

The logo for Expo Q&A 222. 'expo' is in a white lowercase sans-serif font. 'Q&A' is in a white stylized font where the ampersand is a square with a cross inside, and the letters are bold. A registered trademark symbol (®) is to the right of 'Q&A'. '222' is in a white outlined sans-serif font.

# expo Q&A<sup>®</sup> 222

MADRID 31st MAY, 1st & 2nd JUNE 2022

A photograph of a large audience seated in rows of chairs, facing towards the right side of the frame. The audience is diverse in age and appearance. The setting appears to be a large conference hall or auditorium with a high ceiling and stage lights visible in the background.

[www.expoqa.com](http://www.expoqa.com)

# How to implement and test scalable and fault-tolerant applications deployed on Kubernetes

**Micael Gallego**

# ¿Quién soy?



# Desarrollador profesor universitario formador, consultor

 @michael\_gallego

 michael.gallego@urjc.es

 @michaelgallego





# CloudApps

M A S T E R

- **Módulo I: Calidad Software: Diseño, Arquitectura, Pruebas y XP**
- **Módulo II: Servicios Web: Tecnologías, Pruebas y Arquitecturas**
- **Módulo III: Aplicaciones Nativas de la Nube**
- **Módulo IV: DevOps, Integración y Despliegue Continuo**

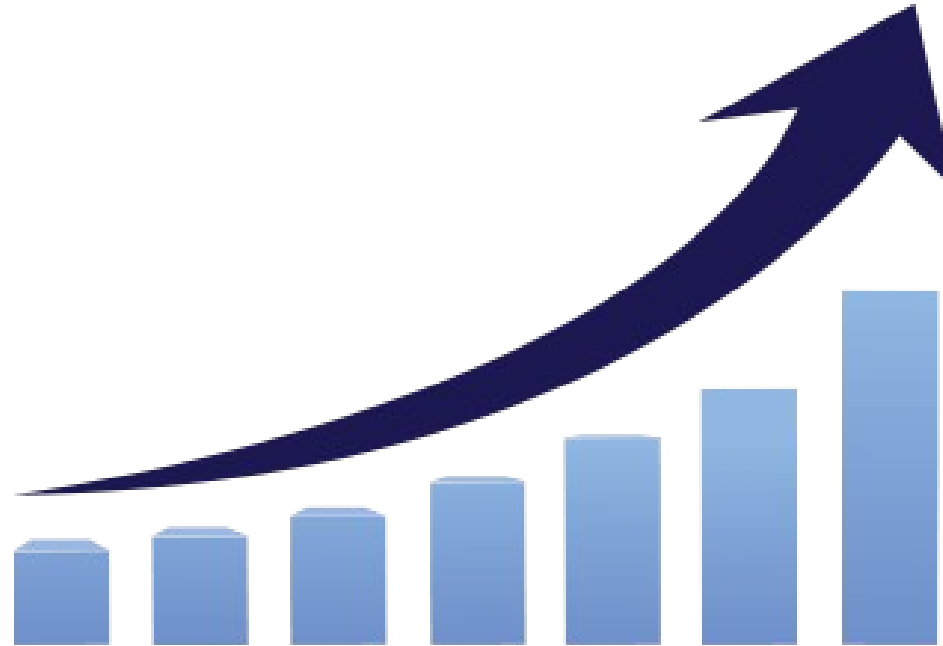
**100% Online / 1 año**  
**Clases en directo**  
**Prácticas semanales**

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

# Escalabilidad y Tolerancia a fallos en Kubernetes



- Escalabilidad en Kubernetes
- Autoescalado
- Escalado con estado
- Pruebas de escalabilidad
- Tolerancia a fallos
- Pruebas de tolerancia a fallos
- Service Mesh para tolerancia a fallos



# Escalabilidad en Kubernetes

# Preparando el entorno

- Las instrucciones están diseñadas para minikube
  - Hay que arrancarlo con la opción

```
$ minikube start --extra-config=kubelet.housekeeping-interval=1s
```

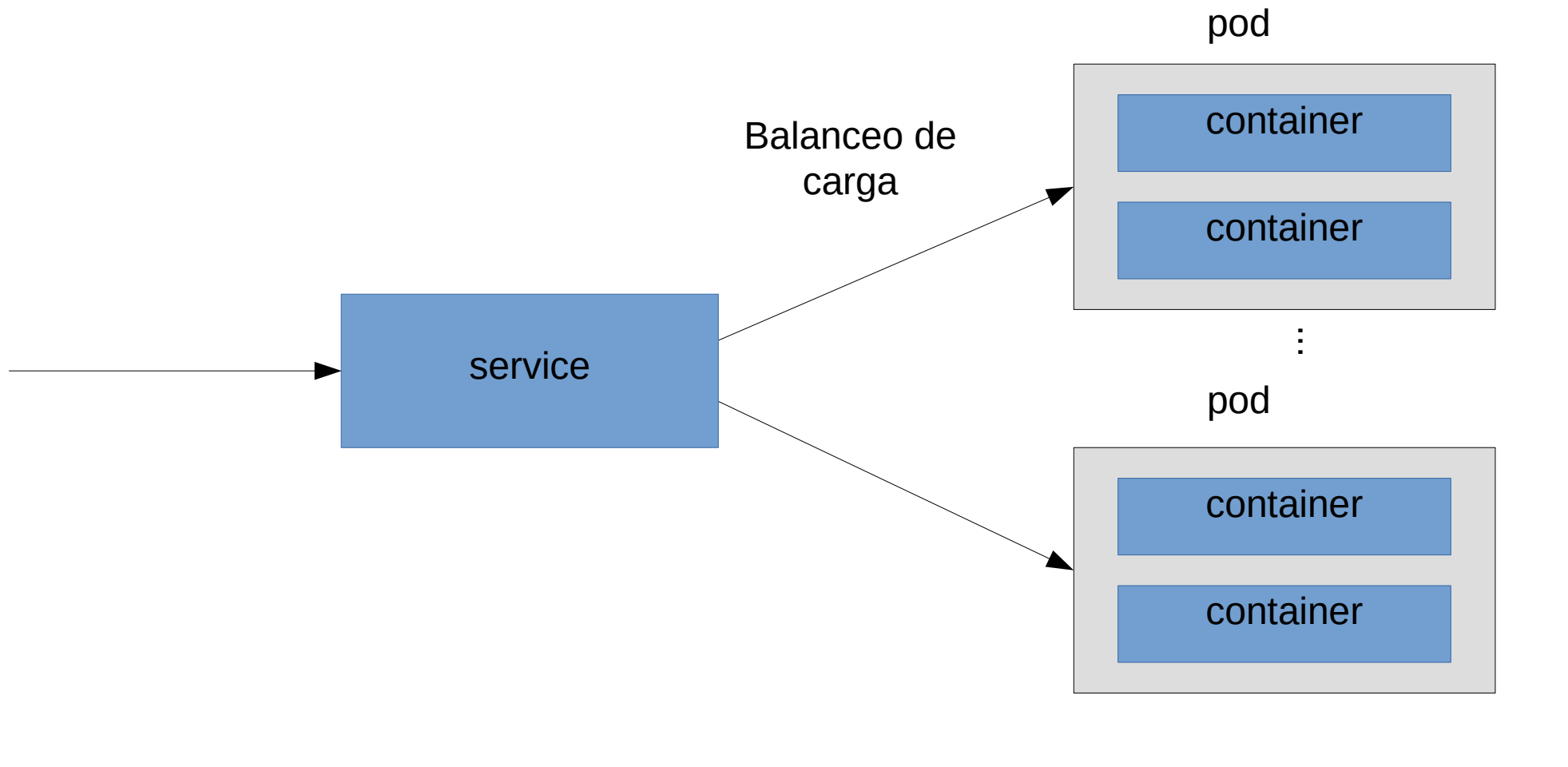
- Esto mitiga el issue reportado en la versión 1.25.2 (última estable al escribir esto)
- Los comandos asumen que se está en la raíz del repositorio

<https://github.com/MasterCloudApps/expoqa2022>

# Escalabilidad



#expoQA22





# 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

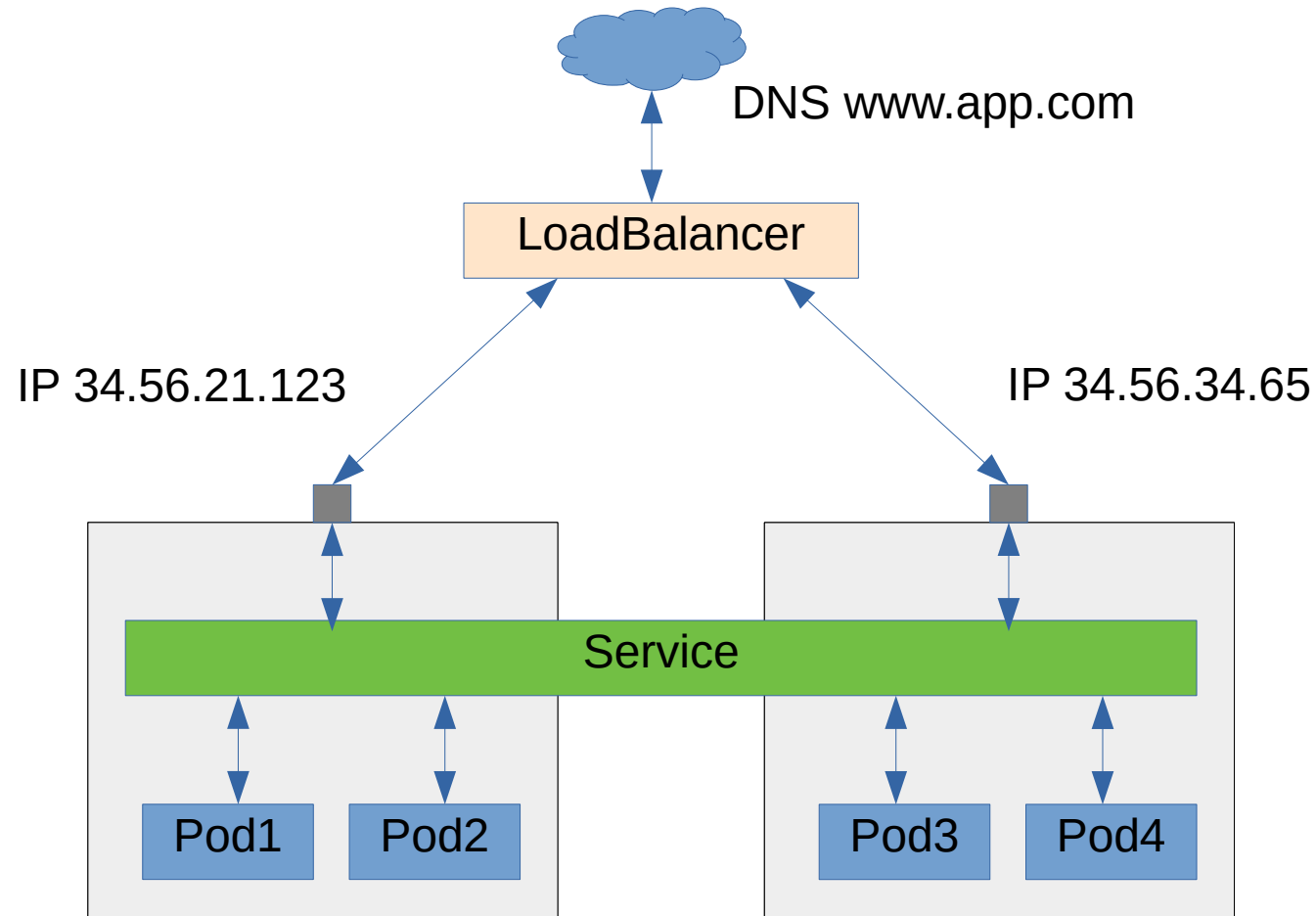


## Proxy-mode

- **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 apply -f ejem1-webgatos/webgatos.yaml  
  
