t3chfest

code URJC

Testeando aplicaciones Kubernetes: escalabilidad y tolerancia a fallos

Micael Gallego

Leganes - 15 Marzo 2019

Quién soy?

Micael Gallego

Developer

Profesor Universitario /
Formador & Consultor

@micael_gallego


@micael_gallego


micael.gallego@urjc.es


@micaelgallego

code URJC

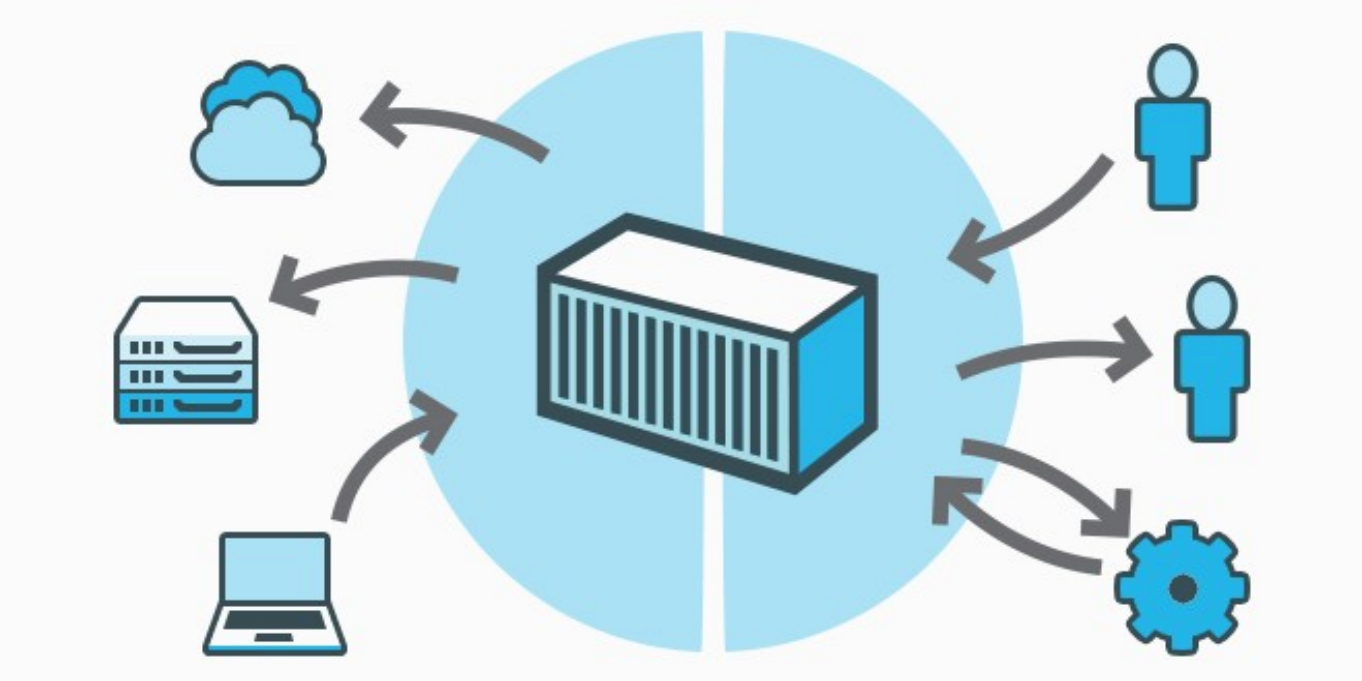
Consultancy / Training

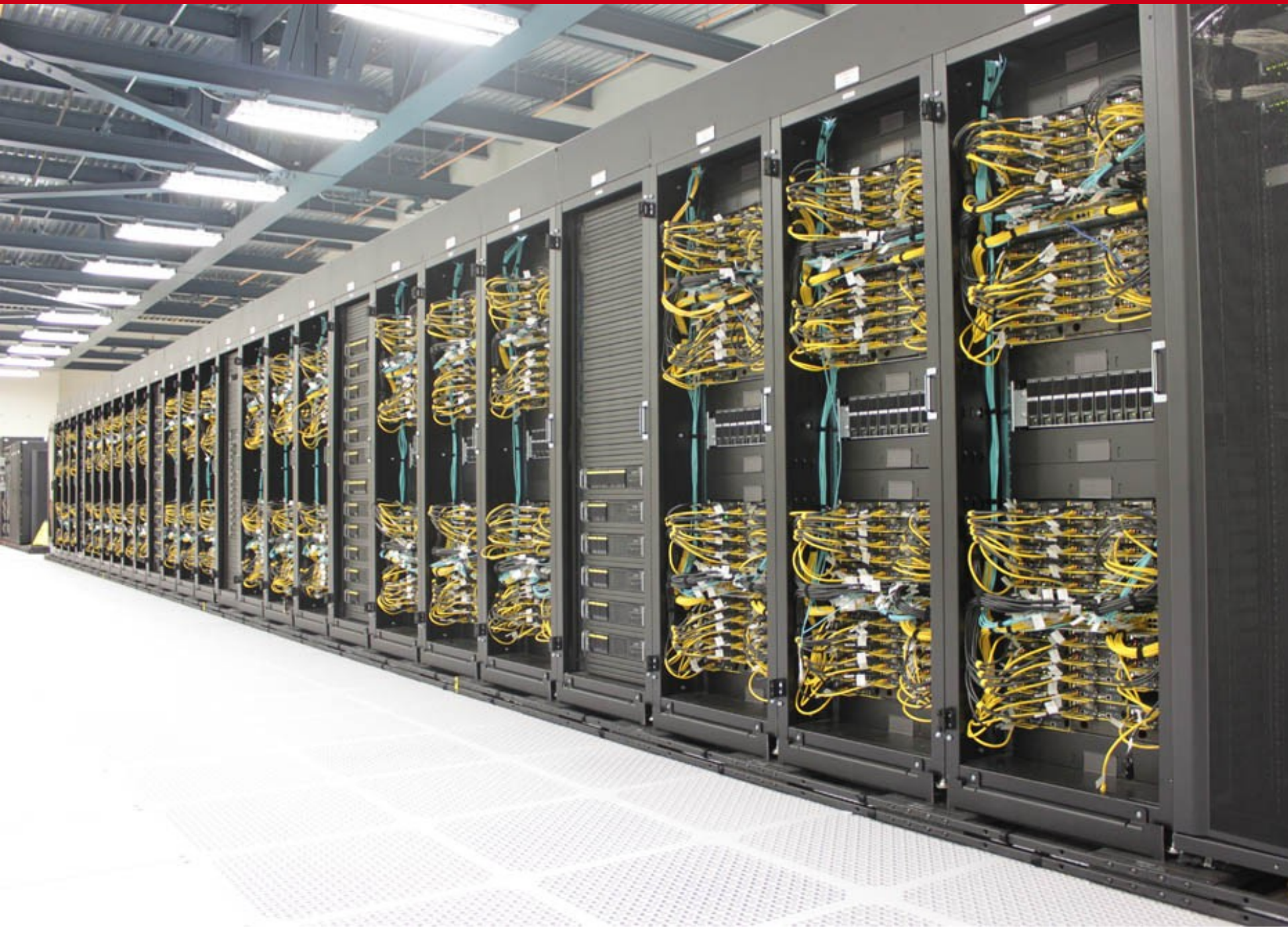
Cloud Computing
Web Technologies
Extreme Programming
Testing / Git / Jenkins
Software Architecture
Concurrent Programming

Universidad
Rey Juan Carlos

http://codeurjc.es

Docker





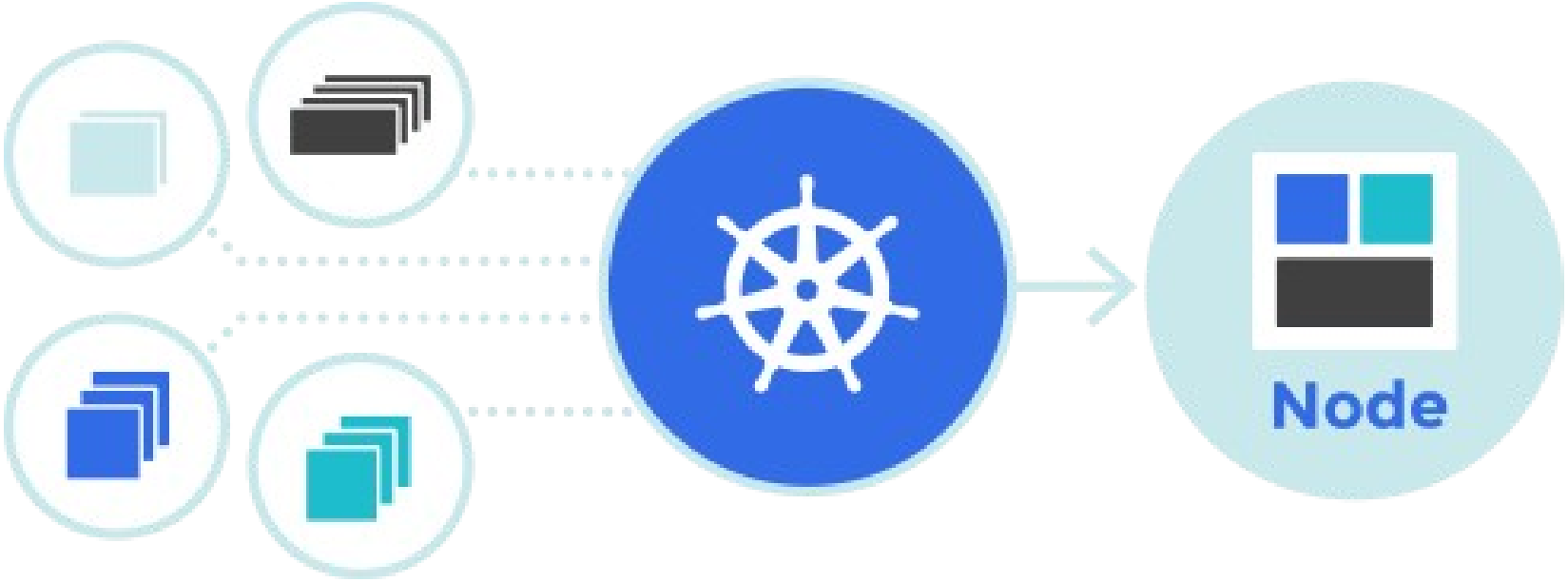
Servicios en producción

- Asistencia en el **despliegue** y en la **actualización**
- **Reinicio** si el servicio finaliza o no responde a las peticiones (***resiliencia***)
- Uso de más recursos hardware la **carga aumenta** (***escalabilidad***)
- **Aprovechamiento** de recursos hardware compartiendo pero sin interferencias (***multitenancy***)
- **Monitorización**

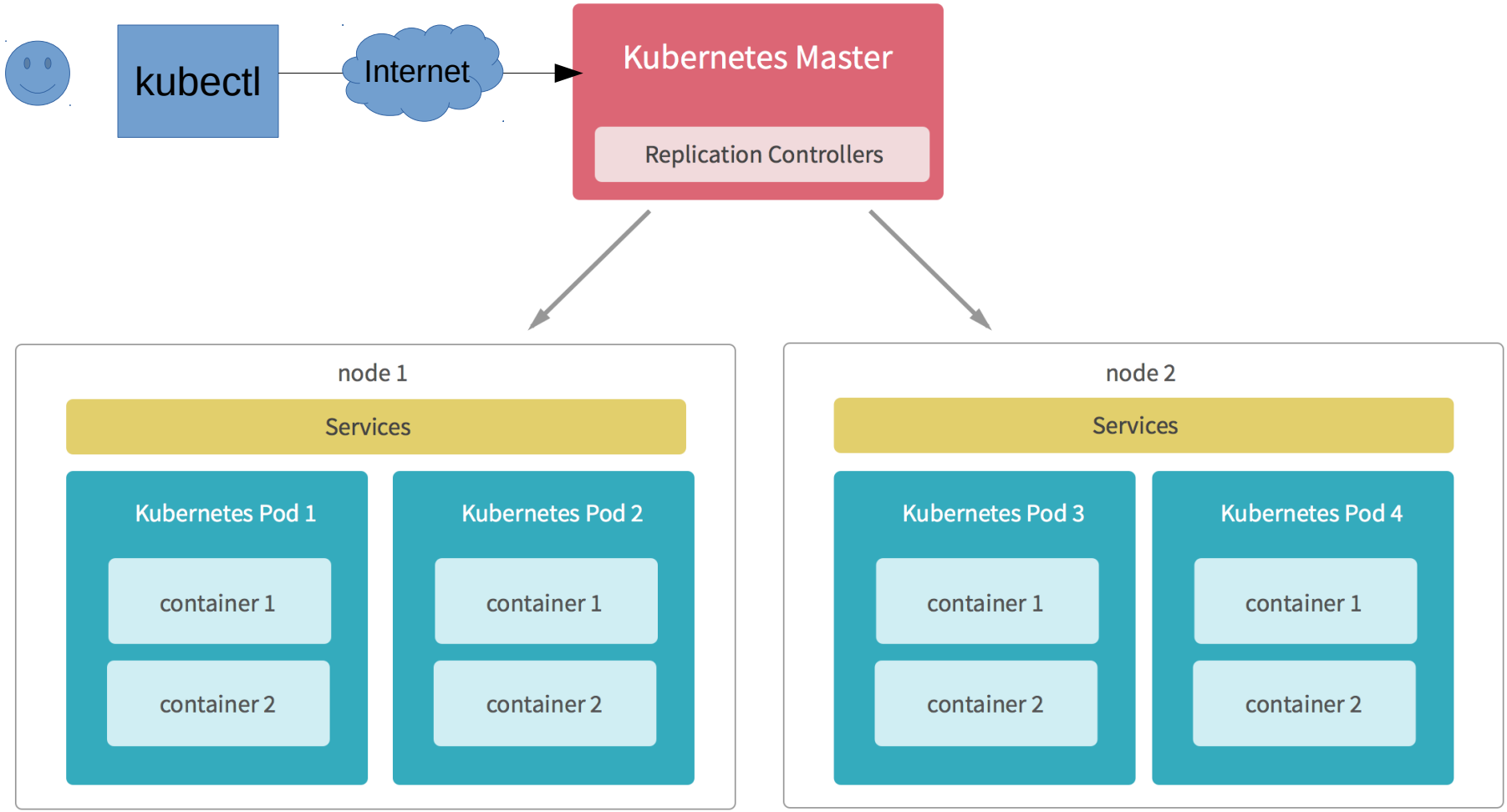
code URJC

kubernetes

Kubernetes



Kubernetes


kubectl
Internet
Kubernetes Master
Replication Controllers
node 1
Services
Kubernetes Pod 1
container 1
container 2
Kubernetes Pod 2
container 1
container 2
node 2
Services
Kubernetes Pod 3
container 1
container 2
Kubernetes Pod 4
container 1
container 2

