



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: 3.497k

Input tokens: 2.874k

Ouput tokens: 0.623k

Command history

kubectl apply -f



sandbox/cycle_20241128_081806/hypothesis/k8s_example-pod-count_pod.yaml --context kind-chaos-eater-cluster -n chaos-eater

kubectl logs k8sexample-pod-count-



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-pod" deleted
service "example-service" 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

```
apiVersion: v1
kind: Service
metadata:
```

Input instructions for your Chaos Engineering



```
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 160ns
- pod/example-pod created
- service/example-service created
Waiting for deployments to stabilize...
- pods is ready.
Deployments stabilized in 3.086 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=project=chaos-eater
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
default	pod/example-pod	1/1	Running	0	5s

NAMESPACE	NAME	TYPE	CLUSTER-IP	EXTERNAL-IP
default	service/example-service	ClusterIP	10.96.31.94	<none>

Summary of each manifest:

nginx/pod.yaml

- This manifest defines a single Pod in Kubernetes.
- The Pod's name is `example-pod`.
- It has a label `app: example`, which can be used to select and manage it.
- The `restartPolicy: Never` means that if the container inside the pod fails, it won't be automatically restarted.
- The Pod contains one container named `example-container`.
- This container uses the `nginx:1.17.1` image, which is a web server.
- The container exposes port 80, which is the standard HTTP port.

nginx/service.yaml

- This manifest defines a Kubernetes Service named "example-service".
- Services act as a stable entry point to a set of Pods, which are the smallest deployable units in Kubernetes.
- 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 through the service.

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: kind: Pod

Issue #1: No Restart Policy for Pod

- details: The pod won't be restarted if the container inside fails. This leads to application downtime.
- manifests having the issues: ['nginx/pod.yaml']
- problematic config: restartPolicy: Never

Issue #2: No Health Checks Defined

- details: The service doesn't check the health of the pod. It might route traffic to an unhealthy pod, leading to errors.
- manifests having the issues: ['nginx/service.yaml', 'nginx/pod.yaml']
- problematic config: Absence of livenessProbe or readinessProbe in pod spec and absence of health checks at the service level.

Application of the manifests:

Thoughts:

The filenames (pod.yaml, service.yaml) and the use of the `nginx` image suggest a web server application. The service exposes port 80, further reinforcing this assumption.

Assumed application:

This application provides a basic web server. It uses Nginx to serve static content or act as a reverse proxy to other services. The service provides a stable endpoint for accessing the Nginx pod, even if the pod is restarted or rescheduled.

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: example-pod-count

Description

The pod 'example-pod' is defined with `restartPolicy: Never`. This means if the pod's container crashes, it won't be restarted automatically, leading to application downtime. Therefore, the number of running pods corresponding to `example-pod` is an important steady state to monitor. We'll name this steady state `example-pod-count` and target it at the `nginx/pod.yaml` manifest.

Current state inspection

I'll use the Kubernetes API (Python) to check the number of running pods with the name 'example-pod'. The script will check the pod count every second for a specified

duration. Since the restartPolicy is 'Never', we expect the pod count to remain at 1 unless an external event terminates the pod. I'll use a short duration (5 seconds) for the check, as a longer duration isn't necessary for this simple scenario.

tool: `k8s` duration: `5`

Inspection script (Python) `k8s_example-pod-count.py` :

```
import kubernetes
import time
import os
from kubernetes import client, config
import argparse

parser = argparse.ArgumentParser()
parser.add_argument('--duration', type=int, default=5, help='Duration of the check')
args = parser.parse_args()

DURATION = min(args.duration, 5) # Limit duration to a maximum of 5 seconds

# 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()

results = []
for i in range(DURATION):
    try:
        pods = v1.list_namespaced_pod(namespace="default", label_selector="app=example")
        pod_count = len(pods.items)
        results.append(pod_count)
        print(f"Second {i+1}: Number of example pods: {pod_count}")
        time.sleep(1)
    except Exception as e:
        print(f"Error checking pod status: {e}")
        results.append(-1) # Append -1 to indicate an error
        time.sleep(1)