$ minikube service webgatos --url  
  
$ kubectl get pods  
  
$ kubectl scale deployment/webgatos --replicas=2  
  
$ kubectl get pods
```

# Escalabilidad y Tolerancia a fallos en Kubernetes



- Escalabilidad en Kubernetes
- **Autoescalado**
- Escalado con estado
- Pruebas de escalabilidad
- Tolerancia a fallos
- Pruebas de tolerancia a fallos
- Service Mesh para tolerancia a fallos

- **Limitar los recursos a los pods**
  - Es buena práctica limitar los recursos de computación que consume un pod
  - Permite conocer si un pod tiene los recursos suficientes en el cluster para ofrecer servicio
  - Evita ataques de DDoS que afecten a otros pods del mismo nodo

<https://cloud.google.com/blog/products/containers-kubernetes/kubernetes-best-practices-resource-requests-and-limits>

# Autoescalado

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

- **Requests:** Recursos mínimos garantizados para el pod en el nodo
- **Limits:** Recursos máximos que podrá usar el pod en el nodo

<https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/>

<https://cloud.google.com/blog/products/containers-kubernetes/kubernetes-best-practices-resource-requests-and-limits>

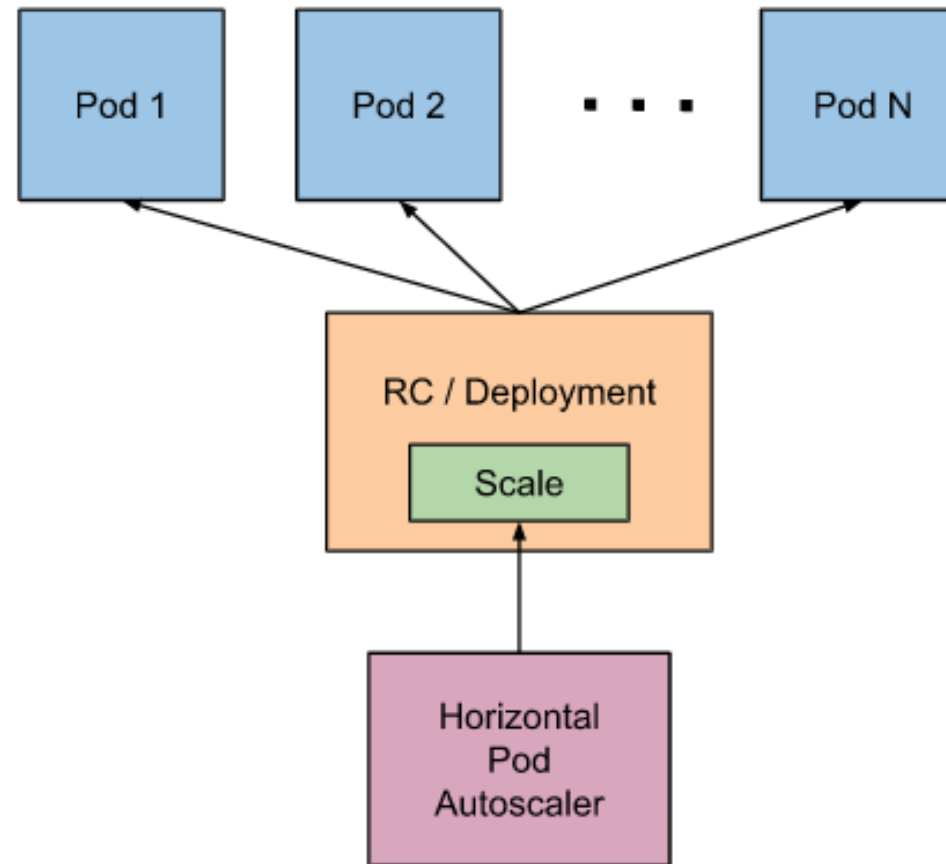
- **Horizontal Pod Autoscaler (HPA)**
  - El número de **réplicas** puede cambiar de forma **automática** en base a alguna **métrica de carga** objetivo
  - Aumenta réplicas si se sobrepasa la métrica
  - Reduce réplicas si se está por debajo
  - Sólo funciona con pods con recursos limitados

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

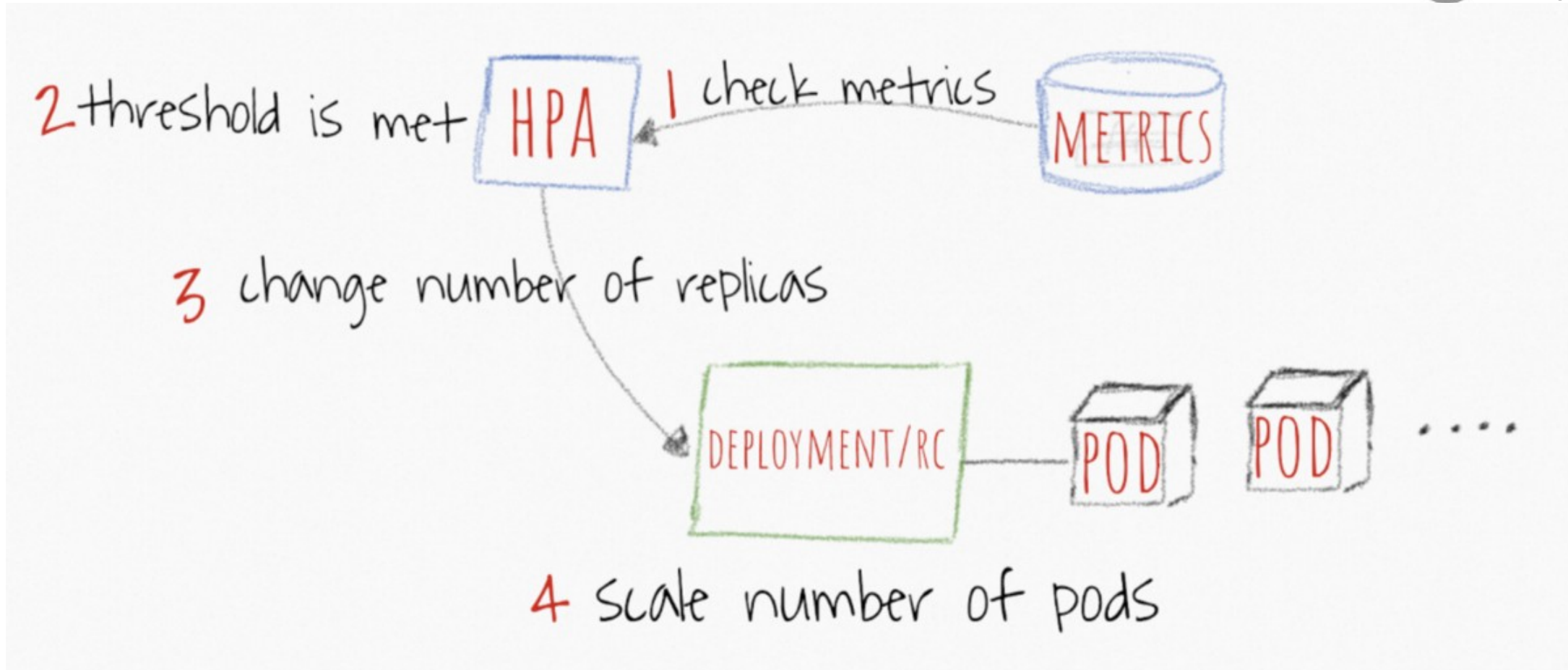


# Autoescalado

- Horizontal Pod Autoscaler (HPA)



# Autoescalado



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

# Autoescalado

- **Horizontal Pod Autoscaler**

- Especifica los límites de escalado (min y max)
- Se define la métrica objetivo



```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: php-apache
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: php-apache
  minReplicas: 1
  maxReplicas: 10
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 10
```

# Demo time

## Web raíz cuadrada con autoscaling

- Se necesita tener activado el metrics-server
  - Minikube

```
$ minikube addons enable metrics-server
```

# Demo time

## Web raíz cuadrada con autoscaling

- Se despliega una aplicación PHP que ejecuta una operación costosa en CPU (un millón de raíces cuadradas)
- Se hacen peticiones a esa aplicación para generar carga de CPU
- Se observa cómo se crean más réplicas de los pods

# Demo time

## Web raíz cuadrada con autoscaling

