



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: 3.404k

Input tokens: 2.859k

Output tokens: 0.545k

Command history

kubectl apply -f



sandbox/cycle_20241128_082317/hypothesis/k8s_nginxPodCount_pod.yaml
--context kind-chaos-eater-cluster -n chaos-eater

kubectl logs k8s_nginxpodcount-pod --



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
```

Input instructions for your Chaos Engineering



```
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 70ns
  - pod/example-pod created
  - service/example-service created
Waiting for deployments to stabilize...
  - pods is ready.
Deployments stabilized in 3.09 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	10.96.151.45	<none>

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, absence of a Deployment or StatefulSet managing the pod.

Issue #1: No Health Checks

- details: The service has no way to determine the health of the pod. It might route traffic to an unhealthy pod, leading to application downtime or errors.
- manifests having the issues: ['nginx/service.yaml', 'nginx/pod.yaml']
- problematic config: Absence 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 use of the nginx image strongly suggest that this application is a web server. The service exposes port 80, further reinforcing this assumption. The pod and service work together to deploy and expose a basic nginx web server.

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.

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: nginxPodCount

Description

The pod's restart policy is set to "Never", which means if the pod crashes, it won't be restarted automatically. This is the most obvious point of failure. Thus, we should first monitor the number of available pods with the label "app: example" to ensure that the pod continues running during the chaos experiment. We will define the steady state related to the number of pods.

Current state inspection

The application is deployed with a Kubernetes Pod and Service. The Pod has the label 'app: example'. To check the number of pods with this label, I will use the Kubernetes Python API. I will set the duration to 5 seconds to check the pod status every second for a short period.

tool: `k8s` duration: `5`

Inspection script (Python) `k8s_nginxPodCount.py` :

```

# Check the number of pods with the label 'app: example'
import kubernetes
import os
import time
import argparse

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

# Initialize the Kubernetes API client
v1 = kubernetes.client.CoreV1Api()

start_time = time.time()
results = []

while time.time() - start_time < args.duration:
    try:
        # Get pods with the label 'app: example'
        pods = v1.list_namespaced_pod("default", label_selector="app=example")
        pod_count = len(pods.items)
        results.append(pod_count)
        print(f"Number of pods with label 'app: example': {pod_count}")
        time.sleep(1)
    except Exception as e:
        print(f"Error: {e}")
        time.sleep(1)

# Summarize the results
print("\nSummary:")
for i, count in enumerate(results):
    print(f"Second {i+1}: {count} pods")

```

Pod `k8snginxpodcount-pod` has failed.

```

Traceback (most recent call last):
  File "/chaos-eater/sandbox/cycle_20241128_082317/hypothesis/k8snginxpodcount_mod1.py", line 15, in <module>
    config.load_incluster_config()
NameError: name 'config' is not defined

```

The error "NameError: name 'config' is not defined" occurs because the `kubernetes.config` module is not imported. To fix this, I will add `'from kubernetes import config'` at the beginning of the script.

tool: `k8s` duration: `5`

Inspection script (Python) `k8snginxPodCount_mod1.py`:

```

from kubernetes import config
import kubernetes
import os
import time
import argparse

```

```

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

# Initialize the Kubernetes API client
v1 = kubernetes.client.CoreV1Api()

start_time = time.time()
results = []

while time.time() - start_time < args.duration:
    try:
        # Get pods with the label 'app: example'
        pods = v1.list_namespaced_pod("default", label_selector="app=example")
        pod_count = len(pods.items)
        results.append(pod_count)
        print(f"Number of pods with label 'app: example': {pod_count}")
        time.sleep(1)
    except Exception as e:
        print(f"Error: {e}")
        time.sleep(1)

# Summarize the results
print("\nSummary:")
for i, count in enumerate(results):
    print(f"Second {i+1}: {count} pods")

```

Pod `k8snginxpodcountmod1-pod` has completed successfully.