Kubernetes en producción

https://cloud.google.com/kubernetes-engine/

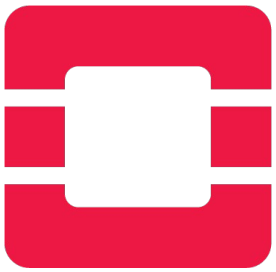
https://www.openshift.com/learn/topics/kubernetes/

https://aws.amazon.com/es/eks/

https://azure.microsoft.com/services/container-service/

code URJC

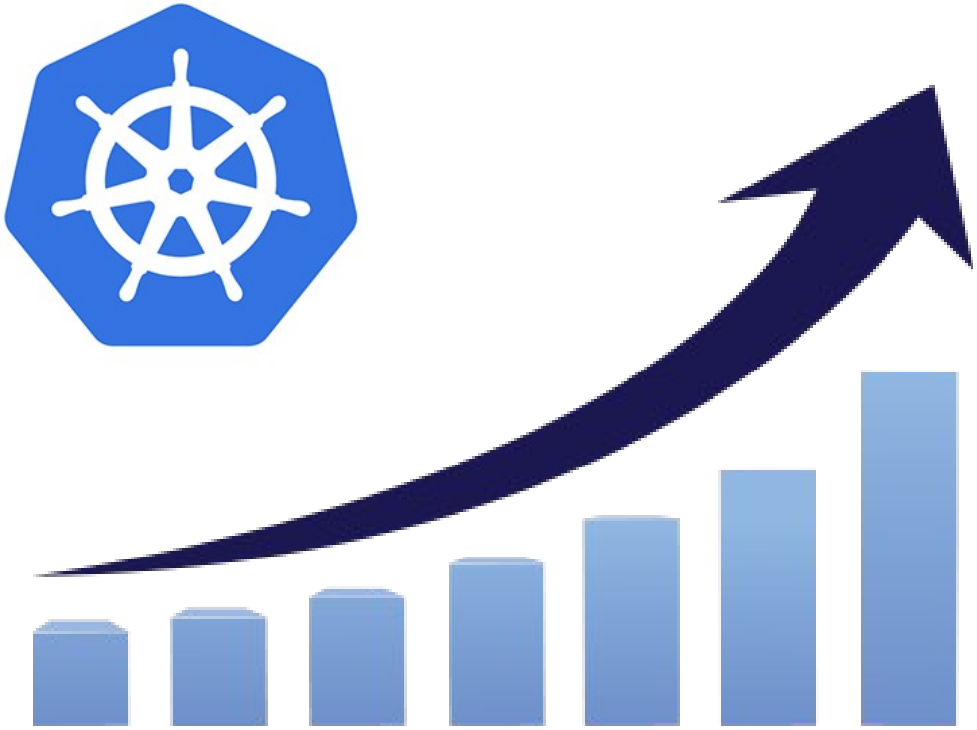
Kubernetes en producción



openstack®

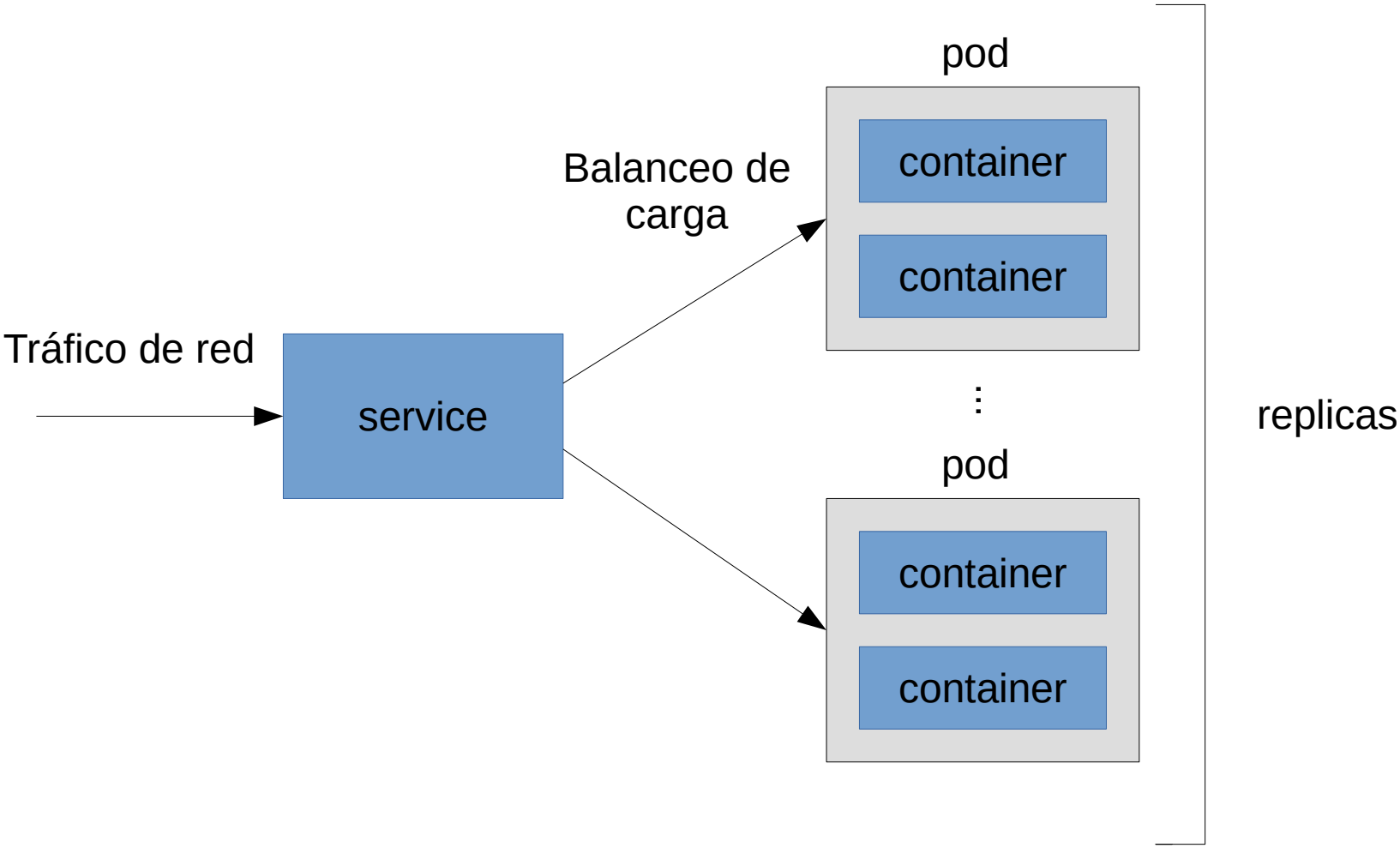
vmware®

@micael_gallego



Escalabilidad en Kubernetes

Escalabilidad



Escalabilidad

- Si un pod no es capaz de atender el tráfico con calidad de servicio, podemos **crear más réplicas de ese pod**
- La carga se **distribuye** entre ellos
- Cada pod puede ejecutarse en **cualquier nodo del cluster**
- Los nodos se pueden **añadir** bajo demanda

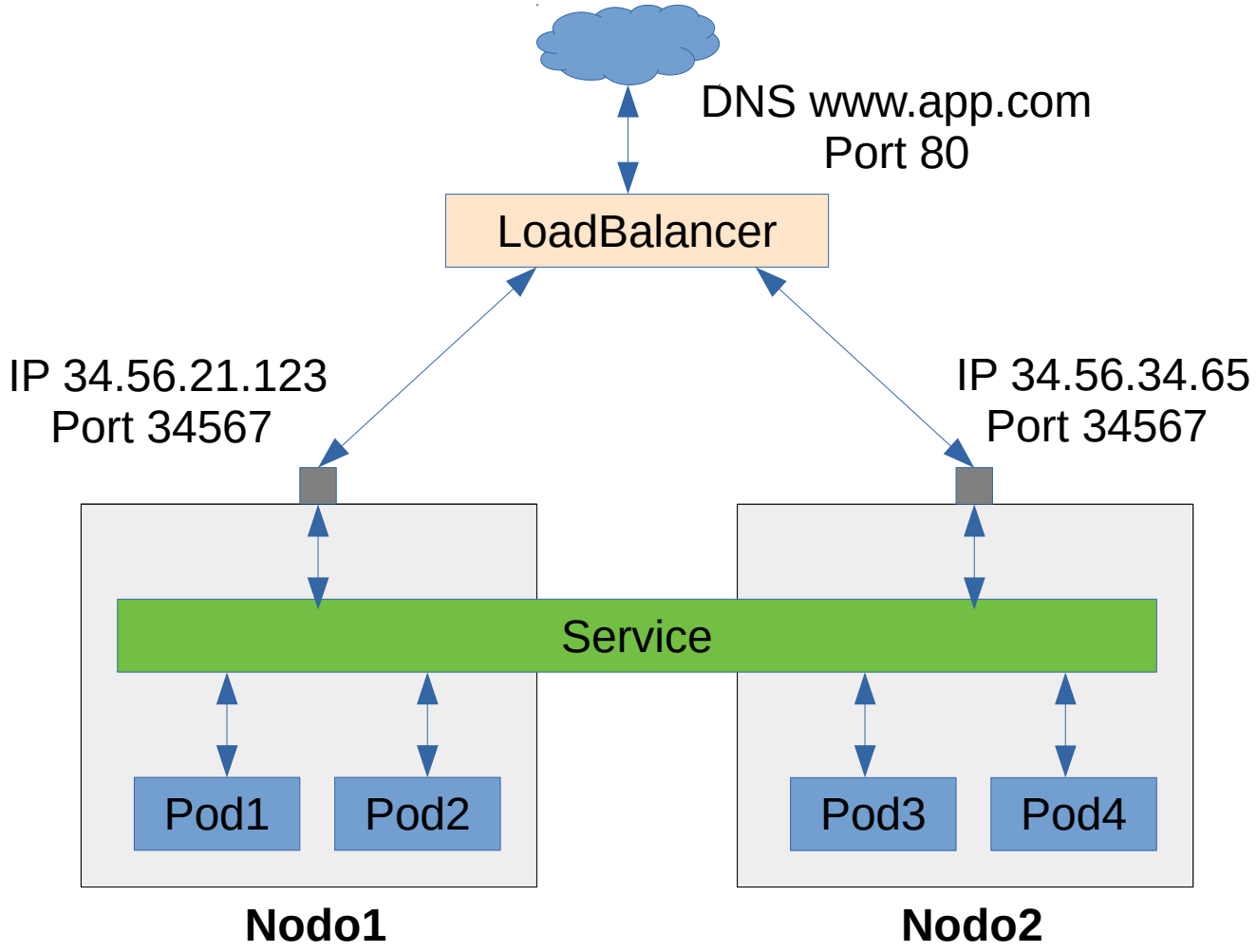
Reparto de carga

Proxy-mode

- **userspace**
  - Round robin
- **iptables**
  - Random
- **ipvs**
  - Round robin
  - Least connection
  - Destination hashing
  - Source hashing
  - Shortest expected delay
  - Never queue

https://kubernetes.io/docs/concepts/services-networking/

Reparto de carga



Demo time

Web gatitos con una réplica y con 2 réplicas

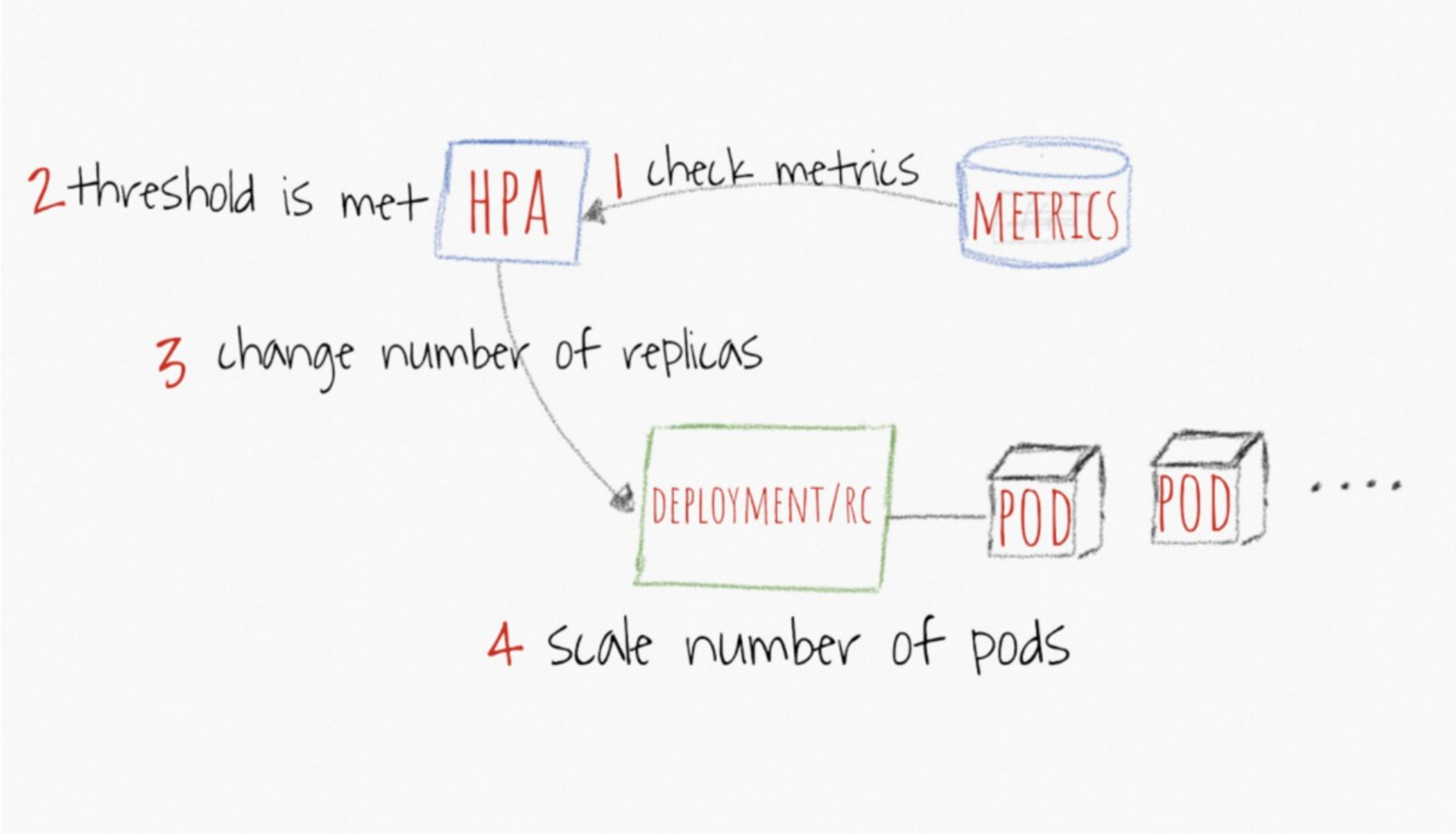
```
$ kubectl create -f webgatos/webgatos.yml

$ minikube service webgatos-service

$ kubectl get pods

$ kubectl scale deployment/webgatos-deploy --replicas=2

$ kubectl get pods
```

Autoescalado

- **Horizontal Pod Autoscaler**
  - El número de **réplicas** puede cambiar de forma **automática** en base a alguna **métrica de carga** objetivo
  - Aumenta réplicas si se sobrepasa la métrica
  - Reduce réplicas si se está por debajo

https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/
https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/

Autoescalado



https://medium.com/magalix/kubernetes-autoscaling-101-cluster-autoscaler-horizontal-pod-autoscaler-and-vertical-pod-2a441d9ad231

Autoescalado

```
apiVersion: apps/v1
kind: Deployment
...
spec:
  ...
  template:
    ...
    spec:
      containers:
      - name: hpa-example
        image: codeurjc/web:v1
        resources:
          limits:
            cpu: 200m
            memory: 128Mi
          requests:
            cpu: 100m
            memory: 64Mi
```

```
apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
...
spec:
  ...
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      targetAverageUtilization: 10
```

Demo time

Web raíz cuadrada con autoscaling

```
$ kubectl create -f hpa/deployment.yaml

$ kubectl create -f hpa/hpa-autoscaling.yaml

$ watch kubectl get pods

$ kubectl run load-generator --generator=run-pod/v1 \
   -it --image=busybox:1.30 /bin/sh

# while true; do wget -q -O- http://php-apache; done
```

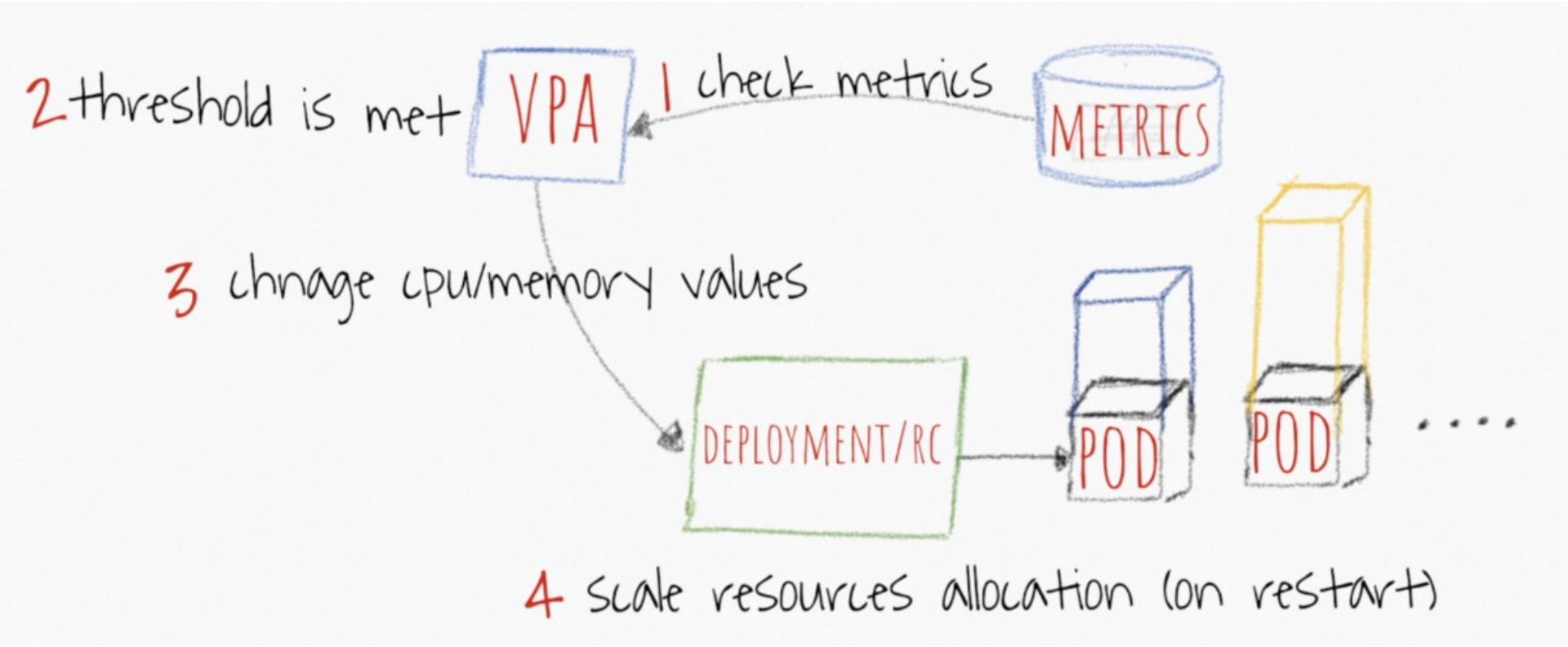
Autoescalado

- **Vertical Pod Autoscaler**
  - Ajusta los recursos solicitados de un pod en base a los recursos usados realmente
  - Para aplicar los nuevos recursos, **reinicia el pod**
  - Los límites máximos no cambian
  - No es compatible con HPA
  - **En desarrollo!!!**

https://cloud.google.com/kubernetes-engine/docs/concepts/verticalpodautoscaler

https://medium.com/magalix/kubernetes-autoscaling-101-cluster-autoscaler-horizontal-pod-autoscaler-and-vertical-pod-2a441d9ad231

