Guía de Instalación y Uso de LitmusChaos

1. Instalación de Dependencias

Necesitás tener instalados los siguientes componentes:

- **kubectl** (cliente de Kubernetes)
- **minikube** (para levantar un clúster local de Kubernetes)
- **helm** (package manager usado para instalar LitmusChaos)

Instalaciones

Podés instalar cada uno siguiendo los links oficiales:

- Instalar kubectl
- Instalar Minikube
- Instalar Helm

> ***Consejo:*** *Instalá primero* `kubectl`*, luego* `minikube`*. Helm podés instalarlo con el package manager de Ubuntu (*`apt`*,* `snap`*, etc).*

---

2. Iniciar Minikube

```
minikube start --driver=docker
```

Una vez que tengas instalado `minikube`, `kubectl` y `helm`, seguí con:

```
helm repo add litmuschaos https://litmuschaos.github.io/litmus-helm/
helm repo list
kubectl create ns litmus
helm install chaos litmuschaos/litmus --namespace=litmus --set
portal.frontend.service.type=NodePort
```

> *Con esto tenés* ***LitmusChaos instalado*** *(versión base).*
> *Podés verificar los pods y servicios con:*
>
> ```
> kubectl get pods -n litmus
> kubectl get svc -n litmus
> ```

---

3. Crear un Proyecto para las Pruebas

Necesitás un proyecto con un `Dockerfile`. Por ejemplo, uno simple con un servidor Node.js que devuelva “Hola”.

Dentro del proyecto:

```
docker build -t my-node-app:local .
minikube image load my-node-app:local
```

Luego aplicá la configuración de despliegue (`deployment.yaml`):

```
kubectl apply -f deployment.yaml
```

Verificá el estado:

```
kubectl get pods
kubectl get svc
```

> *Si en tu `deployment.yaml` pusiste `replicas: 2`, vas a ver dos instancias corriendo.*

---

4. Conectar Litmus con el Clúster

Para abrir el dashboard:

1. Ejecutá el servicio frontend:

   ```
   kubectl port-forward svc/chaos-litmus-frontend-service -n litmus 9091:9091
   ```

2. Abrí el navegador en la URL que indique el comando.

3. Iniciá sesión:

   ```
   Usuario: admin
   Contraseña: litmus
   ```

> *Usuario y contraseña de JUNI:*
> *`admin / @Chaostest1`*
> *(La primera vez te pedirá cambiar la contraseña.)*

> *Si la perdés, hay que entrar a la base de datos de MongoDB para resetearla (complicado).*

---

5. Crear una Service Account

Usá el YAML incluido en el proyecto (`service-account.yaml`):

```
kubectl apply -f service-account.yaml
```

> *Si da error porque ya existe, no pasa nada: Helm suele crearla por defecto.*

Después, creá un **Environment** (por ejemplo, `node-sv`) y activá **Enable Chaos**.

- **Namespace:** `default` (o el que usaste para tu app)
- **Service Account:** la creada recién

Si el comando falla, agregá el namespace explícitamente:

```
kubectl apply -f <archivo>.yaml -n <namespace>
```

---

6. Crear un Probe de Resiliencia

El **Probe** es una verificación que comprueba si tu servicio sigue respondiendo correctamente.

1. Ir a **Resilience Probes** → **New Probe**
2. Tipo: `HTTP`

3. Configurá para que haga una petición a tu endpoint y espere un `200 OK`.

> *Asegurate de usar la IP correcta del clúster (la podés obtener desde los servicios).*

---

7. Crear un Experimento

1. Ir a **Chaos Experiments → New Experiment**
2. Seleccionar el **Environment** creado antes.
3. Elegir **Blank Canvas**
4. En el builder, agregar el experimento `pod-delete`
5. Buscar la label de tu aplicación
6. En la pestaña **Probes**, agregá el probe creado antes y seleccioná **End of test**

Guardá y luego ejecutá el experimento:

```
Run → Start Experiment
```

---

8. Ver Resultados en Tiempo Real

Podés observar los pods en tiempo real:

```
kubectl get pods -l app=<nombre-del-servicio> -w
```

Mientras el experimento corre, verás algo así:

- El pod original (ej. `zn7r7`) es eliminado.
- Kubernetes crea automáticamente uno nuevo (ej. `rnzgd`).
- El probe valida el endpoint → recibe `200 OK` → test exitoso

---

9. Crear un Experimento que Falle Parcialmente

Para ver cómo responde el sistema ante fallos más severos, vamos a crear **dos experimentos encadenados** y **modificar los probes**.

Crear un nuevo probe ("gago-probe")

1. En **Resilience Probes**, creá uno nuevo con el mismo endpoint del anterior.
2. Asegurate de poner correctamente la IP y el endpoint.

Experimento 1 — Pod Delete (básico)

- Dejá todo **default**.
- Este borra **1 pod**.
- Configurá el **probe en modo `continuous`** para que se ejecute periódicamente durante el test.

> *El campo polling interval define cada cuánto se ejecuta el probe.*

Experimento 2 — Pod Delete (total)

- En “**Tune fault**”, poné:

  ```
  Pods affected percentage = 100
  ```

- Esto elimina **todos los pods simultáneamente**.
- También configurá el probe en modo `continuous`.

Durante la ejecución, si mirás los pods con:

```
kubectl get pods -w
```

vas a ver algo como:

- El primer `pod-delete` borra un pod (`hrcmv`). Como el otro (`bcbwt`) sigue vivo, el probe pasa sin problemas.
  Kubernetes levanta otro (`h2w76`) automáticamente.
- En el segundo `pod-delete`, se eliminan ambos pods (`bcbwt` y `h2w76`) simultáneamente.
  Mientras Kubernetes crea los nuevos (`467fk`, `tnx4z`), el probe detecta que no hay respuesta — el servicio se considera **caído**. Por eso el experimento **falla** (como es esperado).

Desinstalación

Si necesitás limpiar todo:

```
helm uninstall chaos -n litmus
kubectl delete ns litmus
minikube delete
```

Notas Finales

- El experimento puede tardar varios minutos en iniciar.
- Si el dashboard no carga, revisá los puertos y el namespace.
- Siempre verificá que `kubectl config current-context` apunte al clúster correcto.