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

Summary:

Threshold

The current state shows that there is 1 pod with the label 'app: example' running. Since the restart policy is 'Never', the threshold should ensure that at least one pod is running. To account for potential transient errors in the Kubernetes API or very short downtime, I'll add a small tolerance by requiring the pod count to be at least 1 for at least 90% of the monitoring period.

The number of pods with the label 'app: example' should be at least 1 for at least 90% of the monitoring period.

Unit test to validate the steady state

To add the threshold assertion, I calculated the percentage of time the pod count was at least 1 during the monitoring period. This percentage is then compared against the threshold (90%). The test passes if the percentage meets or exceeds the threshold, and fails otherwise. I used the Kubernetes Python API to interact with the cluster and

retrieve pod information. The argparse module allows users to specify the duration of the test.

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

class K8sAPIBase:
    def __init__(self):
        # Load Kubernetes configuration based on the environment
        if os.getenv('KUBERNETES_SERVICE_HOST'):
            config.load_incluster_config()
        else:
            config.load_kube_config()

        # Create a Kubernetes API client
        self.v1 = client.CoreV1Api()

class PodCountCheck(K8sAPIBase):
    def test_pod_count(self, duration):
        # Define the threshold percentage
        threshold_percentage = 0.9

        # Initialize variables for tracking pod counts
        results = []
        start_time = time.time()

        # Loop for the specified duration
        while time.time() - start_time < duration:
            try:
                # Get pods with the label 'app: example'
                pods = self.v1.list_namespaced_pod("default", label_selector="app=example")
                pod_count = len(pods.items)
                results.append(pod_count)
                print(f"Number of pods with label 'app: example': {pod_count}")
                time.sleep(1)
            except Exception as e:
                print(f"Error: {e}")
                time.sleep(1)

        # Calculate the percentage of time the pod count was at least 1
        successful_counts = sum(1 for count in results if count >= 1)
        percentage = successful_counts / len(results) if results else 0

        # Summarize the results
        print("\nSummary:")
        for i, count in enumerate(results):
            print(f"Second {i+1}: {count} pods")

        print(f"Percentage of time with at least 1 pod: {percentage}")

        # Assert that the percentage meets the threshold
        assert percentage >= threshold_percentage, f"Pod count was {percentage} which is less than the threshold of {threshold_percentage}"
        print("Test passed!")

def main():
    parser = argparse.ArgumentParser()
```

```

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

test = PodCountCheck()
test.test_pod_count(args.duration)

if __name__ == '__main__':
    main()

```

Pod `unittestnginxpodcountmod0-pod` has failed.

```

Summary:
Percentage of time with at least 1 pod: 0.00%
Traceback (most recent call last):
  File "/chaos-eater/sandbox/cycle_20241128_082317/hypothesis/unittestnginxpodcountmod0.py", line 10, in <module>
    main()
  File "/chaos-eater/sandbox/cycle_20241128_082317/hypothesis/unittestnginxpodcountmod0.py", line 15, in main
    test.test_pod_count(args.duration)
  File "/chaos-eater/sandbox/cycle_20241128_082317/hypothesis/unittestnginxpodcountmod0.py", line 25, in test_pod_count
    assert percentage >= threshold_percentage, f"Pod count was not at least 1 for at least 90% of the time. Actual percentage: {percentage}%"
AssertionError: Pod count was not at least 1 for at least 90% of the time. Actual percentage: 0.00%

```

The error "Pod count was not at least 1 for at least 90% of the time. Actual percentage: 0.00%" indicates that the pod with the label 'app: example' was not found during the test execution. This could be due to a few reasons:

1. The pod might not exist or might not have been deployed yet.
2. The pod might have been deployed in a different namespace than "default".
3. The label selector might be incorrect.

To fix this, I've added a check to ensure that the pod exists before starting the monitoring loop. If the pod is not found, the test will fail immediately with a descriptive message. This helps to pinpoint the issue early on. Additionally, I've included the namespace as an argument to allow for more flexibility in testing.