Autoescalado



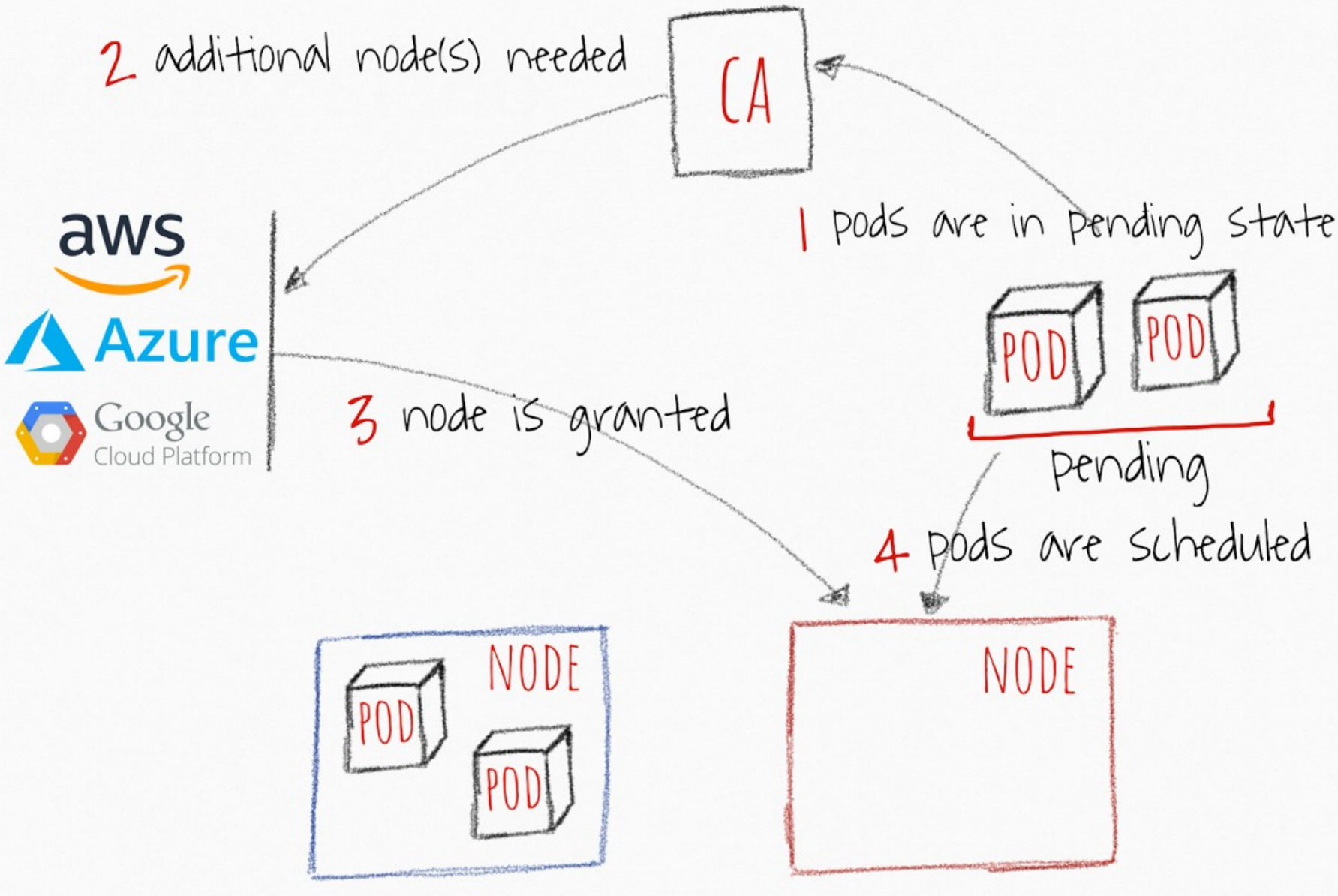
https://medium.com/magalix/kubernetes-autoscaling-101-cluster-autoscaler-horizontal-pod-autoscaler-and-vertical-pod-2a441d9ad231

Autoescalado

- **Cluster autoscaling**
  - El **número de nodos** de un cluster puede cambiar dinámicamente en función del número de pods del cluster
  - Si un nuevo pod que se va a crear **no cabe en el cluster** (por la **reserva de recursos** de cada uno), se crea un nuevo nodo en el cluster
  - Esta funcionalidad requiere de la **instalación de un plugin** en Kubernetes dependiendo del proveedor cloud / VMs.

https://kubernetes.io/docs/tasks/administer-cluster/cluster-management/#cluster-autoscaling

https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler



https://medium.com/magalix/kubernetes-autoscaling-101-cluster-autoscaler-horizontal-pod-autoscaler-and-vertical-pod-2a441d9ad231

Autoescalado

- **Escalado inmediato?**
  - Se tienen que recoger métricas
  - HPA consulta las métricas cada cierto tiempo (por defecto 30 seg)
  - Después de un cambio, hay periodo de espera hasta que todo se estabiliza (3 min por defecto)
  - Los pods pueden tardar en arrancar y estar disponibles
  - Los nodos tardan mucho más en arrancar

Autoescalado

Sobredimensiona un poco si quieres soportar picos de tráfico

O saca la bola de cristal

NETFLIX

https://medium.com/netflix-techblog/scryer-netflixs-predictive-auto-scaling-engine-part-2-bb9c4f9b9385

Escalado con estado

code URJC

- Se asume que los pods de un deployment no tienen estado (**stateless**)
  - Cualquier pod puede ser eliminado (scale in)
  - Un nuevo pod pueda atender tráfico de un usuario que antes atendía otro pod (scale out)
- Pero hay veces que nuestras apps tienen estado (**statefull**)

Qué hacemos?

Demo time

App con estado replicada

```
$ kubectl create -f webapp-stateful/k8s/webapp.yaml

$ minikube service webapp

$ kubectl delete -f webapp-stateful/k8s/webapp.yaml
```

Escalado con estado

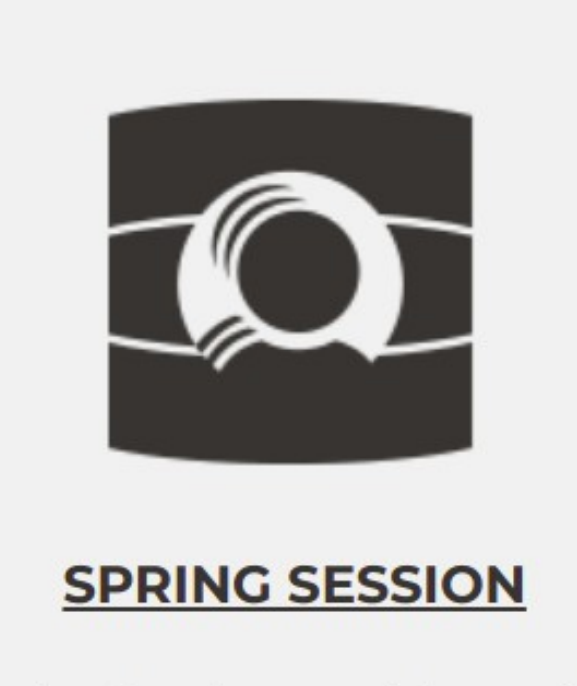
- **Algunas opciones**
  - Convierte la app en stateless > Mueve el estado a otro sitio
  - Crea aplicaciones que repliquen el estado entre sus réplicas
  - Usa mecanismos de escalado con estado en Kubernetes (StatefulSet)
  - Gestiona el estado fuera de Kubernetes (en los servicios proporcionados por el proveedor en la nube

Escalado con estado

- **Convierte la app en stateless**
  - **Web stateful:**
    - App web que mantiene datos del usuario conectado en memoria
    - Peticiones pueden ser atendidas por diferentes pods y se le considera nuevo usuario
  - **Solución:**
    - Los datos de la sesión tienen que estar en un recurso compartido

Escalado con estado

- Convierte la app en stateless



https://spring.io/projects/spring-session

Demo time

App sin estado replicada

```
$ kubectl create -f webapp-stateless/k8s/webapp.yaml

$ minikube service webapp

$ kubectl delete -f webapp-stateless/k8s/webapp.yaml
```

Escalado con estado

- **Tests funcionales**
  - No vale con verificar el status de la petición

```
@Test
public void test() throws InterruptedException {

        String userColor = null;

        for (int i = 0; i < 5; i++) {

                loadPage();

                String color = extractColor();

                if (userColor == null) {
                        userColor = color;
                } else {
                        assertThat(userColor).isEqualTo(color)
                                        .withFailMessage("El color del usuario ha cambiado");
                }

                System.out.println("Loaded page " + (i+1) +" time(s)");

                Thread.sleep(2000);
        }
}
```

Escalado con estado

- Aplicaciones que repliquen estado

Estructuras de datos compartidas y replicadas para Java

https://hazelcast.com/

Escalado con estado

- **Usa mecanismos de escalado stateful**
  - Los recursos **StatefulSet** proporcionan un mecanismo similar a los Deployment pero con características específicas para contenedores con estado:
    - Identificadores de red estables y únicos por pod
    - Almacenamiento persistente estable por pod
    - Despliegue y escalado ordenado de pods
    - Terminado y borrado ordenado de pods

https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/

Escalado con estado

- Usa mecanismos de escalado stateful

Es muy complejo diseñar un sistema distribuido con estado escalable y tolerante a fallos

Mejor usa servicios existentes preparados para trabajar en Kubernetes

Escalado con estado

- MySQL escalables y tolerantes a fallos en K8s

portworx

https://portworx.com/

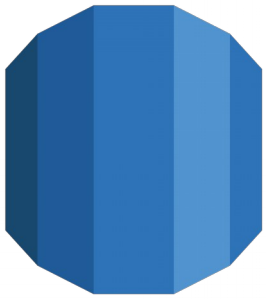
https://vitess.io/

oracle / mysql-operator

https://github.com/oracle/mysql-operator

Escalado con estado

- Gestiona el estado fuera de k8s



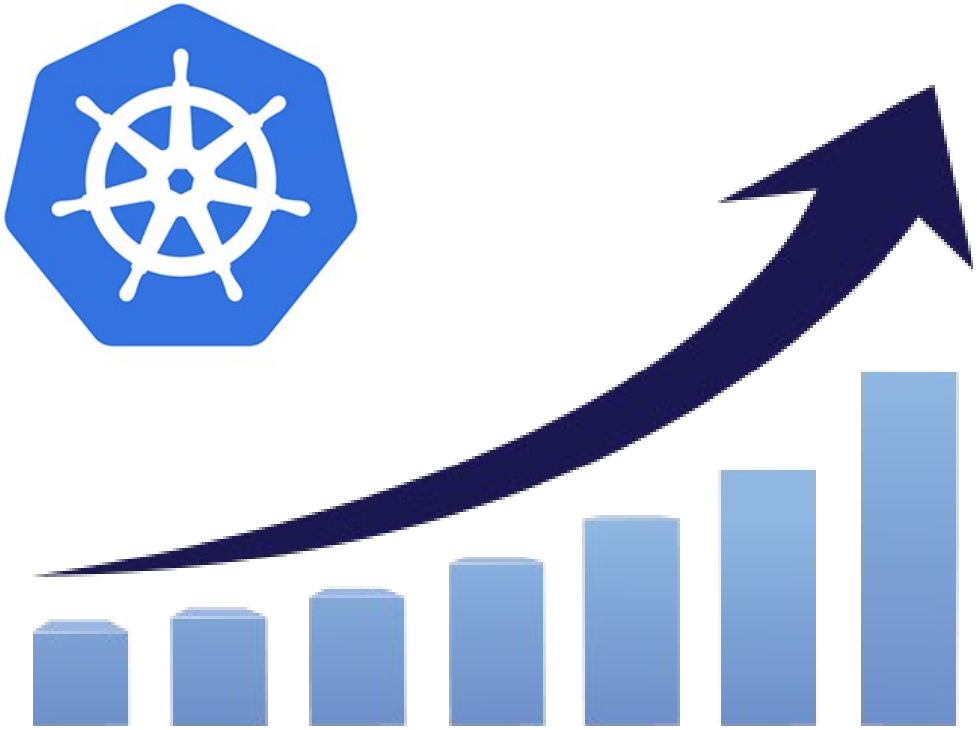
Amazon RDS

Google Cloud

Cloud SQL

Microsoft Azure

SQL Database



Pruebas de Escalabilidad

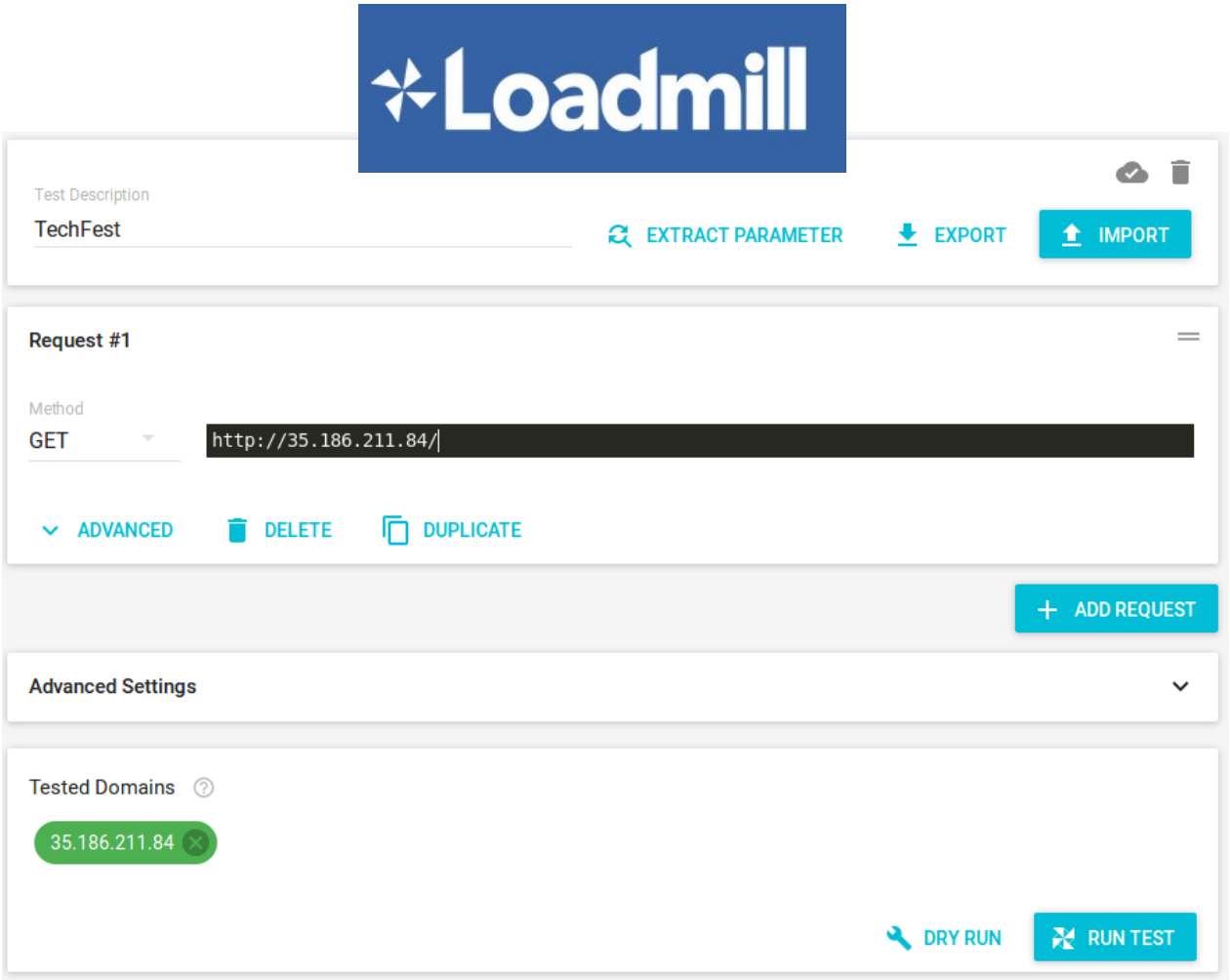
Pruebas de escalabilidad

- Se usan herramientas de **pruebas de carga** para simular **tráfico**
- Se verifica la **calidad de servicio** desde fuera
- Se **observa** que kubernetes (nodos) y las apps (réplicas) escalan como se espera

