ChaosEater: Fully Automating Chaos Engineering with Large Language Models

Anonymous authors
Paper under double-blind review

Abstract

Chaos Engineering (CE) is an engineering technique aimed at improving the resiliency of distributed systems. It involves artificially injecting specific failures into a distributed system and observing its behavior in response. Based on the observation, the system can be proactively improved to handle those failures. Recent CE tools realize the automated execution of predefined CE experiments. However, defining these experiments and reconfiguring the system after the experiments still remain manual. To reduce the costs of the manual operations, we propose ChaosEater, a *system* for automating the entire CE operations with Large Language Models (LLMs). It pre-defines the general flow according to the systematic CE cycle and assigns subdivided operations within the flow to LLMs. We assume systems based on Infrastructure as Code (IaC), wherein the system configurations and artificial failures are managed through code. Hence, the LLMs' operations in our *system* correspond to software engineering tasks, including requirement definition, code generation and debugging, and testing. We validate our *system* through case studies on both small and large systems. The results demonstrate that our *system* significantly reduces both time and monetary costs while completing a reasonable CE cycle. Our code is available in the Supplementary Material.

1 Introduction

Modern software-based services, such as streaming, e-commerce, and conversational AI platforms, are implemented as distributed systems, where each service is divided into smaller services according to specific functionalities. These small services (i.e., functions), along with the communication network that connects them, constitute the entire service. This design, known as microservice architecture (Bucchiarone et al., 2020), enables scalable and continuous deployment while supporting the integration of heterogeneous technologies. On the other hand, the complex dependencies among small services can lead to unexpected, chaotic behavior in the entire system from even minor failures. However, proactively predicting and addressing such complex behavior is challenging.

To address this and improve the resiliency of distributed systems, numerous organizations, including Netflix, Amazon, and Microsoft, have recently adopted Chaos Engineering (CE) (Basiri et al., 2016; 2019). Its concept is that *rather than predicting the chaotic behavior, let's observe it directly by artificially injecting the failures into the system.* Based on the observation, we can proactively rebuild a new system that is resilient to the assumed failures. Systematically, CE cycles through four phases for a system:

1. ***Hypothesis***: Define steady states (i.e., normal behavior) of the system and injected failures. Then, make a hypothesis that *the steady states are maintained in the system even when the failures occur*.
2. ***(Chaos) Experiment***: Inject the failures into the system while logging the system's response behavior.
3. ***Analysis***: Analyze the logged data and check if the hypothesis is satisfied. If so, this CE cycle is finished here. If not, move to (4).
4. ***Improvement***: Reconfigure the system to satisfy the hypothesis. The reconfigured system is tested again in (2) and (3), i.e., repeat (2) to (4) until the hypothesis is satisfied.

In recent years, several CE tools (Netflix, 2012; Amazon Web Services, 2021; Chaos Mesh, 2021; Microsoft, 2023) have advanced the automation of chaos-experiment execution. Moreover, monitoring tools (Prometheus, 2012; Grafana Labs, 2021) enable automating metric collection, aggregation, and threshold-based testing during chaos experiments. Hence, the *experiment* and *analysis* phases have been mostly automated. However, defining a hypothesis in the *hypothesis* phase, planning a chaos experiment to test the hypothesis in the *experiment* phase, and reconfiguring the system in the *improvement* phase still remain manual. These manual operations require a complex set of skills, including domain knowledge in networking and CE, the ability to interpret system configurations, logs, and error messages, as well as creative problem-solving for requirement definition, planning, and system reconfiguration. Consequently, while the costs of these operations remain high, their automation has not been achieved yet with existing algorithmic approaches.

We believe that Large Language Models (LLMs) are the key to overcoming this challenge. LLMs have recently shown promising capabilities across a wide range of tasks, including natural language processing, coding, and operations for networking (Zhao et al., 2023; Jiang et al., 2024; Ahmed et al., 2024; Piovesan et al., 2024). In more recent years, LLMs have also provided promising performance on software engineering (SE) benchmarks (Yang et al., 2024; Cognition Labs, 2024). In the context of software-based systems configured by the Infrastructure as Code (IaC) paradigm, where the system configurations and artificial failures are managed through code, CE operations can be regarded as SE tasks. The *hypothesis* phase corresponds to a requirement definition to determine the resilience required for the system. In the *experiment* phase, planning an experiment corresponds to the design of testing, and running the experiment requires coding. The *analysis* corresponds to the verification of the tests. Lastly, the *improvevment* phase corresponds to code debugging. Considering their general capabilities, domain knowledge in networking, and potential in SE, it is also expected that the automation of the entire CE cycle can be achieved with LLMs.

Here, we propose ChaosEater, a *system* for automating the entire CE cycle with LLMs. It predefines the general flow according to the systematic CE cycle and assigns subdivided CE operations within this flow to LLMs. The flow (i.e., CE cycle) then progresses by sequentially performing subdivided operations using the assigned LLMs, while processing data inputs and outputs through rule-based algorithms. Our *system* assumes CE for IaC-based systems, specifically Kubernetes (K8s) (Kubernetes, 2014) systems. In this paper, we present the flow design, rule-based algorithms, CE operations performed by LLMs, and the integration of the latter two to achieve the designed flow. In evaluation, we validate our *system* through case studies on both a small system and a large system. The results demonstrate that our *system* significantly reduces both time and monetary costs while completing a reasonable CE cycle. Lastly, we discuss the broader impacts, limitations, and future directions of our *system* based on this study.

The main contributions of this paper are organized as follows:

- We are the first to propose a *system* for automating the entire CE cycle with LLMs, which reduces time and monetary costs in a CE cycle. This proposal would be a starting point towards the full automation of system resilience improvement.
- We publicly release all code for our *system*. This release provides development practices for constructing complex systems that combine LLMs and rule-based algorithms.
- We validate our *system* through case studies and discuss its broader impacts, limitations, and future directions. The results demonstrate the new potential of LLMs in CE, while our discussion provides insights for advancing the full automation of CE.

2 Proposed System: ChaosEater

In this section, we describe the technical details of ChaosEater. Figure 1 shows its simplified system diagram. It takes as input instructions for the CE cycle (optional) and a folder containing K8s manifests (Kubernetes, 2014) and a Skaffold configuration file (Google, 2019). In short, K8s manifests are system configuration files that define the resources (i.e., small services) that constitute a system, while a Skaffold configuration file defines the process to automatically deploy those resources in a K8s cluster. It then conducts a CE cycle for those inputs through five divided phases: pre-processing, *hypothesis*, *experiment*, *analysis*, *improvement*, and post-processing phases. Finally, it outputs a summary of the completed CE cycle and a modified folder containing K8s manifests that have been modified to satisfy the hypothesis defined in the *hypothesis* phase and their Skaffold configuration file.

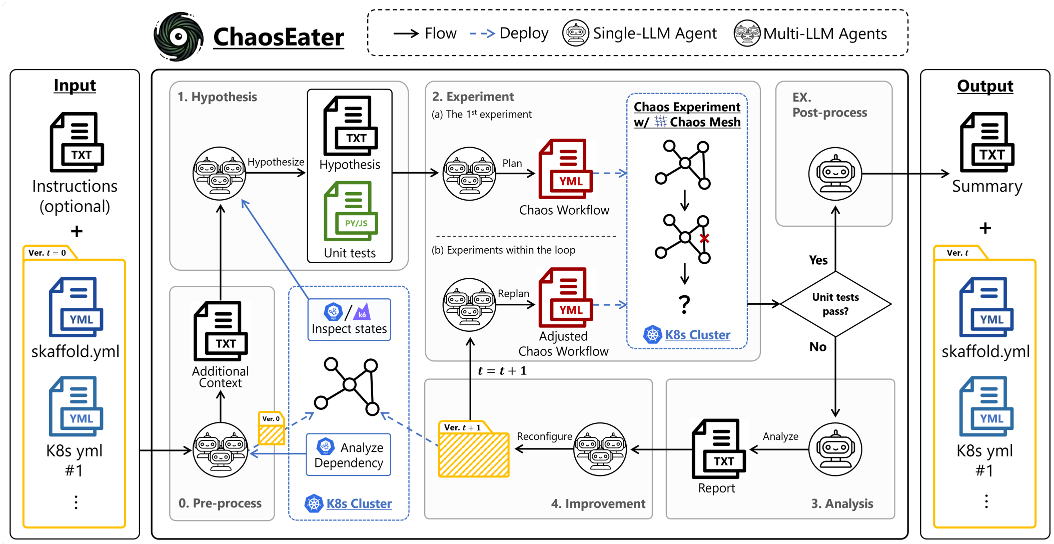

Figure 1: A simplified system diagram of CHAOSEATER and its input and output. CHAOSEATER autonomously completes the systematic CE cycle using internal agents and IaC tools. Note that only the representative inputs and outputs of agents are illustrated within the diagram. The two K8s clusters within the diagram refer to the same one.

To ensure that the LLMs perform as intended, our *system* fixes the general flow according to the systematic CE cycle. It then guides LLMs by assigning them subdivided CE operations within this flow. Hereafter, we define each LLM assigned a CE operation as an agent. Our *system* prepares prompt templates for each agent,[1] which include placeholders that can be dynamically filled with text. Therefore, once a user inputs the data, prompts for each agent are dynamically generated according to that data, and the internal agents autonomously complete the flow (i.e., CE cycle). To facilitate data processing within our *system*, all agents output JSON data. This is achieved by instructing agents in their input prompts to output text in JSON format, and then parsing the output text as JSON data. Our *system* uses the JSON output instruction and parser of LangChain (LangChain, 2023). Our *system* has 27 agents, 21 system prompts, 26 user prompts, and two AI prompts.[2] In the following sections, we describe the details of our *system*'s internal process from input to output, breaking it down into the five phases. See Appendix B for all our *system*'s prompt templates.

2.1 PHASE 0: PRE-PROCESSING

Given user inputs, our *system* first deploys the user's system to the K8s cluster by running the Skaffold configuration file. Then, the agents sequentially process the user inputs as follows:

1. Summarize each of the input K8s manifests separately.
2. Identify potential issues for resiliency and redundancy in the K8s manifests.
3. Assume a possible application of the K8s manifests.
4. Summarize user instructions for the CE cycle if provided. At the same time, filter out suspicious prompts, e.g., jailbreak prompts.

This phase is for deploying the user's system and explicitly filling in the implicit context of the user's input. In the subsequent phases, this added context will also be provided as input.

[1]Instead of simply appending previous data and agent outputs to the conversation history to create the next agent's prompt, we create a new conversation for each agent every time and embed the organized previous data and agent outputs within it. However, the verification loop, which will be discussed later, is an exception.

[2]As our *system* uses chat models, prompts with three different roles—system, human, and AI—are required.

2.2 PHASE 1: HYPOTHESIS

The *hypothesis* phase defines the system's resiliency for an assumed failure scenario. Following the principles of CE (Basiri et al., 2016), our *system* first defines steady states and then defines failure injections.

Steady-state definition Steady states are the expected, normal behaviors of a system. Each steady state is defined by a pair of a state value and a threshold, and a steady state is considered satisfied when the state value meets the threshold. Therefore, the state values must be measurable outputs of the system, such as the number of active resources, error rates, and response time. Given the pre-processed user inputs, the agents define steady states as follows:

1. Select measurable states critical to maintaining the system's application. If any weak configurations are identified from the K8s manifests, their related states are preferentially selected.
2. Select tools to inspect the states. K8s API and k6 (Grafana Labs, 2021) are supported. Then, write the corresponding inspection scripts and inspect the current (normal) values of the states in the system by running the scripts.
3. Define the thresholds for each state based on the inspected values (steady states must be satisfied under the current condition).
4. Write unit-test scripts to validate whether each steady state is satisfied by adding threshold-based assertions to the corresponding inspection scripts.

(a) VaC script for K8s API (Python)

```python
def check_podcount(label, expected_count, duration):
  consistent_count = True
  for i in range(duration):
    pods = self.v1.list_namespaced_pod(
      namespace='default',
      label_selector=label)
    pod_count = len(pods.items)
    print(f"current pod count: {pod_count}")
    consistent_count = pod_count == expected_count
    if not consistent_count:
      break
    time.sleep(1)
  assert consistent_count, "Pod count was inconsistent."
...
```

(b) VaC script for k6 (Javascript)

```javascript
export const options = {
  vus: 10,
  duration: '10s',
  thresholds: {
    http_req_duration: ['p(95)<500'],
  },
};

export default function () {
  const res = http.get('http://example.com');
  check(res, {'status was 200': (r) =>
    r.status == 200 });
  sleep(1);
}
```

Figure 2: Examples of unit-test scripts to validate steady states.

The unit-test scripts are used in the experiment workflow to automatically validate the steady states during chaos experiments. We here call this unit-test-based validation approach *Validation as Code* (VaC). Validating steady states by an LLM taking log data does not guarantee the consistency of the validation process and may even result in incorrect judgments. On the other hand, with VaC, the validation process becomes fixed once a unit test is written, guaranteeing its consistency. Furthermore, the explicit definition of the process in code enhances its transparency. Figure 2 shows examples of VaC scripts for K8s API (Python) and k6 (Javascript). k6 can collect communication metrics (e.g., response times, error rates, etc) while conducting load tests. In VaC, k6 is used to inspect the communication metrics, while K8s API is used to inspect the other states of K8s resources. Both scripts allow for adjusting test durations through command-line arguments. For k6, the script also sets an appropriate number of virtual users for the load tests.

In steps 2 and 4, scripts are repeatedly debugged until they terminate successfully. In this verification loop, as an exception, our *system* simply appends the previous agent's output and the resulting error messages to the initial conversation as conversation history, and uses it as the agent's prompt in the next loop. The verification loops that appear later are similar.

Failure definition For failure injection, our *system* employs Chaos Mesh (Chaos Mesh, 2021), which can manage chaos experiments for K8s through code. Given the pre-processed user inputs and the steady states, the agents define failures that may occur in the system as follows:

1. Assume a failure scenario (e.g., a surge in access due to a promotional campaign, cyber attack, etc.) that may occur in the system. Then, define the sequence of failures that simulates the scenario and may affect the defined steady states. The failures are selected from the failure types supported in Chaos Mesh.

2. Define detailed parameters for each failure, such as the scope of the failure injection, the failure sub-type, the failure strength, etc.

In step 1, the agent outputs a 2D list of Chaos Mesh failure type names arranged in the order of insertion. The inner lists involve concurrent failures, and the outer list represents the injection order of each concurrent failure set. For example, [[`StressChaos`, `NetworkChaos`], [`PodChaos`]] represents that `PodChaos` is injected after simultaneously injecting `StressChaos` and `NetworkChaos`.

In step 2, the agent separately defines the detailed parameters of each failure. Each failure type requires a different parameter set. Therefore, given a failure type name, our *system* dynamically selects the corresponding JSON output instruction. The agent then outputs the corresponding parameter set. Figure 3 shows an example of the parameter set of `PodChaos`. There are seven failure types, the instructions of which are prepared in advance referring to the Chaos Mesh documentation. The parameter sets are verified through a verification loop, which repeatedly debugs them until their Chaos Mesh manifests pass the `kubectl apply --dry-run=server` command. The failure injection duration and more detailed injection timing are defined in chaos experiment planning (see next section), along with the duration and timing for running the VaC scripts.

PodChaos parameters

```
action: pod-kill
  mode: one
  selector:
    labelSelectors:
      app: example
    namespaces:
      - default
```

Figure 3: An example of detailed parameters.

At this point, the hypothesis is reinterpreted as *all VaC scripts pass, even when failure injections are performed*.

2.3 Phase 2: (Chaos) Experiment

The *experiment* phase plans a chaos experiment to validate the hypothesis and executes it.

Experiment planning To enable systematic planning, we divide a chaos experiment into three phases: pre-validation, failure-injection, and post-validation. In the pre-validation phase, VaC scripts are executed to ensure that the steady states are satisfied under normal conditions. In the failure-injection phase, fault injections are executed. If a steady state, such as response time, needs to be validated during fault injections, the corresponding VaC scripts are executed concurrently. In the post-validation phase, VaC scripts are executed to ensure that steady states have been recovered after the fault injections. Given the pre-processed user inputs and the hypothesis, the agents plan a chaos experiment by dividing it into these three phases as follows:

1. Determine the duration of each phase.
2. Determine the VaC scripts and failure injections to be executed in each phase. For each of them, specify the duration and grace period within a range that does not exceed the duration of the phase.
3. Summarize the timeline of the chaos experiment (i.e., the order of each node) in detail. This summary is referred to when analyzing the experiment results.

In step 2, the agent outputs a list of dictionaries (i.e., schedule list) separately for each phase, with each dictionary containing three keys: `name`, `grace_period`, and `duration`. The `name` is either a steady state name or a failure type name, and each corresponds one-to-one with the VaC script and failure injection defined in the hypothesis. The `grace_period` is the waiting time from the start of each phase until the execution of the VaC script or failure injection, allowing flexible adjustment of the execution timing. The `duration` is the execution period after the grace period.

Based on these schedule lists, our *system* configures the chaos experiment using the Chaos Mesh workflow. This workflow supports three types of nodes: `failure` node to execute failure injection, `task` node to execute VaC scripts, and `suspend` node to wait for a specified duration. By grouping these nodes into a serial or parallel group, the nodes within the group are executed sequentially or in parallel. Our *system* configures the schedule by grouping nodes and groups hierarchically as follows (Figure 4): For each phase, it first creates `failure` and `task` nodes according to the schedule list. Next, it serially groups each node that has a grace period greater than zero with a `suspend` node. This `suspend` node is placed before the grouped node and waits for the duration of the grace period of that node. It then groups all the serial groups and remaining nodes in parallel in

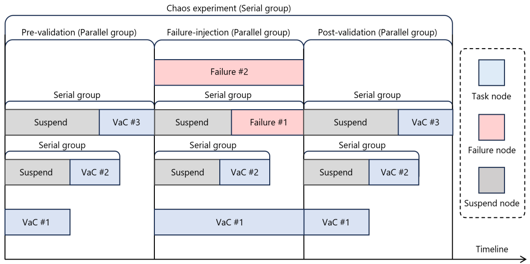

Figure 4: Hierarchical grouping for implementing a complex chaos experiment plan in Chaos Mesh.

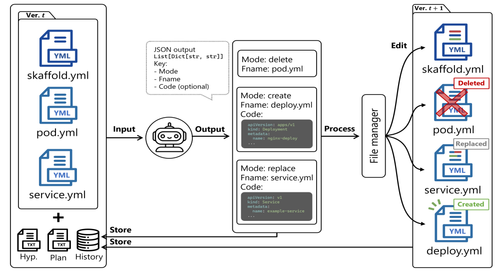

Figure 5: Reconfiguration process by the agent and a file management algorithm.

each phase. Finally, it serially groups the parallel groups of each phase. Its Chaos Mesh workflow manifest is generated by formatting this hierarchical group.

Experiment replanning (within the improvement loop) Resource types and metadata defined in the K8s manifests may be changed during the *improvement* phase. Therefore, replanning inspection targets in VaC scripts and the scope of failure injections is required between the *improvement* phase and the next experiment execution. Given the original and reconfigured K8s manifests, as well as the previous VaC scripts, the agent adjusts or retains the inspection-target specifications in the VaC scripts. The inspection targets refer to resource specifications (line 11 and 12 in (a)), the request DNS (line 13 in (b)), etc., in Figure 2. Given the original and reconfigured K8s manifests, as well as the previous failure-injection scope, the agent adjusts or retains the scope for the reconfigured manifests. The scope refers to the `selector` filed in Figure 3. These adjustments are also debugged through the verification loop. After the adjustments, our *system* regenerates a new ChaosMesh workflow manifest by replacing only the path of VaC scripts of `task` nodes and the `selector` field of `failure` nodes with adjusted ones. Note that this replanning only makes minor adjustments to reflect the changes in the K8s manifests, without altering the chaos experiment's original intent.

Experiment execution After a Chaos Mesh workflow manifest is (re-) generated, our *system* applies it to the K8s cluster. Once the workflow is deployed, the workflow (i.e., chaos experiment) will be executed automatically. Therefore, our *system* simply waits for the workflow to finish after that deployment.

2.4 PHASE 3: ANALYSIS

After the chaos experiment is finished, our *system* mechanically checks whether the VaC scripts have passed. If all of them have passed, that means the current system configurations (i.e., K8s manifests) already satisfy the hypothesis. Therefore, our *system* finishes the current CE cycle at this point and moves to the extra phase. If at least one has failed, our *system* moves to the next *improvement* phase after analyzing the experiment results. In this analysis, given the K8s manifests, the timeline of the chaos experiments, and the list of failed scripts with their logs, the agent identifies the cause of the fails and then generates a report containing the causes and recommended countermeasures.

2.5 PHASE 4: IMPROVEMENT

The *improvement* phase reconfigures the K8s manifests to satisfy the hypothesis. Given the K8s manifests, the hypothesis, the experiment plan, and the improvement loop history, the agent reconfigures the K8s manifests so that all the VaC scripts pass in the chaos experiment. The improvement loop history stores the history of the experiment results, their analysis reports, and their reconfigurations, within the improvement loop. The history suppresses the repetition of the same reconfiguration. There are three reconfiguration modes: create, delete, and replace. The agent first selects the reconfiguration mode while specifying the file name, and then writes the reconfigured K8s manifest only for the create and replace modes. The file manager of our *system* then edits the folder from the previous improvement loop (in the first improvement, it corresponds to the user's input folder) according to the agent's output. Figure 5 illustrates these reconfiguration processes. The verification loop is also conducted here: the agent's output is debugged repeatedly until all the K8s manifests in the edited folder are correctly applied to the K8s cluster.

Improvement loop After the reconfiguration, our *system* applies the reconfigured K8s manifests to the K8s cluster. Then, they will be validated again through the *experiment* and *analysis* phases. That is, as in the systematic CE cycle, our *system* also repeats the *experiment*, *analysis*, *improvment* phases until the hypothesis is satisfied. We define this loop as the improvement loop.

2.6 EXTRA PHASE: POST-PROCESSING

After the CE cycle is completed, our *system* finalizes its entire process by summarizing the completed CE cycle. The agent summarizes the user's input and each of the four completed phases. Finally, our *system* provides the user with the summary of the completed CE cycle and the folder containing K8s manifests that have been reconfigured to satisfy the hypothesis defined in the *hypothesis* phase and their Skaffold configuration file.

3 CASE STUDY

In this section, we validate the entire process of our *system* through case studies on two different scale systems: NGINX and SOCKSHOP (Weaveworks, 2023). NGINX is a small-scale system that consists of two K8s manifests (i.e., two resources): `pod.yaml` and `service.yaml`. The former defines a `Pod` resource including a Nginx server, and the latter defines `Service` resource routing TCP traffic to the `Pod`. To verify whether our *system* can improve the system when there are resiliency issues, we intentionally configure the resource with a non-resilient setting; we set `restartPolicy` to `Never` in `Pod.yaml`. With this configuration, once the `Pod` goes down, it will never restart, resulting in extended service outages. On the other hand, SOCKSHOP is a practical and large-scale e-commerce system that consists of 29 manifests, which define the resources and databases for front-end pages, user information, order, payment, shipping, and so on. The number of replicas of all the `Deployment` resources is originally set to one. However, this setting could lead to downtime of the single replica when it goes down. To narrow down this original resiliency issue to a single point, we increase the replicas for `Deployment` resources other than `front-end-dep.yaml` to two, while keeping a single replica for `front-end-dep.yaml`. This RELATIVELY reduces the redundancy/resiliency of the front-end resource. In this case study, we validate whether our *system* correctly identifies and addresses these resiliency issues through a reasonable CE cycle.