```

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

class K8sAPIBase:
    def __init__(self):
        # Load Kubernetes configuration based on the environment
        if os.getenv('KUBERNETES_SERVICE_HOST'):
            config.load_incluster_config()
        else:
            config.load_kube_config()

        # Create a Kubernetes API client
        self.v1 = client.CoreV1Api()

class PodCountCheck(K8sAPIBase):
    def test_pod_count(self, duration, namespace="default"): # Added namespace argument
        # Define the threshold percentage
        threshold_percentage = 0.9

```

```

# Check if the pod exists before starting the loop
try:
    pods = self.v1.list_namespaced_pod(namespace, label_selector)
    if not pods.items:
        raise AssertionError("No pods found with label 'app: example'")
except client.ApiException as e:
    if e.status == 404:
        raise AssertionError("Namespace '" + namespace + "' does not exist")
    else:
        raise

# Initialize variables for tracking pod counts
results = []
start_time = time.time()

# Loop for the specified duration
while time.time() - start_time < duration:
    try:
        # Get pods with the label 'app: example'
        pods = self.v1.list_namespaced_pod(namespace, label_selector)
        pod_count = len(pods.items)
        results.append(pod_count)
        print(f"Number of pods with label 'app: example': {pod_count}")
        time.sleep(1)
    except Exception as e:
        print(f"Error: {e}")
        time.sleep(1)

# Calculate the percentage of time the pod count was at least 1
successful_counts = sum(1 for count in results if count >= 1)
percentage = successful_counts / len(results) if results else 0

# Summarize the results
print("\nSummary:")
for i, count in enumerate(results):
    print(f"Second {i+1}: {count} pods")

print(f"Percentage of time with at least 1 pod: {percentage}")

# Assert that the percentage meets the threshold
assert percentage >= threshold_percentage, f"Pod count was not high enough: {percentage}"
print("Test passed!")

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

    test = PodCountCheck()
    test.test_pod_count(args.duration, args.namespace) # Pass namespace to test method

if __name__ == '__main__':
    main()

```

Pod `unittestnginxpodcountmod1-pod` has failed.

```

Traceback (most recent call last):
  File "/chaos-eater/sandbox/cycle_20241128_082317/hypothesis/unittestnginxpodcountmod1.py", line 10, in test_pod_count

```


Summary:

```
main()
File "/chaos-eater/sandbox/cycle_20241128_082317/hypothesis/unitte
test.test_pod_count(args.duration, args.namespace) # Pass namesp
File "/chaos-eater/sandbox/cycle_20241128_082317/hypothesis/unitte
assert percentage >= threshold_percentage, f"Pod count was not a
AssertionError: Pod count was not at least 1 for at least 90% of the
Percentage of time with at least 1 pod: 0.00%
```

The previous error, "Pod count was not at least 1 for at least 90% of the time. Actual percentage: 0.00%", even after confirming the pod's existence, suggests a timing issue. The pod might be starting up slowly, or there might be a delay in the Kubernetes API reflecting the pod's status. To address this, I've added a short initial delay before the monitoring loop begins. This gives the pod some time to become ready. Additionally, I've improved the error reporting within the loop to provide more context in case of failures, including the current time and the specific exception encountered.

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

class K8sAPIBase:
    def __init__(self):
        # Load Kubernetes configuration based on the environment
        if os.getenv('KUBERNETES_SERVICE_HOST'):
            config.load_incluster_config()
        else:
            config.load_kube_config()

        # Create a Kubernetes API client
        self.v1 = client.CoreV1Api()

class PodCountCheck(K8sAPIBase):
    def test_pod_count(self, duration, namespace="default"):
        # Define the threshold percentage
        threshold_percentage = 0.9

        # Check if the pod exists before starting the loop
        try:
            pods = self.v1.list_namespaced_pod(namespace, label_selector='app=chaos-eater')
            if not pods.items:
                raise AssertionError("No pods found with label 'app=chaos-eater'")
        except client.ApiException as e:
            if e.status == 404:
                raise AssertionError("Namespace '" + namespace + "' does not exist")
            else:
                raise

        # Add an initial delay to allow the pod to become ready
        print("Waiting for the pod to become ready...")
        time.sleep(5) # Wait for 5 seconds

        # Initialize variables for tracking pod counts
        results = []
        start_time = time.time()
```

```

# Loop for the specified duration
while time.time() - start_time < duration:
    try:
        # Get pods with the label 'app: example'
        pods = self.v1.list_namespaced_pod(namespace, label_selector='app=example')
        pod_count = len(pods.items)
        results.append(pod_count)
        print(f"Number of pods with label 'app: example': {pod_count}")
        time.sleep(1)
    except Exception as e:
        print(f"Error at {time.ctime()}: {e}") # Improved error handling
        time.sleep(1)

# Calculate the percentage of time the pod count was at least 1
successful_counts = sum(1 for count in results if count >= 1)
percentage = successful_counts / len(results) if results else 0

# Summarize the results
print("\nSummary:")
for i, count in enumerate(results):
    print(f"Second {i+1}: {count} pods")

print(f"Percentage of time with at least 1 pod: {percentage}")

# Assert that the percentage meets the threshold
assert percentage >= threshold_percentage, f"Pod count was not at least 1 for at least {threshold_percentage*100}% of the time"
print("Test passed!")

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

    test = PodCountCheck()
    test.test_pod_count(args.duration, args.namespace)

if __name__ == '__main__':
    main()

```

Pod `unittestnginxpodcountmod2-pod` has failed.

```

Traceback (most recent call last):
  File "/chaos-eater/sandbox/cycle_20241128_082317/hypothesis/unittestnginxpodcountmod2.py", line 10, in test_pod_count
    test.test_pod_count(args.duration, args.namespace)
  File "/chaos-eater/sandbox/cycle_20241128_082317/hypothesis/unittestnginxpodcountmod2.py", line 10, in test_pod_count
    assert percentage >= threshold_percentage, f"Pod count was not at least 1 for at least {threshold_percentage*100}% of the time"
AssertionError: Pod count was not at least 1 for at least 90% of the time

```

Summary:

```

Percentage of time with at least 1 pod: 0.00%
main()
File "/chaos-eater/sandbox/cycle_20241128_082317/hypothesis/unittestnginxpodcountmod2.py", line 10, in test_pod_count
    test.test_pod_count(args.duration, args.namespace)
File "/chaos-eater/sandbox/cycle_20241128_082317/hypothesis/unittestnginxpodcountmod2.py", line 10, in test_pod_count
    assert percentage >= threshold_percentage, f"Pod count was not at least 1 for at least {threshold_percentage*100}% of the time"
AssertionError: Pod count was not at least 1 for at least 90% of the time

```

Despite adding an initial delay, the pod count might still be 0 if the pod is not running or is taking an unusually long time to start. The error "Pod count was not at least 1 for at least 90% of the time" indicates that the pod count was 0 for more than 90% of the duration.

least 90% of the time. Actual percentage: 0.00%" indicates that no pods with the specified label were found throughout the duration of the test. To address this, I will implement a retry mechanism within the loop. Instead of immediately failing if no pods are found, the code will retry the pod listing operation multiple times with a short delay between each attempt. This will increase the robustness of the test and handle cases where the pod takes longer to appear in the API.