```
$ kubectl apply -f ejem2-hpa/deployment.yaml
```

```
$ kubectl apply -f ejem2-hpa/hpa-autoscaling.yaml
```

```
$ watch -n 1 kubectl get pods
```

```
$ watch -n 1 kubectl top pod
```

```
$ kubectl run load-generator -it --image=busybox:1.30 /bin/sh
```

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

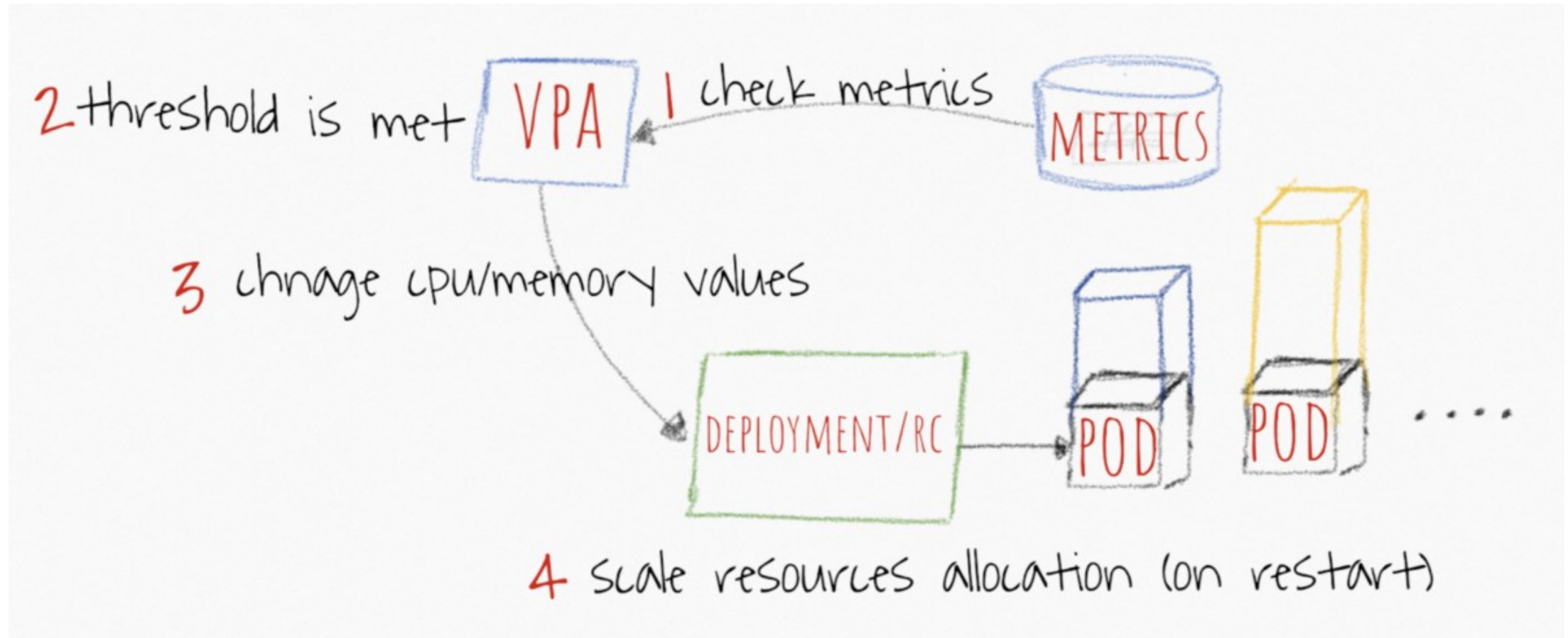
- Vertical Pod Autoscaler

- 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

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

<https://blogs.oracle.com/cloud-infrastructure/verticalpodautoscaler-on-oke>

# Autoescalado



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

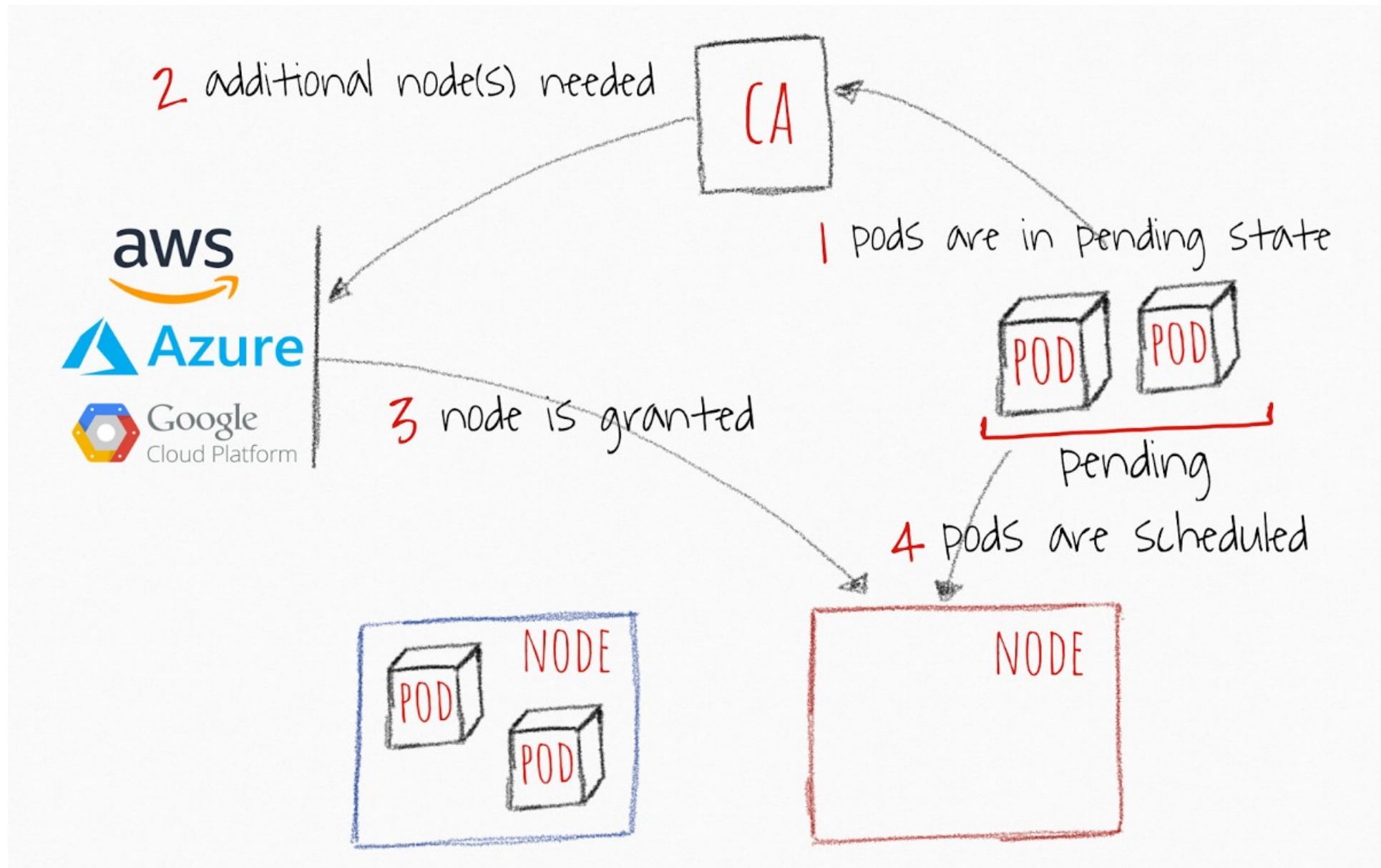


## • Cluster autoscaling

- 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>

- ¿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>

# Autoescalado

- Autoescalado en base a eventos



<https://keda.sh/>

- Kubernetes Event-driven Autoscaling



## Event-driven

Intelligently scale your event-driven application



## Autoscaling Made Simple

Bring rich scaling to every container in your [Kubernetes](#) cluster



## Built-in Scalers

Out-of-the-box scalers for various vendors, databases, messaging systems, telemetry systems, and more



## Multiple Workload Types

Support for variety of workload types such as deployments and jobs



## Vendor-Agnostic

Support for triggers across multiple vendors



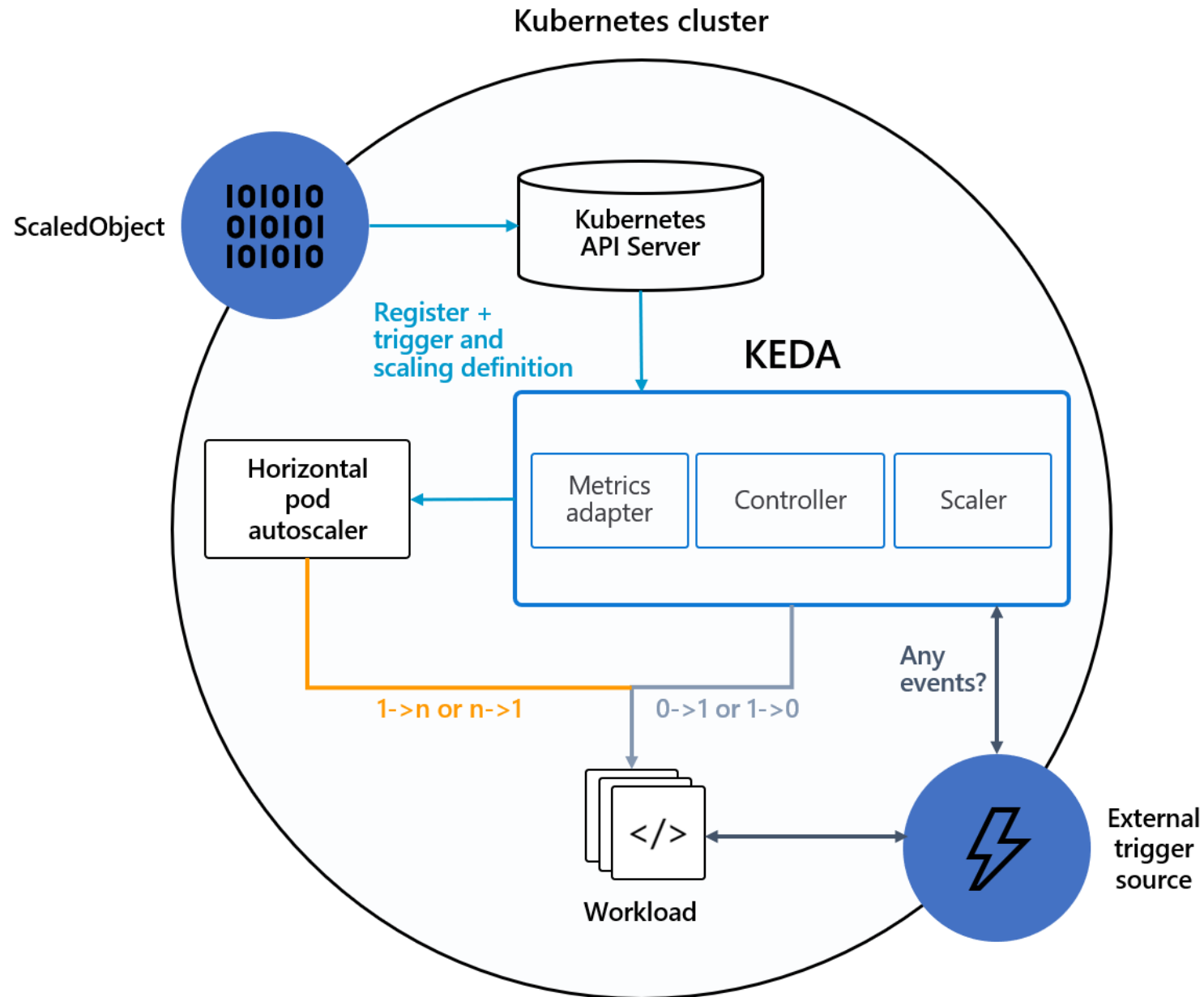
## Azure Functions Support

Run and scale your Azure Functions on Kubernetes in production workloads

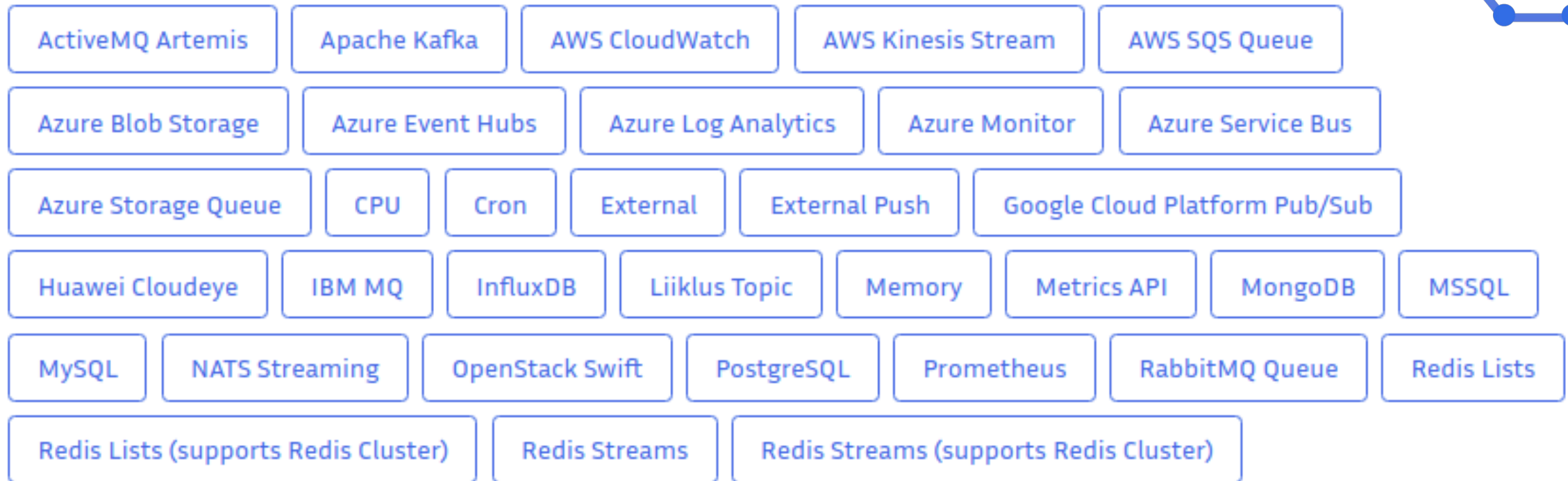
# KEDA



#expoQA22



## • Fuentes de eventos





# Escalabilidad y Tolerancia a fallos en Kubernetes



- Escalabilidad en Kubernetes
- Autoescalado
- **Escalado con estado**
- Pruebas de escalabilidad
- Tolerancia a fallos
- Pruebas de tolerancia a fallos
- Service Mesh para tolerancia a fallos

# Escalado con estado

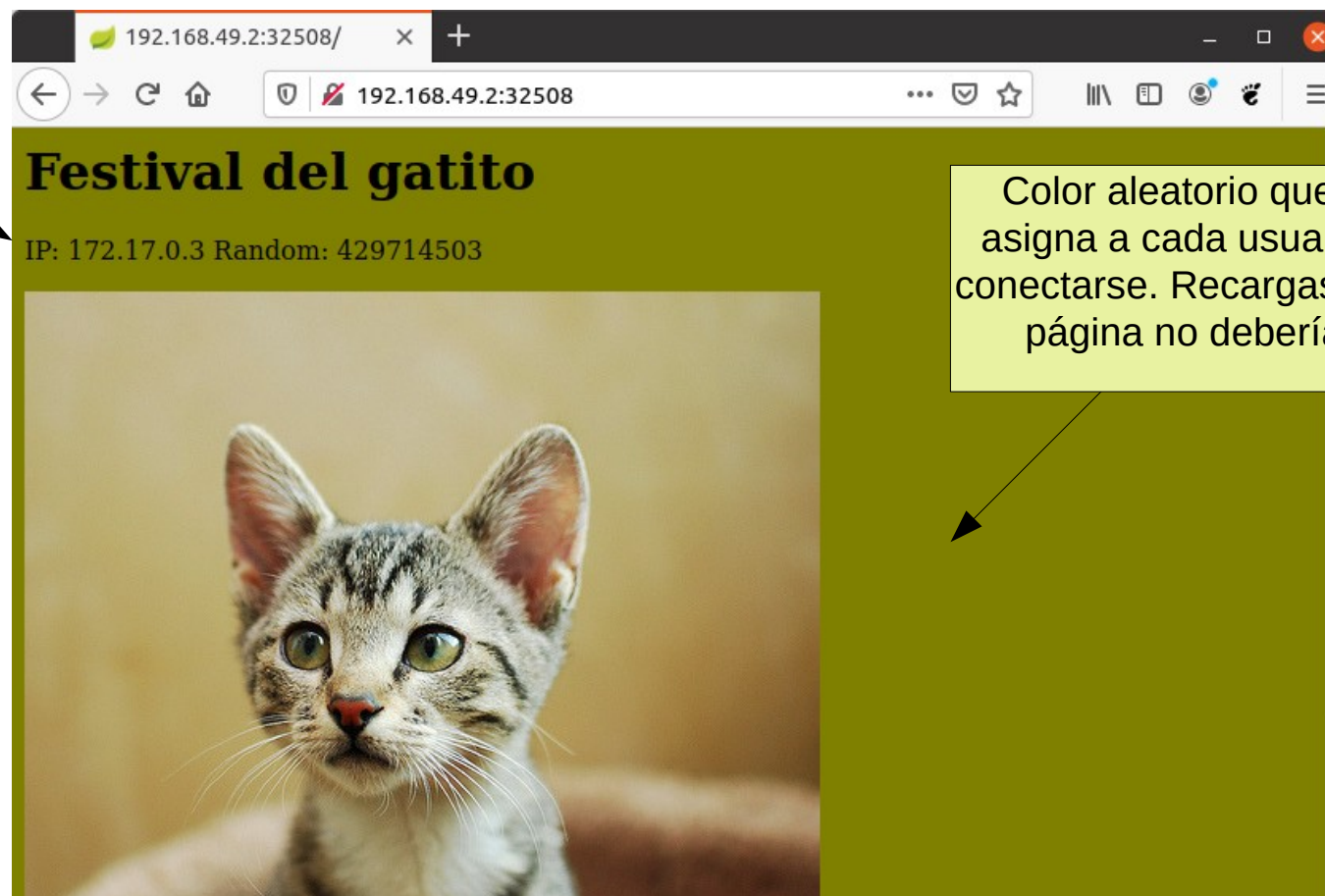
- Se asume que los pods de un deployment no tienen estado (**stateless**)
  - Cualquier pod puede ser eliminado (**scale in**)
  - Un nuevo pod pueda atender tráfico de un usuario que antes atendía otro pod (**scale out**)
- Pero hay veces que nuestras apps tienen estado (**statefull**)

## ¿Qué hacemos?

# Escalado con estado



IP del POD  
que atiende  
la petición



Color aleatorio que se  
asigna a cada usuario al  
conectarse. Recargas de la  
página no deberían

# Escalado con estado