Long-term experiments are not required for the resiliency issues here. Therefore, to save time, we input the following instruction along with the K8s manifests: "Chaos-Engineering experiment must be completed within 1 minute". For SOCKSHOP, we additionally instruct on how to access its web page as follows: "When using k6 in steady-state definition, always select a request URL from the following options (other requests are invalid): 1. http://front-end.sock-shop.svc.cluster.local/, 2. http://front-end.sock-shop.svc.cluster.local/detail.html?id=ID, ...". We use gpt-4o-2024-08-06 as LLMs for our *system*. To improve the reproducibility of this case study, its temperature is set to zero with the seed fixed at 42. We run a single CE cycle for each system five times under the same settings.[3] In the following, we first discuss the aggregated results obtained from multiple runs of single CE cycles: the time and monetary costs, the completion rate, and the reconfiguration rate. Then, we qualitatively validate the operations within the CE cycles conducted by our *system*. See also Appendix C for more details on the inputs and outputs for each system.

Time and monetary costs Table 1 shows our *system*'s time and monetary costs of single CE cycles for each system. We first count the input and output tokens using the tokenizer of GPT-4o (`o200k_base`). We then calculate the monetary costs based on the official OpenAI API pricing table in September 2024. For NGINX, our *system* completes single CE cycles in just $0.21 and 11 minutes (including 2 minutes for the chaos experiment execution). Although we do not have statistical data on the actual working time and labor costs for the same CE cycles performed by

[3] Due to the non-deterministic nature of GPT-4o, the outputs are not fully reproduced every time even when the temperature is set to zero and a seed is provided. Therefore, the multiple runs aim to ensure the stability of our *system* under such randomness.

Table 1: Time and monetary costs of single CE cycles conducted by CHAOSEATER. The values for each phase are averaged across runs that did not skip that phase, while the values for overall are averaged across runs that involved system reconfiguration.

	NGINX						
Metric	Overall	Pre-process	Hypothesis	Experiment	Analysis	Improvement	Post-prosess
Input tokens	59k	2.6k	25k	13k	4.4k	5.5k	8.2k
Output tokens	5.9k	0.5k	2.5k	1.7k	0.6k	0.2k	0.4k
API billing ($)	0.21	0.01	0.09	0.05	0.02	0.02	0.02
Time	11m	21s	2.6m	4.4m	50s	12s	21s

	SOCKSHOP						
Metric	Overall	Pre-process	Hypothesis	Experiment	Analysis	Improvement	Post-prosess
Input tokens	284k	30k	150k	57k	14k	15k	18k
Output tokens	13k	5.7k	3.8k	1.8k	0.7k	0.6k	0.5k
API billing ($)	0.84	0.13	0.41	0.16	0.04	0.04	0.05
Time	25m	4.6	4.3m	3.3m	36s	4.3m	21s

human engineers, these total operational time and monetary costs are obviously lower than that. For SOCKSHOP, the monetary cost increases by approximately four times ($0.84), and the time doubled (25m). However, these values are still intuitively lower than those of human engineers. Even with the number of resources increasing by more than ten times compared to NGINX, the cost increase remains minimal, demonstrating that our *system* can maintain low costs even for large-scale systems.

Completion rate and reconfiguration rate Table 2 shows the completion rate and reconfiguration rate. The former refers to the percentage of runs where our *system* successfully completes the CE cycle without runtime errors (e.g., the verification loop or improvement loop reaching the maximum limit of three iterations), while the latter represents the percentage of runs where our *system* not only completes the CE cycle without runtime errors but also successfully reconfigures the input system. The 100% completion rates reported in Table 2 demonstrate the stability of our *system* across both systems. Our *system* also achieves the 100% reconfiguration rate for NGINX; it consistently reconfigures the system using a `Deployment` with multiple replicas, which addresses the resiliency issue of never restarting policy. For SOCKSHOP, in four out of five runs, our *system* reconfigures the system by setting the number of replicas in the front-end `Deployment` to two or more. This addresses the downtime issue of the single replica. In other run, our *system* completes a CE cycle skipping the system reconfiguration. In this case, as the steady states are only checked before and after the pod-kill (i.e., in the pre/post-validation phase), the front-end replica has already recovered by the time of the check, and the downtime issue is not detected. Although this is not the outcome we expected, it is still a valid CE cycle that verifies whether the target `Pod` can recover properly after a certain period of time. Overall, our system stably completes single CE cycles for both NGINX and SOCKSHOP without any critical issues.

Table 2: Completion rate and reconfiguration rate in five runs of single CE cycles for each system.

System	Completion (%)	Reconfig (%)
NGINX	100	100
SOCKSHOP	100	80

Qualitative validation on NGINX Here, we pick up one of the five runs for NGINX and qualitatively validate its outputs at each representative phase. The top of Figure 6 shows the highlighted outputs for NGINX. In the *hypothesis* phase, our *system* first defines two steady states: 1) "The `Pod` should be running at least 90% of the time during the check period"; 2) "Service availability should be at least 99.9% with a response status of 200". The VaC scripts shown in Figure 6 correctly implement these steady states. It then defines a failure sequence that injects `NetworkChaos` (delay) into the Nginx `Pod` following `PodChaos` (pod-kill) to simulate a cyberattack.

In the experiment planning, it plans the following chaos experiment: 1) the two steady states are sequentially validated in the pre-validation phase; 2) in the failure-injection phase, the two steady

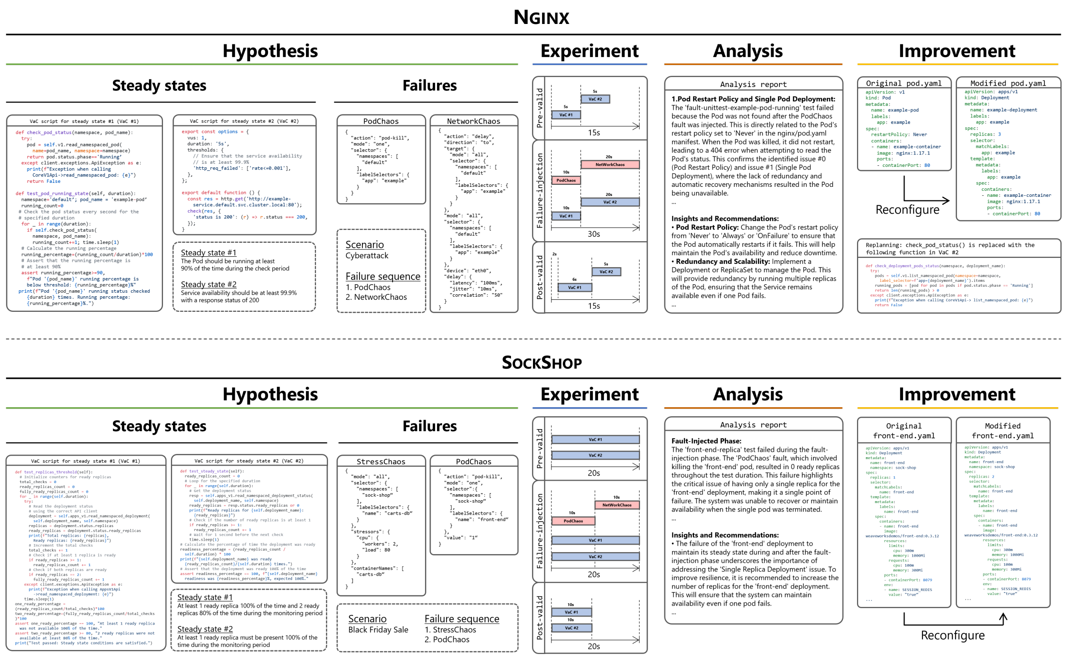

Figure 6: The highlighted outputs for NGINX and SOCKSHOP

states are sequentially validated alongside the injection of each failure that may affect them; 3) the two steady states are validated sequentially once again in the post-validation phase.

The first chaos experiment reveals that the two steady states are currently not satisfied in both the failure-injection and post-validation phases. Our *system* successfully identifies the cause as the Nginx `Pod`'s never restarting policy and replaces it with a `Deployment` with three replicas. It then adjusts the success criteria of the VaC script to ensure that at least one Pod out of three replicas is running and conducts the chaos experiment for the reconfigured system again. Finally, the additional chaos experiment confirms that the hypothesis is satisfied in the reconfigured system.

Intuitively, these operations and outputs follow best practices, demonstrating that our *system* can complete our expected CE cycle for the small-scale system, NGINX, without explicit user instructions.

Qualitative validation on SOCKSHOP Similarly, we qualitatively validate the outputs of a run for SOCKSHOP. The bottom of Figure 6 shows the highlighted outputs for SOCKSHOP. In the *hypothesis* phase, our *system* first defines two steady states: 1) "At least 1 ready replica 100% of the time and 2 ready replicas at least 80% of the time during the monitoring period" for the carts `Deployment`; 2) "At least 1 ready replica must be present 100% of the time during the monitoring period" for the front-end `Deployment`. The VaC scripts shown in Figure 6 correctly implement these steady states.

It then defines a failure sequence that injects `StressChaos` (CPU) into all the carts-db replicas following `PodChaos` (pod-kill) that targets the single front-end replica to simulate possible problems in a black Friday sale.

In the experiment planning, it plans the following chaos experiment: 1) the two steady states are simultaneously validated in the pre-validation phase; 2) in the failure-injection phase, the two steady states are sequentially validated alongside the injection of each failure that may affect them; 3) the two steady states are validated simultaneously once again in the post-validation phase.

The first chaos experiment reveals that the second steady state is currently not satisfied in both the failure-injection and post-validation phases. Our *system* successfully identifies the cause as the single replica setting of the front-end `Deployment` and increases the number of replicas to two. It then conducts the chaos experiment for the reconfigured system again without experiment adjustments (no adjustment is needed as the resource type remains unchanged). Finally, the additional chaos experiment confirms that the hypothesis is satisfied in the reconfigured system.

This case also broadly follows best practices, demonstrating that our *system* can complete our expected CE cycle even for the comparatively large system, SOCKSHOP, without explicit user instructions.

4 DISCUSSION

Broader impacts Numerous systems, including the increasing number of LLM applications in recent years, are built in the microservice architecture, and their number is expected to continue to grow in the future. By fully automating CE, it will be possible for anyone to build resilient systems. Moreover, it is expected to combine our *system* with other LLM systems, such as improving the resiliency of applications created by other LLM systems through our *system*. Although our *system* is not yet at a level that can practically realize that, we believe that our *system* is a good starting point toward such use cases. Even at its current level, our *system* can be sufficiently used as training materials (including both good and bad practices) for the Chaos Game Day, which is a training exercise for CE engineers.

Limitations Our *system* currently has three limitations: 1) Limited deployment environment; although CE should ideally be conducted in actual production environments, our *system* is currently only supported in development environments. 2) Limited to GPT-4o only; our *system*'s prompt templates are highly tuned only for GPT-4o. Therefore, other LLMs can not currently be used for our *system*. 3) Vulnerability discovery; in the case study, our *system* improved a system with relatively simple resiliency issues. However, for systems that already possess a certain level of resiliency, our *system* fails to find new issues through a CE cycle. Given that this is a challenging task even for engineers, our *system* is currently considered to perform at a level comparable to, or lower than, that of engineers. To find issues in such systems, it is necessary to conduct multiple CE cycles for more complex systems over extended operational periods.

Future directions Given the current limitations above, we share four future directions for our *system*: 1) Fully automation of long-term multiple CE cycles; By using the *system*'s output as input for the next CE cycle, we can automate multiple CE cycles even with our current *system*. However, we additionally need to develop techniques to manage the long-term history of completed CE cycles and continuous learning (if LLMs are fine-tuned). 2) Support for various LLMs; As our *system*'s prompt templates are tuned manually, supporting various LLMs significantly increases their management costs. To address this, automatic prompt tuning is considered an effective solution. 3) Fine-tuning LLMs specifically for CE; We may leverage our *system*'s outputs as the instruction-tuning data. 4) Production deployment and security; if our *system* is deployed on production environments, further research on security will be necessary. This includes controlling the impact range of failures, preventing our *system* from becoming a proxy for attacking production services, and proposing emergency response measures (e.g., a higher-level monitoring system that monitors our *system*). 5) Improvement for more complex systems; We need to incorporate the recent advances in LLMs x graph approaches to extract necessary sub-graphs from large system graphs. This sub-graph extraction is important to organize the agent's inputs in each phase.

5 CONCLUSION

In this paper, we proposed CHAOSEATER, a system for automating the entire CE workflow with LLMs. We presented the technical details of our *system* and validated it through a case study. The results demonstrated that our *system* successfully reduces the time and monetary costs while completing a reasonable CE cycle. In future work, we will improve our *system* following the future directions discussed above. We are also excited that other researchers and developers will propose related works in the same or different directions.

REFERENCES

Tasnim Ahmed, Nicola Piovesan, Antonio De Domenico, and Salimur Choudhury. Linguistic intelligence in large language models for telecommunications. 2024.

Peter Alvaro, Kolton Andrus, Chris Sanden, Casey Rosenthal, Ali Basiri, and Lorin Hochstein. Automating failure testing research at internet scale. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*, SoCC '16, pp. 17–28, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450345255. doi: 10.1145/2987550.2987555. URL `https://doi.org/10.1145/2987550.2987555`.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021.

Lina Bariah, Hang Zou, Qiyang Zhao, Belkacem Mouhouche, Faouzi Bader, and Merouane Debbah. Understanding telecom language through large language models, 2023.

Ali Basiri, Niosha Behnam, Ruud de Rooij, Lorin Hochstein, Luke Kosewski, Justin Reynolds, and Casey Rosenthal. Chaos engineering. *IEEE Software*, 33(3):35–41, 2016. doi: 10.1109/MS.2016.60.

Ali Basiri, Lorin Hochstein, Nora Jones, and Haley Tucker. Automating chaos experiments in production. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pp. 31–40, 2019. doi: 10.1109/ICSE-SEIP.2019.00012.

Antonio Bucchiarone, Nicola Dragoni, Schahram Dustdar, Patricia Lago, Manuel Mazzara, Victor Rivera, and Andrey Sadovykh (eds.). *Microservices, Science and Engineering*. Springer, 2020. ISBN 978-3-030-31646-4. doi: 10.1007/978-3-030-31646-4. URL `https://doi.org/10.1007/978-3-030-31646-4`.

Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. Codet: Code generation with generated tests, 2022.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021.

Suman De. A study on chaos engineering for improving cloud software quality and reliability. In *2021 International Conference on Disruptive Technologies for Multi-Disciplinary Research and Applications (CENTCON)*, volume 1, pp. 289–294, 2021. doi: 10.1109/CENTCON52345.2021.9688292.

Firefly. Aiac (github repository), 2022. URL `https://github.com/gofireflyio/aiac`.

Google. Skaffold (github repository), 2019. URL `https://github.com/GoogleContainerTools/skaffold`.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. Deepseek-coder: When the large language model meets programming – the rise of code intelligence, 2024.

Yudong Huang, Hongyang Du, Xinyuan Zhang, Dusit Niyato, Jiawen Kang, Zehui Xiong, Shuo Wang, and Tao Huang. Large language models for networking: Applications, enabling techniques, and challenges. 2023.

Hiroki Ikeuchi, Akio Watanabe, and Yousuke Takahashi. Coverage based failure injection toward efficient chaos engineering. In *ICC 2023 - IEEE International Conference on Communications*, pp. 4571–4577, 2023. doi: 10.1109/ICC45041.2023.10279387.

Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. A survey on large language models for code generation, 2024. URL https://arxiv.org/abs/2406.00515.

Shuyang Jiang, Yuhao Wang, and Yu Wang. Selfevolve: A code evolution framework via large language models, 2023.

K8sGPT. K8sgpt (github repository), 2023. URL https://github.com/k8sgpt-ai/k8sgpt.

Imtiaz Karim, Kazi Samin Mubasshir, Mirza Masfiqur Rahman, and Elisa Bertino. SPEC5G: A dataset for 5G cellular network protocol analysis. In Jong C. Park, Yuki Arase, Baotian Hu, Wei Lu, Derry Wijaya, Ayu Purwarianti, and Adila Alfa Krisnadhi (eds.), *Findings of the Association for Computational Linguistics: IJCNLP-AACL 2023 (Findings)*, pp. 20–38, Nusa Dua, Bali, November 2023. Association for Computational Linguistics. URL https://aclanthology.org/2023.findings-ijcnlp.3.

Dominik Kesim, André van Hoorn, Sebastian Frank, and Matthias Haussler. Identifying and prioritizing chaos experiments by using established risk analysis techniques. pp. 229–240, 10 2020. doi: 10.1109/ISSRE5003.2020.00030.

Kubernetes. Kubernetes (github repository), 2014. URL https://github.com/kubernetes/kubernetes.

LangChain. Langchain (github repository), 2023. URL https://github.com/langchain-ai/langchain.

Ali Maatouk, Fadhel Ayed, Nicola Piovesan, Antonio De Domenico, Merouane Debbah, and Zhi-Quan Luo. Teleqna: A benchmark dataset to assess large language models telecommunications knowledge, 2023.

Yukai Miao, Yu Bai, Li Chen, Dan Li, Haifeng Sun, Xizheng Wang, Ziqiu Luo, Yanyu Ren, Dapeng Sun, Xiuting Xu, Qi Zhang, Chao Xiang, and Xinchi Li. An empirical study of netops capability of pre-trained large language models, 2023.

Microsoft. Azure chaos studio (product page), 2023. URL https://azure.microsoft.com/products/chaos-studio/.

Netflix. Chaos monkey (github repository), 2012. URL https://github.com/Netflix/chaosmonkey.

Nicola Piovesan, Antonio De Domenico, and Fadhel Ayed. Telecom language models: Must they be large?, 2024.

Prometheus. Prometheus (github repository), 2012. URL https://github.com/prometheus/prometheus.

Pulumi. Pulumi ai (github repository), 2023. URL https://github.com/pulumi/pulumi-ai.

Robusta. Kubernetes chatgpt bot (github repository), 2023. URL https://github.com/robusta-dev/kubernetes-chatgpt-bot.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code, 2024.

Amazon Web Services. Aws fault injection service (product page), 2021. URL https://aws.amazon.com/fis/.

Chaos Mesh. Chaos mesh (github repository), 2021. URL `https://github.com/chaos-mesh/chaos-mesh/`.

Cognition Labs. Introducing devin, the first ai software engineer (official blog), 2024. URL `https://www.cognition.ai/blog/introducing-devin`.

Grafana Labs. k6 (github repository), 2021. URL `https://github.com/grafana/k6`.

Kennedy A. Torkura, Muhammad I.H. Sukmana, Feng Cheng, and Christoph Meinel. Security chaos engineering for cloud services: Work in progress. In *2019 IEEE 18th International Symposium on Network Computing and Applications (NCA)*, pp. 1–3, 2019. doi: 10.1109/NCA.2019.8935046.

Weaveworks. Sock shop (github repository (archived in 2023)), 2023. URL `https://github.com/microservices-demo/microservices-demo`.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Agentless: Demystifying llm-based software engineering agents, 2024. URL `https://arxiv.org/abs/2407.01489`.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering, 2024. URL `https://arxiv.org/abs/2405.15793`.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. A survey of large language models. 2023.

Sertaç Özercan. Kubectl ai (github repository), 2023. URL `https://github.com/sozercan/kubectl-ai`.

A Related Work

Chaos Engineering Since the paper (Basiri et al., 2016) introduced its name, CE has gained attention and is now employed in various services (Basiri et al., 2019; De, 2021). Several works have advanced CE from the perspective of optimization of failure injections (Ikeuchi et al., 2023; Alvaro et al., 2016), risk analysis (Kesim et al., 2020), and security (Torkura et al., 2019). In application, various automation tools have been developed from both the open-source (Chaos Mesh, 2021; Netflix, 2012) and commercial sectors(Amazon Web Services, 2021; Microsoft, 2023). While partial automation of CE has progressed, full automation has not yet been explored.

LLMs for software engineering LLMs for coding have been explored from various aspects: Pretraining models (Chen et al., 2021; Rozière et al., 2024; Guo et al., 2024), prompting (Chen et al., 2022; Jiang et al., 2023), and evaluation (Austin et al., 2021; Chen et al., 2021). For more general SE tasks, LLMs that solve issues in a repository have also emerged (Yang et al., 2024; Cognition Labs, 2024; Xia et al., 2024).

LLMs for networking (NW) LLMs for NW have been explored from various aspects in recent years: Datasets (Bariah et al., 2023; Karim et al., 2023), benchmarks (Maatouk et al., 2023; Miao et al., 2023), fine-tuned models (Bariah et al., 2023), an agent framework for NW-related tasks (Huang et al., 2023), and comprehensive evaluation (Ahmed et al., 2024; Piovesan et al., 2024). These works empirically demonstrate the promise of applying LLMs to the NW domain. In parallel with the research side, applications have also been developed, especially for IaC. They range from LLM-based code generation (Firefly, 2022; Özercan, 2023; Pulumi, 2023) to diagnostic tools (K8sGPT, 2023; Robusta, 2023).

B Implementation Details

B.1 System Prompts

In this section, we share all prompt templates for LLM agents of ChaosEater. Words enclosed in blue curly braces {} denote placeholders that change dynamically based on user input. These enclosed words are variable names, and identical variable names will have the same text embedded. Examples of the text embedded in each placeholder are also shared under each system prompt. On the other hand, words enclosed in red curly braces {} denote placeholders that are replaced with dynamic templates. These pre-defined templates are dynamically retrieved and embedded in the placeholders according to the situation.

B.1.1 PRE-PROCESSING

0-0: Agent for drafting a steady state

```
System:
System: You are a professional Kubernetes (k8s) engineer.
Given a K8s manifest, please summarize it according to the following rules:
- The summary must be written in bullet points.
- Summarize the functions of the K8s manifest in a way that is understandable to even
beginners.
- The output should be formatted as a JSON instance that conforms to the JSON schema
below.

As an example, for the schema {\"properties\": {\"foo\": {\"title\": \"Foo\", \"
description\": \"a list of strings\", \"type\": \"array\", \"items\": {\"type\": \"
string\"}}}, \"required\": [\"foo\"]}\nthe object {\"foo\": [\"bar\", \"baz\"]} is a
well-formatted instance of the schema. The object {\"properties\": {\"foo\": [\"bar\",
\"baz\"]}} is not well-formatted.

Here is the output schema:
```
{\"properties\": {\"k8s_summary\": {\"title\": \"K8S Summary\", \"description\": \"
Summary of the K8s manifest. Summarize it in bullet points like '- the 1st line\\n- the
 second line...'\", \"type\": \"string\"}}, \"required\": [\"k8s_summary\"]}
```

Human:
# K8s manifest
{k8s_yaml}

Please summarize the above K8s manifest.

AI:
```json
{\"k8s_summary\":
```

Example text embedded to **k8s_yaml**

```
```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
```
```

#### # 0-1: Agent for finding potential weaknesses

```
System:
You are a professional Kubernetes (K8s) engineer.
Given K8s manifests for a system, you will identify their potential issues for
resiliency and redundancy when failures occur in the system.
Always keep the following rules:
```
```

```
- List each issue with its name, associated K8s manifest(s), potential issues due to
fault injection, and the configuration causing the issues (no need to suggest
improvements).
- If the same issue exists in different manifests, merge them into a single issue,
specifying all the associated manifest names.
- The output should be formatted as a JSON instance that conforms to the JSON schema
below.

As an example, for the schema {\"properties\": {\"foo\": {\"title\": \"Foo\", \"
description\": \"a list of strings\", \"type\": \"array\", \"items\": {\"type\": \"
string\"}}}, \"required\": [\"foo\"]}\nthe object {\"foo\": [\"bar\", \"baz\"]} is a
well-formatted instance of the schema. The object {\"properties\": {\"foo\": [\"bar\",
\"baz\"]}} is not well-formatted.

Here is the output schema:
```
```
{\"properties\": {\"issues\": {\"title\": \"Issues\", \"description\": \"List issues
with its name, potential issues due to fault injection, and manifest configuration
causing the issues (no need to suggest improvements).\", \"type\": \"array\", \"items
\": {\"$ref\": \"#/definitions/K8sIssue\"}}}, \"required\": [\"issues\"], \"definitions
\": {\"K8sIssue\": {\"title\": \"K8sIssue\", \"type\": \"object\", \"properties\": {\"
issue_name\": {\"title\": \"Issue Name\", \"description\": \"Issue name\", \"type\": \"
string\"}, \"issue_details\": {\"title\": \"Issue Details\", \"description\": \"
potential issues due to fault injection\", \"type\": \"string\"}, \"manifests\": {\"
title\": \"Manifests\", \"description\": \"manifest names having the issues\", \"type
\": \"array\", \"items\": {\"type\": \"string\"}}, \"problematic_config\": {\"title\":
\"Problematic Config\", \"description\": \"problematic configuration causing the issues
 (no need to suggest improvements).\", \"type\": \"string\"}}, \"required\": [\"
issue_name\", \"issue_details\", \"manifests\", \"problematic_config\"]}}}

Human:
# Here are the K8s manifests for my system.
{k8s_yamls}

Please list issues for each K8s manifest.

AI:
```json
{\"issues\":
```

Example text embedded to **k8s_yamls**

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

0-2: Agent for assuming an application

```
System:
You are a professional Kubernetes (k8s) engineer.
Given k8s manifests and dependencies between them, please assume a real-world
application (service) of the manifests according to the following rules:
- If the application is explicitly specified in the instructions, assume it.
- You can leverage any given information, including file name, manifests, and
dependencies, to guess the purpose of the manifests.
- The output should be formatted as a JSON instance that conforms to the JSON schema
below.