print("\nSummary:")
for i, count in enumerate(results):
    print(f"Second {i+1}: {'Error' if count == -1 else count}")
```

Pod `k8sexample-pod-count-pod` has completed successfully.

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

Summary:

▶ Threshold

The steady state 'example-pod-count' represents the number of running pods with the label 'app=example'. The current state shows that there is consistently 1 pod running. Since the restart policy is 'Never', we need to ensure that at least 1 pod is running. Given there is no redundancy, the threshold should be exactly 1. We'll add a small tolerance to account for potential delays in reporting or very brief restarts, but the core requirement

is 1 pod. Since the experiment duration is short (1 minute), we'll require this condition to be true for the entire duration of the checks (5 seconds in the provided script).

example-pod-count == 1 for 100% of the 5-second monitoring period

Unit test to validate the steady state

To add the threshold assertion, I will modify the provided inspection script to include a check within the loop. This check will verify that the number of 'example-pod' pods is equal to 1. After the loop, a summary will be printed, and an assertion will be made to ensure that the threshold was met for the entire duration. The threshold will be represented as a percentage (100%) to ensure that the condition (pod count == 1) holds true for all checks.

```
#!/usr/bin/env python3
# Import necessary libraries
import kubernetes
import time
import os
from kubernetes import client, config
import argparse
from unittest_base import K8sAPIBase

class SteadyStateCheck(K8sAPIBase):
    def test_example_pod_count(self):
        # Define the namespace and label selector for the pod
        namespace = "default"
        label_selector = "app=example"

        # Define the expected pod count and the threshold percentage
        expected_pod_count = 1
        threshold_percentage = 100 # 100% of the time

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

        duration = args.duration

        # Initialize results list to store pod counts
        results = []

        # Loop for the specified duration, checking the pod count every second
        for i in range(duration):
            try:
                # List pods matching the label selector in the specified namespace
                pods = self.v1.list_namespaced_pod(namespace=namespace, label_selector=label_selector)
                pod_count = len(pods.items)
                results.append(pod_count)
                print(f"Second {i+1}: Number of example pods: {pod_count}")
                time.sleep(1)
            except Exception as e:
                print(f"Error checking pod status: {e}")
                results.append(-1) # Append -1 to indicate an error
                time.sleep(1)

        # Print a summary of the results
        print("\nSummary:")
        successful_checks = 0
```

```

for i, count in enumerate(results):
    if count == expected_pod_count:
        successful_checks += 1
    print(f"Second {i+1}: {'Error' if count == -1 else count}")

# Calculate the percentage of successful checks
success_percentage = (successful_checks / len(results)) * 100 if len

# Assert that the success percentage meets the threshold
print(f"Success Percentage: {success_percentage:.2f}% over {duration}
assert success_percentage >= threshold_percentage, f"The example-pod

in():
it = SteadyStateCheck()
it.test_example_pod_count()

if __name__ == '__main__':
    main()

```

Pod `unittestexample-pod-countmod0-pod` has failed.

```

Traceback (most recent call last):
  File "/chaos-eater/sandbox/cycle_20241128_081806/hypothesis/unittest
    main()
  File "/chaos-eater/sandbox/cycle_20241128_081806/hypothesis/unittest
    test.test_example_pod_count()
  File "/chaos-eater/sandbox/cycle_20241128_081806/hypothesis/unittest
    assert success_percentage >= threshold_percentage, f"The example
AssertionError: The example-pod-count did not meet the threshold of

Summary:
Success Percentage: 0.00% over 0 seconds

```

The error "AssertionError: The example-pod-count did not meet the threshold of 100% for the entire duration. Only achieved 0.00%" and the summary showing 0 seconds indicate that the duration is not being properly passed or handled. The likely culprit is the missing initialization of `duration` before the `argparse` setup. Moving the `duration = args.duration` line after parsing the arguments should resolve the issue.

```

import necessary libraries
from kubernetes
from time
from os
from kubernetes import client, config
from argparse
from unittest_base import K8sAPIBase

class SteadyStateCheck(K8sAPIBase):
    def test_example_pod_count(self):
        # Define the namespace and label selector for the pod
        namespace = "default"
        label_selector = "app=example"

        # Define the expected pod count and the threshold percentage
        expected_pod_count = 1
        threshold_percentage = 100 # 100% of the time

        # Initialize argparse
        parser = argparse.ArgumentParser()

```

```

parser.add_argument('--duration', type=int, default=5, help='Duration of the test')
args = parser.parse_args()

duration = args.duration # Initialize duration after parsing arguments

# Initialize results list to store pod counts
results = []

# Loop for the specified duration, checking the pod count every second
for i in range(duration):
    try:
        # List pods matching the label selector in the specified namespace
        pods = self.v1.list_namespaced_pod(namespace=namespace, label_selector=label_selector)
        pod_count = len(pods.items)
        results.append(pod_count)
        print(f"Second {i+1}: Number of example pods: {pod_count}")
        time.sleep(1)
    except Exception as e:
        print(f"Error checking pod status: {e}")
        results.append(-1) # Append -1 to indicate an error
        time.sleep(1)