```
@GetMapping("/")
public String mainPage(HttpSession session, Model model) throws UnknownHostException {

    String userColor;

    if(session.isNew()) {
        userColor = colors[nextColor];
        nextColor = (nextColor + 1) % colors.length;
        session.setAttribute("userColor", userColor);
    } else {
        userColor = (String) session.getAttribute("userColor");
    }

    model.addAttribute("userColor", userColor);
    model.addAttribute("ip", InetAddress.getLocalHost().getHostAddress());
    model.addAttribute("random", new Random().nextInt());

    return "index";
}
```

# Escalado con estado

- **Ejemplo 3: Arquitectura no escalable**
  - La aplicación mantiene estado en memoria
  - Si se crean varias réplicas, cuando el usuario recarga la página puede ser atendido por otra réplica y se le asigna un nuevo color → Mal funcionamiento

- Ejemplo 3: Arquitectura no escalable

- Ejecución con 2 réplicas

```
$ kubectl apply -f ejem3-webgatos2-non-scalable/k8s  
  
$ minikube service webapp --url
```

- En **Chrome** la recarga de la página siempre se atiende por el mismo pod con Spring debido a las conexiones persistentes
- En **Firefox** es necesario Ctrl+F5 para realizar otra conexión (y hacer la petición a otro pod)

[https://en.wikipedia.org/wiki/HTTP\\_persistent\\_connection](https://en.wikipedia.org/wiki/HTTP_persistent_connection)

# Escalado con estado

- Ejemplo 3: Arquitectura no escalable



- Ejemplo 3: Arquitectura no escalable

Se testea que  
peticiones  
sucesivas obtienen  
el mismo color

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

    String userColor = null;

    for (int i = 0; i < 10; i++) {

        loadPage();

        String color = extractColor();
        String ip = extractIP();

        System.out.println("Loaded page " + (i+1) + " time(s) with color "+color+" and IP "+ip);

        if (userColor == null) {
            userColor = color;
        } else {
            assertThat(userColor).isEqualTo(color)
                .withFailMessage("El color del usuario ha cambiado");
        }
        Thread.sleep(500);
    }
}
```



# Escalado con estado



- Ejemplo 3: Arquitectura no escalable

```
$ minikube service webapp
```

NAMESPACE	NAME	TARGET PORT	URL
default	webapp	webapp/8080	http://192.168.49.2:31703

```
Opening service default/webapp in default browser...
```

```
$ cd ejem3-webgatos2-non-scalable
```

```
$ mvn test -Dweburl=http://192.168.49.2:31703
```

```
...
```

```
[INFO] Running es.codeurjc.kubetest.E2EHeadlessTest
```

```
Loaded page 1 time(s) with color FFFF00 and IP 172.17.0.2
```

```
Loaded page 2 time(s) with color FFFF00 and IP 172.17.0.2
```

```
Loaded page 3 time(s) with color FFFF00 and IP 172.17.0.2
```

```
Loaded page 4 time(s) with color FF0000 and IP 172.17.0.3
```

```
[ERROR] Tests run: 1, Failures: 1, Errors: 0, Skipped: 0, Time elapsed:
```

```
3.582 s <<< FAILURE! - in es.codeurjc.kubetest.E2EHeadlessTest
```

```
[ERROR] test Time elapsed: 3.575 s <<< FAILURE!
```

```
...
```

# Escalado con estado

- Soluciones

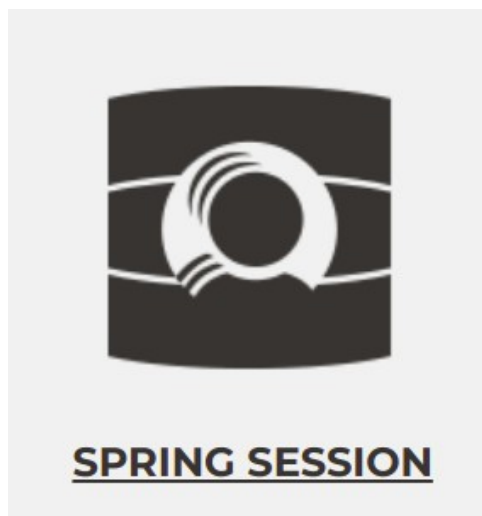
- 1) Mover el estado fuera de la aplicación (convertirla en **stateless**)
- 2) **Replicar** el estado entre las réplicas
- 3) Usar mecanismos de escalado con estado en Kubernetes (**StatefulSet**)