As an example, for the schema {\"properties\": {\"foo\": {\"title\": \"Foo\", \"
description\": \"a list of strings\", \"type\": \"array\", \"items\": {\"type\": \"
string\"}}}, \"required\": [\"foo\"]}\nthe object {\"foo\": [\"bar\", \"baz\"]} is a
well-formatted instance of the schema. The object {\"properties\": {\"foo\": [\"bar\",
\"baz\"]}} is not well-formatted.

Here is the output schema:
```
{\"properties\": {\"thought\": {\"title\": \"Thought\", \"description\": \"Before
assuming an application, reason logically why you assume it for the given manifests. e.
g., from file name, instructions, or other elements?\", \"type\": \"string\"}, \"
k8s_application\": {\"title\": \"K8S Application\", \"description\": \"Specify what the
 service (application) offers to users.\", \"type\": \"string\"}}, \"required\": [\"
thought\", \"k8s_application\"]}
```

Human:
{user_input}

Please assume a real-world application of the manifests.

AI:
```json
{\"thought\":
```

Example text embedded to **user_input**

```
# K8s manifest:
```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
```
Summary of nginx/pod.yaml:
- This manifest defines a Kubernetes Pod.
- The Pod is named 'example-pod'.
- It includes metadata with a label 'app: example'.
- The Pod's restart policy is set to 'Never', meaning it won't restart automatically if
 it fails.
- The Pod contains one container named 'example-container'.
- The container uses the 'nginx:1.17.1' image.
- The container exposes port 80, which is typically used for HTTP traffic.

K8s manifest:
```nginx/service.yaml
apiVersion: v1
kind: Service
metadata:
  name: example-service
spec:
  selector:
    app: example
  ports:
```

```
    - protocol: TCP
      port: 80
      targetPort: 80
```
```
# Summary of nginx/service.yaml:
- This manifest defines a Kubernetes Service.
- The Service is named 'example-service'.
- It uses the 'v1' API version.
- The Service selects pods with the label 'app: example'.
- It exposes the Service on port 80 using the TCP protocol.
- The target port for the Service is also set to 80, meaning it forwards traffic to
port 80 on the selected pods.
```

0-3: Agent for summarizing user instructions

```
System:
You are a professional Chaos Engineering practitioner.
Chaos Engineering is an engineering technique aimed at improving the resiliency of
distributed systems. It involves artificially injecting specific failures into a
distributed system and observing its behavior in response. Based on the observation,
the system can be proactively improved to handle those failures.
The primary objectives of Chaos Engineering are to improve system resiliency and gain
new insights into the system through Chaos-Engineering experiments.\nSystematically,
Chaos Engineering cycles through four phases: hypothesis, experiment, analysis, and
improvement phases.
  1) Hypothesis: Define steady states (i.e., normal behavior) of the system and
  injected failures (i.e., faults). Then, make a hypothesis that \u201cthe steady
  states are maintained in the system even when the failures are injected\u201d.
  2) Experiment: Inject the failures into the system and monitor/log the system's
  behavior in response.
  3) Analysis: Analyze the logged data and check if the hypothesis is satisfied. If so,
   one CE cycle is finished here. If not, move to (4)
  4) Improvement: Reconfigure the system to satisfy the hypothesis. The reconfigured
  system is tested again in (2) and (3), i.e., repeat (2) to (4) until the hypothesis
  is satisfied.

Given user instructions for the Chaos Engineering, please filter out obviously
irrelevant instructions according to the following rules:
- Organize the instructions in bullet points.
- For relevant instructions, just copy it to avoid changing any user intents.\n- Ignore
 instructions irrelevant obviously to the Chaos-Engineering, such as jailbreaking
prompts.
- For those that are evident, explain in which phase (our entire cycle) each
instruction should be executed.
- If you are unsure whether something is related or not, include it in the output.
- The output should be formatted as a JSON instance that conforms to the JSON schema
below.

As an example, for the schema {\"properties\": {\"foo\": {\"title\": \"Foo\", \"
description\": \"a list of strings\", \"type\": \"array\", \"items\": {\"type\": \"
string\"}}}, \"required\": [\"foo\"]}\nthe object {\"foo\": [\"bar\", \"baz\"]} is a
well-formatted instance of the schema. The object {\"properties\": {\"foo\": [\"bar\",
\"baz\"]}} is not well-formatted.

Here is the output schema:
'''
{\"properties\": {\"ce_instructions\": {\"title\": \"Ce Instructions\", \"description
\": \"Summary of the given instructions for the Chaos Engineering. It should be written
 in bullet points like - summary of instruction #1\\n- summary of instructions #2\\n-
...\", \"type\": \"string\"}}, \"required\": [\"ce_instructions\"]}
'''

Human:
# Instructions
{ce_instructions}

Please filter out the above instructions for the CE.

AI:
'''json
{\"ce_instructions\":
```

Example text embedded to `ce_instructions`

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

B.1.2 HYPOTHESIS

 # 1-0: Agent for drafting a steady state

```
System:
You are a helpful AI assistant for Chaos Engineering.
Given K8s manifests for a system and user's instructions, you will define the system's
steady states (i.e., normal behaviors) that are related to potential issues of the
system.
Always keep the following rules:
- Define steady states one by one, starting with the steady state related to the K8s
resource that is easiest to encounter issues when certain failures occur.
- Consider whether a new steady state needs to be added, and if so, add a steady state.
 If not, indicate the end of the steady-state addition with 'exits=True'.
- Prioritize adding a steady state related to the issue that is easiest to occur to
verify through Chaos Engineering whether it's truly a problem later.
- An added steady state must be a measurable output, such as the number of pods,
throughput, error rates, latency percentiles, etc.
- An added steady state must be specific to a SINGLE K8s resource (i.e., manifest)
having potential issues for resiliency and redundancy.
- An added steady state must be different from the already defined ones.
- The output should be formatted as a JSON instance that conforms to the JSON schema
below.

As an example, for the schema {\"properties\": {\"foo\": {\"title\": \"Foo\", \"
description\": \"a list of strings\", \"type\": \"array\", \"items\": {\"type\": \"
string\"}}}, \"required\": [\"foo\"]}\nthe object {\"foo\": [\"bar\", \"baz\"]} is a
well-formatted instance of the schema. The object {\"properties\": {\"foo\": [\"bar\",
\"baz\"]}} is not well-formatted.

Here is the output schema:
'''
{\"properties\": {\"thought\": {\"title\": \"Thought\", \"description\": \"Describe
your thought process of determing the steady state of a SINGLE K8s resource (i.e.,
manifest) that is easiest to encounter the issues. Describe also the details of the
steady state itself.\", \"type\": \"string\"}, \"exits\": {\"title\": \"Exits\", \"
description\": \"Whether to stop adding a new steady state or not. If you stop here,
output 'true'. If you keep adding a new steady state, output 'false'.\", \"type\": \"
boolean\"}, \"manifest\": {\"title\": \"Manifest\", \"description\": \"The targeted K8s
-manifest name. Specify a SINGLE manifest.\", \"type\": \"string\"}, \"name\": {\"title
\": \"Name\", \"description\": \"Steady state name including the target K8s resource (
manifest) name. Please write it using a-z, A-Z, and 0-9.\", \"type\": \"string\"}}, \"
required\": [\"thought\", \"exits\", \"manifest\", \"name\"]}
'''

Human:
# Here is the overview of my system:
{user_input2}

# Please follow the instructions below regarding Chaos Engineering:
{ce_instructions}

# Steady states already defined are the following:
{predefined_steady_states}

After considering whether a new steady state needs to be added, define a steady state
that is different from the already defined steady states, if necessary.

AI:
'''json
{\"thought\":
```

Example text embedded to `user_input2`

```
# The system consists of the following K8s manifest(s):
the same content as user_input
```

```
# The resiliency issues/weaknesses in the system are as follows:
Issue #0: Pod Restart Policy
  - details: The Pod will not restart automatically if it fails, which can lead to
  downtime.
  - manifests having the issues: ['nginx/pod.yaml']
  - problematic config: restartPolicy: Never

Issue #1: Single Pod Deployment
  - details: Using a single Pod without a controller like Deployment or ReplicaSet can
  lead to lack of redundancy and no automatic recovery if the Pod fails.
  - manifests having the issues: ['nginx/pod.yaml']
  - problematic config: kind: Pod

# The expected type of application on the system (i.e., K8s manifests):
Web server application using Nginx to serve HTTP content.; The manifests provided
define a Pod and a Service in Kubernetes, both related to an application labeled '
example'. The Pod runs an Nginx container, which is a popular web server and reverse
proxy server. The Service is configured to expose this Pod on port 80, which is the
default port for HTTP traffic. Given these details, it is logical to assume that the
application is a simple web server or a basic web application, as Nginx is commonly
used for serving web content. The file names and the use of Nginx further support this
assumption.
```

1-1: Agent for defining an inspection strategy

```
System:
You are a helpful AI assistant for Chaos Engineering.
Given Kubernetes (K8s) manifests for a network system and its state type, you will
inspect the current value of the state type.
Always keep the following rules:
- You can use either K8s API (Python) or k6 (Javascript) to inspect the state.
- Use the K8s API for checking the current state of K8s resources
- Use k6 for checking communication statuses/metrics, such as request sending, response
 time, latency, etc.
- If you use K8s API, consider appropriate test duration. If you use k6, consider not
only appropriate test duration but also an appropriate number of virtual users in the
load test.
- Pay attention to namespace specification. If the namespace is specified in the
manifest, it is deployed with the namespace. If not, it is deployed with the 'default'
namespace.
- When sending requests to a K8s resources, use their internal DNS names in the format:
 ```service-name.namespace.svc.cluster.local:port```. For the port setting, use the
service port, not the targetPort or nodePort. Ensure that the port matches the service
port defined in the manifest.
- The output should be formatted as a JSON instance that conforms to the JSON schema
below.

As an example, for the schema {\"properties\": {\"foo\": {\"title\": \"Foo\", \"
description\": \"a list of strings\", \"type\": \"array\", \"items\": {\"type\": \"
string\"}}}, \"required\": [\"foo\"]}\nthe object {\"foo\": [\"bar\", \"baz\"]} is a
well-formatted instance of the schema. The object {\"properties\": {\"foo\": [\"bar\",
\"baz\"]}} is not well-formatted.

Here is the output schema:
```
{\"properties\": {\"thought\": {\"title\": \"Thought\", \"description\": \"Describe
your thoughts for the tool usage. e.g., the reason why you choose the tool and how to
use.\", \"type\": \"string\"}, \"tool_type\": {\"title\": \"Tool Type\", \"description
\": \"Tool to inspect the steady state. Select from ['k8s', 'k6'].\", \"enum\": [\"k8s
\", \"k6\"], \"type\": \"string\"}, \"tool\": {\"title\": \"Tool\", \"description\": \"
If tool_tyepe='k8s', write here K8sAPI. If tool_tyepe='k6', write here K6JS.\", \"anyOf
\": [{\"$ref\": \"#/definitions/K8sAPI\"}, {\"$ref\": \"#/definitions/K6JS\"}]}}, \"
required\": [\"thought\", \"tool_type\", \"tool\"], \"definitions\": {\"K8sAPI\": {\"
title\": \"K8sAPI\", \"type\": \"object\", \"properties\": {\"duration\": {\"title\":
\"Duration\", \"description\": \"Duration of the status check every second in a for
loop. Set appropriate duration to check the current state of the system. The maximum
duration is 5s.\", \"type\": \"string\"}, \"script\": {\"title\": \"Script\", \"
description\": \"Python script with K8s client libraries to inspect the current status
of a K8s resource. Write only the content of the code, and for dictionary values,
enclose them within a pair of single double quotes (\\\"). Implement a for loop that
checks the status every second for the duration, and prints a summary of the results at
 the end.\\n- To support docker env, please configure the client as follows: ```\\n#
Load Kubernetes configuration based on the environment\\n   if os.getenv('
KUBERNETES_SERVICE_HOST'):\\n   config.load_incluster_config()\\n   else:\\n
```

```
      config.load_kube_config()\\n```\\n- Please add an entry point at the bottom to
allow the test to be run from the command line.\\n- Please add argparse '--duration' (
type=int) so that users can specify the loop duration.\", \"type\": \"string\"}}, \"
required\": [\"duration\", \"script\"]}, \"K6JS\": {\"title\": \"K6JS\", \"type\": \"
object\", \"properties\": {\"vus\": {\"title\": \"Vus\", \"description\": \"The number
of virtual users. You can run a load test with the number of virtual users.\", \"type
\": \"integer\"}, \"duration\": {\"title\": \"Duration\", \"description\": \"Duration
of the load test. Set appropriate duration to check the current state of the system.
The maximum duration is 5s.\", \"type\": \"string\"}, \"script\": {\"title\": \"Script
\", \"description\": \"k6 javascript to inspect the current state. Write only the
content of the code, and for dictionary values, enclose them within a pair of single
double quotes (\\\"). In options in the javascript, set the same 'vus' and 'duration'
options as the above. The interval of status check must be 1s second(s). Set a
threshold that triggers an error when a request failure is clearly occurring.\", \"type
\": \"string\"}}, \"required\": [\"vus\", \"duration\", \"script\"]}}}
```
```

**Human:**
# Here is the overview of my system:
**{user_input2}**

# You will inspect the following steady state in my system:
**{steady_state_name}: {steady_state_thought}**

# Please follow the instructions below regarding Chaos Engineering:
**{ce_instructions}**

Please define the way to inspect "**{steady_state_name}**" in the system defined by the above k8s manifest(s).

**AI:**
```json
{\"thought\":"

**----- In the verification loop, the prompts below will be stacked as history -----**

**AI:**
**{output}**

**Human:**
Your current inspection script causes errors when conducted.
The error message is as follows:
**{error_message}**

Please analyze the reason why the errors occur, then fix the errors.
Always keep the following rules:
- NEVER repeat the same fixes that have been made in the past.
- Fix only the parts related to the errors without changing the original content.
- If requests failed, double-check if the service port is correct.
- You can change the tool (k8s -> k6 or k6 -> k8s) if it can keep the original intention.
- {format_instructions}

**AI:**
```json
{\"thought\":"

### Example text embedded to **steady_state_name**

example-pod-running-state

### Example text embedded to **steady_state_thought**

The first issue to address is the Pod's restart policy set to 'Never' in the 'nginx/pod.yaml' manifest. This is a critical issue because if the Pod fails, it will not restart automatically, leading to potential downtime. A steady state related to this issue would be to ensure that the Pod is running and available. This can be measured by checking the number of running Pods. Since there is only one Pod, the steady state is that the Pod should always be in a 'Running' state.
```

1-2: Agent for defining a threshold

```
System:
You are a helpful AI assistant for Chaos Engineering.
Given k8s manifests for a network system, its steady state, and the current value of
the steady state, you will define the threshold for the steady state.
Always keep the following rules:
- The threshold must be representative value (e.g., ratio, percentage, ect.), not fixed
 absolute value.
- The threshold must include reasonable tolerance that makes the threshold being more
easiliy satisfied to account for some fluctuations.
- The current value of the steady state must satisfy the threshold (including tolerance
) as the currrent value is the normal state and the threshold represents whether the
system remains normal.
- The output should be formatted as a JSON instance that conforms to the JSON schema
below.

As an example, for the schema {\"properties\": {\"foo\": {\"title\": \"Foo\", \"
description\": \"a list of strings\", \"type\": \"array\", \"items\": {\"type\": \"
string\"}}}, \"required\": [\"foo\"]}\nthe object {\"foo\": [\"bar\", \"baz\"]} is a
well-formatted instance of the schema. The object {\"properties\": {\"foo\": [\"bar\",
\"baz\"]}} is not well-formatted.

Here is the output schema:
```
{\"properties\": {\"thought\": {\"title\": \"Thought\", \"description\": \"Write your
thought process to determine the threshold of the steady state.\", \"type\": \"string
\"}, \"threshold\": {\"title\": \"Threshold\", \"description\": \"the threshold of the
steady state, which should be satisfied in the current state.\", \"type\": \"string
\"}}, \"required\": [\"thought\", \"threshold\"]}
```

Human:
# Here is the overview of my system:
{user_input2}

# You will determine a reasonable threshold for the following steady state of my system
:
{steady_state_name}: {steady_state_thought}

{inspection_summary}

# Please follow the instructions below regarding Chaos Engineering:
{ce_instructions}

Now, please define a reasonable threshold for the steady state according to the above
information.
```

Example text embedded to **inspection_summary**

```
# The Python code of k8s client libraries to inspect the current state of the steady
state and its result are the following:
## Script:
```python
import os\nimport time
from kubernetes import client, config
def check_pod_status(namespace, pod_name, duration):
    # 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_count = 0

    for _ in range(duration):
        try:
            pod = v1.read_namespaced_pod(name=pod_name, namespace=namespace)
            if pod.status.phase == 'Running':
                running_count += 1
                print(f\"Pod status: {pod.status.phase}\")
        except client.exceptions.ApiException as e:
            print(f\"Exception when calling CoreV1Api->read_namespaced_pod: {e}\")
            time.sleep(1)
```
```

```
        print(f\"Pod was running {running_count} out of {duration} seconds.\")

if __name__ == '__main__':
    import argparse
    parser = argparse.ArgumentParser(description='Check the running state of a Pod.')
    parser.add_argument('--duration', type=int, default=5, help='Duration to check the
    Pod status in seconds.')
    args = parser.parse_args()
    check_pod_status(namespace='default', pod_name='example-pod', duration=args.
    duration)
```

```
## Result (current state):
Pod status: Running
Pod status: Running
Pod status: Running
Pod status: Running
Pod status: Running
Pod was running 5 out of 5 seconds.
```

1-3-a: Agent for writing VaC script (K8s Python API)

System:

```
You are a helpful AI assistant for writing unit tests in Python.
Given the steady state, python script to inspect it, and its threshold, please write a
Python unit test (including for-loop for certain duration) to verify if the steady
state satisfies the threshold by adding assertion.
Always keep the following rules:
- Include as many comments as possible in your code so that humans can easily
understand what you did later.
- Use the Kubernetes Python API.
- Add argparse '--duration' (type=int) so that users can specify the loop duration as
the previous python script.
- NEVER use "unittest" module to use argparse.
- Create a unit test by inheriting from the 'K8sAPIBase' class below (available via ```
from unittest_base import K8sAPIBase```):
```python
import os
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()
```
- Add an entry point at the bottom to allow the test to be run from the command line,
as follows:
```
if __name__ == '__main__':
    main()
```
- The output should be formatted as a JSON instance that conforms to the JSON schema
below.

As an example, for the schema {\"properties\": {\"foo\": {\"title\": \"Foo\", \"
description\": \"a list of strings\", \"type\": \"array\", \"items\": {\"type\": \"
string\"}}}, \"required\": [\"foo\"]}\nthe object {\"foo\": [\"bar\", \"baz\"]} is a
well-formatted instance of the schema. The object {\"properties\": {\"foo\": [\"bar\",
\"baz\"]}} is not well-formatted.

Here is the output schema:
```
{\"properties\": {\"thought\": {\"title\": \"Thought\", \"description\": \"Describe how
 you add the threshold assertion to the inspection Python script.\", \"type\": \"string
\"}, \"code\": {\"title\": \"Code\", \"description\": \"Python unit test code.
Implement a for loop that checks the status every second for the duration, and
implement assertion for the summary at the end.\\n- Please add a Add a entry point at
the bottom to allow the test to be run from the command line.\\n- Please add argparse
'--duration' (type=int) so that users can specify the loop duration. Write only the
```
```

```
content of the code, and for dictionary values, enclose them within a pair of single
double quotes (\\\").\", \"type\": \"string\"}}, \"required\": [\"thought\", \"code\"]}
```

Human:
The steady state:
{steady_state_name}: {steady_state_thought}

The steady state was inspected with the following python code of k8s client libraries:
{script (inspection_summary without results)}

The threshold of the steady state: **{steady_state_threshold}; {steady_state_threshold_description}**

Given the above steady state, command, and threshold, please write a Python unit test to check if the steady state satisfies the threshold.
The threshold in the unit test must exactly match the threshold defined above.
Implement it to support variable durations. Use a representative value (e.g., percentage, ratio, etc.) for the threshold. NEVER use any fixed absolute values for the threshold.

----- In the verification loop, the prompts below will be stacked as history -----

AI:
{output}

User:
Your current unittest cause errors when coducted.
The error message is as follows:
{error_message}

Please analyze the reason why the errors occur, then fix the errors.
Always keep the following rules:
- Ensure that the implementation supports variable durations again.
- NEVER repeat the same fixes that have been made in the past.
- Fix only the parts related to the errors without changing the original content.
- **the same format instructions as in the System role**

1-3-b: Agent for writing VaC script (K6 Javascript)

System:
You are a helpful AI assistant for writing unit tests in k6.
Given a steady state, k6 javascript to inspect it, and its threshold, please write a k6 unit test to verify if the steady state satisfies the threshold by adding threshold options.
Always keep the following rules:
- Include as many comments as possible in your code so that humans can easily understand what you did later.
- Add "thresholds" in "options" section to the given k6 javascript.
- {format_instructions}

Human:
The steady state:
{steady_state_name}: {steady_state_thought}

The steady state can be inspected with the following k6 javascript:
{script (inspection_summary without results)}

The threshold of the steady state: **{steady_state_threshold}; {steady_state_threshold_description}**

Given the above steady state, k6 javascript, and threshold, please write a k6 unit test to check if the steady state satisfies the threshold by adding threshold options.
The threshold in the unit test must exactly match the threshold defined above.

----- In the verification loop, the prompts below will be stacked as history -----

AI:
{output}

User:

```
Your current unittest cause errors when coducted.
The error message is as follows:
{error_message}

Please analyze the reason why the errors occur, then fix the errors.
Always keep the following rules:
- Ensure that the implementation supports variable durations again.
- NEVER repeat the same fixes that have been made in the past.
- Fix only the parts related to the errors without changing the original content.
- the same format instructions as in the System role
```

Example text embedded to `steady_state_threshold`

```
The Pod should be in the 'Running' state at least 90\% of the time during the
observation period.
```

Example text embedded to `steady_state_threshold_description`

```
The steady state we are considering is the 'example-pod-running-state', which requires
the Pod to be in a 'Running' state. The current state shows that the Pod was running 5
out of 5 seconds, which is 100\% of the time. To account for some fluctuations and
ensure the threshold is reasonable, we can set a threshold that allows for a small
percentage of time where the Pod might not be in the 'Running' state due to transient
issues. A reasonable threshold could be that the Pod should be in the 'Running' state
at least 90\% of the time during the observation period. This allows for some tolerance
 while still ensuring the Pod is mostly available.
```

 # 1-4: Agent for drafting failure injection

```
System:
You are a helpful AI assistant for Chaos Engineering.
Given k8s manifests for a system, the steady states of the system, and user's
instructions for Chaos Engineering, you will define the most impactful fault injections
 to reveal potential weaknesses of the system, such as insufficient recovery functions,
 resource allocation, redundancy, etc.
Always keep the following rules:
- First, assume a real-world event that may be most impactful in the the system, such
as promotion campaign, cyber attacks, disasters, etc.
- Then, define the most impactful fault injections to reveal potential weaknesses of
the given system while simulating the assumed real-world event.
- Prioritize fault injections that target the system's weak resources related to the
steady states to verify whether those resources can handle the faults and the steady
states can be maintained.
- The injected faults should be selected from the following fault types of {
ce_tool_name}:
  - PodChaos: simulates Pod failures, such as Pod node restart, Pod's persistent
  unavailablility, and certain container failures in a specific Pod. The supported
  subtypes include 'pod-failure', 'pod-kill', 'container-kill'.
  - NetworkChaos: simulates network failures, such as network latency, packet loss,
  packet disorder, and network partitions.
  - DNSChaos: simulates DNS failures, such as the parsing failure of DNS domain name
  and the wrong IP address returned.
  - HTTPChaos: simulates HTTP communication failures, such as HTTP communication
  latency.
  - StressChaos: simulates CPU race or memory race.
  - IOChaos: simulates the I/O failure of an application file, such as I/O delays, read
   and write failures.
  - TimeChaos: simulates the time jump exception.
- The output should be formatted as a JSON instance that conforms to the JSON schema
below.

