

dispone de un módulo para diseñar GUIs (*wireframing*): salt

integrable con amplia variedad de editores (vi, eclipse...)

implementado en java: **es un jar**

herramientas prometedoras

rmtoo: captura de requisitos en texto plano con generación de artefactos:

- diagramas de dependencias
- listas de requisitos con prioridades y estimación de esfuerzo

taskwarrior: gestión del tiempo desde la línea de comandos y en texto plano

```
$ task add Preparar borrador del informe due:friday
```

herramientas de shell

tmux/wemux: multiplexador de terminales linux/cygwin

permiten compartir una sesión de consola remotamente con ssh

solución remota para el pair programming

zsh: shell moderna con autocompletado, integración con git, atajos de sistema...

vi: editor de texto en modo consola ultra-productivo... una vez que se aprende

¿os atrevéis?

Acceso al repositorio del curso

Miguel Expósito Martín