# Escalado con estado

- **1) Mover el estado fuera de la app**
  - Mueve la información del usuario en memoria a un **recurso compartido**
  - Cualquier réplica que reciba la petición podrá **recuperar los datos** de ese recurso compartido
  - En aplicaciones web la información de un usuario se guarda en la **sesión http** (que suele gestionarse con una cookie)

# Escalado con estado

- 1) Mover el estado fuera de la app



Librería Spring que permite gestionar la sesión http en un recurso compartido



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

# Escalado con estado

- **Ejemplo 4: Arquitectura escalable**
  - Compartir la sesión en MySQL

```
$ kubectl apply -f ejem4-webgatos2-scalable/k8s  
$ minikube service webapp
```

# Escalado con estado

- Ejemplo 4: Arquitectura escalable



# Escalado con estado



- Ejemplo 4: Arquitectura escalable

```
$ minikube service webapp
```

NAMESPACE	NAME	TARGET PORT	URL
default	webapp	webapp/8080	http://192.168.49.2:31703

```
Opening service default/webapp in default browser...
```

```
$ cd ejem4-webgatos2-scalable
```

```
$ mvn test -Dweburl=http://192.168.49.2:31703
```

```
...
```

```
[INFO] Running es.codeurjc.kubetest.E2EHeadlessTest
```

```
Loaded page 1 time(s) with color 0000FF and IP 172.17.0.4
```

```
Loaded page 2 time(s) with color 0000FF and IP 172.17.0.4
```

```
...
```

```
Loaded page 9 time(s) with color 0000FF and IP 172.17.0.4
```

```
Loaded page 10 time(s) with color 0000FF and IP 172.17.0.4
```

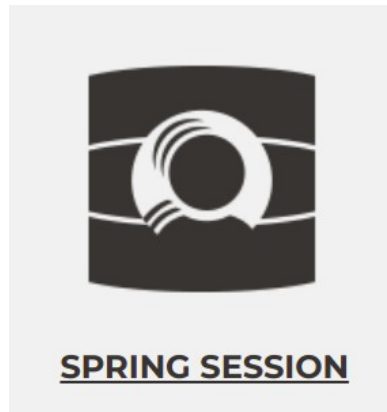
```
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed:
```

```
6.27 s - in es.codeurjc.kubetest.E2EHeadlessTest
```

```
...
```

# Escalado con estado

- 2) Replicar el estado
  - Spring session se puede usar con Hazelcast, un servicio que puede usarse como librería (embebido en la aplicación) que sincroniza la información entre todas las réplicas



<https://hazelcast.com/>

<https://spring.io/projects/spring-session-hazelcast>

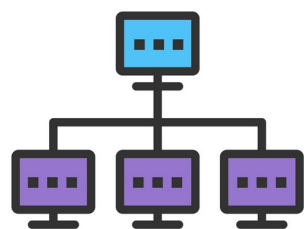


# Escalado con estado



- 2) Replicar el estado

- Hazelcast dispone de mecanismos para descubrir las réplicas de forma automática
- La estrategia concreta depende del entorno



LOCAL  
NETWORK

Usando multicast



Usando API AWS  
(Necesarios permisos)

<https://github.com/hazelcast/hazelcast-aws>



kubernetes

Usando API K8s  
(Necesarios permisos)

<https://github.com/hazelcast/hazelcast-kubernetes>

# Escalado con estado

- 2) Replicar el estado



- Replicar la sesión con Hazelcast

```
$ kubectl apply -f ejem4-webgatos2-scalable-hazelcast/k8s  
  
$ minikube service webapp --url  
  
$ cd ejem4-webgatos2-scalable-hazelcast  
  
$ mvn test -Dwebapp=http://192.168.49.2:31703
```

- 3) Escalado stateful en Kubernetes
  - Los **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

- 3) Escalado stateful en Kubernetes
  - La gestión de servicios stateful en Kubernetes es **compleja** porque depende de las características del **software concreto**
  - Un **operador de Kubernetes** (*operator*) es un plugin que permite gestionar un conjunto de réplicas (cluster) de un servicio stateful

<https://kubernetes.io/docs/concepts/extend-kubernetes/operator/>

# Operadores Kubernetes

- Existen **operadores** para el despliegue y la gestión de los **servicios con estado** más habituales de los servicios de Internet



# Operadores Kubernetes

- Buscadores de operadores



<https://artifacthub.io/packages/search?page=1&operators=true>



<https://operatorhub.io/>

# Operadores Kubernetes

- Recursos personalizados

- Cuando se instala un operador en Kubernetes se añade un nuevo tipo de recurso personalizado (**Custom Resource Definition, CRD**)
- Su gestión es similar a los recursos nativos de Kubernetes

```
apiVersion: rabbitmq.com/v1beta1
kind: RabbitmqCluster
metadata:
  name: rabbitmqcluster-sample
spec:
  replicas: 3
```

# Operadores Kubernetes

- Algunos operadores disponen de un **cliente para consola (CLI)** o un **plugin de kubectl** para simplificar las tareas administrativas
  - **Ejemplo:** Plugin de RabbitMQ

```
$ kubectl rabbitmq create <name>
```

```
$ kubectl rabbitmq list
```



# Operadores Kubernetes

- Official mysql-operator

<https://github.com/mysql/mysql-operator>

- Presslabs MySQL Operator

<https://github.com/presslabs/mysql-operator>

- Percona XtraDB Cluster Operator

<https://www.percona.com/doc/kubernetes-operator-for-pxc/index.html>



# Operadores Kubernetes

- **Official MySQL Operator**
  - Instalación del operador



```
$ helm repo add mysql-operator https://mysql.github.io/mysql-operator/  
$ helm repo update  
$ helm install mysql-operator mysql-operator/mysql-operator --namespace mysql-operator --create-namespace
```

<https://github.com/mysql/mysql-operator>

# Operadores Kubernetes

- Official MySQL Operator
  - Desplegar un cluster de MySQL



ejem5-mysql/mycluster.yaml

```
$ kubectl create secret generic mypws \
  --from-literal=rootUser=root \
  --from-literal=rootHost=% \
  --from-literal=rootPassword="sakila"

$ kubectl apply -f ejem5-mysql/mycluster.yaml
```

```
apiVersion: mysql.oracle.com/v2
kind: InnoDBCluster
metadata:
  name: mycluster
spec:
  secretName: mypws
  tlsUseSelfSigned: true
  instances: 3
  router:
    instances: 1
```

# Operadores Kubernetes

- Official MySQL Operator
  - Monitorizar el despliegue



```
$ kubectl get innodbcluster -watch
```

NAME	STATUS	ONLINE	INSTANCES	ROUTERS	AGE
mycluster	PENDING	0	3	1	45s
mycluster	INITIALIZING	0	3	1	100s
mycluster	INITIALIZING	0	3	1	101s
mycluster	INITIALIZING	0	3	1	101s
mycluster	INITIALIZING	0	3	1	101s
mycluster	INITIALIZING	0	3	1	101s
mycluster	INITIALIZING	0	3	1	101s
mycluster	INITIALIZING	0	3	1	105s
mycluster	ONLINE	1	3	1	106s
mycluster	ONLINE	1	3	1	115s
mycluster	ONLINE	2	3	1	2m1s
mycluster	ONLINE	2	3	1	2m7s
mycluster	ONLINE	3	3	1	2m12s

# Escalabilidad y Tolerancia a fallos en Kubernetes



- Escalabilidad en Kubernetes
- Autoescalado
- Escalado con estado
- **Pruebas de escalabilidad**
- Tolerancia a fallos
- Pruebas de tolerancia a fallos
- Service Mesh para tolerancia a fallos

# 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

- **Artillery.io**

- Herramienta de carga implementada en node
- Configuración de escenario de carga en línea de comandos o fichero yaml
- Open source y comercial

```
$ sudo npm install -g artillery@latest
```





# Pruebas de escalabilidad

- **Artillery.io**

## config.yaml

```
config:
  plugins:
    expect: {}
  environments:
    k8s:
      target: "http://192.168.99.112:30839"
      tls:
        rejectUnauthorized: false
      phases:
        - duration: 120
          arrivalRate: 10
      http:
        pool: 8
      ensure:
        p95: 100
        maxErrorRate: 0
```

## scenario.yaml

```
scenarios:
  - name: "Load test"
    flow:
      - get:
          url: "/"
          expect:
            - statusCode: 200
```

# Pruebas de escalabilidad

- **Desplegamos app y simulamos tráfico**

```
$ kubectl apply -f ejem4-webgatos2-scalable/k8s  
  
$ kubectl scale deployment webapp --replicas=1  
  
$ WEB_APP_URL=$(minikube service webapp --url)  
  
$ cd ejem6-loadtest  
  
$ artillery run -e k8s --config config.yaml -t $WEB_APP_URL scenario.yaml -o output1.json  
  
$ artillery report ./output1.json
```

# Pruebas de escalabilidad

- Desplegamos app y simulamos tráfico

## Summary

Test duration	120 sec
Virtual Users created	1200
Virtual Users completed	1200

## Errors

✓ Test completed without network or OS errors.