As an example, for the schema {\"properties\": {\"foo\": {\"title\": \"Foo\", \"
description\": \"a list of strings\", \"type\": \"array\", \"items\": {\"type\": \"
string\"}}}, \"required\": [\"foo\"]}\nthe object {\"foo\": [\"bar\", \"baz\"]} is a
well-formatted instance of the schema. The object {\"properties\": {\"foo\": [\"bar\",
\"baz\"]}} is not well-formatted.

Here is the output schema:
```

```
{\"properties\": {\"event\": {\"title\": \"Event\", \"description\": \"Consider a real-world fault event that may be most impactful of the system, such as promotion campaign, cyber attacks, disasters, etc.\", \"type\": \"string\"}, \"thought\": {\"title\": \"Thought\", \"description\": \"Write down your thought process to define a sequence of fault injections that exploit the system's weaknesses of while simulating the fault event: 1) how the system's weaknesses affect the steady state; 2) how each fault injection exploit the system's weaknesses; 3) how the sequence simulates the phenamena in the fault event (consider carefully the sequence order). Prioritize fault injections that directly attack the weaknessses of the system, such as insufficient recovery functions, resource allocation, redundancy, etc.\", \"type\": \"string\"}, \"faults\": {\"title\": \"Faults\", \"description\": \"Define a sequence of fault injections that exploit the system's vulnerabilities to the fullest according to the above thoughts. In the inner list, a set of simultaneously injected faults are listed, while in the outer list, the sets are listed in the injection order. For example, [[fault_a], [fault_b, fault_c]] indicates that fault_a is injected, then fault_b and fault_c are injected simultaneously.\", \"type\": \"array\", \"items\": {\"type\": \"array\", \"items\": {\"$ref\": \"#/definitions/Fault\"}}}}, \"required\": [\"event\", \"thought\", \"faults\"], \"definitions\": {\"Fault\": {\"title\": \"Fault\", \"type\": \"object\", \"properties\": {\"name\": {\"title\": \"Name\", \"description\": \"Select a fault type from [\\\"PodChaos\\\", \\\"NetworkChaos\\\", \\\"DNSChaos\\\", \\\"HTTPChaos\\\", \\\"StressChaos\\\", \\\"IOChaos\\\", \\\"TimeChaos\\\"]\", \"enum\": [\"PodChaos\", \"NetworkChaos\", \"DNSChaos\", \"HTTPChaos\", \"StressChaos\", \"IOChaos\", \"TimeChaos\"], \"type\": \"string\"}, \"name_id\": {\"title\": \"Name Id\", \"description\": \"An identifier to prevent name conflicts when the same Fault appears. Assign numbers starting from 0 in sequential order to prevent name conflicts.\", \"type\": \"integer\"}, \"scope\": {\"title\": \"Scope\", \"description\": \"Specify only the fault injection scope (i.e., the target resource where the fault is injected) in advance here.\", \"type\": \"object\", \"additionalProperties\": {\"type\": \"string\"}}}, \"required\": [\"name\", \"name_id\", \"scope\"]}}}
```

Human:
Here is the overview of my system:
{user_input2}

Steady states of the network system defined by the manifests are the following:
{steady_states}

Please follow the instructions below regarding Chaos Engineering as necessary:
{ce_instructions}

Now, please define fault injections to reveal the system's vulnerabilities.

Example text embedded to **steady_states**

Steady states of the network system defined by the manifests are the following:
2 steady states are defined.

1st steady states:
- Name: example-pod-running-state
- Description: The first issue to address is the Pod's restart policy set to 'Never' in the 'nginx/pod.yaml' manifest. This is a critical issue because if the Pod fails, it will not restart automatically, leading to potential downtime. A steady state related to this issue would be to ensure that the Pod is running and available. This can be measured by checking the number of running Pods. Since there is only one Pod, the steady state is that the Pod should always be in a 'Running' state.
- Threshold for the steady state: The Pod should be in the 'Running' state at least 90% of the time during the observation period.; The steady state we are considering is the 'example-pod-running-state', which requires the Pod to be in a 'Running' state. The current state shows that the Pod was running 5 out of 5 seconds, which is 100% of the time. To account for some fluctuations and ensure the threshold is reasonable, we can set a threshold that allows for a small percentage of time where the Pod might not be in the 'Running' state due to transient issues. A reasonable threshold could be that the Pod should be in the 'Running' state at least 90% of the time during the observation period. This allows for some tolerance while still ensuring the Pod is mostly available.
- Whether the steady state meets the threshold is determined by the following Python script with K8s API:

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

```
class TestPodRunningState(K8sAPIBase):
    ...

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Test the running state of a Pod.')
    parser.add_argument('--duration', type=int, default=5, help='Duration to check the
    Pod status in seconds.')
    args = parser.parse_args()
    # Create an instance of the test class and run the test
    test = TestPodRunningState(namespace='default', pod_name='example-pod', duration=
    args.duration)
    test.test_pod_running_state()
```

2nd steady states:
- Name: example-service-http-response-state
- Description: The next issue to address is the lack of redundancy due to the single Pod deployment in the 'nginx/pod.yaml' manifest. This is a significant issue because if the Pod fails, there is no automatic recovery or redundancy, which can lead to service unavailability. A steady state related to this issue would be to ensure that the Service is able to route traffic to the Pod. This can be measured by checking the Service's ability to respond to HTTP requests successfully. Since the Service is supposed to expose the Pod on port 80, the steady state is that the Service should respond with a successful HTTP status code (e.g., 200 OK) for a certain percentage of requests.
- Threshold for the steady state: 95% of HTTP requests should return a 200 OK status.; The steady state for the system is defined as the Service's ability to respond with a successful HTTP status code (200 OK) for a certain percentage of requests. The current state shows that 100% of the requests received a 200 OK status, which indicates a perfectly healthy system. However, to account for potential fluctuations and to ensure the threshold is reasonable, we should allow for some tolerance. A common practice is to set a threshold slightly below the current perfect state to accommodate minor, non-critical issues that might occur during normal operations. Therefore, setting the threshold at 95% ensures that the system is considered healthy as long as it maintains a high level of successful responses, while still allowing for some minor issues.
- Whether the steady state meets the threshold is determined by the following K6 Javascript:

```
import http from 'k6/http';\nimport { check } from 'k6';

export const options = {
  vus: 5,
  duration: '5s',
  thresholds: {
    'http_req_failed': ['rate<0.05'],
  },
};

export default function () {
  const res = http.get('http:\/\/example-service.default.svc.cluster.local:80');
  check(res, {
    'is status 200': (r) => r.status === 200,
  });
}
```

1-5: Agent for determining detailed failure parameters

System:
You are a helpful AI assistant for Chaos Engineering.
Given k8s manifests that define a network system, its steady states, and a fault type that may affect the steady states in the system, please detail the parameters of the fault.
Always keep the following rules:
- Pay attention to namespace specification. If the namespace is specified in the manifest, it is deployed with the namespace. If not, it is deployed with the 'default' namespace.
- The parameters follow the format of Chaos Mesh.

Human:
Here is the overview of my system:
{user_input2}

Steady states of my system:
{steady_states}

```
A fault scenario that may occur in my system and may affect the steady states:
{fault_scenario}

Please follow the instructions below regarding Chaos Engineering as necessary:
{ce_instructions}

Now, please detail the parameters of the fault "{refined_fault_type}".
{detailed_param_instructions}


 ----- In the verification loop, the prompts below will be stacked as history -----


AI:
{output}

Human:
Your current fault parameters cause errors when conducted.
The error message is as follows:
{error_message}

Please analyze the reason why the errors occur, then fix the errors.
Always keep the following rules:
- NEVER repeat the same fixes that have been made in the past.
- Fix only the parts related to the errors without changing the original intent.
```

Example text embedded to `fault_scenario`

```
An assumed fault scenario is as follows:
- Event: Cyber Attack Simulation
- Used Chaos Engineering tool: Chaos Mesh
- Faults to simulate the event: [[{'name': 'PodChaos', 'name_id': 0, 'scope': {'pod': '
example-pod'}}], [{'name': 'NetworkChaos', 'name_id': 0, 'scope': {'service': 'example-
service'}}]]
- Description: Given the system's weaknesses, a cyber attack targeting the web server
could be highly impactful. The Pod's restart policy set to 'Never' and the single Pod
deployment without redundancy are critical vulnerabilities. If the Pod fails, it will
not restart, leading to downtime, and the lack of redundancy means there is no backup
to handle traffic. To simulate a cyber attack, we can inject faults that exploit these
weaknesses. First, we will use PodChaos to simulate a Pod failure, which will test the
system's ability to maintain the 'example-pod-running-state'. Since the Pod will not
restart automatically, this will directly impact the steady state. Next, we will use
NetworkChaos to simulate network latency, which will test the system's ability to
maintain the 'example-service-http-response-state'. This sequence simulates a cyber
attack where the Pod is targeted first, followed by network disruptions, revealing the
system's vulnerabilities in handling such events.
```

Example text embedded to `refined_fault_type`

```
NetworkChaos({'service': 'example-service'})
```

`detailed_param_instructions` for `PodChaos` (template embedded dynamically)

```
The output should be formatted as a JSON instance that conforms to the JSON schema
below.

As an example, for the schema {"properties": {"foo": {"title": "Foo", "description": "a
 list of strings", "type": "array", "items": {"type": "string"}}}, "required": ["foo"]}
the object {"foo": ["bar", "baz"]} is a well-formatted instance of the schema. The
object {"properties": {"foo": ["bar", "baz"]}} is not well-formatted.

Here is the output schema:
'''
{"properties": {"action": {"title": "Action", "description": "Specifies the fault type
from 'pod-failure', 'pod-kill', or 'container-kill'. Note that you may select 'pod-
failure' only when the target Pod's container has livenessProbe and readinessProbe
defined.", "example": "pod-kill", "enum": ["pod-failure", "pod-kill", "container-kill
```

```
"], "type": "string"}, "mode": {"title": "Mode", "description": "Specifies the mode of
the experiment. The mode options include 'one' (selecting a random Pod), 'all' (
selecting all eligible Pods), 'fixed' (selecting a specified number of eligible Pods),
'fixed-percent' (selecting a specified percentage of Pods from the eligible Pods), and
'random-max-percent' (selecting the maximum percentage of Pods from the eligible Pods)
", "example": "one", "enum": ["one", "all", "fixed", "fixed-percent", "random-max-
percent"], "type": "string"}, "value": {"title": "Value", "description": "Provides
parameters for the mode configuration, depending on mode.For example, when mode is set
to fixed-percent, value specifies the percentage of Pods.", "example": "1", "type": "
string"}, "selector": {"title": "Selector", "description": "Specifies the target Pod.",
 "example": null, "allOf": [{"$ref": "#/definitions/Selectors"}]}, "containerNames": {"
title": "Containernames", "description": "When you configure action to container-kill,
this configuration is mandatory to specify the target container name for injecting
faults.", "example": ["prometheus"], "type": "array", "items": {"type": "string"}}}, "
required": ["action", "mode", "selector"], "definitions": {"SetBasedRequirements": {"
title": "SetBasedRequirements", "type": "object", "properties": {"key": {"title": "Key
", "description": "Label key", "type": "string"}, "operator": {"title": "Operator", "
description": "Select an operator.", "enum": ["In", "NotIn", "Exists", "DoesNotExist"],
 "type": "string"}, "values": {"title": "Values", "description": "Label values. The
values set must be non-empty in the case of In and NotIn.", "type": "array", "items":
{"type": "string"}}}, "required": ["key", "operator", "values"]}, "Selectors": {"title
": "Selectors", "type": "object", "properties": {"namespaces": {"title": "Namespaces",
"description": "Specifies the namespace of the experiment's target Pod. If this
selector is None, Chaos Mesh will set it to the namespace of the current Chaos
experiment.", "type": "array", "items": {"type": "string"}}, "labelSelectors": {"title
": "Labelselectors", "description": "Specifies the label-key/value pairs that the
experiment's target Pod must have. If multiple labels are specified, the experiment
target must have all the labels specified by this selector.", "type": "object", "
additionalProperties": {"type": "string"}}, "expressionSelectors": {"title": "
Expressionselectors", "description": "Specifies a set of expressions that define the
label's rules to specifiy the experiment's target Pod.", "example": [{"key": "tier", "
operator": "In", "values": ["cache"]}, {"key": "environment", "operator": "NotIn", "
values": ["dev"]}], "type": "array", "items": {"$ref": "#/definitions/
SetBasedRequirements"}}, "annotationSelectors": {"title": "Annotationselectors", "
description": "Specifies the annotation-key/value pairs that the experiment's target
Pod must have. If multiple annotations are specified, the experiment target must have
all annotations specified by this selector.", "type": "object", "additionalProperties":
 {"type": "string"}}, "fieldSelectors": {"title": "Fieldselectors", "description": "
Specifies the field-key/value pairs of the experiment's target Pod. If multiple fields
are specified, the experiment target must have all fields set by this selector.", "
example": {"metadata.name": "my-pod", "metadata.namespace": "dafault"}, "type": "object
", "additionalProperties": {"type": "string"}}, "podPhaseSelectors": {"title": "
Podphaseselectors", "description": "Specifies the phase of the experiment's target Pod.
 If this selector is None, the target Pod's phase is not limited.", "type": "array", "
items": {"enum": ["Pending", "Running", "Succeeded", "Failed", "Unknown"], "type": "
string"}}, "nodeSelectors": {"title": "Nodeselectors", "description": "Specifies the
node-label-key/value pairs to which the experiment's target Pod belongs.", "type": "
object", "additionalProperties": {"type": "string"}}, "nodes": {"title": "Nodes", "
description": "Specifies the node to which the experiment's target Pod belongs. The
target Pod can only belong to one node in the configured node list. If multiple node
labels are specified, the node to which the experiment's target Pod belongs must have
all labels specified by this selector.", "type": "array", "items": {"type": "string"}},
 "pods": {"title": "Pods", "description": "Specifies the namespaces and list of the
experiment's target Pods. If you have specified this selector, Chaos Mesh ignores other
 configured selectors.", "example": {"default": ["pod-0", "pod-2"]}, "type": "object",
"additionalProperties": {"type": "array", "items": {"type": "string"}}}}}}}
```


detailed_param_instructions for **NetworkChaos** (template embedded dynamically)

```
The output should be formatted as a JSON instance that conforms to the JSON schema
below.

As an example, for the schema {"properties": {"foo": {"title": "Foo", "description": "a
 list of strings", "type": "array", "items": {"type": "string"}}}, "required": ["foo"]}
the object {"foo": ["bar", "baz"]} is a well-formatted instance of the schema. The
object {"properties": {"foo": ["bar", "baz"]}} is not well-formatted.

Here is the output schema:
```


```
{"properties": {"action": {"title": "Action", "description": "Indicates the specific
fault type. Available types include: netem, delay (network delay), loss (packet loss),
duplicate (packet duplicating), corrupt (packet corrupt), partition (network partition)
, and bandwidth (network bandwidth limit). After you specify action field, specify
```

action-related fields for other necessary field configuration.", "example": "Partition
", "enum": ["netem", "delay", "loss", "duplicate", "corrupt", "partition", "bandwidth
"], "type": "string"}, "direction": {"title": "Direction", "description": "Indicates
the direction of target packets. Available vaules include from (the packets from target
), to (the packets to target), and both (the packets from or to target). This parameter
makes Chaos only take effect for a specific direction of packets.", "default": "to", "
example": "both", "enum": ["from", "to", "both"], "type": "string"}, "target": {"title
": "Target", "description": "Used in combination with direction, making Chaos only
effective for some packets. 'from' and 'both' direction cannot be used when targets is
empty in netem action.", "allOf": [{"$ref": "#/definitions/Selector"}]}, "mode": {"
title": "Mode", "description": "Specifies the mode of the experiment. The mode options
include one (selecting a random Pod), all (selecting all eligible Pods), fixed (
selecting a specified number of eligible Pods), fixed-percent (selecting a specified
percentage of Pods from the eligible Pods), and random-max-percent (selecting the
maximum percentage of Pods from the eligible Pods)", "example": "one", "enum": ["one",
"all", "fixed", "fixed-percent", "random-max-percent"], "type": "string"}, "value": {"
title": "Value", "description": "Provides parameters for the mode configuration,
depending on mode. For example, when mode is set to fixed-percent, value specifies the
percentage of Pods.", "example": "1", "type": "string"}, "selector": {"title": "
Selector", "description": "Specifies the target Pod.", "allOf": [{"$ref": "#/
definitions/Selectors"}]}, "externalTargets": {"title": "Externaltargets", "description
": "Indicates the network targets except for Kubernetes, which can be IPv4 addresses or
domains. This parameter only works with direction: to.", "example": ["1.1.1.1", "www.
google.com"], "type": "array", "items": {"type": "string"}}, "device": {"title": "
Device", "description": "Specifies the affected network interface", "example": "eth0",
"type": "string"}, "delay": {"title": "Delay", "description": "When setting action to
delay means simulating network delay fault, you also need to configure this parameters
.", "allOf": [{"$ref": "#/definitions/Deplay"}]}, "loss": {"title": "Loss", "
description": "When setting action to loss means simulating packet loss fault, you can
also configure this parameters.", "allOf": [{"$ref": "#/definitions/Loss"}]}, "
duplicated": {"title": "Duplicated", "description": "When setting action to duplicate,
meaning simulating package duplication, you can also set this parameters.", "allOf":
[{"$ref": "#/definitions/Duplicate"}]}, "corrupt": {"title": "Corrupt", "description":
"When setting action to corrupt means simulating package corruption fault, you can also
configure the following parameters.", "allOf": [{"$ref": "#/definitions/Corrupt"}]}, "
rate": {"title": "Rate", "description": "When setting action to rate means simulating
bandwidth rate fault, you also need to configure this parameters. This action is
similar to bandwidth/rate below, however, the key distinction is that this action can
combine with other netem actions listed above. However, if you require more control
over the bandwidth simulation such as limiting the buffer size, select the bandwidth
action.", "allOf": [{"$ref": "#/definitions/Rate"}]}, "bandwidth": {"title": "Bandwidth
", "description": "When setting 'action' to 'bandwidth' means simulating bandwidth
limit fault, you also need to configure this parameters. This action is mutually
exclusive with any netem action defined above. If you need to inject bandwidth rate
along with other network failures such as corruption, use the rate action instead.", "
allOf": [{"$ref": "#/definitions/Bandwidth"}]}}}, "required": ["action", "mode", "
selector"], "definitions": {"SetBasedRequirements": {"title": "SetBasedRequirements", "
type": "object", "properties": {"key": {"title": "Key", "description": "Label key", "
type": "string"}, "operator": {"title": "Operator", "description": "Select an operator
.", "enum": ["In", "NotIn", "Exists", "DoesNotExist"], "type": "string"}, "values": {"
title": "Values", "description": "Label values. The values set must be non-empty in the
case of In and NotIn.", "type": "array", "items": {"type": "string"}}}, "required": ["
key", "operator", "values"]}, "Selectors": {"title": "Selectors", "type": "object", "
properties": {"namespaces": {"title": "Namespaces", "description": "Specifies the
namespace of the experiment's target Pod. If this selector is None, Chaos Mesh will set
it to the namespace of the current Chaos experiment.", "type": "array", "items": {"
type": "string"}}, "labelSelectors": {"title": "Labelselectors", "description": "
Specifies the label-key/value pairs that the experiment's target Pod must have. If
multiple labels are specified, the experiment target must have all the labels specified
by this selector.", "type": "object", "additionalProperties": {"type": "string"}}, "
expressionSelectors": {"title": "Expressionselectors", "description": "Specifies a set
of expressions that define the label's rules to specifiy the experiment's target Pod.",
"example": [{"key": "tier", "operator": "In", "values": ["cache"]}, {"key": "
environment", "operator": "NotIn", "values": ["dev"]}], "type": "array", "items": {"
$ref": "#/definitions/SetBasedRequirements"}}, "annotationSelectors": {"title": "
Annotationselectors", "description": "Specifies the annotation-key/value pairs that the
experiment's target Pod must have. If multiple annotations are specified, the
experiment target must have all annotations specified by this selector.", "type": "
object", "additionalProperties": {"type": "string"}}, "fieldSelectors": {"title": "
Fieldselectors", "description": "Specifies the field-key/value pairs of the experiment'
s target Pod. If multiple fields are specified, the experiment target must have all
fields set by this selector.", "example": {"metadata.name": "my-pod", "metadata.
namespace": "dafault"}, "type": "object", "additionalProperties": {"type": "string"}},
"podPhaseSelectors": {"title": "Podphaseselectors", "description": "Specifies the phase
of the experiment's target Pod. If this selector is None, the target Pod's phase is
not limited.", "type": "array", "items": {"enum": ["Pending", "Running", "Succeeded", "
Failed", "Unknown"], "type": "string"}}, "nodeSelectors": {"title": "Nodeselectors", "
description": "Specifies the node-label-key/value pairs to which the experiment's