Pruebas de escalabilidad

- Herramientas de carga y verificación de la calidad de servicio

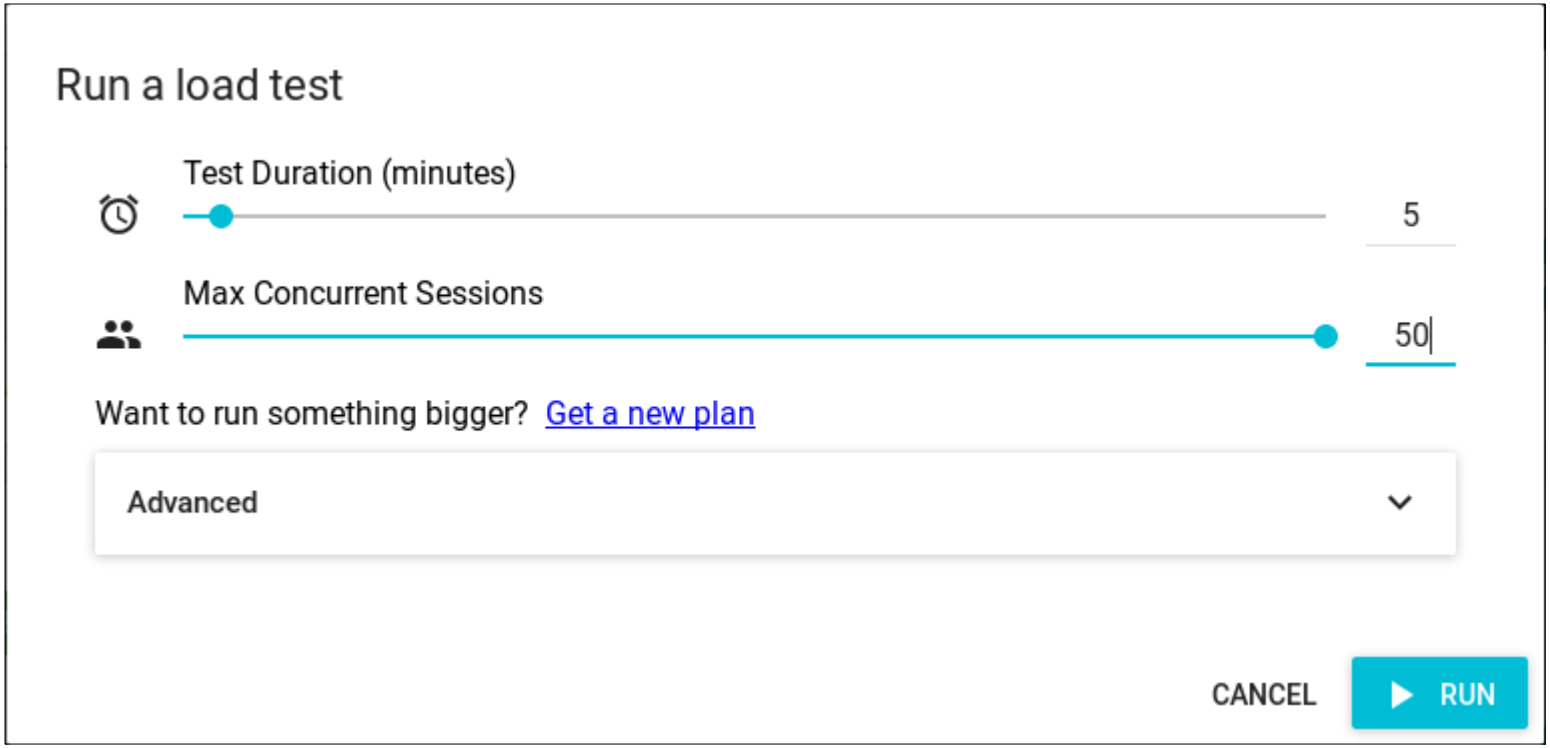
Pruebas de escalabilidad



@micael_gallego

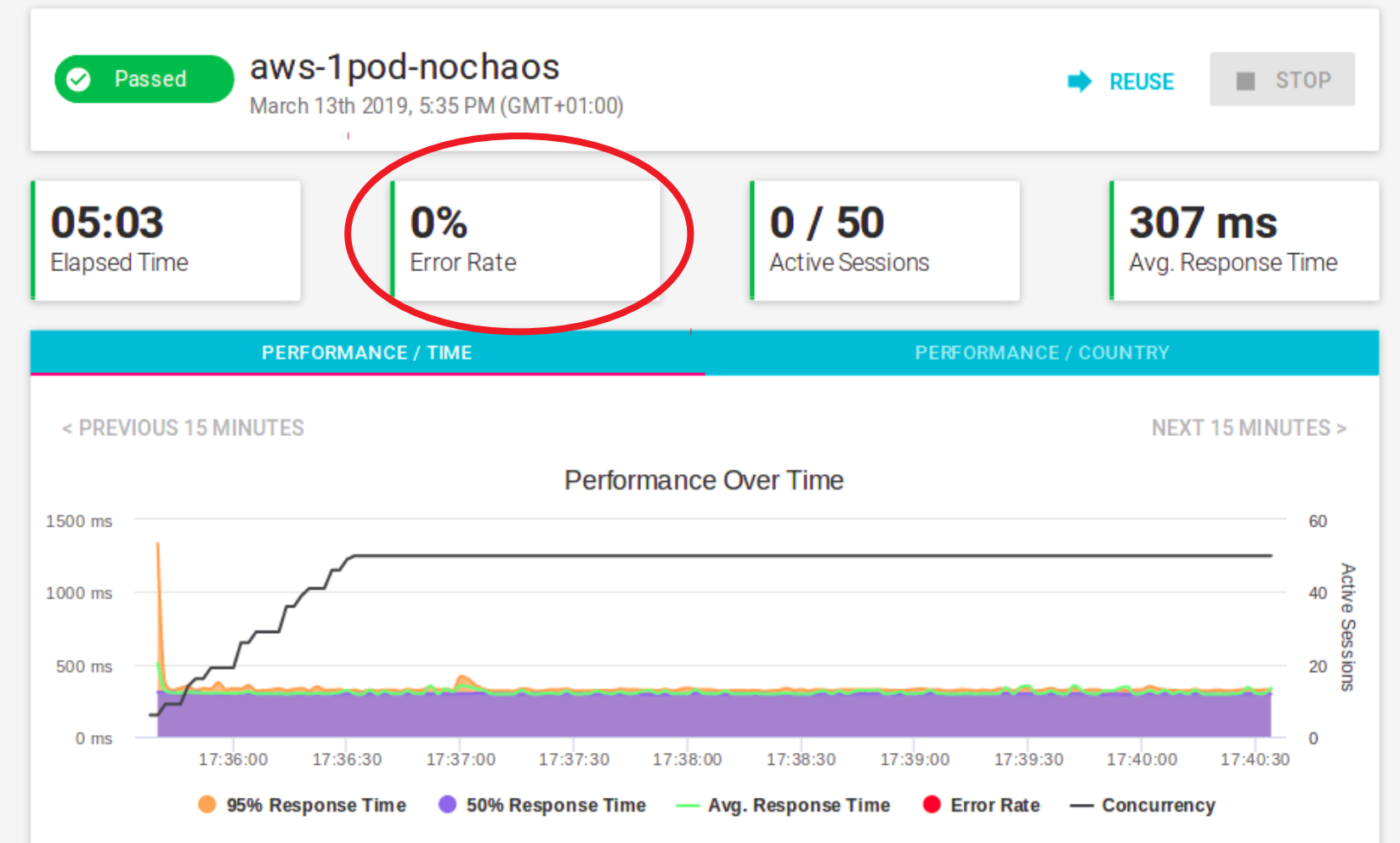
https://www.loadmill.com

Pruebas de escalabilidad



- Test de 5 minutos de duración
- 50 sesiones simultáneas haciendo peticiones http

Pruebas de escalabilidad



Pruebas de escalabilidad

code URJC

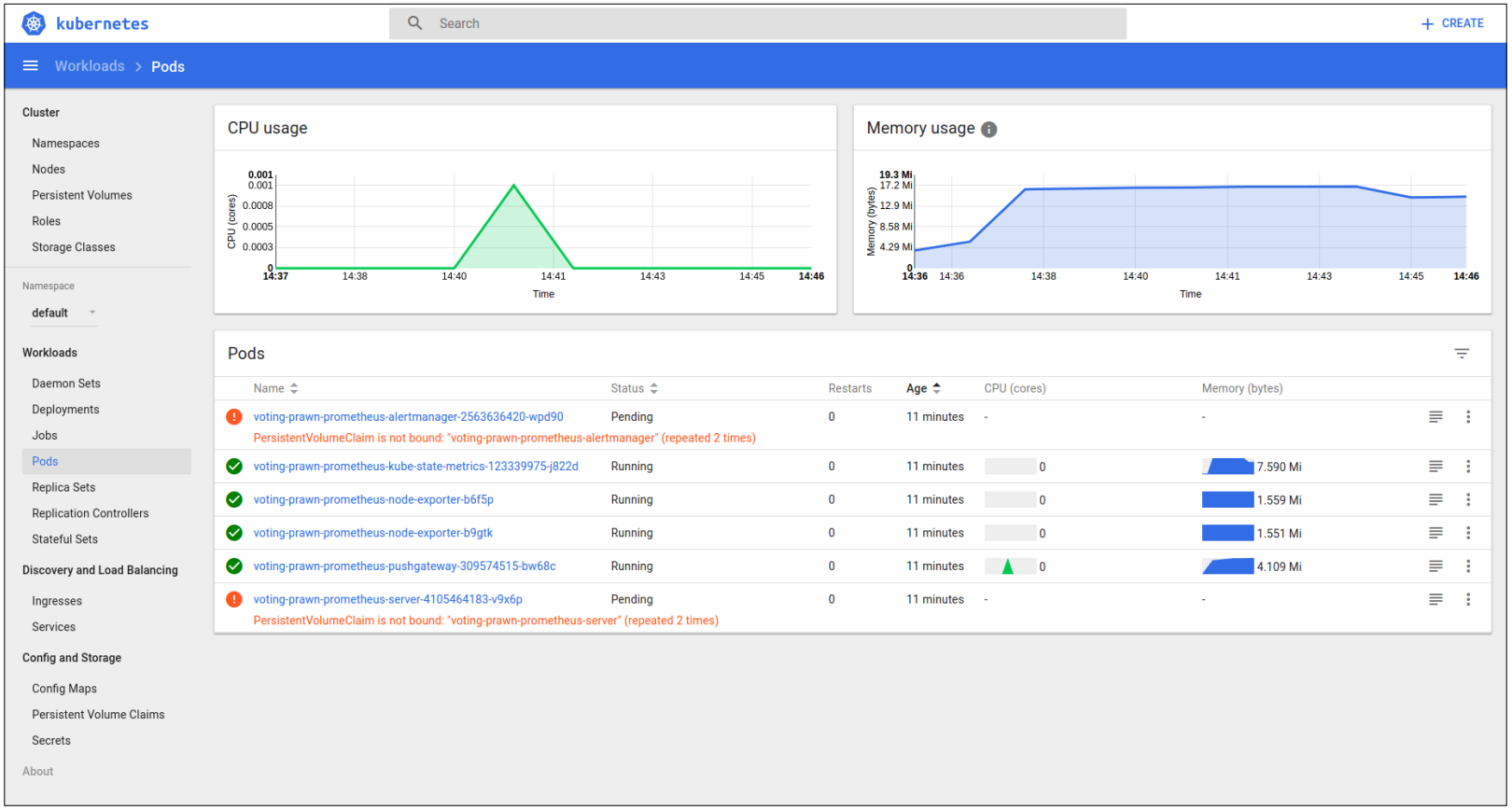
¿Qué ocurre en el cluster?

Observabilidad

- Monitorización (CPU, memoria…)
- Alertas y visualización
- Agregación de logs
- Registro de la comunicación entre servicios