# Escalabilidad y Tolerancia a fallos en Kubernetes



- Escalabilidad en Kubernetes
- Autoescalado
- Escalado con estado
- Pruebas de escalabilidad
- **Tolerancia a fallos**
- Pruebas de tolerancia a fallos
- Service Mesh para tolerancia a fallos



# Tolerancia a fallos

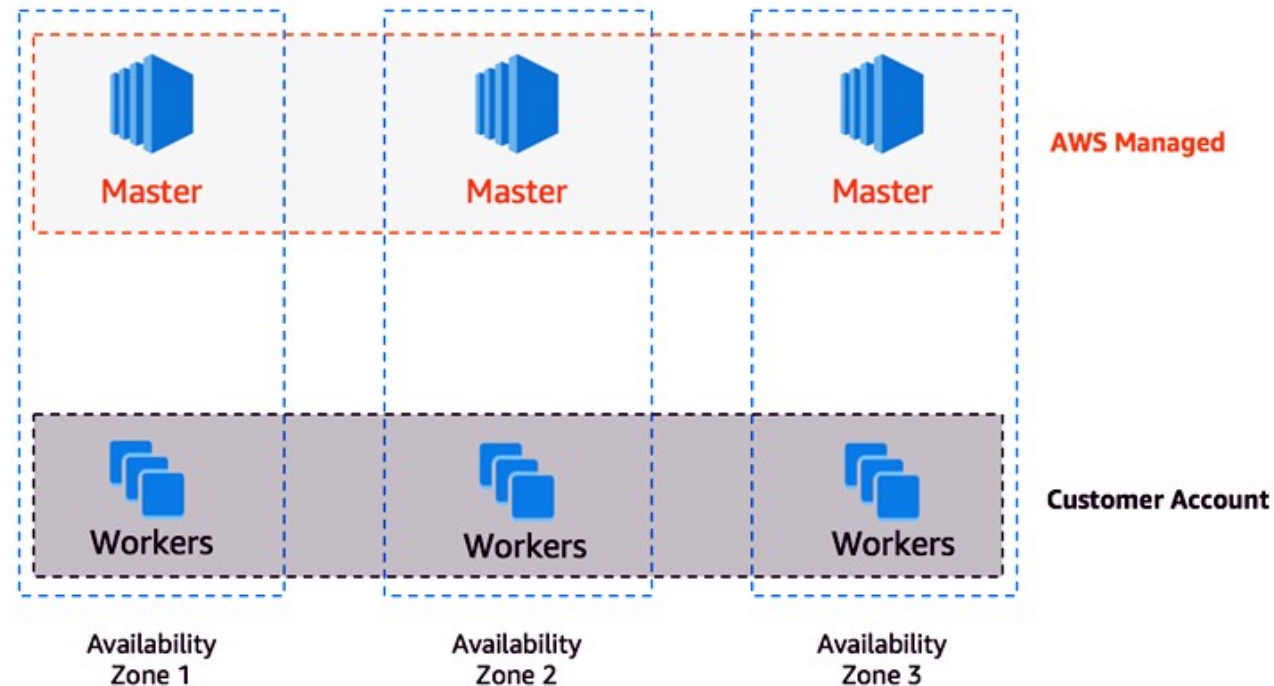
# Tolerancia a fallos

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



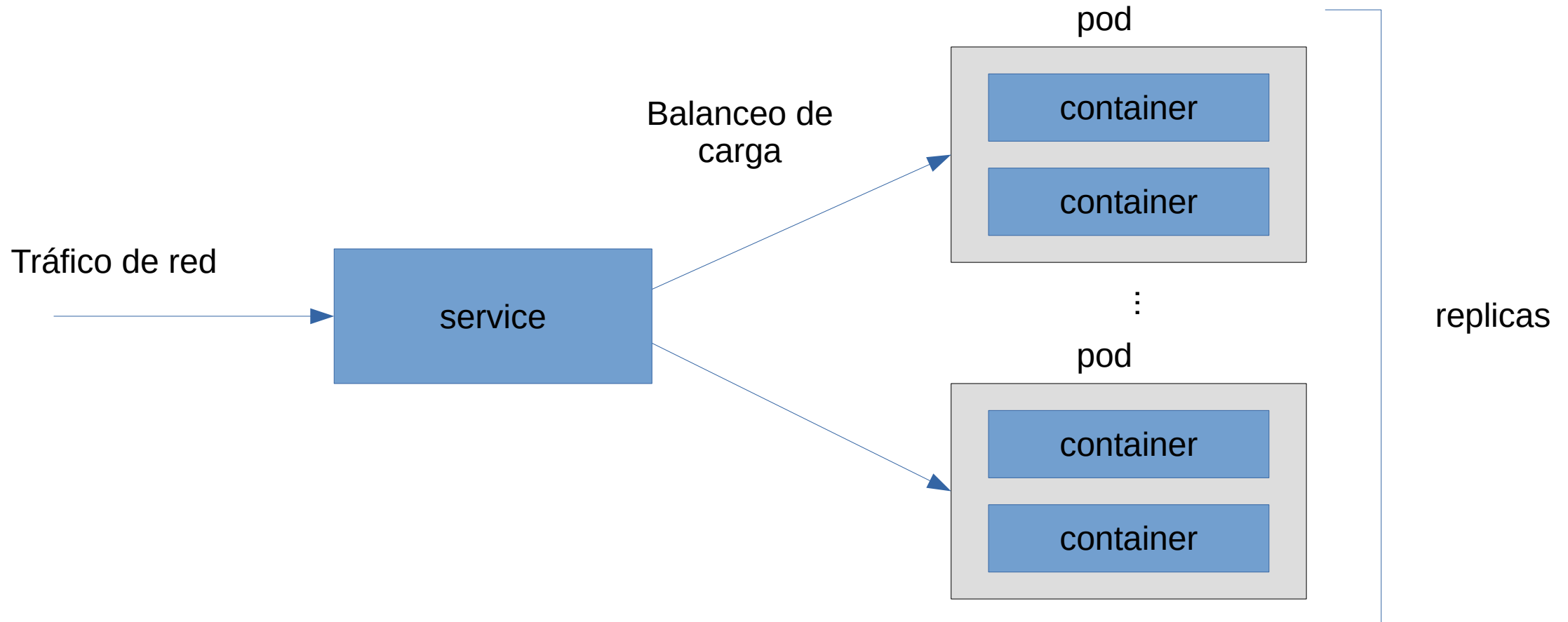
**A distributed, reliable key-value store for the  
most critical data of a distributed system**

<https://coreos.com/etcd/>



# Tolerancia a fallos

Las **réplicas de los pods** ejecutándose 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

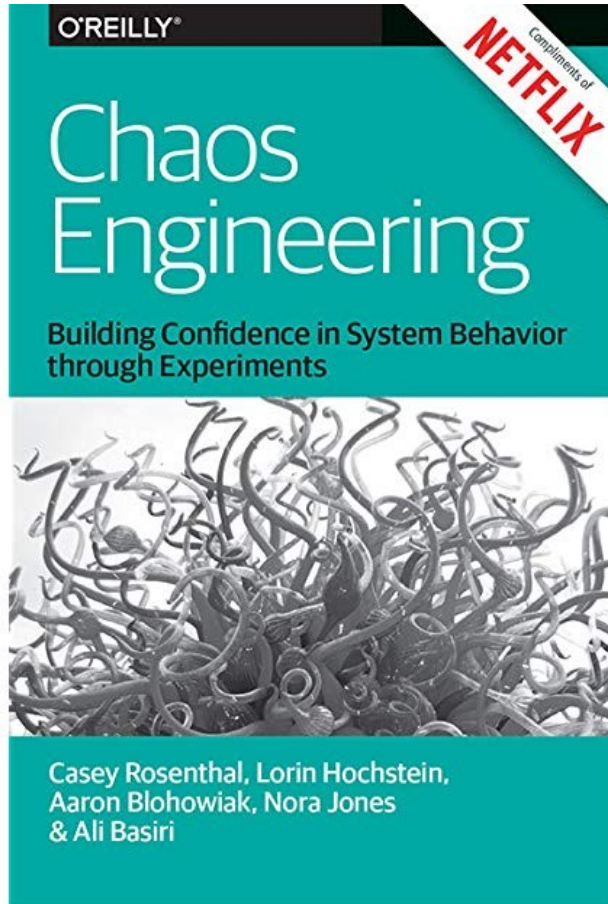
<https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/>

# Escalabilidad y Tolerancia a fallos en Kubernetes



- Escalabilidad en Kubernetes
- Autoescalado
- Escalado con estado
- Pruebas de escalabilidad
- Tolerancia a fallos
- **Pruebas de tolerancia a fallos**
- Service Mesh para 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/>