```
target Pod belongs.", "type": "object", "additionalProperties": {"type": "string"}}, "
nodes": {"title": "Nodes", "description": "Specifies the node to which the experiment's
 target Pod belongs. The target Pod can only belong to one node in the configured node
list. If multiple node labels are specified, the node to which the experiment's target
Pod belongs must have all labels specified by this selector.", "type": "array", "items
": {"type": "string"}}, "pods": {"title": "Pods", "description": "Specifies the
namespaces and list of the experiment's target Pods. If you have specified this
selector, Chaos Mesh ignores other configured selectors.", "example": {"default": ["pod
-0", "pod-2"]}, "type": "object", "additionalProperties": {"type": "array", "items": {"
type": "string"}}}}}, "Selector": {"title": "Selector", "type": "object", "properties":
 {"mode": {"title": "Mode", "description": "Specifies the mode of the experiment. The
mode options include one (selecting a random Pod), all (selecting all eligible Pods),
fixed (selecting a specified number of eligible Pods), fixed-percent (selecting a
specified percentage of Pods from the eligible Pods), and random-max-percent (selecting
 the maximum percentage of Pods from the eligible Pods)", "example": "one", "enum": ["
one", "all", "fixed", "fixed-percent", "random-max-percent"], "type": "string"}, "
selector": {"title": "Selector", "description": "Specifies the target Pod.", "example":
 null, "allOf": [{"$ref": "#/definitions/Selectors"}]}}, "required": ["mode", "selector
"]}, "Reorder": {"title": "Reorder", "type": "object", "properties": {"reorder": {"
title": "Reorder", "description": "Indicates the probability to reorder", "default":
"0", "example": "0.5", "type": "string"}, "correlation": {"title": "Correlation", "
description": "Indicates the correlation between this time's length of delay time and
the previous time's length of delay time. Range of value: [0, 100]", "default": "0", "
example": "50", "type": "string"}, "gap": {"title": "Gap", "description": "Indicates
the gap before and after packet reordering", "default": 0, "example": 5, "type": "
integer"}}}, "Deplay": {"title": "Deplay", "type": "object", "properties": {"latency":
{"title": "Latency", "description": "Indicates the network latency", "example": "2ms",
"type": "string"}, "correlation": {"title": "Correlation", "description": "Indicates
the correlation between the current latency and the previous one. Range of value: [0,
100]. Specify only the number. NEVER include any units.", "example": "50", "type": "
string"}, "jitter": {"title": "Jitter", "description": "Indicates the range of the
network latency", "example": "1ms", "type": "string"}, "reorder": {"title": "Reorder",
"description": "Indicates the status of network packet reordering", "allOf": [{"$ref":
"#/definitions/Reorder"}]}}}, "Loss": {"title": "Loss", "type": "object", "properties":
 {"loss": {"title": "Loss", "description": "Indicates the probability of packet loss.
Range of value: [0, 100]. Specify only the number. NEVER include any units.", "default
": "0", "example": "50", "type": "string"}, "correlation": {"title": "Correlation", "
description": "Indicates the correlation between the probability of current packet loss
 and the previous time's packet loss. Range of value: [0, 100]. Specify only the number
. NEVER include any units.", "default": "0", "example": "50", "type": "string"}}}, "
Duplicate": {"title": "Duplicate", "type": "object", "properties": {"duplicate": {"
title": "Duplicate", "description": "Indicates the probability of packet duplicating.
Range of value: [0, 100]. Specify only the number. NEVER include any units.", "default
": "0", "example": "50", "type": "string"}, "correlation": {"title": "Correlation", "
description": "Indicates the correlation between the probability of current packet
duplicating and the previous time's packet duplicating. Range of value: [0, 100].
Specify only the number. NEVER include any units.", "default": "0", "example": "50", "
type": "string"}}}, "Corrupt": {"title": "Corrupt", "type": "object", "properties": {"
corrupt": {"title": "Corrupt", "description": "Indicates the probability of packet
corruption. Range of value: [0, 100]. Specify only the number. NEVER include any units
.", "default": "0", "example": "50", "type": "string"}, "correlation": {"title": "
Correlation", "description": "Indicates the correlation between the probability of
current packet corruption and the previous time's packet corruption. Range of value:
[0, 100]. Specify only the number. NEVER include any units.", "default": "0", "example
": "50", "type": "string"}}}, "Rate": {"title": "Rate", "type": "object", "properties":
 {"rate": {"title": "Rate", "description": "Indicates the rate of bandwidth limit.
Allows bit, kbit, mbit, gbit, tbit, bps, kbps, mbps, gbps, tbps unit. bps means bytes
per second", "example": "1mbps", "type": "string"}}}, "Bandwidth": {"title": "Bandwidth
", "type": "object", "properties": {"rate": {"title": "Rate", "description": "Indicates
 the rate of bandwidth limit. Allows bit, kbit, mbit, gbit, tbit, bps, kbps, mbps, gbps
, tbps unit. bps means bytes per second", "example": "1mbps", "type": "string"}, "limit
": {"title": "Limit", "description": "Indicates the number of bytes waiting in queue",
"example": 1, "type": "integer"}, "buffer": {"title": "Buffer", "description": "
Indicates the maximum number of bytes that can be sent instantaneously", "example": 1,
"type": "integer"}, "peakrate": {"title": "Peakrate", "description": "Indicates the
maximum consumption of bucket (usually not set)", "example": 1, "type": "integer"}, "
minburst": {"title": "Minburst", "description": "Indicates the size of peakrate bucket
(usually not set)", "example": 1, "type": "integer"}}}}}
```


detailed_param_instructions for **DNSChaos** (template embedded dynamically)

```
The output should be formatted as a JSON instance that conforms to the JSON schema
below.
```

As an example, for the schema {"properties": {"foo": {"title": "Foo", "description": "a list of strings", "type": "array", "items": {"type": "string"}}}, "required": ["foo"]} the object {"foo": ["bar", "baz"]} is a well-formatted instance of the schema. The object {"properties": {"foo": ["bar", "baz"]}} is not well-formatted.

Here is the output schema:


```
{"properties": {"action": {"title": "Action", "description": "Defines the behavior of DNS fault from 'random' or 'error'. When the value is random, DNS service returns a random IP address; when the value is error, DNS service returns an error.", "example": "random", "enum": ["random", "error"], "type": "string"}, "mode": {"title": "Mode", "description": "Specifies the mode of the experiment. The mode options include 'one' (selecting a random Pod), 'all' (selecting all eligible Pods), 'fixed' (selecting a specified number of eligible Pods), 'fixed-percent' (selecting a specified percentage of Pods from the eligible Pods), and 'random-max-percent' (selecting the maximum percentage of Pods from the eligible Pods)", "example": "one", "enum": ["one", "all", "fixed", "fixed-percent", "random-max-percent"], "type": "string"}, "value": {"title": "Value", "description": "Provides parameters for the mode configuration, depending on mode. For example, when mode is set to fixed-percent, value specifies the percentage of Pods.", "example": "1", "type": "string"}, "patterns": {"title": "Patterns", "description": "Selects a domain template that matches faults. The fault is applyed to these domains. Placeholder ? and wildcard * are supported, but the wildcard in patterns configuration must be at the end of string. For example, chaos-mes*.org. is an invalid configuration. When patterns is not configured, faults are injected for all domains.", "example": "google.com, chaos-mesh.org, github.com", "type": "array", "items": {"type": "string"}}, "selector": {"title": "Selector", "description": "Specifies the target Pod.", "example": null, "allOf": [{"$ref": "#/definitions/Selectors"}]}}, "required": ["selector"], "definitions": {"SetBasedRequirements": {"title": "SetBasedRequirements", "type": "object", "properties": {"key": {"title": "Key", "description": "Label key", "type": "string"}, "operator": {"title": "Operator", "description": "Select an operator.", "enum": ["In", "NotIn", "Exists", "DoesNotExist"], "type": "string"}, "values": {"title": "Values", "description": "Label values. The values set must be non-empty in the case of In and NotIn.", "type": "array", "items": {"type": "string"}}}, "required": ["key", "operator", "values"]}, "Selectors": {"title": "Selectors", "type": "object", "properties": {"namespaces": {"title": "Namespaces", "description": "Specifies the namespace of the experiment's target Pod. If this selector is None, Chaos Mesh will set it to the namespace of the current Chaos experiment.", "type": "array", "items": {"type": "string"}}, "labelSelectors": {"title": "Labelselectors", "description": "Specifies the label-key/value pairs that the experiment's target Pod must have. If multiple labels are specified, the experiment target must have all the labels specified by this selector.", "type": "object", "additionalProperties": {"type": "string"}}, "expressionSelectors": {"title": "Expressionselectors", "description": "Specifies a set of expressions that define the label's rules to specifiy the experiment's target Pod.", "example": [{"key": "tier", "operator": "In", "values": ["cache"]}, {"key": "environment", "operator": "NotIn", "values": ["dev"]}], "type": "array", "items": {"$ref": "#/definitions/SetBasedRequirements"}}, "annotationSelectors": {"title": "Annotationselectors", "description": "Specifies the annotation-key/value pairs that the experiment's target Pod must have. If multiple annotations are specified, the experiment target must have all annotations specified by this selector.", "type": "object", "additionalProperties": {"type": "string"}}, "fieldSelectors": {"title": "Fieldselectors", "description": "Specifies the field-key/value pairs of the experiment's target Pod. If multiple fields are specified, the experiment target must have all fields set by this selector.", "example": {"metadata.name": "my-pod", "metadata.namespace": "dafault"}, "type": "object", "additionalProperties": {"type": "string"}}, "podPhaseSelectors": {"title": "Podphaseselectors", "description": "Specifies the phase of the experiment's target Pod. If this selector is None, the target Pod's phase is not limited.", "type": "array", "items": {"enum": ["Pending", "Running", "Succeeded", "Failed", "Unknown"], "type": "string"}}, "nodeSelectors": {"title": "Nodeselectors", "description": "Specifies the node-label-key/value pairs to which the experiment's target Pod belongs.", "type": "object", "additionalProperties": {"type": "string"}}, "nodes": {"title": "Nodes", "description": "Specifies the node to which the experiment's target Pod belongs. The target Pod can only belong to one node in the configured node list. If multiple node labels are specified, the node to which the experiment's target Pod belongs must have all labels specified by this selector.", "type": "array", "items": {"type": "string"}}, "pods": {"title": "Pods", "description": "Specifies the namespaces and list of the experiment's target Pods. If you have specified this selector, Chaos Mesh ignores other configured selectors.", "example": {"default": ["pod-0", "pod-2"]}, "type": "object", "additionalProperties": {"type": "array", "items": {"type": "string"}}}}}}}
```


detailed_param_instructions for HTTPChaos (template embedded dynamically)

The output should be formatted as a JSON instance that conforms to the JSON schema below.

````
As an example, for the schema {"properties": {"foo": {"title": "Foo", "description": "a
 list of strings", "type": "array", "items": {"type": "string"}}}, "required": ["foo"]}
the object {"foo": ["bar", "baz"]} is a well-formatted instance of the schema. The
object {"properties": {"foo": ["bar", "baz"]}} is not well-formatted.

Here is the output schema:
```
````



````
{"properties": {"mode": {"title": "Mode", "description": "Specifies the mode of the
experiment. The mode options include one (selecting a random Pod), all (selecting all
eligible Pods), fixed (selecting a specified number of eligible Pods), fixed-percent (
selecting a specified percentage of Pods from the eligible Pods), and random-max-
percent (selecting the maximum percentage of Pods from the eligible Pods)", "example":
"one", "enum": ["one", "all", "fixed", "fixed-percent", "random-max-percent"], "type":
"string"}, "value": {"title": "Value", "description": "Provides parameters for the mode
 configuration, depending on mode. For example, when mode is set to fixed-percent,
value specifies the percentage of Pods.", "example": "1", "type": "string"}, "target":
{"title": "Target", "description": "Specifies whether the target of fault injection is
Request or Response. The target-related fields (replace.path, replace.method, replace.
queries, patch.queries) should be configured at the same time.", "example": "Request",
"enum": ["Request", "Response"], "type": "string"}, "port": {"title": "Port", "
description": "The TCP port that the target service listens on.", "example": 80, "type
": "integer"}, "code": {"title": "Code", "description": "Specifies the status code
responded by target. If not specified, the fault takes effect for all status codes by
default. This configuration is effective only when the 'target' is set to 'Response'",
"example": 200, "type": "integer"}, "path": {"title": "Path", "description": "Specify
the URI path of the target request. Supports Matching wildcards. If not specified, the
fault takes effect on all paths by default.", "example": "/api/*", "type": "string"}, "
method": {"title": "Method", "description": "Specify the HTTP method of the target
request method. If not specified, the fault takes effect for all methods by default.",
"example": "GET", "type": "string"}, "request_headers": {"title": "Request Headers", "
description": "Matches request headers to target.", "example": {"Content-Type": "
application/json"}, "type": "object", "additionalProperties": {"type": "string"}}, "
abort": {"title": "Abort", "description": "Abort fault. Indicates whether to inject the
 fault that interrupts the connection.", "default": false, "example": true, "type": "
boolean"}, "delay": {"title": "Delay", "description": "Deplay fault. Specifies the time
 for a latency fault.", "default": "0", "example": "10s", "type": "string"}, "replace":
 {"title": "Replace", "description": "Replace fault. Specifies replaced contents.", "
allOf": [{"$ref": "#/definitions/Replace"}]}, "patch": {"title": "Patch", "description
": "Patch fault. Specifies patch contents.", "allOf": [{"$ref": "#/definitions/Patch
"}]}}, "required": ["mode", "target", "port"], "definitions": {"Replace": {"title": "
Replace", "type": "object", "properties": {"headers": {"title": "Headers", "description
": "Specifies the key pair used to replace the request headers or response headers.", "
example": {"Content-Type": "application/xml"}, "type": "object", "additionalProperties
": {"type": "string"}}, "body": {"title": "Body", "description": "Specifies request
body or response body to replace the fault (Base64 encoded).", "example": "
eyJmb28iOiAiYmFyIn0K", "type": "string"}, "path": {"title": "Path", "description": "
Specifies the URI path used to replace content.", "example": "/api/v2", "type": "string
"}, "method": {"title": "Method", "description": "Specifies the replaced content of the
 HTTP request method.", "example": "DELETE", "type": "string"}, "queries": {"title": "
Queries", "description": "Specifies the replaced key pair of the URI query.", "type": "
array", "items": {"type": "array", "items": {"type": "string"}}}, "code": {"title": "
Code", "description": "Specifies the replaced content of the response status code. This
 configuration is effective only when the 'target' is set to 'Response'.", "example":
404, "type": "integer"}}}, "PatchBody": {"title": "PatchBody", "type": "object", "
properties": {"type": {"title": "Type", "description": "Specifies the type of patch
faults of the request body or response body. Currently, it only supports JSON.", "
example": "JSON", "type": "string"}, "value": {"title": "Value", "description": "
Specifies the fault of the request body or response body with patch faults.", "example
": "{\"foo\": \"bar\"}", "type": "string"}}}, "Patch": {"title": "Patch", "type": "
object", "properties": {"headers": {"title": "Headers", "description": "Specifies the
attached key pair of the request headers or response headers with patch faults.", "
example": [["Set-Cookie", "one cookie"]], "type": "array", "items": {"type": "array", "
items": {"type": "string"}}}, "body": {"title": "Body", "description": "Patch body.", "
allOf": [{"$ref": "#/definitions/PatchBody"}]}, "queries": {"title": "Queries", "
description": "Specifies the attached key pair of the URI query with patch faults.", "
example": [["foo", "bar"]], "type": "array", "items": {"type": "array", "items": {"type
": "string"}}}}}}}
```
````


detailed_param_instructions for **StressChaos** (template embedded dynamically)

```
The output should be formatted as a JSON instance that conforms to the JSON schema
below.
```

```
As an example, for the schema {"properties": {"foo": {"title": "Foo", "description": "a
 list of strings", "type": "array", "items": {"type": "string"}}}, "required": ["foo"]}
the object {"foo": ["bar", "baz"]} is a well-formatted instance of the schema. The
object {"properties": {"foo": ["bar", "baz"]}} is not well-formatted.

Here is the output schema:
```

{"properties": {"mode": {"title": "Mode", "description": "Specifies the mode of the
experiment. The mode options include ’one’ (selecting a random Pod), ’all’ (selecting
all eligible Pods), ’fixed’ (selecting a specified number of eligible Pods), ’fixed-
percent’ (selecting a specified percentage of Pods from the eligible Pods), and ’random
-max-percent’ (selecting the maximum percentage of Pods from the eligible Pods)", "
example": "one", "enum": ["one", "all", "fixed", "fixed-percent", "random-max-percent
"], "type": "string"}, "value": {"title": "Value", "description": "Provides parameters
for the mode configuration, depending on mode.For example, when mode is set to fixed-
percent, value specifies the percentage of Pods.", "example": "1", "type": "string"}, "
stressors": {"title": "Stressors", "description": "Specifies the stress of CPU or
memory", "dafault": null, "allOf": [{"$ref": "#/definitions/Stressors"}]}, "
stressngStressors": {"title": "Stressngstressors", "description": "Specifies the stres-
ng parameter to reach richer stress injection", "example": "--clone 2", "type": "string
"}, "containerNames": {"title": "Containernames", "description": "Specifies the name of
 the container into which the fault is injected.", "example": ["nginx"], "type": "array
", "items": {"type": "string"}}, "selector": {"title": "Selector", "description": "
Specifies the target Pod.", "allOf": [{"$ref": "#/definitions/Selectors"}]}}, "required
": ["mode", "selector"], "definitions": {"MemoryStressor": {"title": "MemoryStressor",
"type": "object", "properties": {"workers": {"title": "Workers", "description": "
Specifies the number of threads that apply memory stress", "example": 1, "type": "
integer"}, "size": {"title": "Size", "description": "Specifies the memory size to be
occupied or a percentage of the total memory size. The final sum of the occupied memory
 size is size.", "example": "256MB", "type": "string"}, "oomScoreAdj": {"title": "
Oomscoreadj", "description": "Specifies the oom_score_adj of the stress process.", "
example": -1000, "type": "integer"}}}, "CPUStressor": {"title": "CPUStressor", "type":
"object", "properties": {"workers": {"title": "Workers", "description": "Specifies the
number of threads that apply CPU stress", "example": 1, "type": "integer"}, "load": {"
title": "Load", "description": "Specifies the percentage of CPU occupied. 0 means that
no additional CPU is added, and 100 refers to full load. The final sum of CPU load is
workers * load.", "example": 50, "type": "integer"}}}, "Stressors": {"title": "
Stressors", "type": "object", "properties": {"memory": {"title": "Memory", "description
": "Specifies the memory stress", "allOf": [{"$ref": "#/definitions/MemoryStressor"}]},
 "cpu": {"title": "Cpu", "description": "Specifies the CPU stress", "allOf": [{"$ref":
"#/definitions/CPUStressor"}]}}}, "SetBasedRequirements": {"title": "
SetBasedRequirements", "type": "object", "properties": {"key": {"title": "Key", "
description": "Label key", "type": "string"}, "operator": {"title": "Operator", "
description": "Select an operator.", "enum": ["In", "NotIn", "Exists", "DoesNotExist"],
 "type": "string"}, "values": {"title": "Values", "description": "Label values. The
values set must be non-empty in the case of In and NotIn.", "type": "array", "items":
{"type": "string"}}}, "required": ["key", "operator", "values"]}, "Selectors": {"title
": "Selectors", "type": "object", "properties": {"namespaces": {"title": "Namespaces",
"description": "Specifies the namespace of the experiment’s target Pod. If this
selector is None, Chaos Mesh will set it to the namespace of the current Chaos
experiment.", "type": "array", "items": {"type": "string"}}, "labelSelectors": {"title
": "Labelselectors", "description": "Specifies the label-key/value pairs that the
experiment’s target Pod must have. If multiple labels are specified, the experiment
target must have all the labels specified by this selector.", "type": "object", "
additionalProperties": {"type": "string"}}, "expressionSelectors": {"title": "
Expressionselectors", "description": "Specifies a set of expressions that define the
label’s rules to specifiy the experiment’s target Pod.", "example": [{"key": "tier", "
operator": "In", "values": ["cache"]}, {"key": "environment", "operator": "NotIn", "
values": ["dev"]}], "type": "array", "items": {"$ref": "#/definitions/
SetBasedRequirements"}}, "annotationSelectors": {"title": "Annotationselectors", "
description": "Specifies the annotation-key/value pairs that the experiment’s target
Pod must have. If multiple annotations are specified, the experiment target must have
all annotations specified by this selector.", "type": "object", "additionalProperties":
 {"type": "string"}}, "fieldSelectors": {"title": "Fieldselectors", "description": "
Specifies the field-key/value pairs of the experiment’s target Pod. If multiple fields
are specified, the experiment target must have all fields set by this selector.", "
example": {"metadata.name": "my-pod", "metadata.namespace": "dafault"}, "type": "object
", "additionalProperties": {"type": "string"}}, "podPhaseSelectors": {"title": "
Podphaseselectors", "description": "Specifies the phase of the experiment’s target Pod.
 If this selector is None, the target Pod’s phase is not limited.", "type": "array", "
items": {"enum": ["Pending", "Running", "Succeeded", "Failed", "Unknown"], "type": "
string"}}, "nodeSelectors": {"title": "Nodeselectors", "description": "Specifies the
node-label-key/value pairs to which the experiment’s target Pod belongs.", "type": "
object", "additionalProperties": {"type": "string"}}, "nodes": {"title": "Nodes", "
description": "Specifies the node to which the experiment’s target Pod belongs. The
target Pod can only belong to one node in the configured node list. If multiple node
labels are specified, the node to which the experiment’s target Pod belongs must have

````
all labels specified by this selector.", "type": "array", "items": {"type": "string"}},
 "pods": {"title": "Pods", "description": "Specifies the namespaces and list of the
experiment's target Pods. If you have specified this selector, Chaos Mesh ignores other
 configured selectors.", "example": {"default": ["pod-0", "pod-2"]}, "type": "object",
"additionalProperties": {"type": "array", "items": {"type": "string"}}}}}}}
```
````


detailed_param_instructions for IOChaos (template embedded dynamically)

````
The output should be formatted as a JSON instance that conforms to the JSON schema
below.

As an example, for the schema {"properties": {"foo": {"title": "Foo", "description": "a
 list of strings", "type": "array", "items": {"type": "string"}}}, "required": ["foo"]}
the object {"foo": ["bar", "baz"]} is a well-formatted instance of the schema. The
object {"properties": {"foo": ["bar", "baz"]}} is not well-formatted.

Here is the output schema:
```
{"properties": {"action": {"title": "Action", "description": "Indicates the specific
type of faults. Only latency, fault, attrOverride, and mistake are supported.", "
example": "latency", "enum": ["latency", "fault", "attrOverride", "mistake"], "type": "
string"}, "mode": {"title": "Mode", "description": "Specifies the mode of the
experiment. The mode options include one (selecting a random Pod), all (selecting all
eligible Pods), fixed (selecting a specified number of eligible Pods), fixed-percent (
selecting a specified percentage of Pods from the eligible Pods), and random-max-
percent (selecting the maximum percentage of Pods from the eligible Pods)", "example":
"one", "enum": ["one", "all", "fixed", "fixed-percent", "random-max-percent"], "type":
"string"}, "selector": {"title": "Selector", "description": "Specifies the target Pod
.", "allOf": [{"$ref": "#/definitions/Selectors"}]}, "value": {"title": "Value", "
description": "Provides parameters for the mode configuration, depending on mode. For
example, when mode is set to fixed-percent, value specifies the percentage of Pods.", "
example": "1", "type": "string"}, "volumePath": {"title": "Volumepath", "description":
"The mount point of volume in the target container. Must be the root directory of the
mount.", "example": "/var/run/etcd", "type": "string"}, "path": {"title": "Path", "
description": "The valid range of fault injections, either a wildcard or a single file.
 If not specified, the fault is valid for all files by default", "example": "/var/run/
etcd/*/", "type": "string"}, "methods": {"title": "Methods", "description": "Type of
the file system call that requires injecting fault. Supported method types: ['lookup',
'forget', 'getattr', 'setattr', 'readlink', 'mknod', 'mkdir', 'unlink', 'rmdir', '
symlink', 'rename', 'link', 'open', 'read', 'write', 'flush', 'release', 'fsync', '
opendir', 'readdir', 'releasedir', 'fsyncdir', 'statfs', 'setxattr', 'getxattr', '
listxattr', 'removexattr', 'access', 'create', 'getlk', 'setlk', 'bmap']. All Types by
default.", "example": ["READ"], "type": "array", "items": {"type": "string"}}, "percent
": {"title": "Percent", "description": "Probability of failure per operation, in %.", "
default": 100, "example": 100, "type": "integer"}, "containerNames": {"title": "
Containernames", "description": "Specifies the name of the container into which the
fault is injected.", "type": "array", "items": {"type": "string"}}, "deplay": {"title":
 "Deplay", "description": "Specify when the 'action' is set to 'latency'. Specific
delay time.", "type": "string"}, "errno": {"title": "Errno", "description": "Specify
when the 'action' is set to 'fault'. Returned error number: 1: Operation not permitted,
 2: No such file or directory, 5: I/O error, 6: No such device or address, 12: Out of
memory, 16: Device or resource busy, 17: File exists, 20: Not a directory, 22: Invalid
argument, 24: Too many open files, 28: No space left on device", "type": "integer"}, "
attr": {"title": "Attr", "description": "Specify when the 'action' is set to '
attrOverride'. Specific property override rules.", "allOf": [{"$ref": "#/definitions/
AttrOverrideSpec"}]}, "mistake": {"title": "Mistake", "description": "Specify when the
'action' is set to 'mistake'. Specific error rules.", "allOf": [{"$ref": "#/definitions
/MistakeSpec"}]}}, "required": ["action", "mode", "volumePath", "attr", "mistake"], "
definitions": {"SetBasedRequirements": {"title": "SetBasedRequirements", "type": "
object", "properties": {"key": {"title": "Key", "description": "Label key", "type": "
string"}, "operator": {"title": "Operator", "description": "Select an operator.", "enum
": ["In", "NotIn", "Exists", "DoesNotExist"], "type": "string"}, "values": {"title": "
Values", "description": "Label values. The values set must be non-empty in the case of
In and NotIn.", "type": "array", "items": {"type": "string"}}}, "required": ["key", "
operator", "values"]}, "Selectors": {"title": "Selectors", "type": "object", "
properties": {"namespaces": {"title": "Namespaces", "description": "Specifies the
namespace of the experiment's target Pod. If this selector is None, Chaos Mesh will set
 it to the namespace of the current Chaos experiment.", "type": "array", "items": {"
type": "string"}}, "labelSelectors": {"title": "Labelselectors", "description": "
Specifies the label-key/value pairs that the experiment's target Pod must have. If
multiple labels are specified, the experiment target must have all the labels specified
 by this selector.", "type": "object", "additionalProperties": {"type": "string"}}, "
expressionSelectors": {"title": "Expressionselectors", "description": "Specifies a set
of expressions that define the label's rules to specifiy the experiment's target Pod.",
````

