



ChaosEater demo

Live Q&A

Settings



Model

google/gemini-1.5-pro-latest



Token

Hugging Face token



Cluster selection

kind-chaos-eater-cluster



Clean the cluster

☒ Clean the cluster before run

☒ Clean the cluster after run

☒ New deployment

Max # steady states

3



Max retries

3



Seed (GPTs only)

42



Usage:

Total billing: \$0.02

Total tokens: 3.552k

Input tokens: 2.858k

Ouput tokens: 0.694k

Command history

kubectl apply -f
sandbox/cycle_20241128_081601/hypothesis/k8s_NumberOfRunningPods_example_pod.yaml --context kind-chaos-eater-cluster -n chaos-eater

kubectl logs



Your instructions for Chaos Engineering:

The Chaos-Engineering experiment must be completed within 1 minute.



Phase 0: Preprocessing

Cleaning the cluster `kind-chaos-eater-cluster` ... Done

```
$ kubectl delete workflow --all --context kind-chaos-eater-cluster -n chaos-eater
No resources found
$ kubectl delete workflownode --all --context kind-chaos-eater-cluster -n chaos-eater
No resources found
$ kubectl delete deployments --all --context kind-chaos-eater-cluster -n chaos-eater
No resources found
$ kubectl delete pods --all --context kind-chaos-eater-cluster -n chaos-eater
No resources found
$ kubectl delete services --all --context kind-chaos-eater-cluster -n chaos-eater
No resources found
```

```
$ kubectl delete all --all-namespaces --context kind-chaos-eater-cluster -l app=example
pod "example-deployment-7bbcc69fbf-9v88z" deleted
pod "example-deployment-7bbcc69fbf-c7wrg" deleted
pod "example-deployment-7bbcc69fbf-p8zzf" deleted
service "example-service" deleted
deployment.apps "example-deployment" deleted
```

K8s manifest(s) to be deployed:

`nginx/pod.yaml`

```
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
  labels:
    app: example
spec:
  restartPolicy: Never
  containers:
  - name: example-container
    image: nginx:1.17.1
    ports:
    - containerPort: 80
```

`nginx/service.yaml`

Input instructions for your Chaos Engineering



```
spec:
  selector:
    app: example
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

Deploying resources... Done

```
$ skaffold run --kube-context kind-chaos-eater-cluster -l project=chaos-eater
No tags generated
Starting test...
Starting deploy...
Loading images into kind cluster nodes...
Images loaded in 501ms
- pod/example-pod created
- service/example-service created
Waiting for deployments to stabilize...
- pods is ready.
Deployments stabilized in 2.099 seconds
You can also run [skaffold run --tail] to get the logs
```

Resource statuses

```
$ kubectl get all --all-namespaces --context kind-chaos-eater-cluster --selector=app=example
```

Summary of each manifest:

nginx/pod.yaml

- This manifest defines a Kubernetes Pod named `example-pod`.
- A Pod is the smallest deployable unit in Kubernetes, like a container for your application.
- This Pod uses the `nginx:1.17.1` image, which is a popular web server.
- The Pod exposes port 80, which is the standard port for HTTP traffic.
- The `restartPolicy: Never` means that if the container inside the pod fails, it won't be automatically restarted.

nginx/service.yaml

- This manifest defines a Kubernetes Service named "example-service".
- A Kubernetes Service acts as a load balancer and provides a stable IP address for accessing Pods.
- This service targets Pods with the label "app: example".
- It exposes port 80 on the service, which maps to port 80 on the target Pods.
- This allows external traffic to access the application running on the Pods with the label "app: example" on port 80.

Resiliency issues/weaknesses in the manifests:

Issue #0: No Redundancy for Pod

- details: If the pod crashes or the node it's running on fails, the application becomes unavailable.
- manifests having the issues: ['nginx/pod.yaml']

- problematic config: restartPolicy: Never

Issue #1: No Replica for Pod

- details: The application is running as a single pod, which means a single point of failure. If the pod crashes or the node it's running on fails, the application becomes unavailable.
- manifests having the issues: ['nginx/pod.yaml']
- problematic config: Absence of a Deployment or other controller managing multiple replicas of the pod.

Issue #2: No Health Check

- details: The service has no way to determine the health of the underlying pod. It might route traffic to an unhealthy pod, leading to application downtime.
- manifests having the issues: ['nginx/pod.yaml', 'nginx/service.yaml']
- problematic config: Lack of livenessProbe or readinessProbe in the pod spec and no health checks configured in the service.

Application of the manifests:

Thoughts:

The filenames (nginx/pod.yaml and nginx/service.yaml) and the container image (nginx:1.17.1) clearly indicate that this application is intended to serve web content using the Nginx web server. The service.yaml exposes port 80, further reinforcing this assumption.

Assumed application:

This application provides a basic web server using Nginx. It can serve static web content or act as a reverse proxy to other services. The provided manifests create a single pod for the web server and a service to expose it. This setup is suitable for simple web applications or testing environments. For production, it would typically be extended with features like deployments for managing multiple pods, ingress for external access, and configmaps for managing configuration.

Summary of your instructions for Chaos Engineering:

- The Chaos-Engineering experiment must be completed within 1 minute. (This instruction should be considered in the experiment phase.)

Phase 1: Hypothesis

Steady-state definition

Steady state #1: NumberOfRunningPods_example

Description

The pod's restart policy is set to "Never", which means if the pod's container crashes, it won't be restarted automatically. This is the most likely point of failure. Thus, I'll define the steady state of the number of pods running with the label "app: example". This steady state is directly tied to the 'nginx/pod.yaml' manifest and will immediately reflect any issues with the pod's availability.

Current state inspection

I'll use the Kubernetes API (Python) with the official Python client library to check the number of running pods. This allows me to directly query the Kubernetes API server for the current state of the pods. I'll use a loop to periodically check the pod count for a specified duration, providing a summary at the end. This approach is suitable for monitoring the state of Kubernetes resources over time.

tool: `k8s` duration: `5`

Inspection script (Python) `k8s_NumberOfRunningPods_example.py` :