- Podemos ver los nodos de nuestro cluster en su web

### Clients

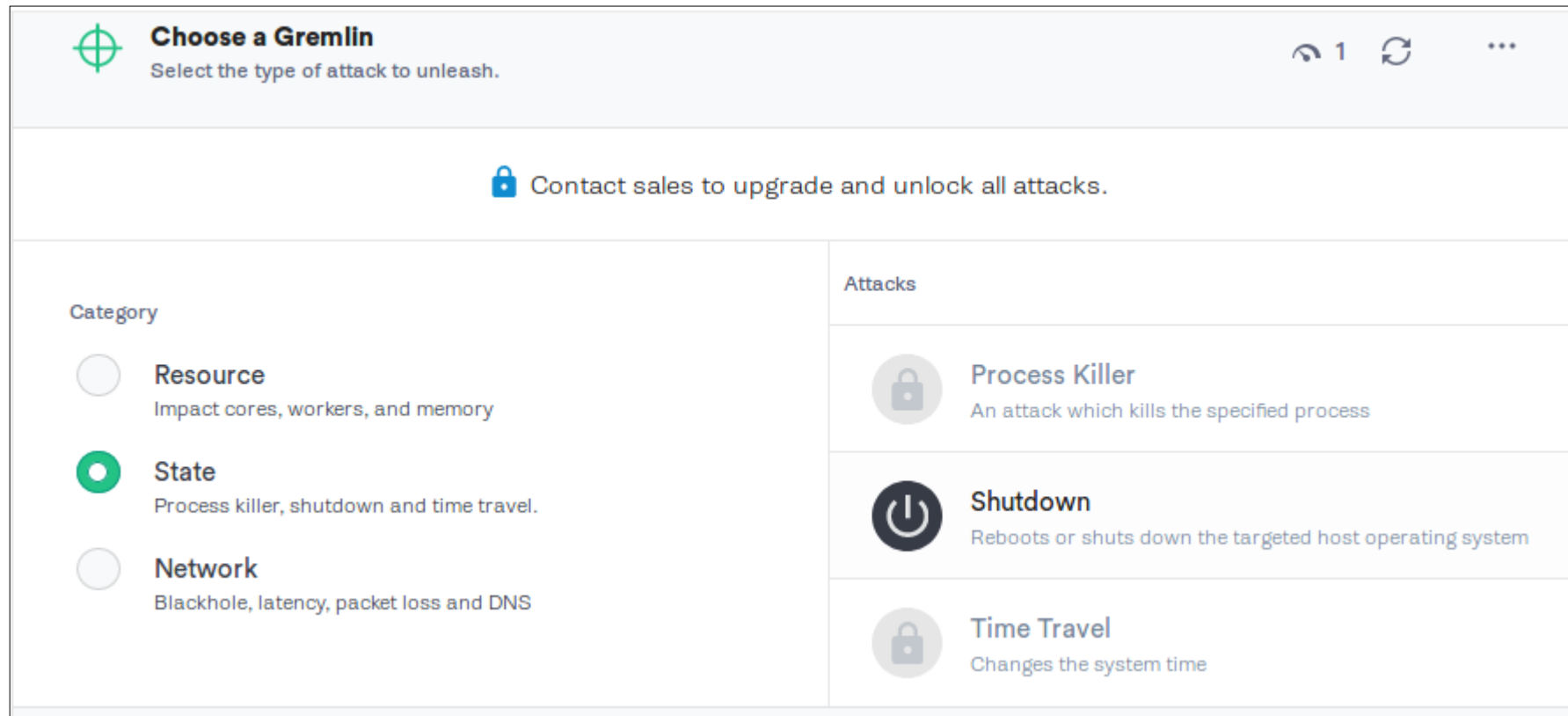
#### Infrastructure

 Search Infrastructure Clients

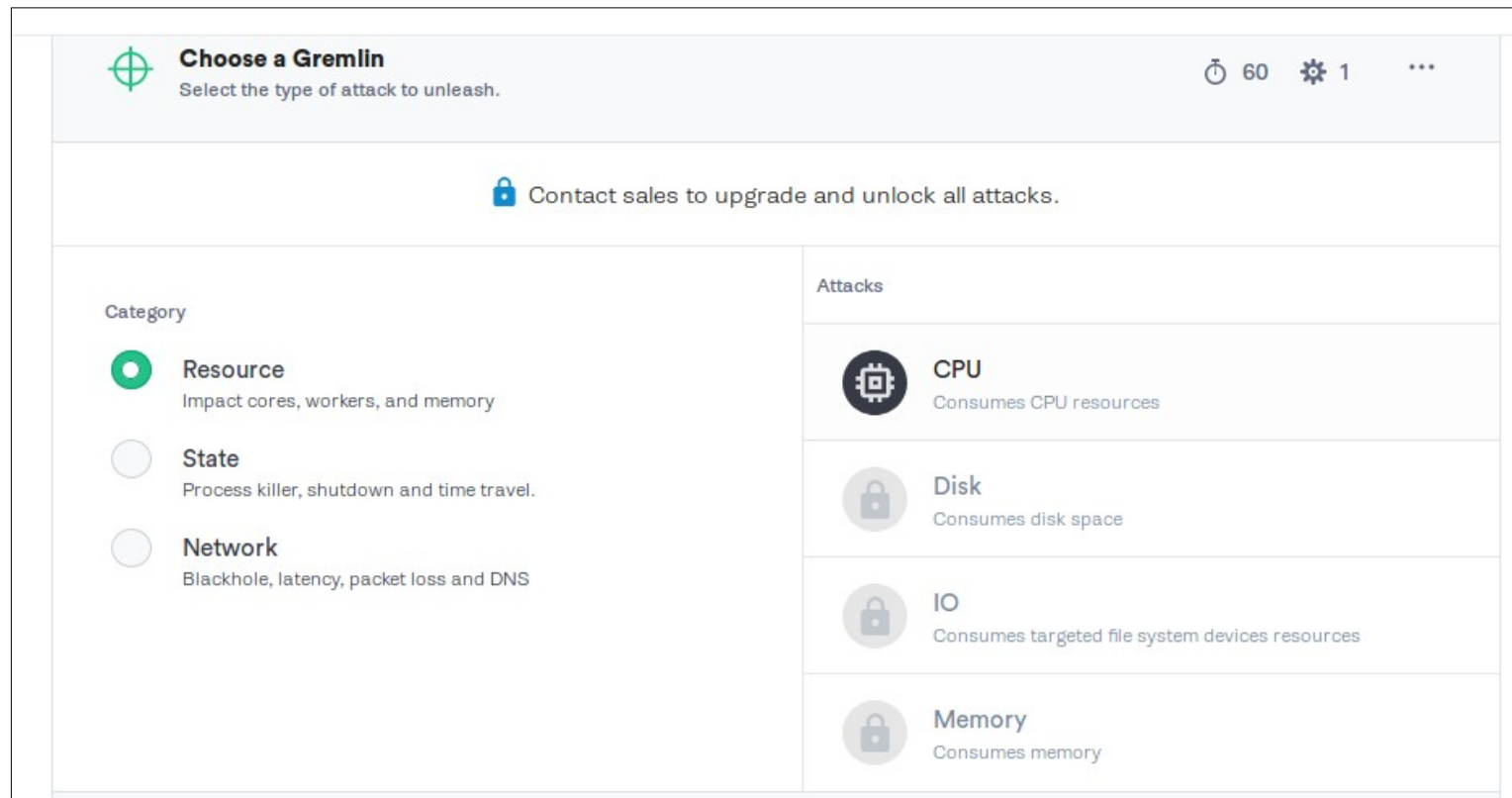
 gke-standard-cluster-2-default-pool-b6855ebb-j648 active	Deactivate
Docker gremlin-client-version:2.9.2 local-ip:10.36.0.11 local-hostname:gremlin-s9wvd	
 gke-standard-cluster-2-default-pool-b6855ebb-4lk9 active	Deactivate
Docker gremlin-client-version:2.9.2 local-ip:10.36.2.6 local-hostname:gremlin-dxqht	
 gke-standard-cluster-2-default-pool-b6855ebb-b6m2 active	Deactivate
Docker gremlin-client-version:2.9.2 local-ip:10.36.1.4 local-hostname:gremlin-7jccd	

# Gremlin

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

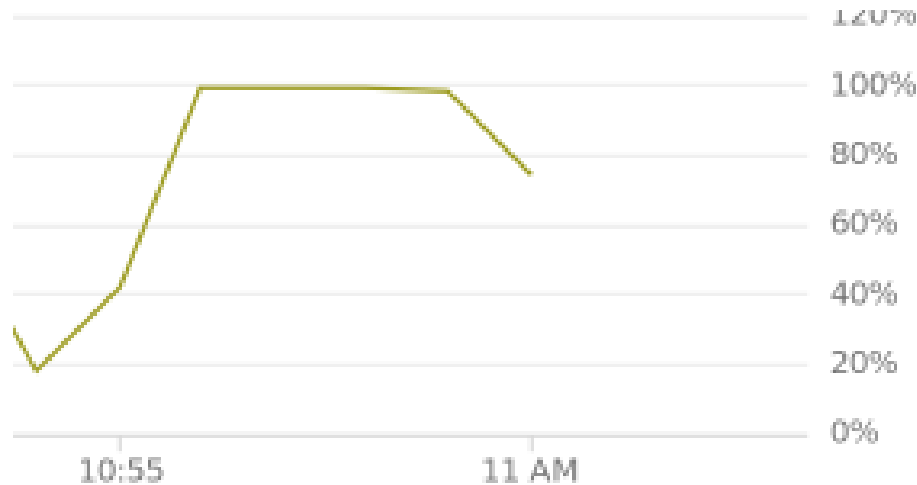


- Tipos de caos (versión gratuita)
  - Añadir carga de CPU



- 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 la aplicación de webgatos con **un único pod**
- Se eliminará un pod con label **app=webapp** cada **15 segundos**

```
$ kubectl apply -f ejem4-webgatos2-scalable/k8s  
  
$ kubectl scale deployment webapp --replicas=1  
  
$ TAG=app VALUE=webapp NAMESPACE=default DELAY=15 ejem7-chaos/chaos.sh
```



# Pod-Chaos-Monkey

- Vemos que los pods se **terminan** (por la herramienta de caos) y se **regeneran** (por el Deployment) cada 15 segundos

```
Every 1,0s: kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
webapp-cf8b764c4-8wfgx	1/1	Terminating	0	19s
webapp-cf8b764c4-jnftm	0/1	ContainerCreating	0	1s
webapp-cf8b764c4-mcqqr	1/1	Running	0	39s

# Pod-Chaos-Monkey

- Usamos artillery para simular usuarios accediendo y monitorizar los errores

```
$ cd ejem6-loadtest  
  
$ WEB_APP_URL=$(minikube service webapp --url)  
  
$ artillery run -e k8s --config config.yaml -t $WEB_APP_URL scenario.yaml -o output2.json  
  
$ artillery report ./output2.json
```

# Pod-Chaos-Monkey

## Prueba con caos: 1 pod

### Summary

Test duration	130 sec
Virtual Users created	1200
Virtual Users completed	282

### Errors

ECONNREFUSED	897
Failed expectations for request <a href="http://192.168.49.2:30145/">http://192.168.49.2:30145/</a>	7
ECONNRESET	14

# Pod-Chaos-Monkey

- Configuramos **2 réplicas** y volvemos a ejecutar la prueba

```
$ kubectl scale deployment webapp --replicas=2  
  
$ artillery run -e k8s --config config.yaml -t $WEB_APP_URL scenario.yaml -o output3.json  
  
$ artillery report ./output3.json
```

# Pod-Chaos-Monkey



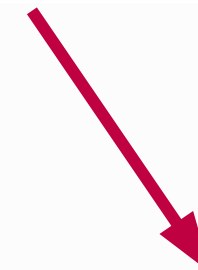
## Prueba con caos: 2 pods

### Summary

Test duration	130 sec
Virtual Users created	1200
Virtual Users completed	864

### Errors

ECONNREFUSED	321
ECONNRESET	9
Failed expectations for request http://192.168.49.2:30145/	6



# Pod-Chaos-Monkey



## Prueba con caos: 1 pod

### Summary

Test duration	130 sec
Virtual Users created	1200
Virtual Users completed	282

### Errors

ECONNREFUSED	897
Failed expectations for request http://192.168.49.2:30145/	7
ECONNRESET	14

## Prueba con caos: 2 pods

### Summary

Test duration	130 sec
Virtual Users created	1200
Virtual Users completed	864

### Errors

ECONNREFUSED	321
ECONNRESET	9
Failed expectations for request http://192.168.49.2:30145/	6

# 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'
    ...
```

habilitado

identificador

Modo de  
muerte

Valor de  
muerte

# Escalabilidad y Tolerancia a fallos en Kubernetes



- Escalabilidad en Kubernetes
- Autoescalado
- Escalado con estado
- Pruebas de escalabilidad
- Tolerancia a fallos
- Pruebas de tolerancia a fallos
- **Service Mesh para tolerancia a fallos**

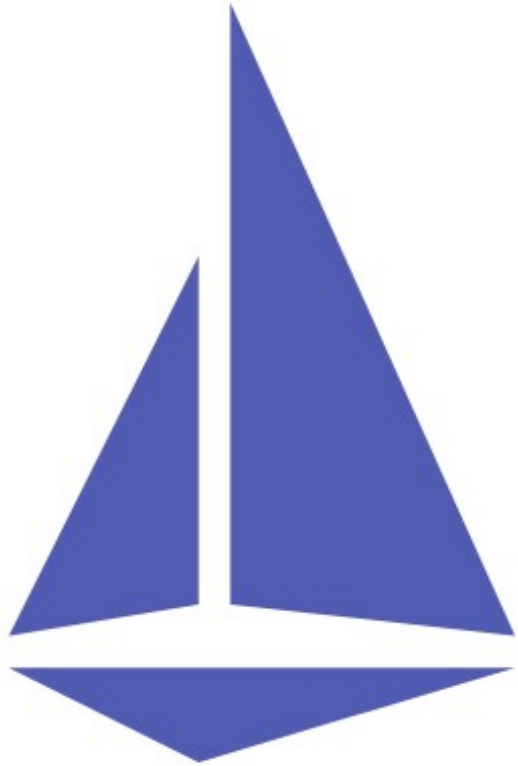
# ¿Cómo ser más tolerantes a los fallos?