```
"example": [{"key": "tier", "operator": "In", "values": ["cache"]}, {"key": "
environment", "operator": "NotIn", "values": ["dev"]}], "type": "array", "items": {"
$ref": "#/definitions/SetBasedRequirements"}}, "annotationSelectors": {"title": "
Annotationselectors", "description": "Specifies the annotation-key/value pairs that the
 experiment's target Pod must have. If multiple annotations are specified, the
experiment target must have all annotations specified by this selector.", "type": "
object", "additionalProperties": {"type": "string"}}, "fieldSelectors": {"title": "
Fieldselectors", "description": "Specifies the field-key/value pairs of the experiment'
s target Pod. If multiple fields are specified, the experiment target must have all
fields set by this selector.", "example": {"metadata.name": "my-pod", "metadata.
namespace": "dafault"}, "type": "object", "additionalProperties": {"type": "string"}},
"podPhaseSelectors": {"title": "Podphaseselectors", "description": "Specifies the phase
 of the experiment's target Pod. If this selector is None, the target Pod's phase is
not limited.", "type": "array", "items": {"enum": ["Pending", "Running", "Succeeded", "
Failed", "Unknown"], "type": "string"}}, "nodeSelectors": {"title": "Nodeselectors", "
description": "Specifies the node-label-key/value pairs to which the experiment's
target Pod belongs.", "type": "object", "additionalProperties": {"type": "string"}}, "
nodes": {"title": "Nodes", "description": "Specifies the node to which the experiment's
 target Pod belongs. The target Pod can only belong to one node in the configured node
list. If multiple node labels are specified, the node to which the experiment's target
Pod belongs must have all labels specified by this selector.", "type": "array", "items
": {"type": "string"}}, "pods": {"title": "Pods", "description": "Specifies the
namespaces and list of the experiment's target Pods. If you have specified this
selector, Chaos Mesh ignores other configured selectors.", "example": {"default": ["pod
-0", "pod-2"]}, "type": "object", "additionalProperties": {"type": "array", "items": {"
type": "string"}}}}}, "TimeSpec": {"title": "TimeSpec", "type": "object", "properties":
 {"sec": {"title": "Sec", "description": "Timestamp in seconds. Specify either sec or
nsec.", "type": "integer"}, "nsec": {"title": "Nsec", "description": "Timestamp in
nanoseconds. Specify either sec or nsec.", "type": "integer"}}}, "AttrOverrideSpec": {"
title": "AttrOverrideSpec", "type": "object", "properties": {"ino": {"title": "Ino", "
description": "ino number", "type": "integer"}, "size": {"title": "Size", "description
": "File size", "type": "integer"}, "blocks": {"title": "Blocks", "description": "
Number of blocks that the file uses", "type": "integer"}, "atime": {"title": "Atime", "
description": "Last access time", "allOf": [{"$ref": "#/definitions/TimeSpec"}]}, "
mtime": {"title": "Mtime", "description": "Last modified time", "allOf": [{"$ref": "#/
definitions/TimeSpec"}]}, "ctime": {"title": "Ctime", "description": "Last status
change time", "allOf": [{"$ref": "#/definitions/TimeSpec"}]}, "kind": {"title": "Kind",
 "description": "File type, see fuser::FileType", "type": "string"}, "perm": {"title":
"Perm", "description": "File permissions in decimal", "type": "integer"}, "nlink": {"
title": "Nlink", "description": "Number of hard links", "type": "integer"}, "uid": {"
title": "Uid", "description": "User ID of the owner", "type": "integer"}, "gid": {"
title": "Gid", "description": "Group ID of the owner", "type": "integer"}, "rdev": {"
title": "Rdev", "description": "Device ID", "type": "integer"}}}, "MistakeSpec": {"
title": "MistakeSpec", "type": "object", "properties": {"filling": {"title": "Filling",
 "description": "The wrong data to be filled. Only zero (fill 0) or random (fill random
 bytes) are supported.", "type": "string"}, "maxOccurrences": {"title": "Maxoccurrences
", "description": "Maximum number of errors in each operation.", "example": 1, "type":
"integer"}, "maxLength": {"title": "Maxlength", "description": "Maximum length of each
error (in bytes).", "example": 1, "type": "integer"}}, "required": ["filling", "
maxOccurrences", "maxLength"]}}}
```


detailed_param_instructions for **TimeChaos** (template embedded dynamically)

The output should be formatted as a JSON instance that conforms to the JSON schema below.

As an example, for the schema {"properties": {"foo": {"title": "Foo", "description": "a list of strings", "type": "array", "items": {"type": "string"}}}, "required": ["foo"]} the object {"foo": ["bar", "baz"]} is a well-formatted instance of the schema. The object {"properties": {"foo": ["bar", "baz"]}} is not well-formatted.

Here is the output schema:


```
{"properties": {"timeOffset": {"title": "Timeoffset", "description": "Specifies the
length of time offset.", "example": "-5m", "type": "string"}, "clockIds": {"title": "
Clockids", "description": "Specifies the ID of clock that will be offset. See the
clock_gettime documentation for details.", "default": ["CLOCK_REALTIME"], "example": ["
CLOCK_REALTIME", "CLOCK_MONOTONIC"], "type": "array", "items": {"type": "string"}}, "
mode": {"title": "Mode", "description": "Specifies the mode of the experiment. The mode
 options include 'one' (selecting a random Pod), 'all' (selecting all eligible Pods), '
fixed' (selecting a specified number of eligible Pods), 'fixed-percent' (selecting a
specified percentage of Pods from the eligible Pods), and 'random-max-percent' (
selecting the maximum percentage of Pods from the eligible Pods)", "example": "one", "
enum": ["one", "all", "fixed", "fixed-percent", "random-max-percent"], "type": "string
```

```
"}, "value": {"title": "Value", "description": "Provides parameters for the mode
configuration, depending on mode. For example, when mode is set to fixed-percent, value
 specifies the percentage of Pods.", "example": "1", "type": "string"}, "containerNames
": {"title": "Containernames", "description": "Specifies the name of the container into
 which the fault is injected.", "example": ["nginx"], "type": "array", "items": {"type
": "string"}}, "selector": {"title": "Selector", "description": "Specifies the target
Pod.", "example": null, "allOf": [{"$ref": "#/definitions/Selectors"}]}}, "required":
["timeOffset", "mode", "selector"], "definitions": {"SetBasedRequirements": {"title": "
SetBasedRequirements", "type": "object", "properties": {"key": {"title": "Key", "
description": "Label key", "type": "string"}, "operator": {"title": "Operator", "
description": "Select an operator.", "enum": ["In", "NotIn", "Exists", "DoesNotExist"],
 "type": "string"}, "values": {"title": "Values", "description": "Label values. The
values set must be non-empty in the case of In and NotIn.", "type": "array", "items":
{"type": "string"}}}, "required": ["key", "operator", "values"]}, "Selectors": {"title
": "Selectors", "type": "object", "properties": {"namespaces": {"title": "Namespaces",
"description": "Specifies the namespace of the experiment's target Pod. If this
selector is None, Chaos Mesh will set it to the namespace of the current Chaos
experiment.", "type": "array", "items": {"type": "string"}}, "labelSelectors": {"title
": "Labelselectors", "description": "Specifies the label-key/value pairs that the
experiment's target Pod must have. If multiple labels are specified, the experiment
target must have all the labels specified by this selector.", "type": "object", "
additionalProperties": {"type": "string"}}, "expressionSelectors": {"title": "
Expressionselectors", "description": "Specifies a set of expressions that define the
label's rules to specifiy the experiment's target Pod.", "example": [{"key": "tier", "
operator": "In", "values": ["cache"]}, {"key": "environment", "operator": "NotIn", "
values": ["dev"]}], "type": "array", "items": {"$ref": "#/definitions/
SetBasedRequirements"}}, "annotationSelectors": {"title": "Annotationselectors", "
description": "Specifies the annotation-key/value pairs that the experiment's target
Pod must have. If multiple annotations are specified, the experiment target must have
all annotations specified by this selector.", "type": "object", "additionalProperties":
 {"type": "string"}}, "fieldSelectors": {"title": "Fieldselectors", "description": "
Specifies the field-key/value pairs of the experiment's target Pod. If multiple fields
are specified, the experiment target must have all fields set by this selector.", "
example": {"metadata.name": "my-pod", "metadata.namespace": "dafault"}, "type": "object
", "additionalProperties": {"type": "string"}}, "podPhaseSelectors": {"title": "
Podphaseselectors", "description": "Specifies the phase of the experiment's target Pod.
 If this selector is None, the target Pod's phase is not limited.", "type": "array", "
items": {"enum": ["Pending", "Running", "Succeeded", "Failed", "Unknown"], "type": "
string"}}, "nodeSelectors": {"title": "Nodeselectors", "description": "Specifies the
node-label-key/value pairs to which the experiment's target Pod belongs.", "type": "
object", "additionalProperties": {"type": "string"}}, "nodes": {"title": "Nodes", "
description": "Specifies the node to which the experiment's target Pod belongs. The
target Pod can only belong to one node in the configured node list. If multiple node
labels are specified, the node to which the experiment's target Pod belongs must have
all labels specified by this selector.", "type": "array", "items": {"type": "string"}},
 "pods": {"title": "Pods", "description": "Specifies the namespaces and list of the
experiment's target Pods. If you have specified this selector, Chaos Mesh ignores other
 configured selectors.", "example": {"default": ["pod-0", "pod-2"]}, "type": "object",
"additionalProperties": {"type": "array", "items": {"type": "string"}}}}}}}
'''
```

B.1.3 EXPERIMENT

 # 2-0: Agent for determining time schedule

```
System:
You are a helpful AI assistant for Chaos Engineering.
Given k8s manifests that define a network system, its steady states, and faults that
may affect the steady states in the system, you will design a Chaos Engineering
experiment for them.
First, you will determine the time schedule for the Chaos Engineering experiment.
Always keep the following rules:
- The experiment is divided into three phases: pre-validation, fault-injection, and
post-validation phases: pre-validation to ensure that the system satisfies the steady
states fault injection; fault-injection to observe the system's behavior during fault
injection; post-validation to ensure that the system has returned to its steady states
after fault injection.
- The output should be formatted as a JSON instance that conforms to the JSON schema
below.

As an example, for the schema {\"properties\": {\"foo\": {\"title\": \"Foo\", \"
description\": \"a list of strings\", \"type\": \"array\", \"items\": {\"type\": \"
string\"}}}, \"required\": [\"foo\"]}\nthe object {\"foo\": [\"bar\", \"baz\"]} is a
```

```
well-formatted instance of the schema. The object {\"properties\": {\"foo\": [\"bar\",
\"baz\"]}} is not well-formatted.

Here is the output schema:
```


```
{\"properties\": {\"thought\": {\"title\": \"Thought\", \"describe\": \"Think about the
 total time and the reasonable time allocation for each phase that you are about to
design, and explain your thought process in detail.\", \"type\": \"string\"}, \"
total_time\": {\"title\": \"Total Time\", \"description\": \"Total time of the entire
chaos experiment. total_time should equal to the sum of pre_validation_time,
fault_injection_time, and post_validation_time.\", \"example\": \"10m\", \"type\": \"
string\"}, \"pre_validation_time\": {\"title\": \"Pre Validation Time\", \"description
\": \"Total time of validation before fault injection.\", \"example\": \"2m\", \"type
\": \"string\"}, \"fault_injection_time\": {\"title\": \"Fault Injection Time\", \"
description\": \"Total time of fault injection.\", \"example\": \"6m\", \"type\": \"
string\"}, \"post_validation_time\": {\"title\": \"Post Validation Time\", \"
description\": \"Total time of validation after fault injection.\", \"example\": \"2m
\", \"type\": \"string\"}}, \"required\": [\"thought\", \"total_time\", \"
pre_validation_time\", \"fault_injection_time\", \"post_validation_time\"]}
```


Human:
```
# Here is the overview of my system:
{user_input2}

# Steady states of my system:
{steady_states}

# A fault scenario that may occur in my system and may affect the steady states:
{detailed_fault_scenario}

# Please follow the instructions below regarding Chaos Engineering as necessary:
{ce_instructions}

Now, please plan a Chaos Engineering experiment to check the network system's
resiliency that the steady states are remained during fault injection.
```

Example text embedded to **detailed_fault_scenario**

```
An assumed fault scenario is as follows:
- Event: Cyber Attack Simulation\n- Used Chaos Engineering tool: Chaos Mesh
- Faults to simulate the event: [[Fault(name='PodChaos', name_id=0, params={'action': '
pod-kill', 'mode': 'one', 'selector': {'namespaces': ['default'], 'labelSelectors': {'
app': 'example'}}})], [Fault(name='NetworkChaos', name_id=0, params={'action': 'delay',
 'mode': 'all', 'selector': {'namespaces': ['default'], 'labelSelectors': {'app': '
example'}}, 'direction': 'to', 'delay': {'latency': '100ms', 'jitter': '10ms'}})]]
- Description: Given the system's weaknesses, a cyber attack targeting the web server
could be highly impactful. The Pod's restart policy set to 'Never' and the single Pod
deployment without redundancy are critical vulnerabilities. If the Pod fails, it will
not restart, leading to downtime, and the lack of redundancy means there is no backup
to handle traffic. To simulate a cyber attack, we can inject faults that exploit these
weaknesses. First, we will use PodChaos to simulate a Pod failure, which will test the
system's ability to maintain the 'example-pod-running-state'. Since the Pod will not
restart automatically, this will directly impact the steady state. Next, we will use
NetworkChaos to simulate network latency, which will test the system's ability to
maintain the 'example-service-http-response-state'. This sequence simulates a cyber
attack where the Pod is targeted first, followed by network disruptions, revealing the
system's vulnerabilities in handling such events.
```

2-1: Agent for scheduling each experiment phase (pre-validation, failure-injection, and post-validation phases)

System:
```
You are a helpful AI assistant for Chaos Engineering.
Given k8s manifests that define a network system, its steady states, and faults that
may affect the steady states in the system, you will design a Chaos Engineering
experiment for them.
The experiment is divided into three phases: pre-validation, fault-injection, and post-
validation phases: pre-validation to ensure that the system satisfies the steady states
 fault injection; fault-injection to observe the system's behavior during fault
```

```
injection; post-validation to ensure that the system has returned to its steady states
after fault injection.
Here, you will detail the {phase_name}.
Always keep the following rules:
- {phase_planning_instructions}

Human:
# Here is the overview of my system:
{user_input}

# Steady states of my system:
{steady_states}

# A fault scenario that may occur in my system and may affect the steady states:
{detailed_fault_scenario}

# Please follow the instructions below regarding Chaos Engineering as necessary:
{ce_instructions}

Now, please detail the {phase_name}. Note that the phase's total time is {
phase_total_time}.
```

Example text embedded to phase_name

```
pre-validation phase
```

Example text embedded to phase_total_time

```
10s
```

phase_planning_instructions for the pre-validation and post-validation phases

```
The output should be formatted as a JSON instance that conforms to the JSON schema
below.

As an example, for the schema {\"properties\": {\"foo\": {\"title\": \"Foo\", \"
description\": \"a list of strings\", \"type\": \"array\", \"items\": {\"type\": \"
string\"}}}, \"required\": [\"foo\"]}\nthe object {\"foo\": [\"bar\", \"baz\"]} is a
well-formatted instance of the schema. The object {\"properties\": {\"foo\": [\"bar\",
\"baz\"]}} is not well-formatted.

Here is the output schema:
```
```
{\"properties\": {\"thought\": {\"title\": \"Thought\", \"description\": \"Describe in
detail the timeline for when each fault injection and each unit test (for verifying
steady-state) will be executed. For example, explain which fault injections/unit tests
will be executed simultaneously, and whether certain fault injections/unit tests will
be executed at staggered timings. Additionally, explain the thought process that led
you to this approach.\", \"type\": \"string\"}, \"unit_tests\": {\"title\": \"Unit
Tests\", \"description\": \"The list of unit test schedule.\", \"type\": \"array\", \"
items\": {\"$ref\": \"#/definitions/UnitTest\"}}}, \"required\": [\"thought\", \"
unit_tests\"], \"definitions\": {\"UnitTest\": {\"title\": \"UnitTest\", \"type\": \"
object\", \"properties\": {\"name\": {\"title\": \"Name\", \"description\": \"Steady
state name to be verified by a unit test.\", \"type\": \"string\"}, \"grace_period\":
{\"title\": \"Grace Period\", \"description\": \"Time elapsed from the start of the
current phase to the beginning of the unit test.\", \"example\": \"0s\", \"type\": \"
string\"}, \"duration\": {\"title\": \"Duration\", \"description\": \"Duration of the
unit test. (grace_period + duration) should not exceed the current phase's total time
.\", \"example\": \"2m\", \"type\": \"string\"}}, \"required\": [\"name\", \"
grace_period\", \"duration\"]}}}
```

phase_planning_instructions for the fault-injection phases

```
The output should be formatted as a JSON instance that conforms to the JSON schema below.

As an example, for the schema {\"properties\": {\"foo\": {\"title\": \"Foo\", \"description\": \"a list of strings\", \"type\": \"array\", \"items\": {\"type\": \"string\"}}}, \"required\": [\"foo\"]}\nthe object {\"foo\": [\"bar\", \"baz\"]} is a well-formatted instance of the schema. The object {\"properties\": {\"foo\": [\"bar\", \"baz\"]}} is not well-formatted.

Here is the output schema:
```

```
{\"properties\": {\"thought\": {\"title\": \"Thought\", \"description\": \"Describe in detail the timeline for when each fault injection and each unit test (for verifying steady-state) will be executed. For example, explain which fault injections/unit tests will be executed simultaneously, and whether certain fault injections/unit tests will be executed at staggered timings. Additionally, explain the thought process that led you to this approach.\", \"type\": \"string\"}, \"fault_injection\": {\"title\": \"Fault Injection\", \"description\": \"The list of fault injection schedules.\", \"type\": \"array\", \"items\": {\"$ref\": \"#/definitions/FaultInjection\"}}, \"unit_tests\": {\"title\": \"Unit Tests\", \"description\": \"The list of unit test schedule.\", \"type\": \"array\", \"items\": {\"$ref\": \"#/definitions/UnitTest\"}}}, \"required\": [\"thought\", \"fault_injection\", \"unit_tests\"], \"definitions\": {\"FaultInjection\": {\"title\": \"FaultInjection\", \"type\": \"object\", \"properties\": {\"name\": {\"title\": \"Name\", \"description\": \"Select a fault type from [\\\"PodChaos\\\", \\\"NetworkChaos\\\", \\\"DNSChaos\\\", \\\"HTTPChaos\\\", \\\"StressChaos\\\", \\\"IOChaos\\\", \\\"TimeChaos\\\"]\", \"enum\": [\"PodChaos\", \"NetworkChaos\", \"DNSChaos\", \"HTTPChaos\", \"StressChaos\", \"IOChaos\", \"TimeChaos\"], \"type\": \"string\"}, \"name_id\": {\"title\": \"Name Id\", \"description\": \"An identifier to prevent name conflicts when the same Fault appears. Assign numbers starting from 0 in sequential order to prevent name conflicts.\", \"type\": \"integer\"}, \"grace_period\": {\"title\": \"Grace Period\", \"description\": \"Time elapsed from the start of the current phase to the beginning of the fault injection.\", \"example\": \"0s\", \"type\": \"string\"}, \"duration\": {\"title\": \"Duration\", \"description\": \"Duration of the unit test. (grace_period + duration) should not exceed the current phase’s total time.\", \"example\": \"2m\", \"type\": \"string\"}}, \"required\": [\"name\", \"name_id\", \"grace_period\", \"duration\"]}, \"UnitTest\": {\"title\": \"UnitTest\", \"type\": \"object\", \"properties\": {\"name\": {\"title\": \"Name\", \"description\": \"Steady state name to be verified by a unit test.\", \"type\": \"string\"}, \"grace_period\": {\"title\": \"Grace Period\", \"description\": \"Time elapsed from the start of the current phase to the beginning of the unit test.\", \"example\": \"0s\", \"type\": \"string\"}, \"duration\": {\"title\": \"Duration\", \"description\": \"Duration of the unit test. (grace_period + duration) should not exceed the current phase’s total time.\", \"example\": \"2m\", \"type\": \"string\"}}, \"required\": [\"name\", \"grace_period\", \"duration\"]}}}
```


2-2: Agent for summarizing the planned experiment

```
System:
You are a helpful AI assistant for Chaos Engineering.
Given a Chaos-Engineering-experiment plan, you will summarize it in detail according to the following rules:
- In each phase, describe in detail the timeline for when each fault injection/unit test (for verifying steady-state) will be executed. For example, summarize which fault injections/unit tests will be executed simultaneously, and whether certain fault injections/unit tests will be executed at staggered timings.
- Be sure to specify both each fault injection/unit test and their corresponding workflow names.
- When explaining the timeline, provide a detailed description using specific values for duration, grace period, etc. Rephrase the specific values in a way that everyone can easily understand.
- The meanings of each value are as follows:
  - Grace Period: Time elapsed from the start of the current phase to the beginning of the fault injection/unit test.
  - Duration: Duration of the fault injection/unit test. (grace_period + duration) should not exceed the corresponding phase’s total time.
- Never output bullet points.
- The output should be formatted as a JSON instance that conforms to the JSON schema below.
As an example, for the schema {\"properties\": {\"foo\": {\"title\": \"Foo\", \"description\": \"a list of strings\", \"type\": \"array\", \"items\": {\"type\": \"string\"}}}, \"required\": [\"foo\"]}\nthe object {\"foo\": [\"bar\", \"baz\"]} is a
```

```
well-formatted instance of the schema. The object {\"properties\": {\"foo\": [\"bar\",
\"baz\"]}} is not well-formatted.

Here is the output schema:
```


```
{\"properties\": {\"summary\": {\"title\": \"Summary\", \"description\": \"The summary
of the given Chaos-Engineering-experiment plan.\", \"type\": \"string\"}}, \"required
\": [\"summary\"]}
```


```
Human:
# Here is my Chaos-Engineering-experiment plan:
## Time Schedule
{time_schedule_overview}

## Pre-validation Phase
{pre_validation_overview}

## Fault-injection Phase
{fault_injection_overview}

## Post-validation phase
{post_validation_overview}

Please summarize the above plan.
```

Example text embedded to time_schedule_overview

```
Given the constraints of the experiment needing to be completed within 1 minute, we
need to carefully allocate time to each phase to ensure that we can effectively
validate the system's steady states before and after the fault injection, as well as
observe the system's behavior during the fault injection. The pre-validation phase is
crucial to establish a baseline that the system is in its expected steady state before
any faults are introduced. The fault injection phase is where we introduce the chaos to
 observe how the system behaves under stress. Finally, the post-validation phase is
necessary to ensure that the system returns to its steady state after the faults are
removed. Given the short total time of 1 minute, a reasonable allocation could be 10
seconds for pre-validation, 40 seconds for fault injection, and 10 seconds for post-
validation. This allocation allows us to have a brief but sufficient observation period
 for each phase, ensuring that we can gather meaningful insights from the experiment.
```

Example text embedded to pre_validation_overview

```
In the pre-validation phase, we need to ensure that the system is in its expected
steady states before we proceed with fault injection. Given the constraints, we have 10
 seconds to perform these checks. We have two steady states to verify: the 'example-pod
-running-state' and the 'example-service-http-response-state'.