Observabilidad

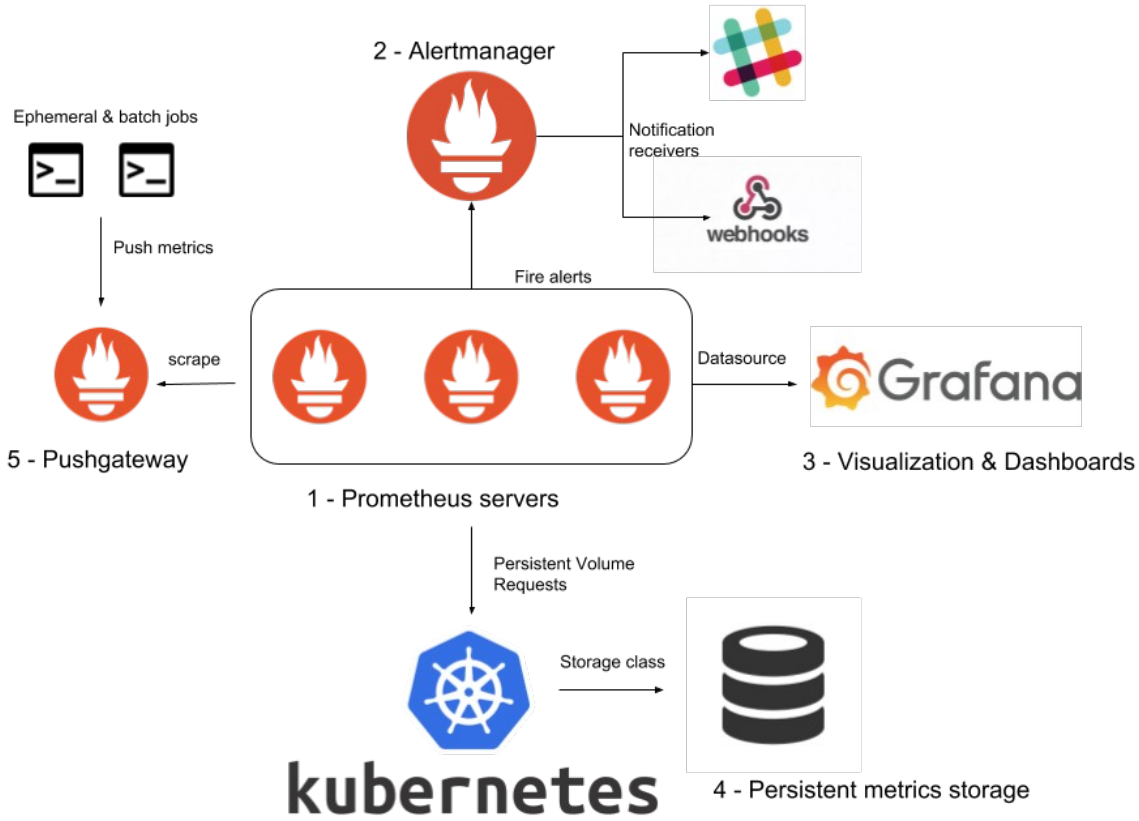
- Monitorización y visualización



Kubernetes Dashboard

Observabilidad

- Monitorización y visualización

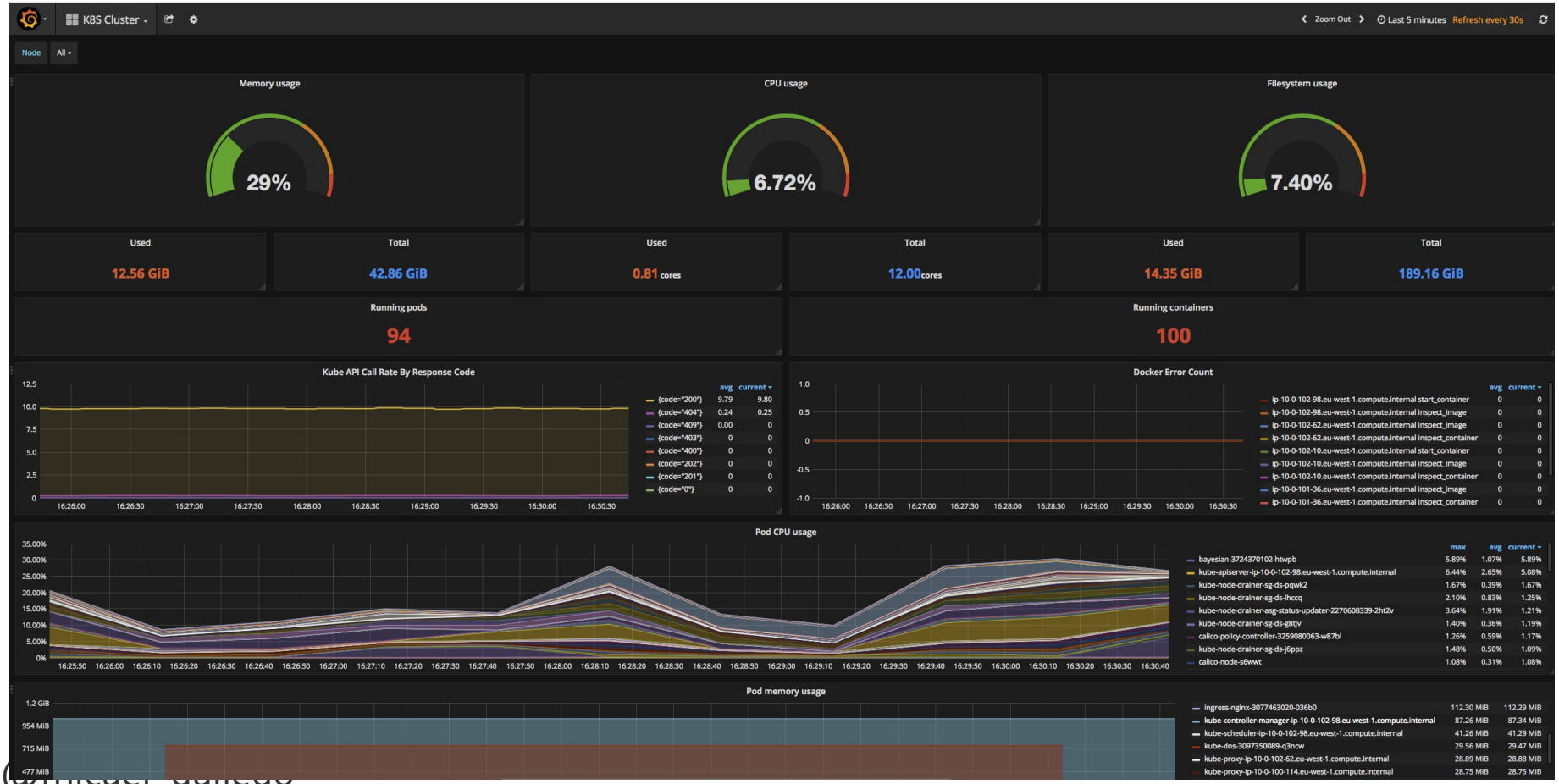


@micael_gallego

Prometheus + Grafana

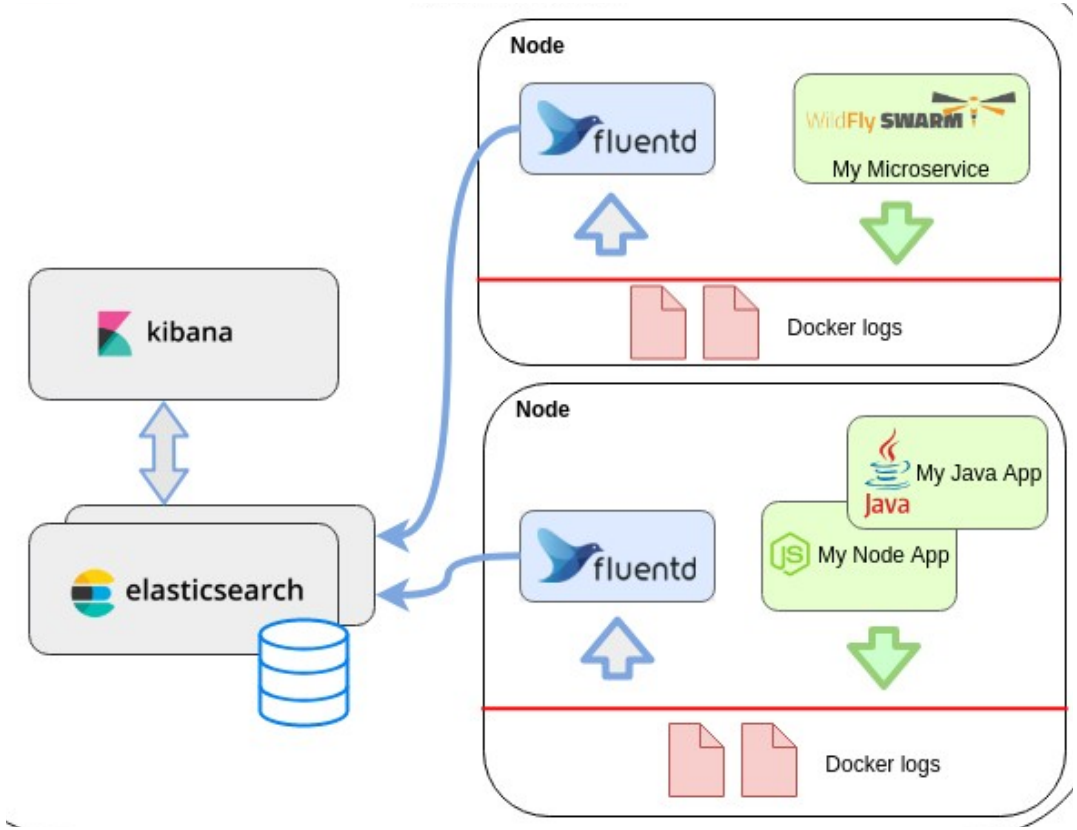
Observabilidad

- Monitorización y visualización



Observabilidad

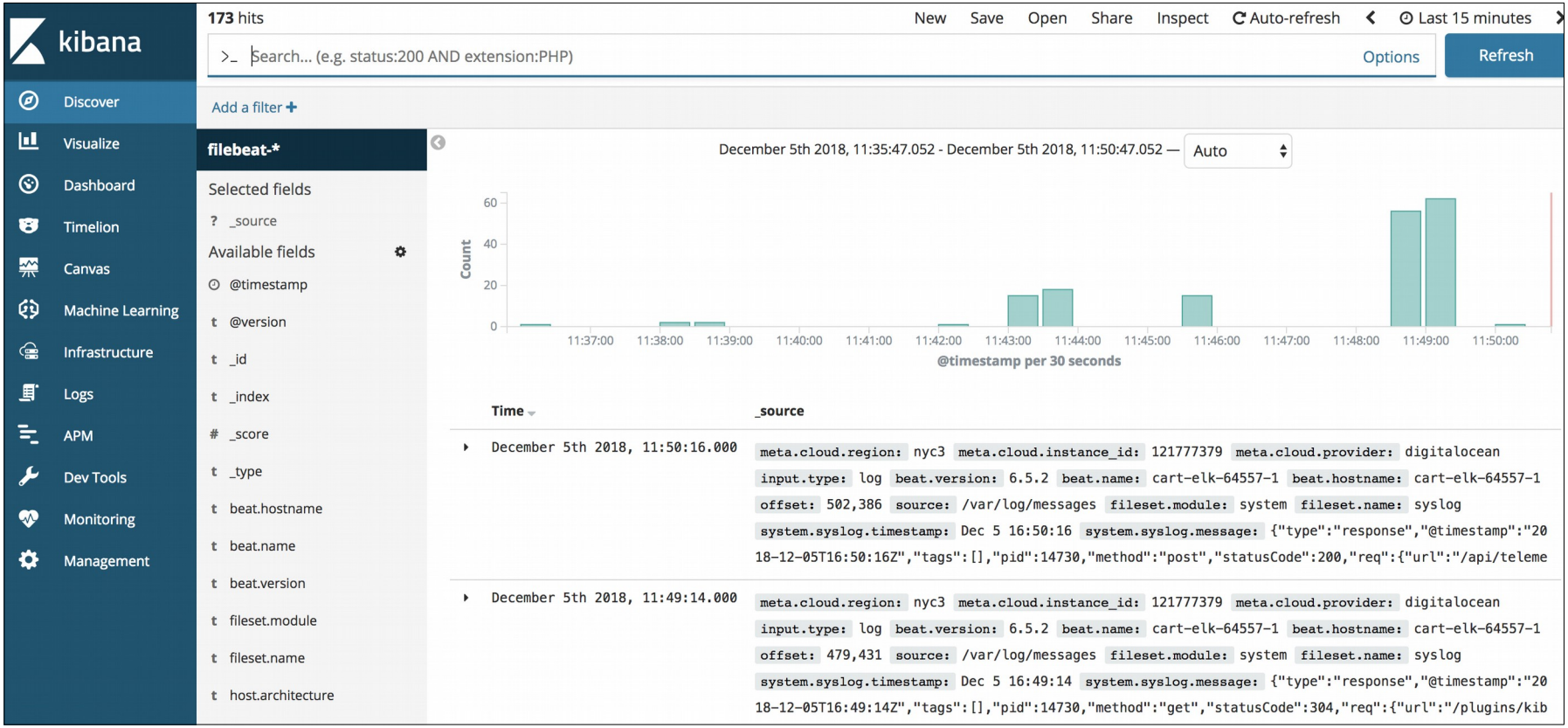
- Agregración de logs



ElasticSearch + FluentD + Kibana

Observabilidad

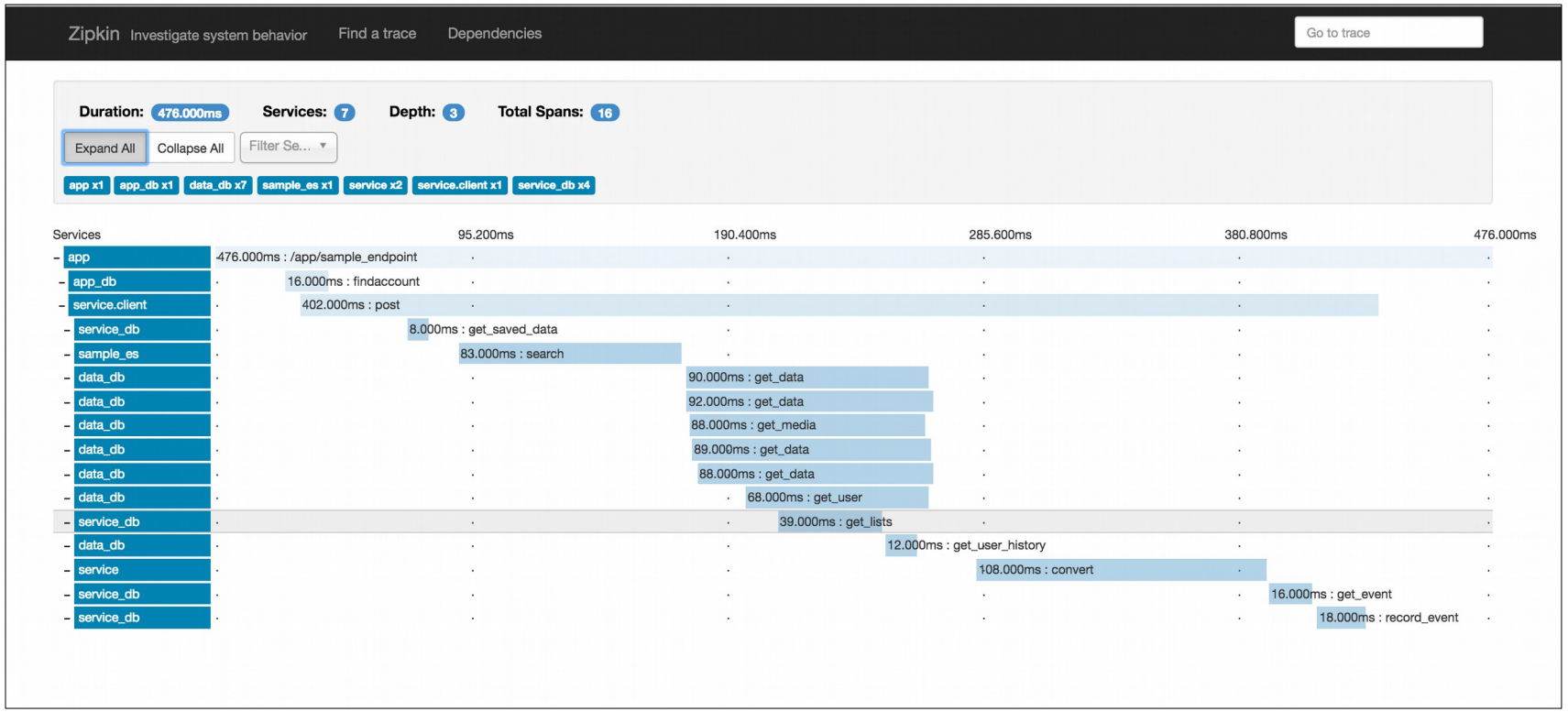
- Agregración de logs



ElasticSearch + FluentD + Kibana

Observabilidad

- Registro de comunicación entre servicios



Zipkin

Observabilidad

- **¿Tenemos que adaptar las aplicaciones?**
  - No requiere cambiar las aplicaciones
    - Monitorización de métricas de sistema
    - Visualización y alertas
    - Recogida de logs

Observabilidad

- **¿Tenemos que adaptar las aplicaciones?**
  - Requiere cambios en las aplicaciones
    - Monitorización de métricas de negocio
    - Generación de logs con información relevante (a ser posible estructurada)
    - Comunicación entre servicios: Enviar trazas a zipkin

Observabilidad

- Para testing automático necesitamos APIs en los tests
  - ElasticSearch

    https://www.elastic.co/guide/en/elasticsearch/reference/current/docs.html
  - Prometheus

    https://prometheus.io/docs/prometheus/latest/querying/api/
  - Kuebernetes

    https://kubernetes.io/docs/reference/using-api/client-libraries/
  - Zipkin

    https://zipkin.io/zipkin-api/

Observabilidad

- Kubernetes

Go	github.com/kubernetes/client-go/
Python	github.com/kubernetes-client/python/
Java	github.com/kubernetes-client/java
dotnet	github.com/kubernetes-client/csharp
JavaScript	github.com/kubernetes-client/javascript

```java
public class Example {
    public static void main(String[] args) throws IOException, ApiException{
        ApiClient client = Config.defaultClient();
        Configuration.setDefaultApiClient(client);

        CoreV1Api api = new CoreV1Api();
        V1PodList list = api.listPodForAllNamespaces(null, null, null, null, null, null, null, null, null);
        for (V1Pod item : list.getItems()) {
            System.out.println(item.getMetadata().getName());
        }
    }
}
```

https://kubernetes.io/docs/reference/using-api/client-libraries/

Tolerancia a fallos

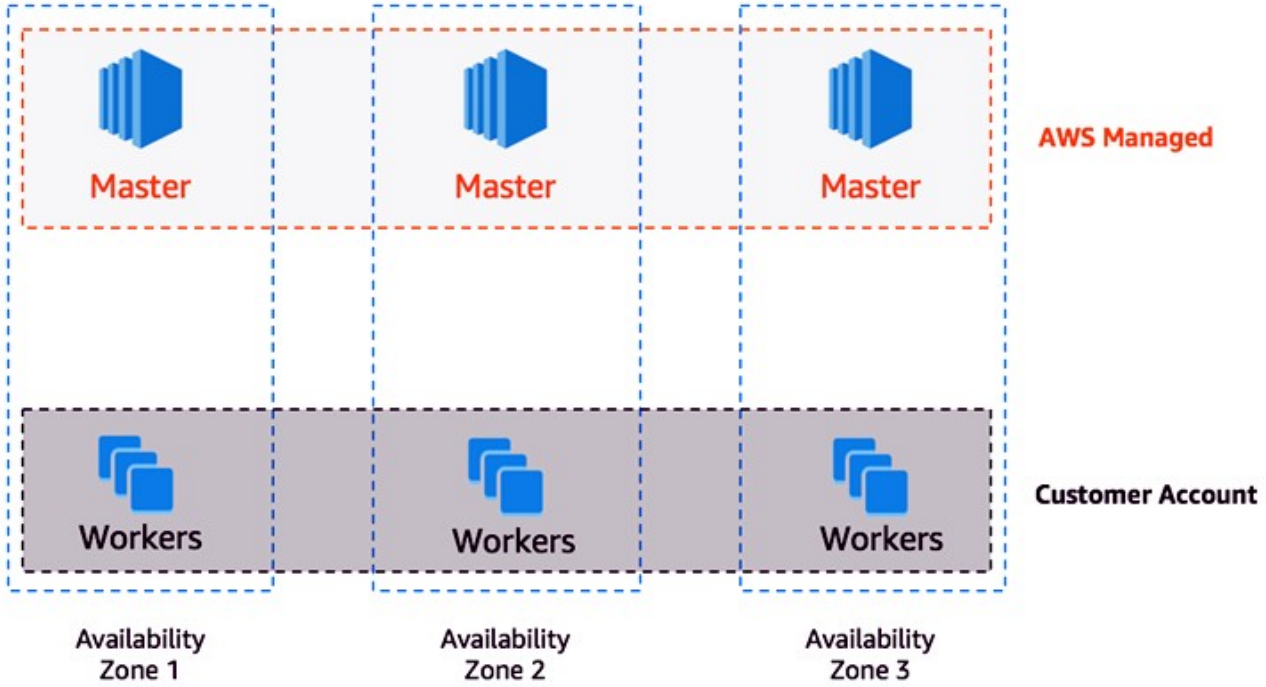
Tolerancia a fallos

Para ser tolerante a fallos se necesita ser **redundante**

Tiene que haber varias copias de cada elemento desacopladas entre sí para que un fallo en una no afecte a las demás