# Print a summary of the results
print("\nSummary:")
successful_checks = 0
for i, count in enumerate(results):
    if count == expected_pod_count:
        successful_checks += 1
    print(f"Second {i+1}: {'Error' if count == -1 else count}")

# Calculate the percentage of successful checks
success_percentage = (successful_checks / len(results)) * 100 if len(results) > 0 else 0

# Assert that the success percentage meets the threshold
print(f"Success Percentage: {success_percentage:.2f}% over {duration} seconds")
assert success_percentage >= threshold_percentage, f"The example-pod-count did not meet the threshold of {threshold_percentage}%"

def main():
    test = SteadyStateCheck()
    test.test_example_pod_count()

if __name__ == '__main__':
    main()

```

Pod `unittestexample-pod-countmod1-pod` has failed.

Traceback (most recent call last):

File `"/chaos-eater/sandbox/cycle_20241128_081806/hypothesis/unittestexample-pod-countmod1-pod.py"`, line 10, in `<module>`

Summary:

Success Percentage: 0.00% over 0 seconds

main()

File `"/chaos-eater/sandbox/cycle_20241128_081806/hypothesis/unittestexample-pod-countmod1-pod.py"`, line 10, in `<module>`

test.test_example_pod_count()

File `"/chaos-eater/sandbox/cycle_20241128_081806/hypothesis/unittestexample-pod-countmod1-pod.py"`, line 10, in `<module>`

assert success_percentage >= threshold_percentage, f"The example-pod-count did not meet the threshold of {threshold_percentage}%"

AssertionError: The example-pod-count did not meet the threshold of 100%

The previous error was fixed by ensuring the duration is initialized correctly. However, the current error, "AssertionError: The example-pod-count did not meet the threshold of 100% for the entire duration. Only achieved 0.00%", suggests that the pod count check is failing. This could be due to the pod not being ready or available immediately after deployment. To address this, we'll add a check within the loop to wait for the pod to become ready before counting it towards the success percentage. We'll use the pod's status conditions to determine readiness.

```
# Import necessary libraries
import kubernetes
import time
import os
from kubernetes import client, config
import argparse
from unittest_base import K8sAPIBase

class SteadyStateCheck(K8sAPIBase):
    def test_example_pod_count(self):
        # Define the namespace and label selector for the pod
        namespace = "default"
        label_selector = "app=example"

        # Define the expected pod count and the threshold percentage
        expected_pod_count = 1
        threshold_percentage = 100 # 100% of the time

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

        duration = args.duration

        # Initialize results list to store pod counts
        results = []

        # Loop for the specified duration, checking the pod count every second
        for i in range(duration):
            try:
                # List pods matching the label selector in the specified namespace
                pods = self.v1.list_namespaced_pod(namespace=namespace, label_selector=label_selector)
                pod_count = 0 # Initialize pod_count for this iteration
                for pod in pods.items:
                    # Check if the pod is in the 'Running' phase and ready
                    if pod.status.phase == "Running" and any(condition.type == "Ready" for condition in pod.status.conditions):
                        pod_count += 1
                results.append(pod_count)
                print(f"Second {i+1}: Number of example pods: {pod_count}")
                time.sleep(1)
            except Exception as e:
                print(f"Error checking pod status: {e}")
                results.append(-1) # Append -1 to indicate an error
                time.sleep(1)

        # Print a summary of the results
        print("\nSummary:")
        successful_checks = 0
        for i, count in enumerate(results):
            if count == expected_pod_count:
                successful_checks += 1
```



```

        print(f"Second {i+1}: {'Error' if count == -1 else count}")

    # Calculate the percentage of successful checks
    success_percentage = (successful_checks / len(results)) * 100

    # Assert that the success percentage meets the threshold
    print(f"Success Percentage: {success_percentage:.2f}% over {duration} seconds")
    assert success_percentage >= threshold_percentage, f"The example-pod-count did not meet the threshold of {threshold_percentage}% success"

def main():
    test = SteadyStateCheck()
    test.test_example_pod_count()

if __name__ == '__main__':
    main()

```

Pod `unittestexample-pod-countmod2-pod` has failed.