The 'example-pod-running-state' requires us to check that the Pod is in the 'Running'
state at least 90% of the time. We will use the provided Python script to verify this.
Since the script checks the Pod status every second, we can run it for 5 seconds to
gather sufficient data for validation.

The 'example-service-http-response-state' requires us to ensure that 95% of HTTP
requests return a 200 OK status. We will use the K6 script to simulate HTTP requests to
 the service. The script is configured to run for 5 seconds with 5 virtual users, which
 should provide enough data to validate this steady state.

Both unit tests will be executed simultaneously to maximize the use of the 10-second
window. This approach ensures that we efficiently validate both steady states within
the given time constraint, allowing us to proceed confidently to the fault injection
phase.
- Verified Steady State #0: '''example-pod-running-state'''
  - Workflow Name: '''pre-unittest-example-pod-running-state'''
  - Grace Period: '''0s'''
  - Duration: '''5s'''
- Verified Steady State #1: '''example-service-http-response-state'''
  - Workflow Name: '''pre-unittest-example-service-http-response-state'''
  - Grace Period: '''0s'''
  - Duration: '''5s'''
```

Example text embedded to **fault_injection_overview**

```
In this fault-injection phase, we aim to simulate a cyber attack by injecting two types
 of faults: PodChaos and NetworkChaos. The goal is to observe how these faults impact
the system's steady states. Given the 40-second time constraint, we will stagger the
fault injections to maximize the observation of their effects.

First, we will inject the PodChaos fault to simulate a Pod failure. This will directly
test the 'example-pod-running-state' steady state, as the Pod's restart policy is set
to 'Never', meaning it will not automatically recover. We will start this fault
injection immediately at the beginning of the phase and let it run for 20 seconds.

Simultaneously, we will run the unit test for the 'example-pod-running-state' to verify
 the impact of the PodChaos fault on the Pod's availability.\n\nAfter 20 seconds, we
will inject the NetworkChaos fault to simulate network latency. This will test the '
example-service-http-response-state' steady state by introducing delays in HTTP
responses. We will let this fault run for the remaining 20 seconds of the phase.

During the NetworkChaos fault injection, we will also run the unit test for the '
example-service-http-response-state' to verify the impact of network latency on the
Service's ability to respond to HTTP requests.

This staggered approach allows us to isolate the effects of each fault on the
respective steady states, providing clear insights into the system's vulnerabilities
and resilience.
- Verified Steady State #0: ```example-pod-running-state```
  - Workflow Name: ```fault-unittest-example-pod-running-state```
  - Grace Period: ```0s```
  - Duration: ```20s```
- Verified Steady State #1: ```example-service-http-response-state```
  - Workflow Name: ```fault-unittest-example-service-http-response-state```
  - Grace Period: ```20s```
  - Duration: ```20s```

- Injected Faults #0: ```PodChaos```
  - Workflow Name: ```fault-podchaos```
  - Grace Period: ```0s```
  - Duration: ```20s```

- Injected Faults #1: ```NetworkChaos```
  - Workflow Name: ```fault-networkchaos```
  - Grace Period: ```20s```
  - Duration: ```20s```
```

Example text embedded to **post_validation_overview**

```
In the post-validation phase, we need to ensure that the system has returned to its
steady states after the fault injection. Given the 10-second total time for this phase,
 we will conduct unit tests to verify each steady state. The tests will be executed
sequentially due to the short duration, ensuring that each steady state is verified
independently. The first test will check the 'example-pod-running-state', ensuring the
Pod is running. The second test will verify the 'example-service-http-response-state',
ensuring the Service responds with a 200 OK status. Each test will have a brief grace
period to allow the system to stabilize after the fault injection, followed by a short
duration to perform the checks. This approach ensures that both steady states are
validated within the available time, confirming the system's recovery.
- Verified Steady State #0: ```example-pod-running-state```
  - Workflow Name: ```post-unittest-example-pod-running-state```
  - Grace Period: ```1s```
  - Duration: ```4s```

- Verified Steady State #1: ```example-service-http-response-state```
  - Workflow Name: ```post-unittest-example-service-http-response-state```
  - Grace Period: ```5s```
  - Duration: ```4s```
```

2-3: Agent for adjusting a failure scope

System:

```
You are a helpful AI assistant for Chaos Engineering.
```

````
Given a previous K8s manifests, a Chaos-Engineering-experiment plan for it, and the
current K8s manifests, you will determine whether we need to adujst the scope of fault
injections for the current K8s manifests.
Always keep the following rules:
- Cosider how you must change or keep the scope (i.e., target) of the fault injecttion
comparing the previous K8s manifests and the current K8s manifests.
- You only make minor adjustments related to resource changes, metadata change, etc, so
 NEVER make any scope changes that alter the original goal of the chaos experiment.
- The output should be formatted as a JSON instance that conforms to the JSON schema
below.

As an example, for the schema {\"properties\": {\"foo\": {\"title\": \"Foo\", \"
description\": \"a list of strings\", \"type\": \"array\", \"items\": {\"type\": \"
string\"}}}, \"required\": [\"foo\"]}\nthe object {\"foo\": [\"bar\", \"baz\"]} is a
well-formatted instance of the schema. The object {\"properties\": {\"foo\": [\"bar\",
\"baz\"]}} is not well-formatted.

Here is the output schema:
```
{\"properties\": {\"thought\": {\"title\": \"Thought\", \"description\": \"Describe why
 you need to change/keep the scope of the fault injection for the current K8s manifests
.\", \"type\": \"string\"}, \"selector\": {\"title\": \"Selector\", \"description\": \"
Adjust the scope (target) of the fault injection comparing the differeneces between the
 current and previous manifests. \", \"allOf\": [{\"$ref\": \"#/definitions/Selectors
\"}]}}, \"required\": [\"thought\", \"selector\"], \"definitions\": {\"
SetBasedRequirements\": {\"title\": \"SetBasedRequirements\", \"type\": \"object\", \"
properties\": {\"key\": {\"title\": \"Key\", \"description\": \"Label key\", \"type\":
\"string\"}, \"operator\": {\"title\": \"Operator\", \"description\": \"Select an
operator.\", \"enum\": [\"In\", \"NotIn\", \"Exists\", \"DoesNotExist\"], \"type\": \"
string\"}, \"values\": {\"title\": \"Values\", \"description\": \"Label values. The
values set must be non-empty in the case of In and NotIn.\", \"type\": \"array\", \"
items\": {\"type\": \"string\"}}}, \"required\": [\"key\", \"operator\", \"values\"]},
\"Selectors\": {\"title\": \"Selectors\", \"type\": \"object\", \"properties\": {\"
namespaces\": {\"title\": \"Namespaces\", \"description\": \"Specifies the namespace of
 the experiment's target Pod. If this selector is None, Chaos Mesh will set it to the
namespace of the current Chaos experiment.\", \"type\": \"array\", \"items\": {\"type
\": \"string\"}}, \"labelSelectors\": {\"title\": \"Labelselectors\", \"description\":
\"Specifies the label-key/value pairs that the experiment's target Pod must have. If
multiple labels are specified, the experiment target must have all the labels specified
 by this selector.\", \"type\": \"object\", \"additionalProperties\": {\"type\": \"
string\"}}, \"expressionSelectors\": {\"title\": \"Expressionselectors\", \"description
\": \"Specifies a set of expressions that define the label's rules to specifiy the
experiment's target Pod.\", \"example\": [{\"key\": \"tier\", \"operator\": \"In\", \"
values\": [\"cache\"]}, {\"key\": \"environment\", \"operator\": \"NotIn\", \"values\":
 [\"dev\"]}], \"type\": \"array\", \"items\": {\"$ref\": \"#/definitions/
SetBasedRequirements\"}}, \"annotationSelectors\": {\"title\": \"Annotationselectors\",
 \"description\": \"Specifies the annotation-key/value pairs that the experiment's
target Pod must have. If multiple annotations are specified, the experiment target must
 have all annotations specified by this selector.\", \"type\": \"object\", \"
additionalProperties\": {\"type\": \"string\"}}, \"fieldSelectors\": {\"title\": \"
Fieldselectors\", \"description\": \"Specifies the field-key/value pairs of the
experiment's target Pod. If multiple fields are specified, the experiment target must
have all fields set by this selector.\", \"example\": {\"metadata.name\": \"my-pod\",
\"metadata.namespace\": \"dafault\"}, \"type\": \"object\", \"additionalProperties\":
{\"type\": \"string\"}}, \"podPhaseSelectors\": {\"title\": \"Podphaseselectors\", \"
description\": \"Specifies the phase of the experiment's target Pod. If this selector
is None, the target Pod's phase is not limited.\", \"type\": \"array\", \"items\": {\"
enum\": [\"Pending\", \"Running\", \"Succeeded\", \"Failed\", \"Unknown\"], \"type\":
\"string\"}}, \"nodeSelectors\": {\"title\": \"Nodeselectors\", \"description\": \"
Specifies the node-label-key/value pairs to which the experiment's target Pod belongs
.\", \"type\": \"object\", \"additionalProperties\": {\"type\": \"string\"}}, \"nodes
\": {\"title\": \"Nodes\", \"description\": \"Specifies the node to which the
experiment's target Pod belongs. The target Pod can only belong to one node in the
configured node list. If multiple node labels are specified, the node to which the
experiment's target Pod belongs must have all labels specified by this selector.\", \"
type\": \"array\", \"items\": {\"type\": \"string\"}}, \"pods\": {\"title\": \"Pods\",
\"description\": \"Specifies the namespaces and list of the experiment's target Pods.
If you have specified this selector, Chaos Mesh ignores other configured selectors.\",
\"example\": {\"default\": [\"pod-0\", \"pod-2\"]}, \"type\": \"object\", \"
additionalProperties\": {\"type\": \"array\", \"items\": {\"type\": \"string\"}}}}}}}\n
```

Human:
# Here is the previous K8s manifests of my system:
{prev_k8s_yamls}

# Here is a planned Chaos Engineering:
{experiment_plan_summary}
````

```
# Here is the current K8s menifests of my system:
{curr_k8s_yamls}

# Here is the scope of a fault injection for the previous manifests.
{curr_fault_injection}

Now, please adjust the scope of the fault injection for the current manifests. Note
that you here focus on the 'selector' parameter (i.e., scope).
{format_instructions}
```

2-4: Agent for adjusting a VaC script

```
System:
You are a helpful AI assistant for Chaos Engineering.
Given the previous K8s manifests, a previous unit test to verify whether the steady
state satisfies the threshold, and the reconfigured K8s manifests, you will determine
whether the unit test requires adjustment to account for the changes in the
reconfigured manifests, and adjust it as necessary.
Always keep the following rules:
- First, consider which K8s manifest resource is the target of the unit test. If there
are changes to that manifest, update the unit test as necessary. If there are no
changes, the unit test should not require modification.
- You may only make minor adjustments to K8s API, HTTP, or DNS request to account for
changes in resource types, parameter seetings, metadata, etc.
- The reconfiguration was made so that the system satisfy the threshold value in the
previous unit test, so the threshold value or other parameters must remain unchanged in
 the new unit test. For example, suppose the number of replicas was reconfigured from 1
 to 3 in order to maintain a steady state with more than 1 active pod at all times. In
such cases, changing the threshold value from 1 to 3 would alter the intent of this
steady state, so the threshold value must remain unchanged (i.e., more than 1 active
pod)."
- If redundancy has been newly added, the unit test should verify whether the steady
state is maintained by the entire redundancy.
- If the unit test's content needs no changes and only function or variable names need
to be changed, leave them as they are to save output costs.
- The output should be formatted as a JSON instance that conforms to the JSON schema
below.

As an example, for the schema {\"properties\": {\"foo\": {\"title\": \"Foo\", \"
description\": \"a list of strings\", \"type\": \"array\", \"items\": {\"type\": \"
string\"}}}, \"required\": [\"foo\"]}\nthe object {\"foo\": [\"bar\", \"baz\"]} is a
well-formatted instance of the schema. The object {\"properties\": {\"foo\": [\"bar\",
\"baz\"]}} is not well-formatted.

Here is the output schema:
'''
{\"properties\": {\"thought\": {\"title\": \"Thought\", \"description\": \"Describe
your thought process for determining whether the unit test requires adjustment to
account for the changes in the reconfigured manifests: First, consider which K8s
manifest resource is the target of the unit test. If there are changes to that manifest
, update the unit test as necessary. If there are no changes, the unit test should not
require modification. If the unit test needs updating, describe also how you modify the
 inspection method according to the differences between the previous and reconfigured
manifests. If the modification is not required, describe the reason.\", \"type\": \"
string\"}, \"code\": {\"title\": \"Code\", \"description\": \"If the unit test needs
updating, write a new unit test code with the inspection method modified. Write only
the content of the code without enclosing it in a code block. If not, this field is not
 required.\", \"type\": \"string\"}}, \"required\": [\"thought\"]}
'''

Human:
# Here is the previous K8s manifests of my system:
{prev_k8s_yamls}

# Here is the reconfigured K8s manifests of my system:
{curr_k8s_yamls}

# Here is the unit test for the previous manifests.
{prev_unittest}

Now, please determine whether the unit test requires adjustment to account for the
changes in the reconfigured manifests, and adjust it as necessary.
```

```
----- In the verification loop, the prompts below will be stacked as history -----

AI:
{output}

Human:
Your current unit test causes errors when conducted.
The error message is as follows:
{error_message}

This unit test should be succeeded.
Please analyze the reason why the errors occur, then fix the errors.
Always keep the following rules:
- NEVER repeat the same fixes that have been made in the past.
- Fix only the parts related to the errors without changing the original intent.
- {format_instructions}
```

Example text embedded to **`experiment_plan_summary`**

```
The Chaos Engineering experiment is structured into three phases: pre-validation, fault
 injection, and post-validation, all to be completed within a total of 1 minute.

In the pre-validation phase, which lasts for 10 seconds, two unit tests are executed
simultaneously to verify the system's steady states before any faults are introduced.
The 'example-pod-running-state' is checked using a Python script to ensure the Pod is
in the 'Running' state at least 90% of the time. This test runs for 5 seconds.
Concurrently, the 'example-service-http-response-state' is verified using a K6 script
to simulate HTTP requests, ensuring 95% of requests return a 200 OK status. This test
also runs for 5 seconds. Both tests start immediately at the beginning of the phase.

The fault injection phase spans 40 seconds and involves two staggered fault injections.
 Initially, the PodChaos fault is injected to simulate a Pod failure, running for the
first 20 seconds. Simultaneously, the 'example-pod-running-state' unit test is
conducted to observe the impact of this fault. After 20 seconds, the NetworkChaos fault
 is introduced to simulate network latency, running for the remaining 20 seconds.
During this period, the 'example-service-http-response-state' unit test is executed to
assess the effect of network delays. This staggered approach allows for isolated
observation of each fault's impact on the system.

Finally, the post-validation phase, lasting 10 seconds, ensures the system returns to
its steady states after fault removal. The tests are conducted sequentially. The '
example-pod-running-state' is verified first, with a 1-second grace period followed by
a 4-second test duration. Subsequently, the 'example-service-http-response-state' is
checked, starting after a 5-second grace period and running for 4 seconds. This
sequence confirms the system's recovery to its expected steady states.
```

B.1.4 Analysis

3-0: Agent for analyzing an experiment results

```
System:
You are a helpful AI assistant for Chaos Engineering.
Given K8s manifests for a network system, its hypothesis, the overview of a Chaos-
Engineeering experiment, and the experimental results, you will analyze the
experimental results.
Always keep the following rules:
- Analyze step by step why the test(s) failed, based on the system configurations (
manifests) and the flow of the experiment.
- Specify the cause while mentioning the corresponding system configurations and the
corresponding phenomena in the Chaos-Engineering experiment.
- The analysis report here will be used for reconfiguring the system later to avoid the
 failures and improve resiliency. Therefore, make carefully the report rich in insights
 so that it will be helpful at that time.
- When providing insights and reconfiguration recommendations, limit them to areas
related to the failed test.
- The output should be formatted as a JSON instance that conforms to the JSON schema
below.

As an example, for the schema {\"properties\": {\"foo\": {\"title\": \"Foo\", \"
description\": \"a list of strings\", \"type\": \"array\", \"items\": {\"type\": \"
```

```
string\"}}}, \"required\": [\"foo\"]}\nthe object {\"foo\": [\"bar\", \"baz\"]} is a
well-formatted instance of the schema. The object {\"properties\": {\"foo\": [\"bar\",
\"baz\"]}} is not well-formatted.

Here is the output schema:
```
```
{\"properties\": {\"report\": {\"title\": \"Report\", \"description\": \"Analysis of
the experiment result.\", \"type\": \"string\"}}, \"required\": [\"report\"]}
```
```

```
Human:
# Here is the overview of my system:
{user_input2}

# Here is the hypothesis for my system:
The hypothesis is "The steady states of the sytem are maintained even when the fault
scenario occurs (i.e., when the faults are injected)".
The steady states here are as follows:
{steady_states}

The fault scenario here is as follows:
{detailed_fault_scenario}

# Here is the overview of my Chaos-Engineering experiment to verify the hypothesis:
{experiment_plan_summary}
```

For the first analysis, the following prompt is added

```
# The experiment's results are as follows:
{experiment_result}

Now, please analyze the results and provide an analysis report rich in insights.
```

For the second and subsequent analyses, the following prompt is added

---

```
# The update history for the above K8s manifests is the following:
{reconfig_history}

# The experiment's results in the latest K8s manifests are as follows:
{experiment_result}

Now, please analyze the results and provide an analysis report rich in insights.
```

Example text embedded to **experiment_result**

```
Passed unittests:
- pre-unittest-example-pod-running-state
- pre-unittest-example-service-http-response-state

Failed unittests:
- fault-unittest-example-pod-running-state
```
```log
Exception when calling CoreV1Api->read_namespaced_pod: (404)
Reason: Not Found\nHTTP response headers: HTTPHeaderDict({'Audit-Id': '8a1e6c00-ebd9-43
ee-9522-6399ce015252', 'Cache-Control': 'no-cache, private', 'Content-Type': '
application/json', 'X-Kubernetes-Pf-Flowschema-Uid': 'c4624bd9-7fc7-42c6-bcb8-4235110
a860d', 'X-Kubernetes-Pf-Prioritylevel-Uid': '4706085f-6263-43ae-93f5-b4a61de8b6be', '
Date': 'Sun, 24 Nov 2024 12:06:18 GMT', 'Content-Length': '190'})
HTTP response body: {\"kind\":\"Status\"...', 'X-Kubernetes-Pf-Flowschema-Uid': '
c4624bd9-7fc7-42c6-bcb8-4235110a860d', 'X-Kubernetes-Pf-Prioritylevel-Uid': '4706085f
-6263-43ae-93f5-b4a61de8b6be', 'Date': 'Sun, 24 Nov 2024 12:06:37 GMT', 'Content-Length
': '190'})\nHTTP response body: {\"kind\":\"Status\",\"apiVersion\":\"v1\",\"metadata
\":{},\"status\":\"Failure\",\"message\":\"pods \\\"example-pod\\\" not found\",\"
reason\":\"NotFound\",\"details\":{\"name\":\"example-pod\",\"kind\":\"pods\"},\"code
\":404}

Pod was running 0 out of 20 seconds, which is 0.00% of the time.
```
```

```
```

- fault-unittest-example-service-http-response-state
```log
time=\"2024-11-24T12:06:38Z\" level=warning msg=\"Request Failed\" error=\"Get \\\"http
:\/\/example-service.default.svc.cluster.local:80\\\": dial tcp 10.96.255.84:80:
connect: connection refused\"\ntime=\"2024-11-24T12:06:38Z\" level=warning msg=\"
Request Failed\" error=\"Get \\\"http:\/\/example-service.default.svc.cluster.local
:80\\\": dial tcp 10.96.255.84:80: connect: connection refused\"\ntime=\"2024-11-24T12
:06:38Z\" level=warning msg=\"Request Failed\" error=\"Get \\\"http:\/\/example-service
.default.svc.cluster.local:8... level=error msg=\"thresholds on metrics '
http_req_failed' have been crossed
```
```

### B.1.5 Improvement

**# 4-0: Agent for reconfiguring K8s manifests**

```
System:
You are a helpful AI assistant for Chaos Engineering.
Given K8s manifests that define a network system, its hypothesis, the overview of a
Chaos-Engineeering experiment, and the experiment's results, you will reconfigure the
system based on analysis of the experiment's results.
Always keep the following rules:
- NEVER change the original intention (its description) of the original version of the
system.
- NEVER do the same reconfiguration as in the history.
- Start with simple reconfiguration, and if the hypothesis is still not satisfied,
gradually try more complex reconfigurations.
- The output should be formatted as a JSON instance that conforms to the JSON schema
below.

As an example, for the schema {\"properties\": {\"foo\": {\"title\": \"Foo\", \"
description\": \"a list of strings\", \"type\": \"array\", \"items\": {\"type\": \"
string\"}}}, \"required\": [\"foo\"]}\nthe object {\"foo\": [\"bar\", \"baz\"]} is a
well-formatted instance of the schema. The object {\"properties\": {\"foo\": [\"bar\",
\"baz\"]}} is not well-formatted.

Here is the output schema:
```
{\"properties\": {\"thought\": {\"title\": \"Thought\", \"description\": \"Describe
your plan to modify the K8s manifests.\", \"type\": \"string\"}, \"modified_k8s_yamls
\": {\"title\": \"Modified K8S Yamls\", \"description\": \"The list of modified K8s
manifests (yamls). If you create a new manifest to modify resources in an existing
manifest, make sure to delete the existing manifest before creating the new one.\", \"
type\": \"array\", \"items\": {\"$ref\": \"#/definitions/ModK8sYAML\"}}}, \"required\":
 [\"thought\", \"modified_k8s_yamls\"], \"definitions\": {\"ModK8sYAML\": {\"title\":
\"ModK8sYAML\", \"type\": \"object\", \"properties\": {\"mod_type\": {\"title\": \"Mod
Type\", \"description\": \"Modification type. Select from ['replace', 'create', 'delete
']. The 'replace' replaces/overwites the content of an exisiting yaml. The 'create'
creates a new yaml. The 'delete' deletes an existing yaml.\", \"enum\": [\"replace\",
\"create\", \"delete\"], \"type\": \"string\"}, \"fname\": {\"title\": \"Fname\", \"
description\": \"The file name of the modified yaml. If mod_type is 'replace' or '
delete', the name must match an existing yaml's name. If mod_type='create', name the
file appropriately to avoid overlapping with existing yamls' names.\", \"type\": \"
string\"}, \"explanation\": {\"title\": \"Explanation\", \"description\": \"If mod_type
 is 'delete', explain why you need to delete the yaml. If mod_type is 'replace',
explain which part you should modify from the original conde and why. If mod_type is '
create', explain whether it is a completely new resource or a replacement resouce for
an existing resource. If it is a replacement, also explain the differences and the
reasons for them, just like with 'replace'.\", \"type\": \"string\"}, \"code\": {\"
title\": \"Code\", \"description\": \"If mod_type is 'delete', this field is not
required. Otherwise, write the content of a K8s YAML manifest modified to pass all the
unit tests. Write only the content of the code, and for dictionary values, enclose them
 within a pair of single double quotes (\\\").\", \"type\": \"string\"}}, \"required\":
 [\"mod_type\", \"fname\", \"explanation\"]}}}
```

Human:
Here is the overview of my system (original version):
{user_input2}

Here is the hypothesis for my system:
```
```

```
The hypothesis is "The steady states of the sytem are maintained even when the fault
scenario occurs (i.e., when the faults are injected)".
The steady states here are as follows:
{steady_states}

The fault scenario here is as follows:
{detailed_fault_scenario}

# Here is the overview of my Chaos-Engineering experiment to verify the hypothesis:
{experiment_plan_summary}

# The experiment's results of the original system are as follows:
{experiment_result}

First, please analyze the results and provide an analysis report rich in insights.