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

class K8sAPIBase:
    def __init__(self):
        # Load Kubernetes configuration based on the environment
        if os.getenv('KUBERNETES_SERVICE_HOST'):
            config.load_incluster_config()
        else:
            config.load_kube_config()

        # Create a Kubernetes API client
        self.v1 = client.CoreV1Api()

class PodCountCheck(K8sAPIBase):
    def test_pod_count(self, duration, namespace="default"):
        # Define the threshold percentage
        threshold_percentage = 0.9

        # Initialize variables for tracking pod counts
        results = []
        start_time = time.time()

        # Loop for the specified duration
        while time.time() - start_time < duration:
            try:
                # Retry mechanism to handle cases where the pod takes time to appear
                retries = 3
                for i in range(retries):
                    pods = self.v1.list_namespaced_pod(namespace, label_selector="app=example")
                    if pods.items:
                        break # Pod found, exit retry loop
                else:
                    if i < retries - 1:
                        print(f"No pods found, retrying ({i+1})/3")
                        time.sleep(2) # Wait before retrying
                    else:
                        raise AssertionError(f"No pods found with label 'app=example' in namespace '{namespace}'")

                pod_count = len(pods.items)
                results.append(pod_count)
                print(f"Number of pods with label 'app: example': {pod_count}")
                time.sleep(1)
            except Exception as e:
                print(f"Error at {time.ctime()}: {e}")
                time.sleep(1)

        # Calculate the percentage of time the pod count was at least threshold_percentage
        successful_counts = sum(1 for count in results if count >= threshold_percentage * len(results))
        percentage = successful_counts / len(results)
        print(f"Pod count test result: {percentage * 100}%")
```

```

percentage = successful_counts / len(results) if results else 0

# Summarize the results
print("\nSummary:")
for i, count in enumerate(results):
    print(f"Second {i+1}: {count} pods")

print(f"Percentage of time with at least 1 pod: {percentage}")

# Assert that the percentage meets the threshold
assert percentage >= threshold_percentage, f"Pod count was not at least 1 for at least {threshold_percentage*100}% of the time"
print("Test passed!")

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

    test = PodCountCheck()
    test.test_pod_count(args.duration, args.namespace)

if __name__ == '__main__':
    main()

```

Pod `unittestnginxpodcountmod3-pod` has failed.

```

Summary:
Percentage of time with at least 1 pod: 0.00%
Traceback (most recent call last):
  File "/chaos-eater/sandbox/cycle_20241128_082317/hypothesis/unittestnginxpodcountmod3.py", line 10, in <module>
    main()
  File "/chaos-eater/sandbox/cycle_20241128_082317/hypothesis/unittestnginxpodcountmod3.py", line 14, in main
    test.test_pod_count(args.duration, args.namespace)
  File "/chaos-eater/sandbox/cycle_20241128_082317/hypothesis/unittestnginxpodcountmod3.py", line 20, in test_pod_count
    assert percentage >= threshold_percentage, f"Pod count was not at least 1 for at least {threshold_percentage*100}% of the time"
AssertionError: Pod count was not at least 1 for at least 90% of the time

```

AssertionError: MAX_MOD_COUNT_EXCEEDED: 3

Traceback:

```

File "/opt/conda/lib/python3.10/site-packages/streamlit/runtime/scriptrunner/script_runner.py", line 512, in _exec
    exec(code, module.__dict__)
File "/workspace/app/projects/chaos-eater/ChaosEater_demo.py", line 431, in <module>
    main()
File "/workspace/app/projects/chaos-eater/ChaosEater_demo.py", line 397, in main
    output = st.session_state.chaoseater.run_ce_cycle(
File "/workspace/app/projects/chaos-eater/chaos_eater/chaos_eater.py", line 14, in run_ce_cycle
    hypothesis_logs, hypothesis = self.hypothesizer.hypothesize(
File "/workspace/app/projects/chaos-eater/chaos_eater/hypothesis/hypothesizer.py", line 14, in hypothesize
    steady_state_logs, steady_states = self.steady_state_definer.define_steady_state(
File "/workspace/app/projects/chaos-eater/chaos_eater/hypothesis/steady_state_definer.py", line 14, in define_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}"
```