Tolerancia a fallos

- Despliegue redundante y aislado



https://aws.amazon.com/es/eks/

Tolerancia a fallos

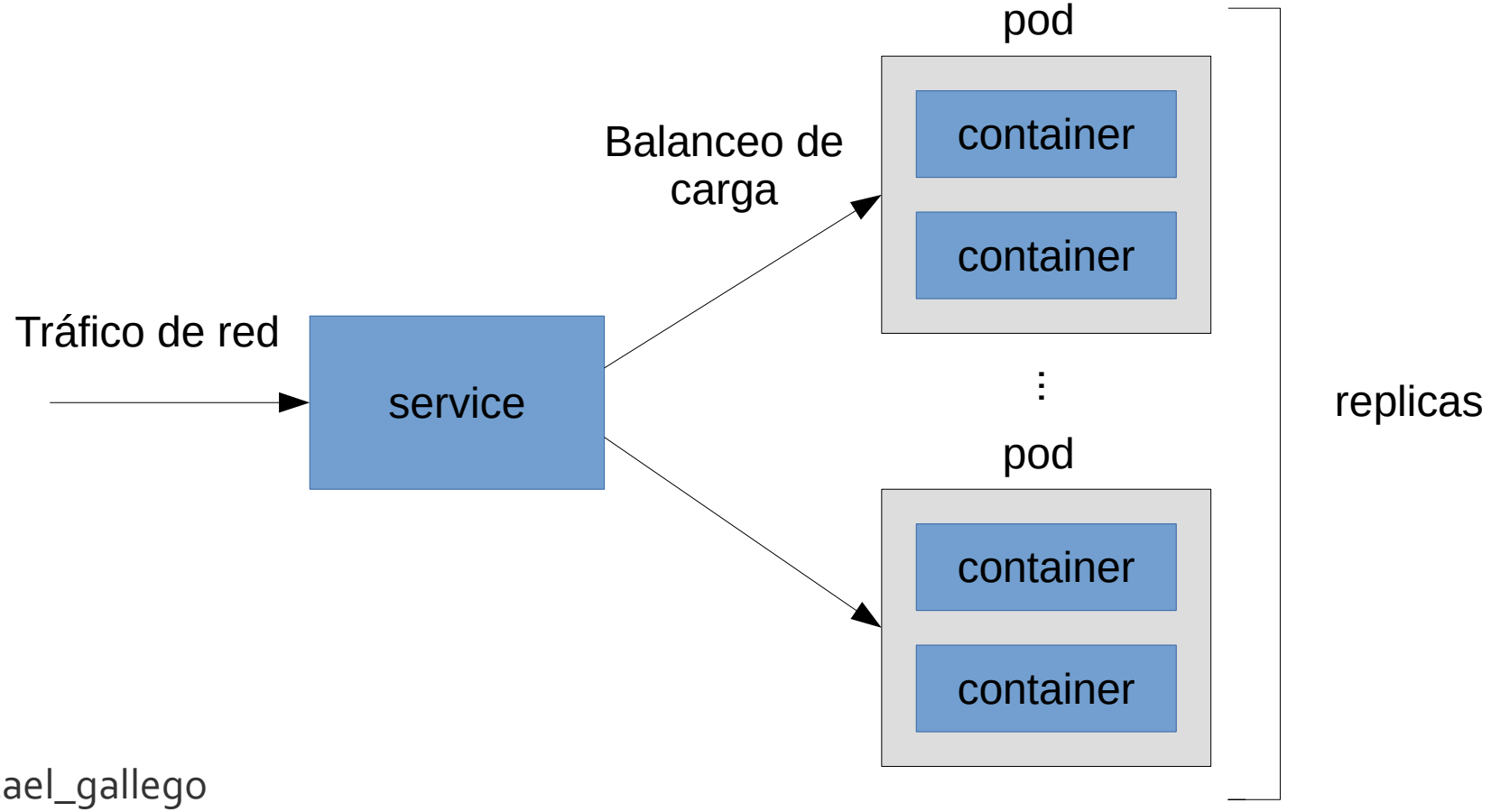
- Los servicios stateful de k8s son redundantes

A distributed, reliable key-value store for the most critical data of a distributed system

https://coreos.com/etcd/

Tolerancia a fallos

Las **réplicas de los pods** ejecutándo en **diferentes nodos** del cluster ofrecen **redundancia** a nivel de aplicación



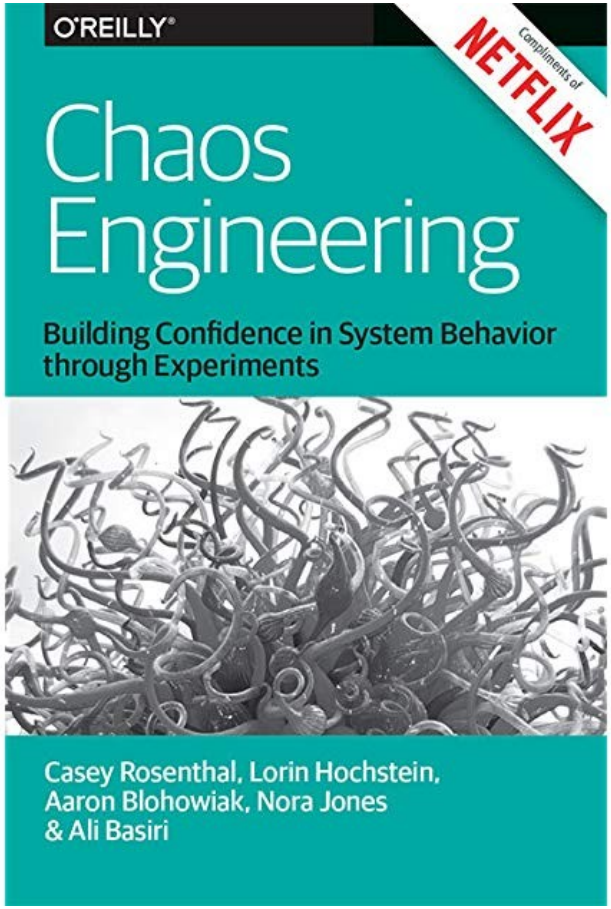
Tolerancia a fallos

- Si un **nodo del cluster falla**
  - Los **pods ejecutando** en ese nodo se crean en otro **nodo disponible**
  - El nodo fallido se **"sustituye"** por un nuevo nodo en el que se pueden volver a desplegar pods

https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/

Pruebas de tolerancia a fallos

Pruebas de tolerancia a fallos



"Limited scope, continuous, disaster recovery"

Russ Miles

http://principlesofchaos.org/

https://medium.com/russmiles/chaos-engineering-for-the-business-17b723f26361

Pruebas de tolerancia a fallos

- Introducir errores **aleatorios** e **impredecibles**
- Su objetivo es proporcionar datos sobre al **estabilidad de los sistemas**
- Sabemos que los **sistemas** van a fallar
- Por qué no introducir nosotros los fallos para ver cómo de **resistentes** son nuestros sistemas

Pruebas de tolerancia a fallos

- **Chaos monkeys** de Netflix
  - Creada en 2011
  - Desconectaba nodos de AWS en los sistemas en producción de Netflix
  - Se comprobaba cómo se comportaba el resto del sistema ante la caída de una instancia

https://github.com/netflix/chaosmonkey

Pruebas de tolerancia a fallos

- Ejemplos de fallos que podemos inducir (caos)
  - Para una Máquina Virtual
  - Incrementar la latencia en la red
  - Forzar errores HTTP
  - Aumentar la carga de CPU
  - Simular disco lleno
  - Etc…

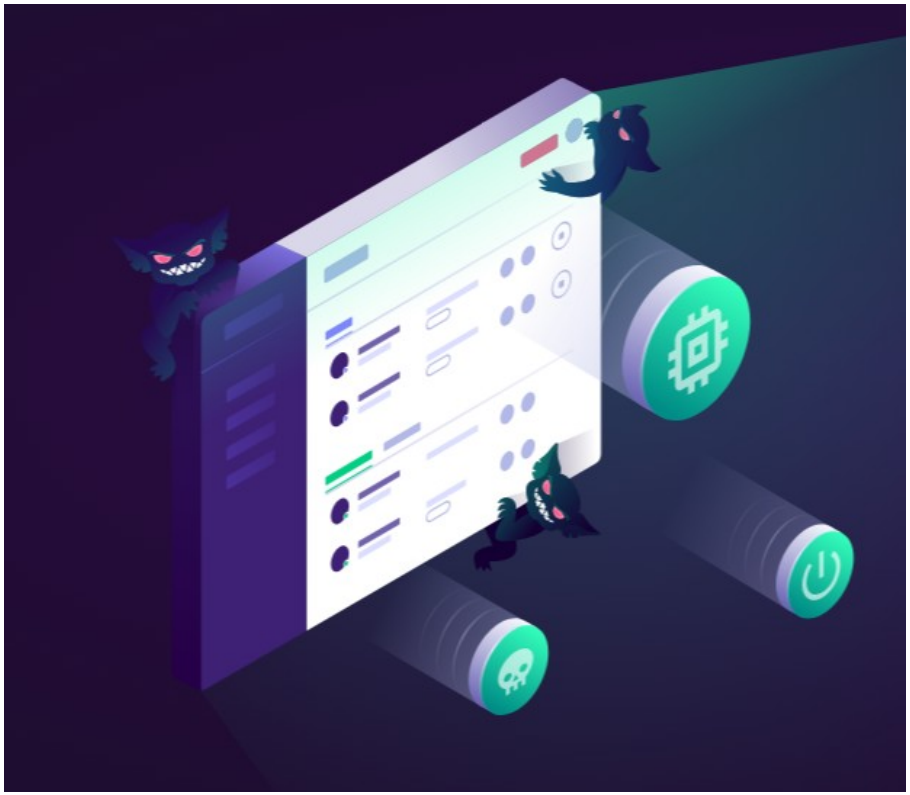
Pruebas de tolerancia a fallos

- **Metodología**
  - Partimos de un **estado estable**
  - Inducimos **caos**
  - **Comparamos** el comportamiento del estado estable frente al estado problemático
  - Las herramientas de **observabilidad** son esenciales

Herramientas de Caos para Kubernetes

Gremlin

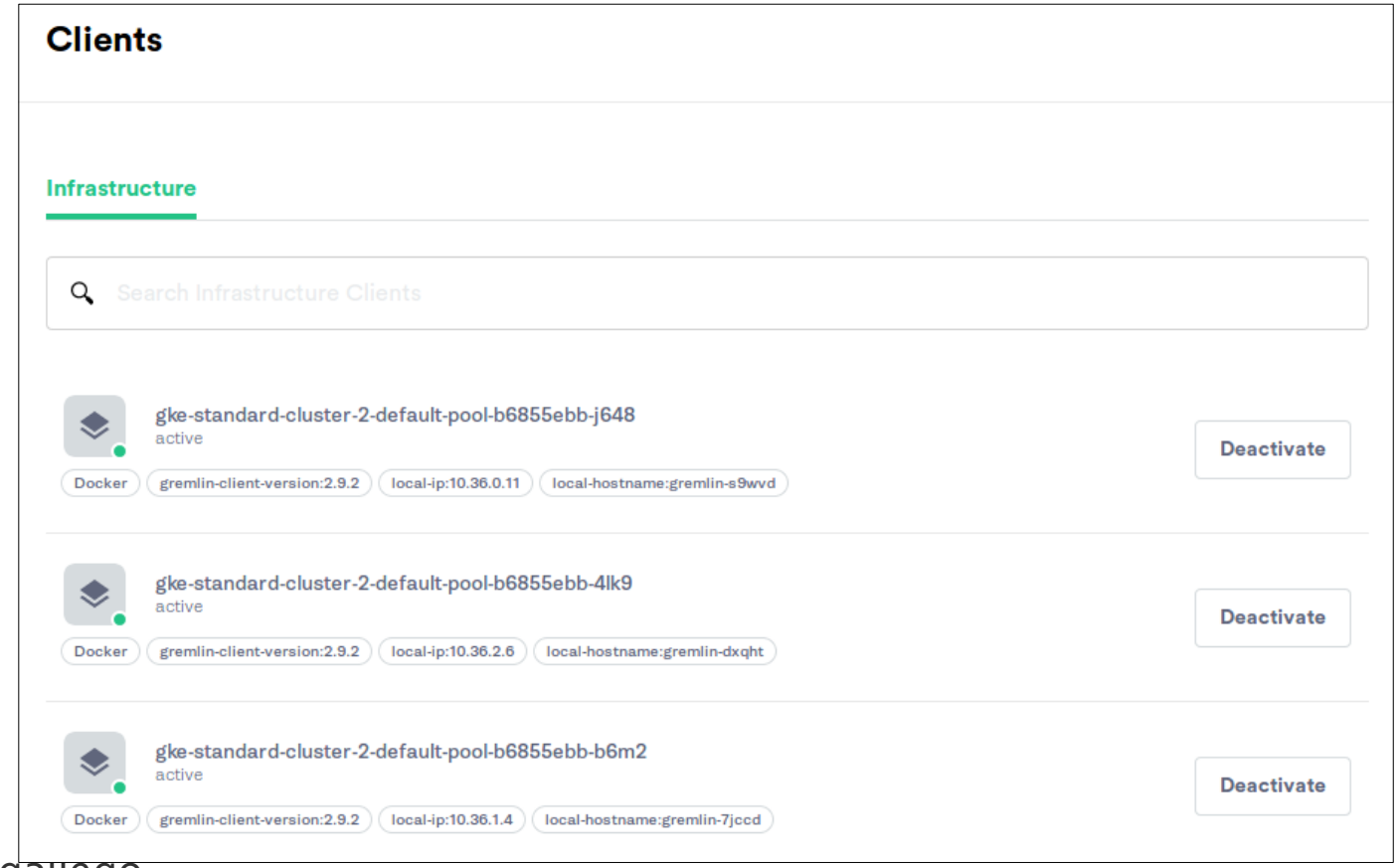
- **Chaos as a Service**
- Basado en cliente/ servidor
- Es necesario instalar un componente en nuestro cluster k8s