# 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)?



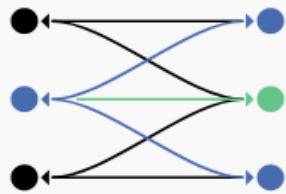
# Istio

Connect, secure, control, and observe services.

<https://istio.io/>

- Es una implementación de **service mesh** 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

## Ofrece muchos más servicios de red



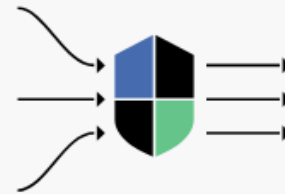
### Connect

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



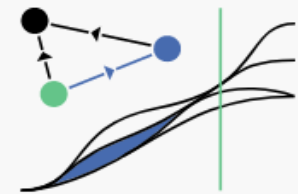
### Secure

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.

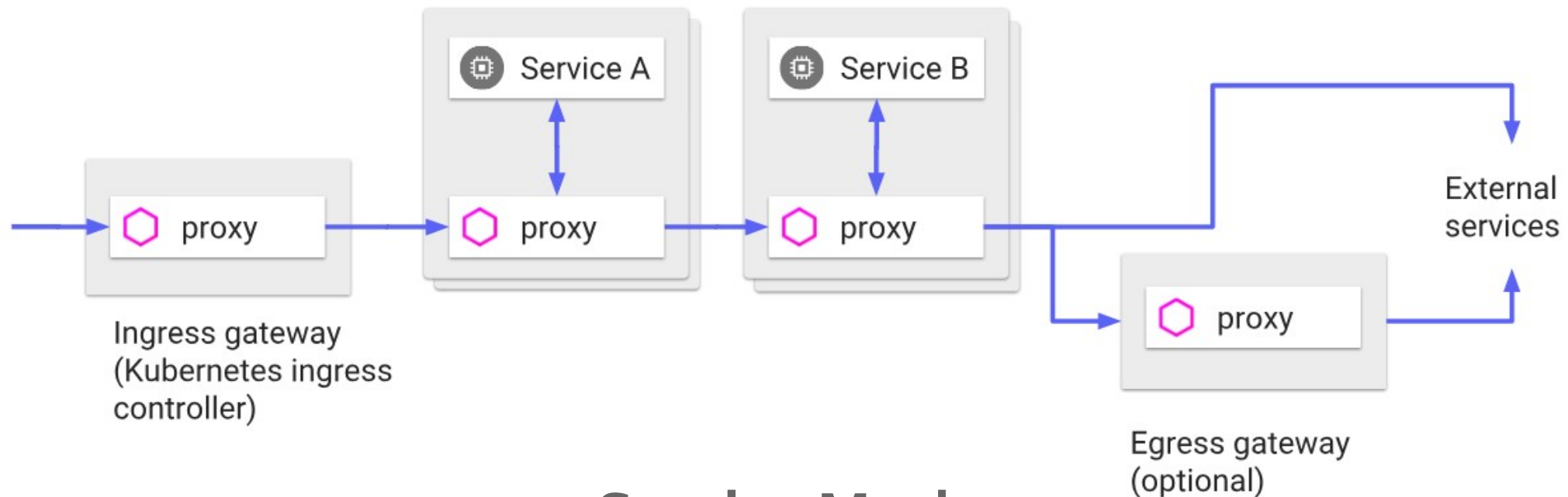


### Observe

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

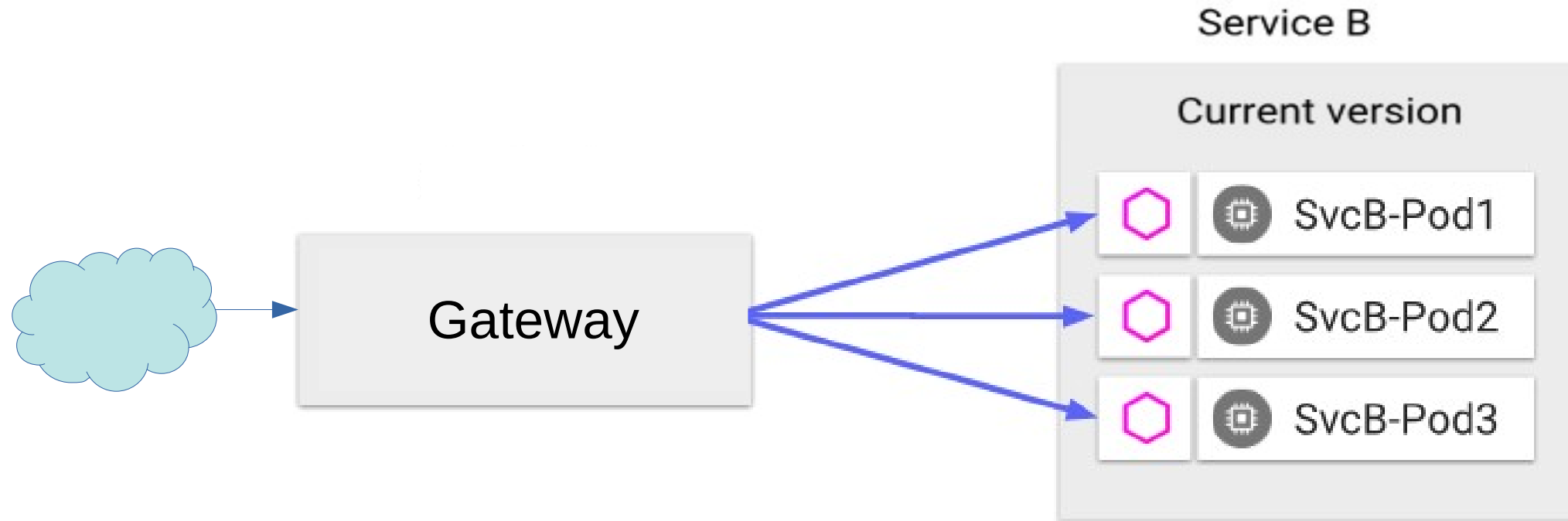


Para controlar el tráfico, Istio pone un **micro-proxy** al lado de cada **pod**, a la **entrada** y a la **salida** de cada nodo del cluster



## Service Mesh

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



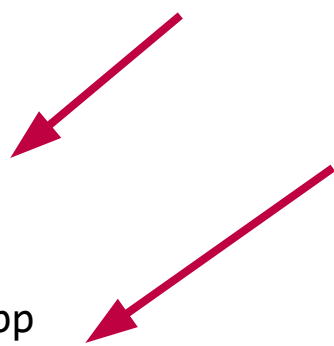
<https://istio.io/>

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: istio-gateway
spec:
  selector:
    istio: ingressgateway
  servers:
    - port:
        number: 80
        name: http
        protocol: HTTP
      hosts:
        - "*"

```

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: istio-virtualservice
spec:
  hosts:
    - "*"
  gateways:
    - istio-gateway
  http:
    - match:
        - uri:
            prefix: /
      route:
        - destination:
            host: webapp
            port:
              number: 8080

```



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

- 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*)
  - **Evitar nuevas peticiones en caso de errores** para que se puedan recuperar si están sobrecargados



- Reintentos

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: istio-virtualservice
spec:
  hosts:
    - "*"
  gateways:
    - istio-gateway
  http:
    - match:
        - uri:
            prefix: /
      route:
        - destination:
            host: webapp
            port:
              number: 8080
      retries:
        attempts: 5
        perTryTimeout: 2s
```

Retries config

- Pruebas de pod-chaos-monkey **sin reintentos**

## Prueba con caos: 1 pod

### Summary

Test duration	130 sec
Virtual Users created	1200
Virtual Users completed	282

### Errors

ECONNREFUSED	897
Failed expectations for request http://192.168.49.2:30145/	7
ECONNRESET	14

## Prueba con caos: 2 pods

### Summary

Test duration	130 sec
Virtual Users created	1200
Virtual Users completed	864

### Errors

ECONNREFUSED	321
ECONNRESET	9
Failed expectations for request http://192.168.49.2:30145/	6

- Instalamos istio en minikube

```
$ minikube start --memory=16384 --cpus=4 --driver=virtualbox
$ minikube tunnel
$ helm repo add istio https://istio-release.storage.googleapis.com/charts
$ helm repo update
$ kubectl create namespace istio-system
$ helm install istio-base istio/base -n istio-system
$ helm install istiod istio/istiod -n istio-system --wait
$ kubectl create namespace istio-ingress
$ kubectl label namespace istio-ingress istio-injection=enabled
$ helm install istio-ingress istio/gateway -n istio-ingress --wait
```

<https://istio.io/latest/docs/setup/platform-setup/minikube/>  
<https://istio.io/latest/docs/setup/install/helm/>



- Despliegue de la web de gatos con istio y caos

```
$ kubectl delete deployments, services --all  
$ kubectl label namespace default istio-injection=enabled --overwrite  
$ kubectl apply -f ejem4-webgatos2-scalable-hazelcast/k8s  
$ kubectl apply -f ejem8-istio  
$ TAG=app VALUE=webapp NAMESPACE=default DELAY=15 ejem7-chaos/chaos.sh
```

- Pruebas de carga

```
$ WEB_APP_URL=http://$(kubectl get svc istio-ingress \
  -n istio-ingress -o jsonpath='{.status.loadBalancer.ingress[0].ip}')/

$ cd ejem6-loadtest

$ artillery run -e k8s --config config.yaml -t $WEB_APP_URL scenario.yaml -o output4.json

$ artillery report ./output4.json
```

## Prueba con caos: 2 pod + istio

### Summary

Test duration	130 sec
Virtual Users created	1200
Virtual Users completed	1198

### Errors

Failed expectations for request <a href="http://10.96.83.187/">http://10.96.83.187/</a>	2
--	---



- 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



- 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
```

# Conclusiones

- **Crear una aplicación escalable y tolerante a fallos no es sencillo**
  - Stateless vs statfull
- **Kubernetes nos ofrece muchas facilidades**
  - Deployments, HorizontalPodAutoscaler, Istio
- **Siempre hay que verificar que el comportamiento es el esperado**
  - Peticiones OK, funcionamiento correcto

# How to implement and test scalable and fault-tolerant applications deployed on Kubernetes

**Micael Gallego**

The background image shows a large audience of people seated in rows of chairs, facing a stage area. The room has a high ceiling with numerous stage lights. The entire image is overlaid with a semi-transparent blue filter and large, light-colored curved graphic elements.

# THANK YOU FOR ATTENDING

nexo **qa**  
events

[www.expoqa.com](http://www.expoqa.com)