```
# Check the number of running pods with label 'app=example'
import kubernetes
import time
import os
from kubernetes import client, config
import argparse

def main():
    parser = argparse.ArgumentParser()
    parser.add_argument('--duration', type=int, default=5, help='Duration in seconds')
    args = parser.parse_args()

    # Load Kubernetes configuration based on the environment
    if os.getenv('KUBERNETES_SERVICE_HOST'):
        config.load_incluster_config()
    else:
        config.load_kube_config()

    v1 = client.CoreV1Api()

    running_counts = []
    for i in range(args.duration):
        try:
            pods = v1.list_namespaced_pod("default", label_selector="app=example")
            running_pods = [pod for pod in pods.items if pod.status.phase == 'Running']
            running_count = len(running_pods)
            running_counts.append(running_count)
            print(f"Number of running pods at time {i+1}s: {running_count}")
            time.sleep(1)
        except Exception as e:
            print(f"Error checking pod status: {e}")
            running_counts.append(-1) # Append -1 to indicate an error
            time.sleep(1)


    print("\nSummary:")
    for i, count in enumerate(running_counts):
        print(f"Time {i+1}s: {count if count != -1 else 'Error'}")

if __name__ == "__main__":
    main()
```

Pod `k8snumberofrunningpodsexample-pod` has completed successfully.

The inspection script's results (current states) are as follows:

Summary:

 Threshold

The current setup has a single pod and no redundancy. The restart policy is "Never", so a crash won't result in an automatic restart. Therefore, the number of running pods should be 1. To account for momentary fluctuations in the pod status, I'll allow for a short period where the pod count might be 0, but it should be 1 for at least 80% of the monitoring duration (which is reasonable for a 1-minute experiment).

The number of running pods with the label 'app=example' should be equal to 1 for at least 80% of the monitoring period.

Unit test to validate the steady state

I added a threshold assertion to the provided Python script. The threshold is that the number of running pods with the label 'app=example' should be equal to 1 for at least 80% of the monitoring period. I implemented this by calculating the percentage of time the running pod count was 1 and asserting that this percentage is greater than or equal to 80%. The script now takes a '--duration' argument to specify the monitoring duration, and the assertion is performed at the end of the monitoring period. The script also provides a summary of the pod counts at each time step.

TypeError: write() argument must be str, not None

Traceback:

```
File "/opt/conda/lib/python3.10/site-packages/streamlit/runtime/scriptrunner/s
exec(code, module.__dict__)
File "/workspace/app/projects/chaos-eater/ChaosEater_demo.py", line 431, in <m
main()
File "/workspace/app/projects/chaos-eater/ChaosEater_demo.py", line 397, in ma
output = st.session_state.chaoseater.run_ce_cycle(
File "/workspace/app/projects/chaos-eater/chaos_eater/chaos_eater.py", line 14
hypothesis_logs, hypothesis = self.hypothesizer.hypothesize(
File "/workspace/app/projects/chaos-eater/chaos_eater/hypothesis/hypothesizer.
steady_state_logs, steady_states = self.steady_state_definer.define_steady
File "/workspace/app/projects/chaos-eater/chaos_eater/hypothesis/steady_state_
unittest_log, unittest = self.unittest_agent.write_unittest(
File "/workspace/app/projects/chaos-eater/chaos_eater/hypothesis/llm_agents/st
write_file(file_path, unittest["code"])
File "/workspace/app/projects/chaos-eater/chaos_eater/utils/functions.py", lin
f.write(content)
```