https://www.gremlin.com/

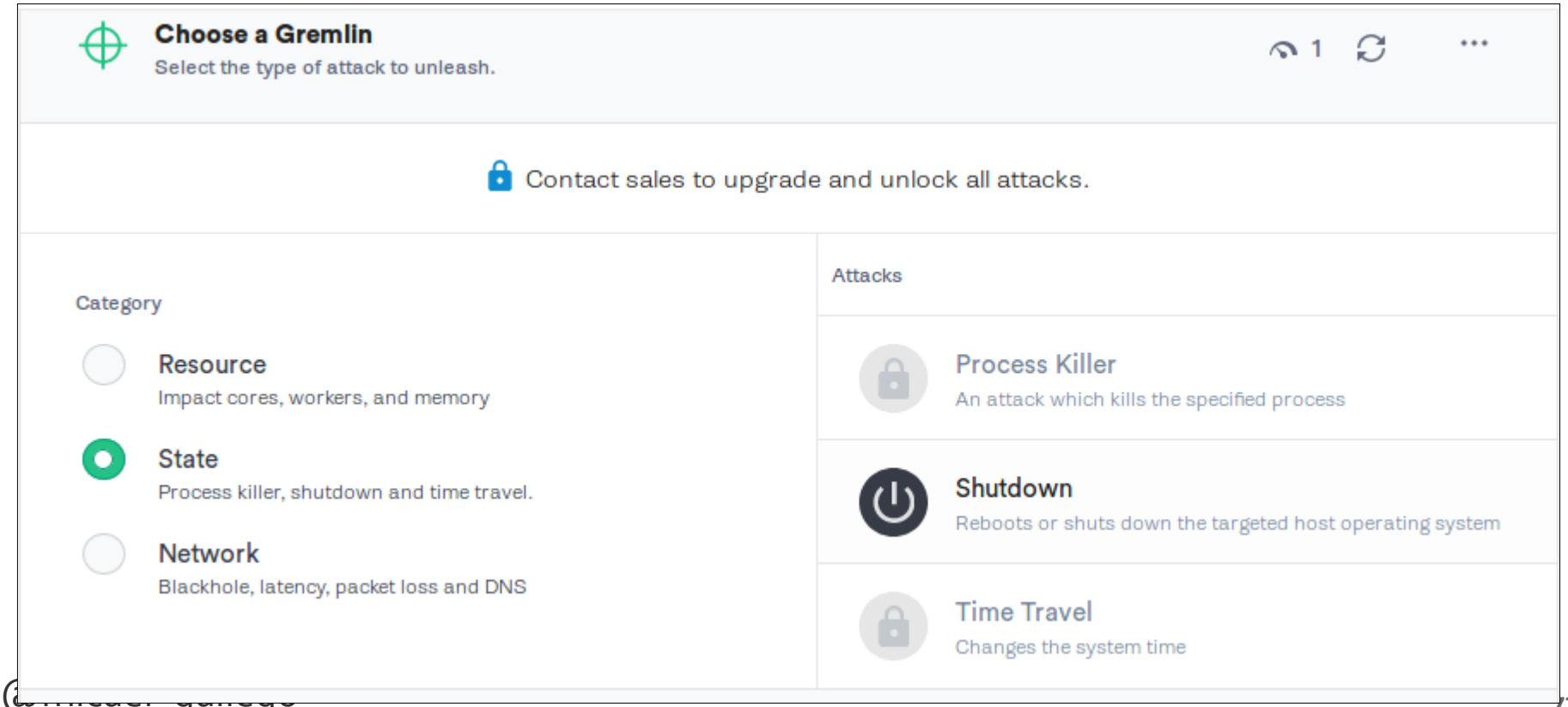
Gremlin

- Podemos ver los nodos de nuestro cluster en su web



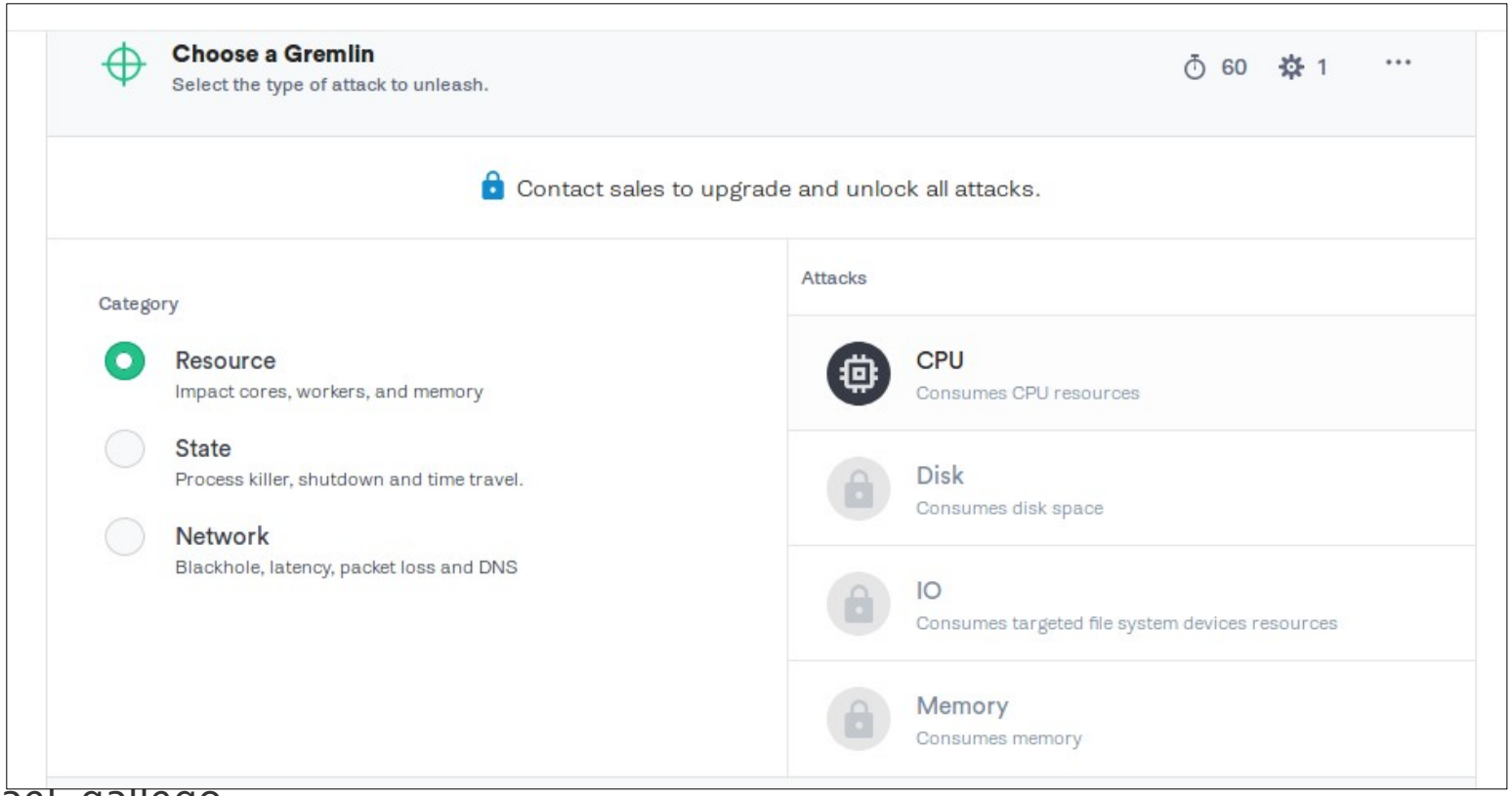
Gremlin

- **Tipos de caos (versión gratuita)**
  - Apagar un nodo



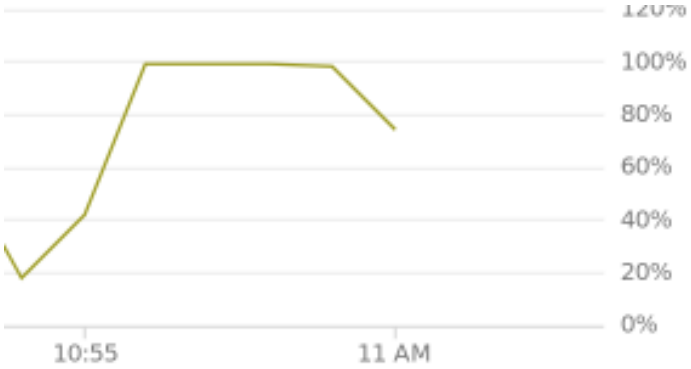
Gremlin

- **Tipos de caos (versión gratuita)**
  - Añadir carga de CPU



Gremlin

- **Carga de CPU**
  - Si lanzamos una prueba de carga elegimos un nodo de los disponibles, un tiempo de ejecución y cuántos nodos usar.
  - En las métricas de nuestro cluster podemos ver cómo la carga de CPU aumenta



Pod-Chaos-Monkey

- **Elimina un pod** cada cierto tiempo
- El temporizador puede ser **configurado**
- Podemos así comprobar como se comporta nuestra app cuando los pods se mueren súbitamente
- Script de bash que usa kubectl

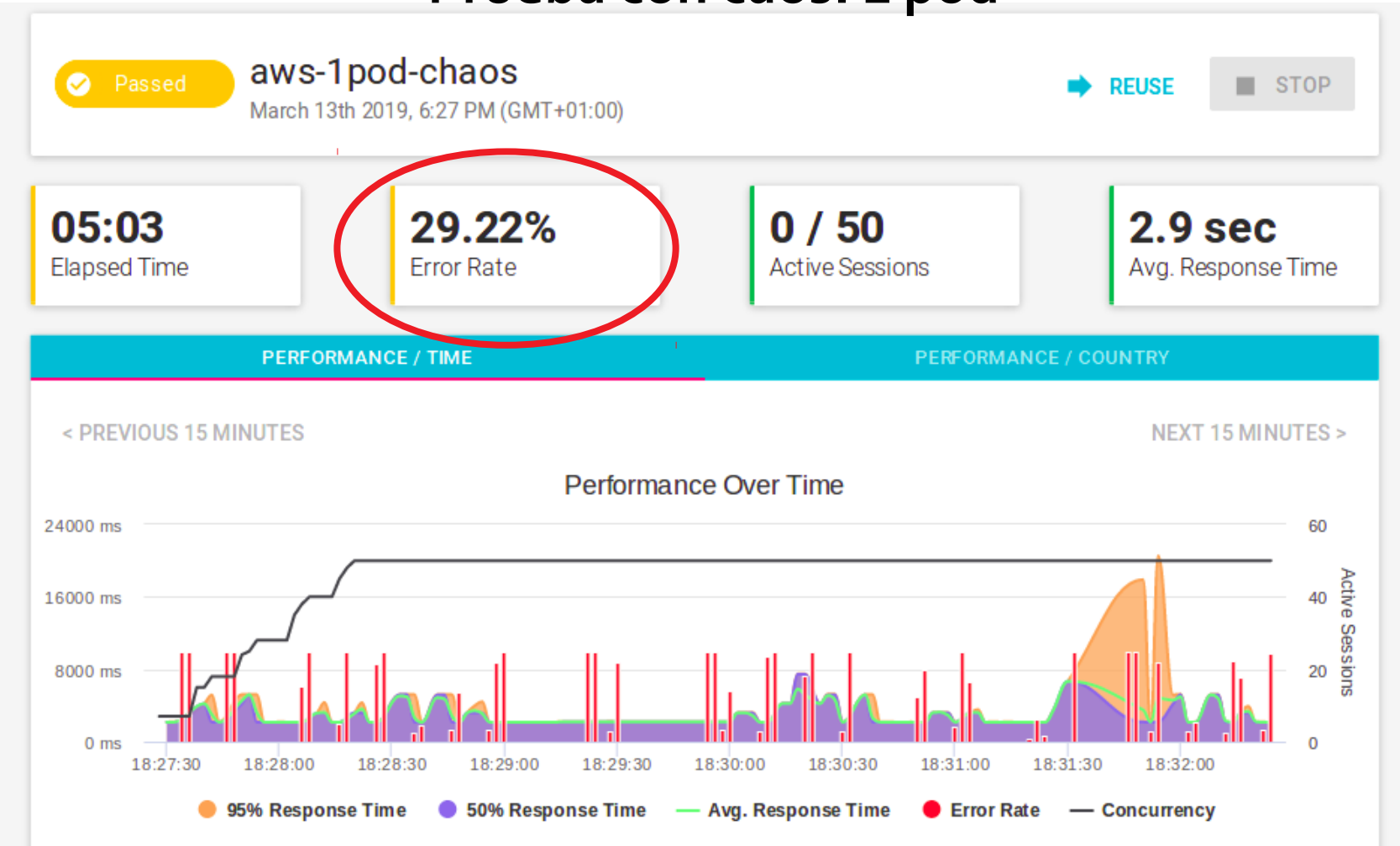
https://github.com/jnewland/kubernetes-pod-chaos-monkey

Pod-Chaos-Monkey

- Desplegamos una aplicación con **un único pod**
- La app tarda **2 seg** en procesar cada petición
- Usaremos **LoadMill** para simular la carga y analizar la calidad de servicio
- Se eliminará un pod cada **30 segundos**

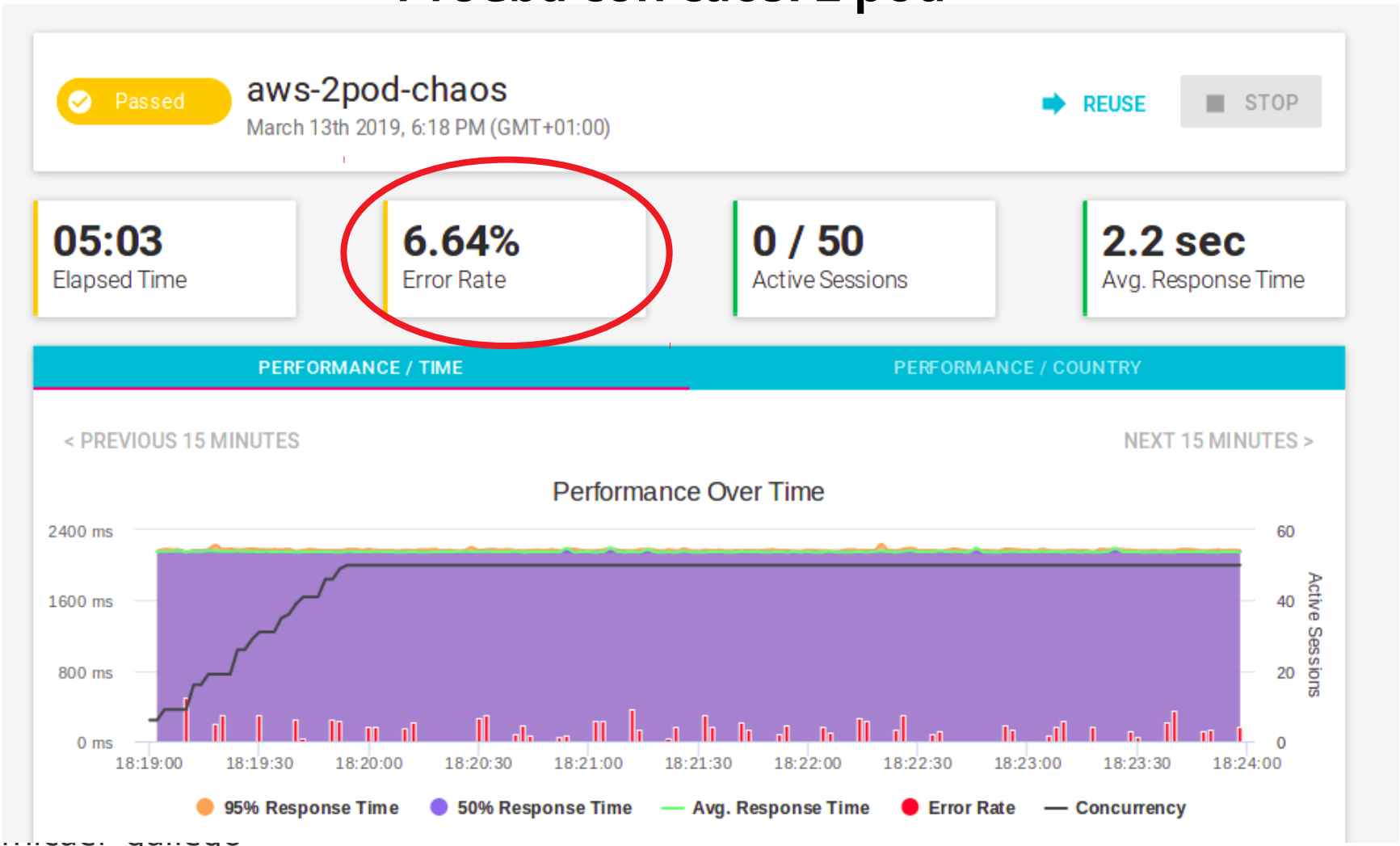
Pod-Chaos-Monkey

Prueba con caos: 1 pod



Pod-Chaos-Monkey

Prueba con caos: 2 pod



Pod-Chaos-Monkey

- Vemos que los pods se regeneran cada 30 segundos

```
node-micro-6fd4b5f997-n2xkp   0/1   Terminating   0     31s
node-micro-6fd4b5f997-ntjpc   1/1   Running   0     2s
node-micro-6fd4b5f997-n2xkp   0/1   Terminating   0     42s
node-micro-6fd4b5f997-n2xkp   0/1   Terminating   0     42s
node-micro-6fd4b5f997-ntjpc   1/1   Terminating   0     30s
node-micro-6fd4b5f997-8jm9j   0/1   Pending   0     0s
node-micro-6fd4b5f997-8jm9j   0/1   Pending   0     0s
node-micro-6fd4b5f997-8jm9j   0/1   ContainerCreating   0     0s
node-micro-6fd4b5f997-ntjpc   0/1   Terminating   0     30s
node-micro-6fd4b5f997-8jm9j   1/1   Running   0     1s
node-micro-6fd4b5f997-ntjpc   0/1   Terminating   0     32s
node-micro-6fd4b5f997-ntjpc   0/1   Terminating   0     32s
node-micro-6fd4b5f997-8jm9j   1/1   Terminating   0     31s
node-micro-6fd4b5f997-jf7z4   0/1   Pending   0     0s
node-micro-6fd4b5f997-jf7z4   0/1   Pending   0     0s
node-micro-6fd4b5f997-jf7z4   0/1   ContainerCreating   0     0s
node-micro-6fd4b5f997-8jm9j   0/1   Terminating   0     32s
node-micro-6fd4b5f997-8jm9j   0/1   Terminating   0     32s
node-micro-6fd4b5f997-jf7z4   1/1   Running   0     2s
node-micro-6fd4b5f997-8jm9j   0/1   Terminating   0     41s
node-micro-6fd4b5f997-8jm9j   0/1   Terminating   0     41s
node-micro-6fd4b5f997-jf7z4   1/1   Terminating   0     30s
node-micro-6fd4b5f997-qjv5x   0/1   Pending   0     0s
node-micro-6fd4b5f997-qjv5x   0/1   Pending   0     0s
node-micro-6fd4b5f997-qjv5x   0/1   ContainerCreating   0     0s
node-micro-6fd4b5f997-jf7z4   0/1   Terminating   0     31s
node-micro-6fd4b5f997-qjv5x   1/1   Running   0     3s
node-micro-6fd4b5f997-jf7z4   0/1   Terminating   0     40s
node-micro-6fd4b5f997-jf7z4   0/1   Terminating   0     40s
node-micro-6fd4b5f997-qjv5x   1/1   Terminating   0     31s
node-micro-6fd4b5f997-gqjvc   0/1   Pending   0     0s
node-micro-6fd4b5f997-gqjvc   0/1   Pending   0     0s
node-micro-6fd4b5f997-gqjvc   0/1   ContainerCreating   0     0s
node-micro-6fd4b5f997-qjv5x   0/1   Terminating   0     32s
```

Kube-Monkey

- Es la versión de **Netflix Chaos Monkey** aplicada a k8s
- Se puede planificar para que sólo actúe en ciertas franjas horarias
- Se suelen usar horas valle

https://github.com/asobti/kube-monkey

Kube-Monkey

- En los deployments se especifican **labels** para configurar Kube-Monkey
  - Si está habilitado o no para ser eliminada por kube-monkey
  - El identificador de la app
  - El modo de muerte: Puedes matar un solo pod o todos o un porcentaje
  - Etc…