```

Summary:
Traceback (most recent call last):
  File "/chaos-eater/sandbox/cycle_20241128_081806/hypothesis/unittestexample-pod-countmod2-pod.py", line 10, in main
    test.test_example_pod_count()
  File "/chaos-eater/sandbox/cycle_20241128_081806/hypothesis/unittestexample-pod-countmod2-pod.py", line 25, in test_example_pod_count
    assert success_percentage >= threshold_percentage, f"The example-pod-count did not meet the threshold of {threshold_percentage}% success"
AssertionError: The example-pod-count did not meet the threshold of 100% success
Success Percentage: 0.00% over 0 seconds

```

The previous fix attempted to check pod readiness. However, the error persists, indicating the pod might still not be ready within the test's timeframe. The duration being 0 seconds in the error message suggests an issue with how duration is handled or the loop not running at all. While the previous fix correctly initialized `duration` after parsing arguments, the error output suggests the `duration` is still 0. This could be due to an argparse issue or the argument not being passed correctly. To be sure the loop runs as expected, we'll add a default value to the duration argument in argparse and print the duration before the loop starts. This will help diagnose if the duration is being set correctly.

```

# Import necessary libraries
import kubernetes
import time
import os
from kubernetes import client, config
import argparse
from unittest_base import K8sAPIBase

class SteadyStateCheck(K8sAPIBase):
    def test_example_pod_count(self):
        # Define the namespace and label selector for the pod
        namespace = "default"
        label_selector = "app=example"

        # Define the expected pod count and the threshold percentage
        expected_pod_count = 1
        threshold_percentage = 100 # 100% of the time

        # Initialize argparse

```

```

parser = argparse.ArgumentParser()
parser.add_argument('--duration', type=int, default=5, help=
args = parser.parse_args()
duration = args.duration

print(f"Running test for {duration} seconds") # Print duration

# Initialize results list to store pod counts
results = []

# Loop for the specified duration, checking the pod count every second
for i in range(duration):
    try:
        # List pods matching the label selector in the specified namespace
        pods = self.v1.list_namespaced_pod(namespace=namespace, label_selector=label_selector)
        pod_count = 0
        for pod in pods.items:
            if pod.status.phase == "Running" and any(condition in pod.labels for condition in conditions):
                pod_count += 1
        results.append(pod_count)
        print(f"Second {i+1}: Number of example pods: {pod_count}")
        time.sleep(1)
    except Exception as e:
        print(f"Error checking pod status: {e}")
        results.append(-1) # Append -1 to indicate an error
        time.sleep(1)

# Print a summary of the results
print("\nSummary:")
successful_checks = 0
for i, count in enumerate(results):
    if count == expected_pod_count:
        successful_checks += 1
    print(f"Second {i+1}: {'Error' if count == -1 else count}")

# Calculate the percentage of successful checks
success_percentage = (successful_checks / len(results)) * 100

# Assert that the success percentage meets the threshold
print(f"Success Percentage: {success_percentage:.2f}% over {duration} seconds")
assert success_percentage >= threshold_percentage, f"The example-pod-count did not meet the threshold of {threshold_percentage}%"

def main():
    test = SteadyStateCheck()
    test.test_example_pod_count()

if __name__ == '__main__':
    main()

```

Pod `unittestexample-pod-countmod3-pod` has failed.

```

Traceback (most recent call last):
  File "/chaos-eater/sandbox/cycle_20241128_081806/hypothesis/unittestexample.py", line 10, in <module>
    main()
  File "/chaos-eater/sandbox/cycle_20241128_081806/hypothesis/unittestexample.py", line 20, in test_example_pod_count
    test.test_example_pod_count()
  File "/chaos-eater/sandbox/cycle_20241128_081806/hypothesis/unittestexample.py", line 30, in test_example_pod_count
    assert success_percentage >= threshold_percentage, f"The example-pod-count did not meet the threshold of {threshold_percentage}%"
AssertionError: The example-pod-count did not meet the threshold of 100%
Running test for 0 seconds

```

Summary:

Success Percentage: 0.00% over 0 seconds

AssertionError: MAX_MOD_COUNT_EXCEEDED: 3

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
assert mod_count < max_mod_loop, f"MAX_MOD_COUNT_EXCEEDED: {max_mod_loop}"
```