Kube-Monkey

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: monkey-victim
  namespace: app-namespace
spec:
  template:
    metadata:
      labels:
        kube-monkey/enabled: enabled
        kube-monkey/identifier: monkey-victim
        kube-monkey/kill-mode: "fixed"
        kube-monkey/kill-value: '1'
[... omitted ...]
```

habilitado

identificador

Modo de muerte

Valor de muerte

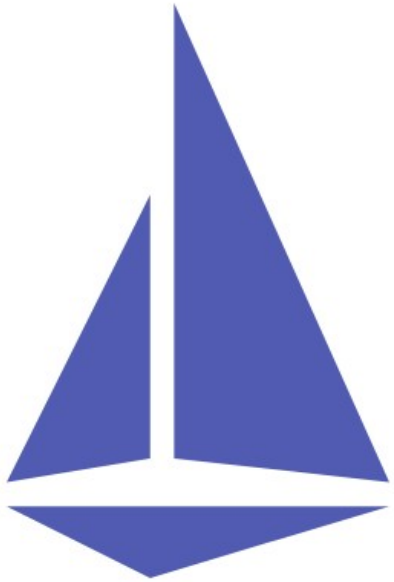
¿Cómo ser más tolerantes a los fallos?

Mejora en tolerancia a fallos

- **¿Por qué el cliente recibe errores cuando hay dos pods?**
  - Porque el pod que atiende la petición muere y se envía el error al cliente
- **¿Se podría reintentar la petición en este caso para que llegue al otro pod?**
  - El cliente podría reintentar, pero no el pod

Mejora en tolerancia a fallos

¿Podría reintentar el balanceador (servicio)?



Istio

Connect, secure, control, and observe services.

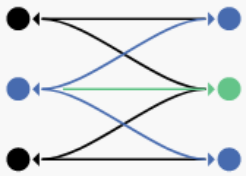
Istio

- Es un **plugin de Kubernetes** que gestiona las peticiones de red que llegan a un pod (y las que salen de él)
- Mejora la **tolerancia a fallos**
  - Permite reintentar peticiones fallidas
  - Permite proteger servicios para que sean más tolerantes a fallos (Circuit breaker, políticas…)
  - Permite controlar los timeouts de las peticiones

Istio

code URJC

Ofrece muchos más servicios de red

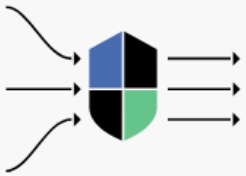


Connect

Intelligently control the flow of traffic and API calls between services, conduct a range of tests, and upgrade gradually with red/black deployments.

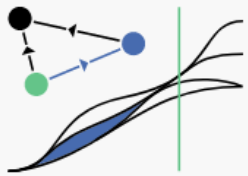
Secure

Automatically secure your services through managed authentication, authorization, and encryption of communication between services.



Control

Apply policies and ensure that they're enforced, and that resources are fairly distributed among consumers.



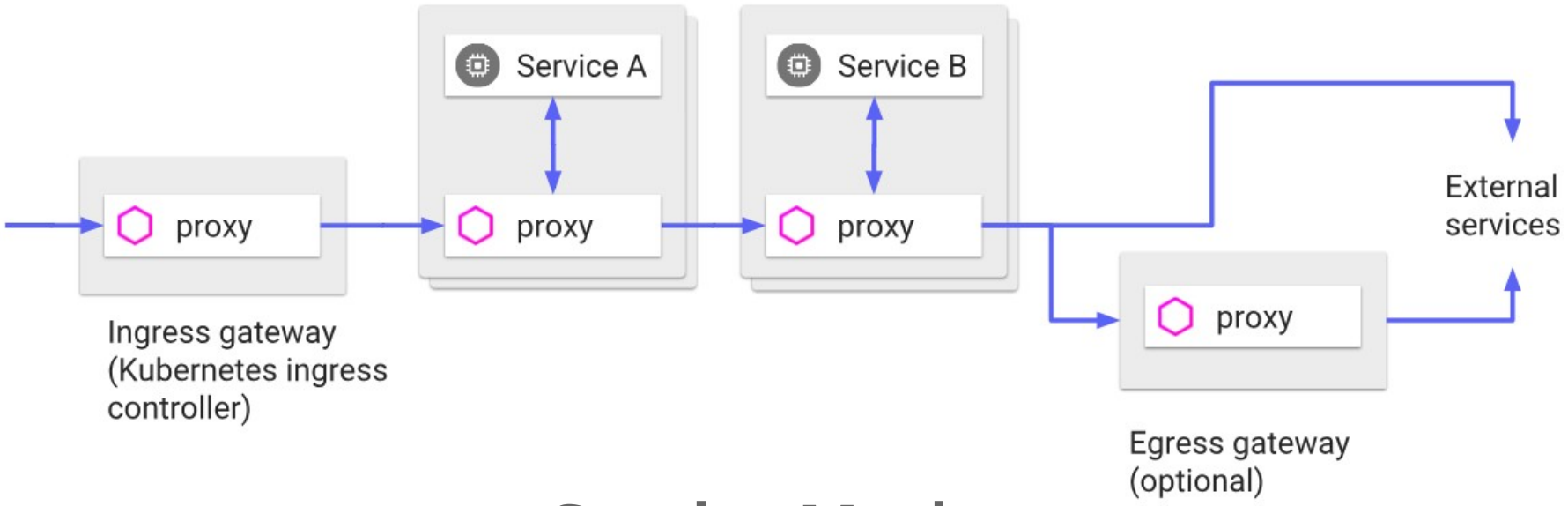
Observe

See what's happening with rich automatic tracing, monitoring, and logging of all your services.

https://istio.io/

Istio

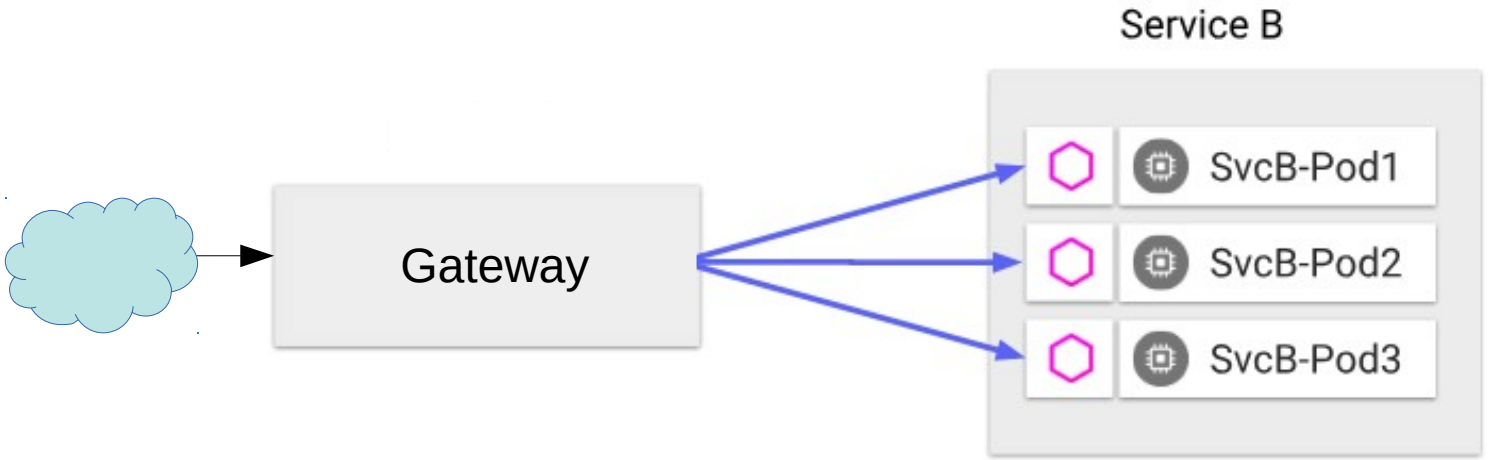
Para controlar el tráfico, Istio pone un **micro-proxy** al lado de cada **pod**, a la **entrada** y a la **salida** de cada nodo del cluster



Service Mesh

Istio

Para poder aplicar Istio a las llamadas externas, se usa un Istio **Gateway** en vez de un Ingress



https://istio.io/

Istio

```
apiVersion: networking.istio.io/v1alph
kind: Gateway
metadata:
  name: httpbin-gateway
spec:
  selector:
    istio: ingressgateway
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - "httpbin.example.com"
```

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: httpbin
spec:
  hosts:
  - "httpbin.example.com"
  gateways:
  - httpbin-gateway
  http:
  - match:
    - uri:
        prefix: /status
    - uri:
        prefix: /delay
    route:
    - destination:
        port:
          number: 8000
        host: httpbin
```

Istio

- Mejoras en tolerancia a fallos con Istio a los **clientes del servicio**
  - Timeout de las peticiones (para no esperar demasiado)
  - Reintentos en peticiones fallidas

Istio

- Mejoras en tolerancia a fallos con Istio al **servicio**
  - **Reintentos** con tiempo de espera (para no sobrecargar el servicio)
  - Limitar el número de **conexiones concurrentes**
  - Verificación de la **salud** de los servicios (*health-check*)
  - ***Circuit-breakers*** a los pods para que se recuperen antes de atender nuevas peticiones

Istio

- Reintentos

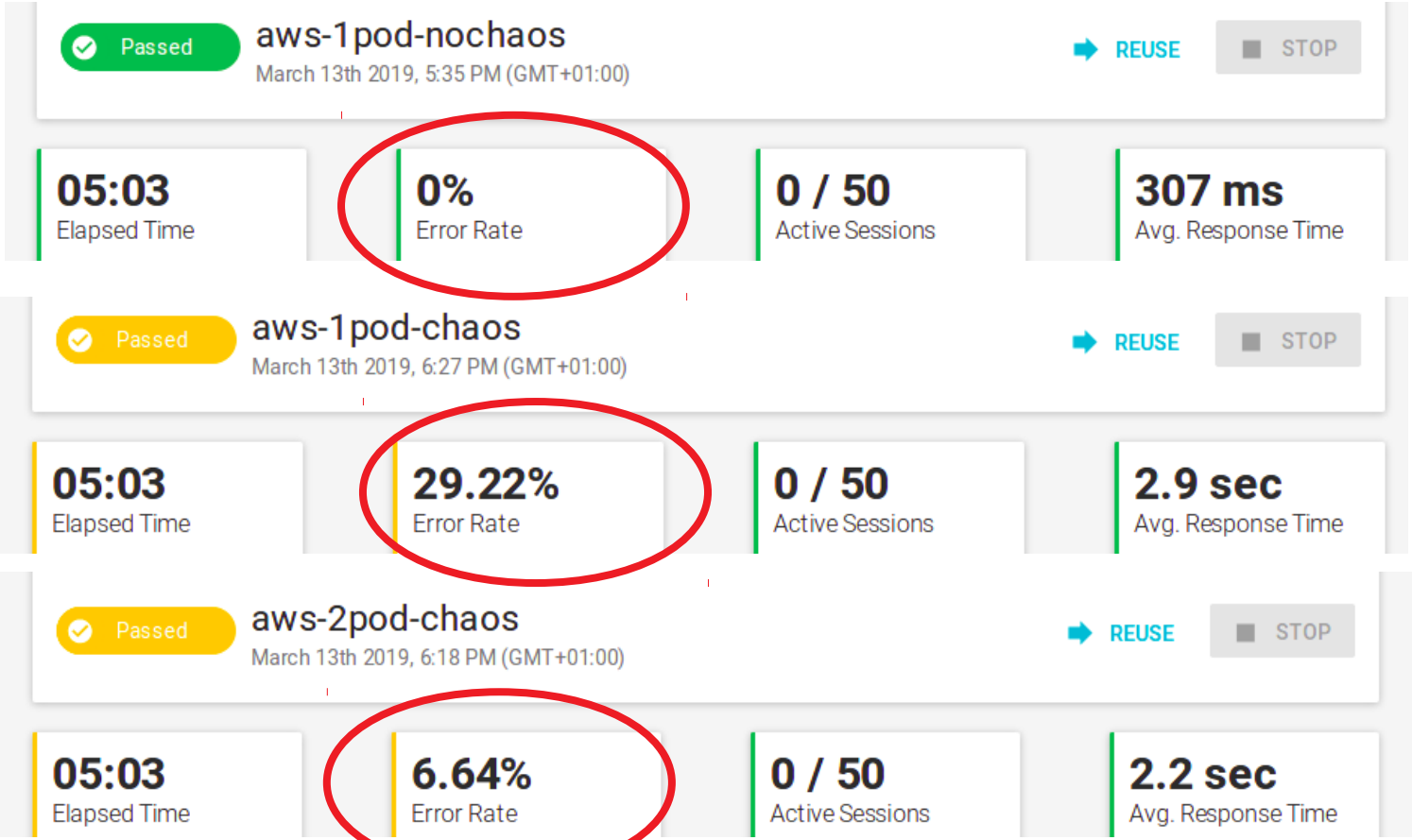
```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: ratings
spec:
  hosts:
    - ratings
  http:
  - match:
    - uri:
        prefix: /app
  - route:
    - destination:
        host: ratings
        port:
          number: 80
    retries:
      attempts: 3
      perTryTimeout: 2s
```

service

Retries config

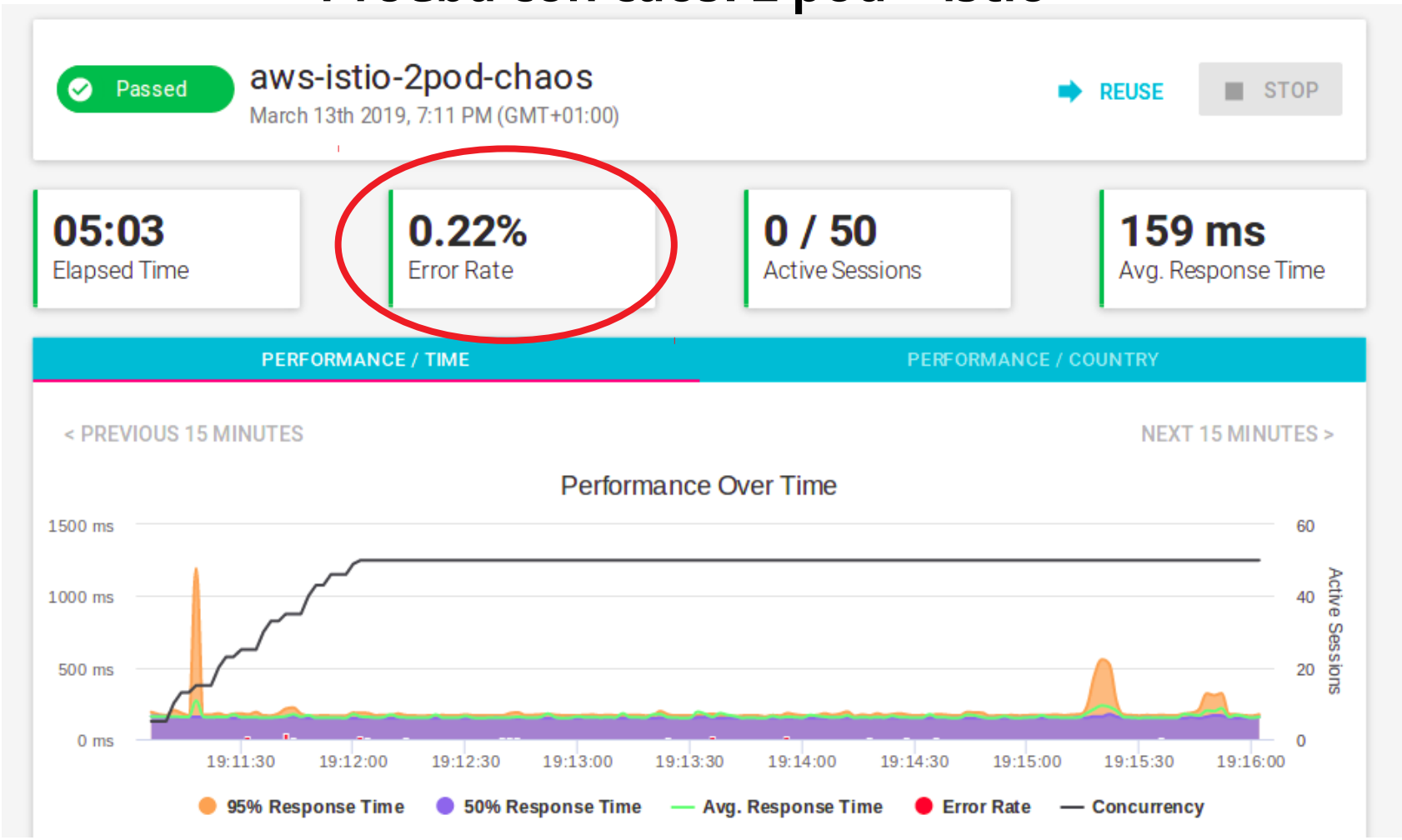
Istio

- Pruebas de pod-chaos-monkey sin reintentos



Istio

Prueba con caos: 2 pod + istio



Istio

- Circuit breacker

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: ratings
spec:
  hosts:
    - ratings
  http:
  - match:
    - uri:
        prefix: /app
    route:
    - destination:
        host: ratings
        subset: v1
```

Tenemos que crear un recurso nuevo

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: ratings
spec:
  host: ratings
  subsets:
  - name: v1
    trafficPolicy:
      connectionPool:
        tcp:
          maxConnections: 100
```

Istio

- Istio también se puede usar como herramienta para **insertar caos en las peticiones de red**
  - Insertar errores de API rest
  - Insertar latencia

- Podemos hacer **pruebas de tolerancia a fallos** con Istio

Istio

- **Errores en peticiones de red**
  - Creamos gateway, deployment y service
  - Creamos virtual service

```
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: nginx-color-http-error
spec:
  hosts:
  - "*"
  gateways:
  - nginx-color-gateway
  http:
  - match:
    - uri:
        prefix: /app
    route:
    - destination:
        host: nginx
        port:
          number: 80
    fault:
      abort:
        percent: 50
        httpStatus: 503
```

Demo time
Inyección de errores

```
$ kubectl config use-context istio.k8s.codeurjc.es

$ kubectl create -f nginx-color-gateway.yaml

$ kubectl apply -f nginx-color2.yml

$ kubectl apply -f nginx-color2-vs.yml

$ kubectl apply -f nginx-color2-vs-error.yml
```

ElasTest

https://elastest.io/

Universidad Rey Juan Carlos

code URJC

Laboratorio de Software

Master en Desarrollo y Despliegue de Aplicaciones en la Nube

Online / Clases en directo

Septiembre 2019

https://www.codeurjc.es/mastercloudapps/

t3chfest

code URJC

Testeando aplicaciones Kubernetes: escalabilidad y tolerancia a fallos

Gracias!!!

Micael Gallego

Leganes - 15 Marzo 2019