AI:
# Here is my analysis report:
{analysis_report}

Human:
Then, please reconfigure the system to avoid the fails (improve resiliency).


------------------------------------------------------------------------------------------
 In the improvement loop, the prompts below will be stacked as improvement history
------------------------------------------------------------------------------------------


AI:
```json
{output}
```"""

Human:
Here is the K8s menifests of the modified system (version={mod_version}):
{k8s_yamls_mod}

The experiment's results of the modified system were as follows:
{experiment_result_mod}

Please analyze the results and provide an analysis report rich in insights again.

AI:
Here is my analysis report:
{analysis_report_mod}

Human:
Then, please reconfigure the system to avoid the fails (improve resiliency).

 In the verification loop, the prompts below will be stacked as verification history
 Verification history is kept within a single verification loop and is reset in every
 improvement iteration

AI:
```json
{output}
```

User:
Your current unittest causes errors when conducted.
The error message is as follows:
{error_message}

Please analyze the reason why the errors occur, then fix the errors.
Always keep the following rules:
- Ensure that the implementation supports variable durations again.
- NEVER repeat the same fixes that have been made in the past.
- Fix only the parts related to the errors without changing the original content.
- the same format instruction as in the System role
```

Example text embedded to `analysis_report`

```
The Chaos Engineering experiment aimed to test the system's resilience by simulating a
cyber attack through PodChaos and NetworkChaos. The experiment results indicate
failures in maintaining the defined steady states during and after the fault injection
phase. Here is a detailed analysis of the results:

1. **Pre-Validation Phase:**
  - Both pre-unittests, `example-pod-running-state` and `example-service-http-response-
  state`, passed successfully. This indicates that the system was in a healthy state
  before the fault injection, with the Pod running 100% of the time and the Service
  responding with a 200 OK status for all requests.

2. **Fault Injection Phase:**
  - **PodChaos Fault:** The `fault-unittest-example-pod-running-state` failed because
  the Pod was not found (404 error) during the test. This is expected due to the
  PodChaos action 'pod-kill', which terminated the Pod. Since the Pod's restart policy
  is set to 'Never', it did not restart, leading to 0% uptime during the test. This
  highlights the critical issue of having a single Pod with no redundancy or automatic
  recovery mechanism.
  - **NetworkChaos Fault:** The `fault-unittest-example-service-http-response-state`
  also failed. The logs show repeated connection refused errors, indicating that the
  Service could not route traffic to the Pod, as it was not running. This failure is a
  direct consequence of the Pod being unavailable, demonstrating the lack of redundancy
   and the impact of a single point of failure.

3. **Post-Validation Phase:**
  - Both post-unittests, `example-pod-running-state` and `example-service-http-response
  -state`, failed. The Pod was still not found, and the Service continued to refuse
  connections. This indicates that the system did not recover to its steady states
  after the faults were removed, primarily due to the Pod's restart policy and the
  absence of a controller to manage Pod lifecycle and redundancy.

**Insights and Recommendations:**
  - **Pod Restart Policy:** The Pod's restart policy should be changed from 'Never' to
  'Always' or 'OnFailure' to ensure automatic recovery in case of failures.
  - **Redundancy and Scalability:** Implement a Deployment or ReplicaSet to manage the
  Pod. This will provide redundancy and ensure that a minimum number of Pods are always
   running, improving the system's resilience to failures.
  - **Service Availability:** Ensure that the Service can handle traffic even if one
  Pod fails by having multiple replicas. This can be achieved by scaling the Deployment
   to have more than one replica.
  - **Monitoring and Alerts:** Implement monitoring and alerting mechanisms to detect
  and respond to Pod failures promptly, minimizing downtime.

By addressing these issues, the system can improve its resilience and maintain its
steady states even during fault scenarios.\nHuman: Then, please reconfigure the system
to avoid the fails (improve resiliency).
```

B.1.6 POST-PROCESSING

 # EX: Agent for summarizing a completed CE cycle

```
System:
You are a helpful AI assistant for Chaos Engineering.
Given a summary of a Chaos Engineering cycle, please elaborate the summary.
The output should be formatted as a JSON instance that conforms to the JSON schema
below.

As an example, for the schema {\"properties\": {\"foo\": {\"title\": \"Foo\", \"
description\": \"a list of strings\", \"type\": \"array\", \"items\": {\"type\": \"
string\"}}}, \"required\": [\"foo\"]}\nthe object {\"foo\": [\"bar\", \"baz\"]} is a
well-formatted instance of the schema. The object {\"properties\": {\"foo\": [\"bar\",
\"baz\"]}} is not well-formatted.

Here is the output schema:
```
{\"properties\": {\"summary\": {\"title\": \"Summary\", \"type\": \"string\"}}, \"
required\": [\"summary\"]}
```

Human:
Here is the overview of a Chaos Engineering Cycle:
# Here is a Chaos Engineering cycle
```

```
## Step 0. User-input understanding
### Here is the overview of user inputs:
{user_input2}

## Step 1. Hypothesis definition
### Here is the overview of the hypothesis for the system:
The hypothesis is "The steady states of the system are maintained even when the fault
scenario occurs (i.e., when the faults are injected)".
The steady states here are as follows:
{steady_states}

The fault scenario here is as follows:
{detailed_fault_scenario}

## Step 2.1. Chaos-Engineering experiment
### Here is the overview of my Chaos-Engineering experiment to verify the hypothesis:
{experiment_plan_summary}

## Step 2.2, 3, 4. Experiment execution, analysis and improvement (reconfiguring the
system to satisfy the hypothesis)
### Here is the improvement history:
{improvement_history}

Please elaborate the above summary of a Chaos Engineering Cycle.
```

Example text embedded to **improvement_history**

```
### Experiment result (1st try)
the same content as experiment_result

### Analysis report (1st try)
the same content as analysis_report

### Improvement result (1st try)
1 K8s manifests are modified:
- The K8s manifest 'nginx/pod.yaml' was replaced.

Replace the Pod manifest with a Deployment manifest to provide redundancy and automatic
 recovery. The Deployment will manage the Pod lifecycle, ensuring that a specified
number of replicas are always running, which addresses both the restart policy and
redundancy issues.
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: example-deployment
  labels:
    app: example
spec:
  replicas: 3
  selector:
    matchLabels:
      app: example
    template:
      metadata:
        labels:
          app: example
      spec:
        containers:
        - name: example-container
          image: nginx:1.17.1
          ports:
          - containerPort: 80
```

Experiment result (2nd try)
Passed unittests:
- pre-unittest-example-pod-running-state
- pre-unittest-example-service-http-response-state
- fault-unittest-example-pod-running-state
- fault-unittest-example-service-http-response-state
- post-unittest-example-pod-running-state
- post-unittest-example-service-http-response-state

Failed unittests:
```

C FULL RESULTS OF CASE STUDIES

In this section, we provide full inputs and outputs for each run studied in the case study. The inputs include instructions, Skaffold configuration files, K8s manifests. The outputs include VaC scripts, Chaos Mesh workflow manifests, reconfigured K8s manifests, adjusted VaC scripts in the experiment replanning phase. In the case study, we pick up the fourth run for NGINX and the fifth run for SOCKSHOP. For complete dialogues, see casestudy_complete_dialogues/Nginx/ChaosEater_GPT4o_nginx_4.pdf for NGINX and casestudy_complete_dialogues/Sockshop/ChaosEater_GPT4o_sockshop_5.pdf in the Supplementary Material. The results of other runs can also be found in the same folders. Table 3 shows the statistics of NGINX and SOCKSHOP.

Table 3: Statistics of the systems for the case study. Tokens are counted by the tokenizer `cl100k_base`.

System	# manifests	# lines	# words	# tokens
NGINX	2	24	373	115
SOCKSHOP	29	869	17696	4605

C.1 NGINX

C.1.1 INPUTS

Instructions

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

skaffold.yaml

```
apiVersion: skaffold/v3
kind: Config
metadata:
  name: nginx-app
manifests:
  rawYaml:
    - pod.yaml
    - service.yaml
```

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
```

service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: example-service
spec:
  selector:
    app: example
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

C.1.2 Outputs

VaC script for the steady state #1: The Pod should be running at least 90% of the time during the check period

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

class TestPodRunningState(K8sAPIBase):
    def __init__(self):
        super().__init__()

    def check_pod_status(self, namespace, pod_name):
        try:
            pod = self.v1.read_namespaced_pod(name=pod_name, namespace=namespace)
            return pod.status.phase == 'Running'
        except client.exceptions.ApiException as e:
            print(f"Exception when calling CoreV1Api->read_namespaced_pod: {e}")
            return False

    def test_pod_running_state(self, duration):
        namespace = 'default'
        pod_name = 'example-pod'
        running_count = 0

        # Check the pod status every second for the specified duration
        for _ in range(duration):
            if self.check_pod_status(namespace, pod_name):
                running_count += 1
            time.sleep(1)

        # Calculate the running percentage
        running_percentage = (running_count / duration) * 100

        # Assert that the running percentage is at least 90%
        assert running_percentage >= 90, f"Pod '{pod_name}' running percentage is below
         threshold: {running_percentage}%"

        print(f"Pod '{pod_name}' running status checked {duration} times. Running
        percentage: {running_percentage}%.")


def main():
    parser = argparse.ArgumentParser(description='Test if a pod is running at least 90%
     of the time.')
    parser.add_argument('--duration', type=int, default=5, help='Duration to check the
    pod status in seconds.')
    args = parser.parse_args()

    test = TestPodRunningState()
    test.test_pod_running_state(args.duration)


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

VaC script for the steady state #2: Service availability should be at least 99.9% with a response status of 200

```
import http from 'k6/http';
import { check } from 'k6';

export const options = {
  vus: 1,
  duration: '5s',
  thresholds: {
    // Ensure that the service availability is at least 99.9%
    'http_req_failed': ['rate<=0.001'], // 0.1% failure rate corresponds to 99.9%
    availability
  },
};

export default function () {
  const res = http.get('http://example-service.default.svc.cluster.local:80');
  check(res, {
    'status is 200': (r) => r.status === 200,
  });
}
```

Chaos Mesh workflow manifest

```
apiVersion: chaos-mesh.org/v1alpha1
kind: Workflow
metadata:
  name: chaos-experiment-20241124-132854
spec:
  entry: the-entry
  templates:
    #-------------------------------
    # entry point of whole workflow
    #-------------------------------
    - name: the-entry
      templateType: Serial
      deadline: 30m51s
      children:
        - pre-validation-phase
        - fault-injection-phase
        - post-validation-phase

    #-------------------------------------
    # Entry point of pre-validation-phase
    #-------------------------------------
    - name: pre-validation-phase
      templateType: Serial
      deadline: 10m10s
      children:
        - pre-validation-overlapped-workflows

    - name: pre-validation-suspend-workflow
      templateType: Serial
      deadline: 5m10s
      children:
        - pre-validation-suspend
        - pre-unittest-example-service-availability

    - name: pre-validation-suspend
      templateType: Suspend
      deadline: 5s

    - name: pre-validation-overlapped-workflows
      templateType: Parallel
      deadline: 5m10s
      children:
        - pre-unittest-example-pod-running
        - pre-validation-suspend-workflow

    # Definitions of children of pre-validation-phase
    - name: pre-unittest-example-pod-running
      templateType: Task
      deadline: 5m5s
```

```yaml
    task:
      container:
        name: pre-unittest-example-pod-running-container
        image: chaos-eater/k8sapi:1.0
        imagePullPolicy: IfNotPresent
        command: ["/bin/bash", "-c"]
        args: ["python /chaos-eater/sandbox/cycle_20241124_132128/unittest_example-pod-running_mod0.py --duration 5"]
        volumeMounts:
          - name: pvc-volume
            mountPath: /chaos-eater
      volumes:
        - name: pvc-volume
          persistentVolumeClaim:
            claimName: pvc

  - name: pre-unittest-example-service-availability
    templateType: Task
    deadline: 5m5s
    task:
      container:
        name: pre-unittest-example-service-availability-container
        image: grafana/k6:latest
        command: ["k6", "run", "--duration", "5s", "--quiet", "/chaos-eater/sandbox/cycle_20241124_132128/unittest_example-service-availability_mod0.js"]
        volumeMounts:
          - name: pvc-volume
            mountPath: /chaos-eater
      volumes:
        - name: pvc-volume
          persistentVolumeClaim:
            claimName: pvc

  #--------------------------------------
  # Entry point of fault-injection-phase
  #--------------------------------------
  - name: fault-injection-phase
    templateType: Serial
    deadline: 10m30s
    children:
      - fault-injection-overlapped-workflows

  - name: fault-injection-parallel-workflow
    templateType: Parallel
    deadline: 5m10s
    children:
      - fault-unittest-example-pod-running
      - fault-podchaos

  - name: fault-injection-suspend-workflow
    templateType: Serial
    deadline: 5m30s
    children:
      - fault-injection-suspend
      - fault-injection-parallel-workflows

  - name: fault-injection-suspend
    templateType: Suspend
    deadline: 10s

  - name: fault-injection-parallel-workflows
    templateType: Parallel
    deadline: 5m20s
    children:
      - fault-unittest-example-service-availability
      - fault-networkchaos

  - name: fault-injection-overlapped-workflows
    templateType: Parallel
    deadline: 5m30s
    children:
      - fault-injection-parallel-workflow
      - fault-injection-suspend-workflow

  # Definitions of children of pre-validation-phase
  # unit tests
  - name: fault-unittest-example-pod-running
    templateType: Task
```

```
    deadline: 5m10s
    task:
      container:
        name: fault-unittest-example-pod-running-container
        image: chaos-eater/k8sapi:1.0
        imagePullPolicy: IfNotPresent
        command: ["/bin/bash", "-c"]
        args: ["python /chaos-eater/sandbox/cycle_20241124_132128/unittest_example-
        pod-running_mod0.py --duration 10"]
        volumeMounts:
          - name: pvc-volume
            mountPath: /chaos-eater
      volumes:
        - name: pvc-volume
          persistentVolumeClaim:
            claimName: pvc

  - name: fault-unittest-example-service-availability
    templateType: Task
    deadline: 5m20s
    task:
      container:
        name: fault-unittest-example-service-availability-container
        image: grafana/k6:latest
        command: ["k6", "run", "--duration", "20s", "--quiet", "/chaos-eater/sandbox/
        cycle_20241124_132128/unittest_example-service-availability_mod0.js"]
        volumeMounts:
          - name: pvc-volume
            mountPath: /chaos-eater
      volumes:
        - name: pvc-volume
          persistentVolumeClaim:
            claimName: pvc

  # fault_injections
  - name: fault-podchaos
    templateType: PodChaos
    deadline: 10s
    podChaos:
      action: pod-kill
      mode: one
      selector:
        labelSelectors:
          app: example
        namespaces:
          - default


  - name: fault-networkchaos
    templateType: NetworkChaos
    deadline: 20s
    networkChaos:
      action: delay
      delay:
        correlation: '50'
        jitter: 10ms
        latency: 100ms
      device: eth0
      direction: to
      mode: all
      selector:
        labelSelectors:
          app: example
        namespaces:
          - default
      target:
        mode: all
        selector:
          labelSelectors:
            app: example
          namespaces:
            - default


  #-------------------------------------
  # Entry point of post-validation phase
  #-------------------------------------
  - name: post-validation-phase
```

```
      templateType: Serial
      deadline: 10m11s
      children:
        - post-validation-overlapped-workflows

    - name: post-validation-suspend-workflow
      templateType: Serial
      deadline: 5m8s
      children:
        - post-validation-suspend
        - post-unittest-example-pod-running

    - name: post-validation-suspend
      templateType: Suspend
      deadline: 2s

    - name: post-validation-suspend-workflow2
      templateType: Serial
      deadline: 5m11s
      children:
        - post-validation-suspend2
        - post-unittest-example-service-availability

    - name: post-validation-suspend2
      templateType: Suspend
      deadline: 6s

    - name: post-validation-overlapped-workflows
      templateType: Parallel
      deadline: 5m11s
      children:
        - post-validation-suspend-workflow
        - post-validation-suspend-workflow2

    # Definitions of children of pre-validation-phase
    - name: post-unittest-example-pod-running
      templateType: Task
      deadline: 5m6s
      task:
        container:
          name: post-unittest-example-pod-running-container
          image: chaos-eater/k8sapi:1.0
          imagePullPolicy: IfNotPresent
          command: ["/bin/bash", "-c"]
          args: ["python /chaos-eater/sandbox/cycle_20241124_132128/unittest_example-pod-running_mod0.py --duration 6"]
          volumeMounts:
            - name: pvc-volume
              mountPath: /chaos-eater
        volumes:
          - name: pvc-volume
            persistentVolumeClaim:
              claimName: pvc

    - name: post-unittest-example-service-availability
      templateType: Task
      deadline: 5m5s
      task:
        container:
          name: post-unittest-example-service-availability-container
          image: grafana/k6:latest
          command: ["k6", "run", "--duration", "5s", "--quiet", "/chaos-eater/sandbox/cycle_20241124_132128/unittest_example-service-availability_mod0.js"]
          volumeMounts:
            - name: pvc-volume
              mountPath: /chaos-eater
        volumes:
          - name: pvc-volume
            persistentVolumeClaim:
              claimName: pvc
```

Reconfigured pod.yaml

```
apiVersion: apps/v1
kind: Deployment
```

```yaml
metadata:
  name: example-deployment
  labels:
    app: example
spec:
  replicas: 3
  selector:
    matchLabels:
      app: example
  template:
    metadata:
      labels:
        app: example
    spec:
      containers:
      - name: example-container
        image: nginx:1.17.1
        ports:
        - containerPort: 80
```

Adjusted VaC script for the steady state #1: The Pod should be running at least 90% of the time during the check period

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

class TestDeploymentRunningState(K8sAPIBase):
    def __init__(self):
        super().__init__()

    def check_deployment_pods_status(self, namespace, deployment_name):
        try:
            pods = self.v1.list_namespaced_pod(namespace=namespace, label_selector=f'app={deployment_name}').items
            running_pods = [pod for pod in pods if pod.status.phase == 'Running']
            return len(running_pods) > 0
        except client.exceptions.ApiException as e:
            print(f"Exception when calling CoreV1Api->list_namespaced_pod: {e}")
            return False

    def test_deployment_running_state(self, duration):
        namespace = 'default'
        deployment_name = 'example'
        running_count = 0

        # Check the deployment pods status every second for the specified duration
        for _ in range(duration):
            if self.check_deployment_pods_status(namespace, deployment_name):
                running_count += 1
            time.sleep(1)

        # Calculate the running percentage
        running_percentage = (running_count / duration) * 100

        # Assert that the running percentage is at least 90%
        assert running_percentage >= 90, f"Deployment '{deployment_name}' running percentage is below threshold: {running_percentage}%"

        print(f"Deployment '{deployment_name}' running status checked {duration} times. Running percentage: {running_percentage}%.")


def main():
    parser = argparse.ArgumentParser(description='Test if a deployment has at least one pod running at least 90% of the time.')
    parser.add_argument('--duration', type=int, default=5, help='Duration to check the deployment pods status in seconds.')
    args = parser.parse_args()

    test = TestDeploymentRunningState()
    test.test_deployment_running_state(args.duration)

```

```python

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

C.2 SockShop

C.2.1 Inputs

Instructions

```
- The Chaos-Engineering experiment must be completed within 1 minute.
- When using k6 in steady-state definition, always select a request URL from the
following options (other requests are invalid):
1. http://front-end.sock-shop.svc.cluster.local/
2. http://front-end.sock-shop.svc.cluster.local/catalogue?size=10
3. http://front-end.sock-shop.svc.cluster.local/detail.html?id=<ID>
   Replace <ID> with an available ID: [03fef6ac-1896-4ce8-bd69-b798f85c6e0b, 3395a43e-2
   d88-40de-b95f-e00e1502085b, 510a0d7e-8e83-4193-b483-e27e09ddc34d, 808a2de1-1aaa-4c25
   -a9b9-6612e8f29a38, 819e1fbf-8b7e-4f6d-811f-693534916a8b, 837ab141-399e-4c1f-9abc-
   bace40296bac, a0a4f044-b040-410d-8ead-4de0446aec7e, d3588630-ad8e-49df-bbd7-3167
   f7efb246, zzz4f044-b040-410d-8ead-4de0446aec7e]
4. http://front-end.sock-shop.svc.cluster.local/category/
5. http://front-end.sock-shop.svc.cluster.local/category?tags=<TAG>
   Replace <TAG> with an available tag: [magic, action, blue, brown, black, sport,
   formal, red, green, skin, geek]
6. http://front-end.sock-shop.svc.cluster.local/basket.html
```

skaffold.yaml

```yaml
apiVersion: skaffold/v3
kind: Config
metadata:
  name: sock-shop-app
manifests:
  rawYaml:
    - manifests/00-sock-shop-ns.yaml
    - manifests/01-carts-dep.yaml
    - manifests/02-carts-svc.yaml
    - manifests/03-carts-db-dep.yaml
    - manifests/04-carts-db-svc.yaml
    - manifests/05-catalogue-dep.yaml
    - manifests/06-catalogue-svc.yaml
    - manifests/07-catalogue-db-dep.yaml
    - manifests/08-catalogue-db-svc.yaml
    - manifests/09-front-end-dep.yaml
    - manifests/10-front-end-svc.yaml
    - manifests/11-orders-dep.yaml
    - manifests/12-orders-svc.yaml
    - manifests/13-orders-db-dep.yaml
    - manifests/14-orders-db-svc.yaml
    - manifests/15-payment-dep.yaml
    - manifests/16-payment-svc.yaml
    - manifests/17-queue-master-dep.yaml
    - manifests/18-queue-master-svc.yaml
    - manifests/19-rabbitmq-dep.yaml
    - manifests/20-rabbitmq-svc.yaml
    - manifests/21-session-db-dep.yaml
    - manifests/22-session-db-svc.yaml
    - manifests/23-shipping-dep.yaml
    - manifests/24-shipping-svc.yaml
    - manifests/25-user-dep.yaml
    - manifests/26-user-svc.yaml
    - manifests/27-user-db-dep.yaml
    - manifests/28-user-db-svc.yaml
```

manifests/00-sock-shop-ns.yaml

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: sock-shop
```

manifests/01-carts-dep.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: carts
  labels:
    name: carts
  namespace: sock-shop
spec:
  replicas: 2
  selector:
    matchLabels:
      name: carts
  template:
    metadata:
      labels:
        name: carts
    spec:
      containers:
      - name: carts
        image: weaveworksdemos/carts:0.4.8
        env:
         - name: JAVA_OPTS
           value: -Xms64m -Xmx128m -XX:+UseG1GC -Djava.security.egd=file:/dev/urandom -
           Dspring.zipkin.enabled=false
        resources:
          limits:
            cpu: 300m
            memory: 500Mi
          requests:
            cpu: 100m
            memory: 200Mi
        ports:
        - containerPort: 80
        securityContext:
          runAsNonRoot: true
          runAsUser: 10001
          capabilities:
            drop:
              - all
            add:
              - NET_BIND_SERVICE
          readOnlyRootFilesystem: true
        volumeMounts:
        - mountPath: /tmp
          name: tmp-volume
      volumes:
        - name: tmp-volume
          emptyDir:
            medium: Memory
      nodeSelector:
        beta.kubernetes.io/os: linux
```

manifests/02-carts-svc.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: carts
  annotations:
        prometheus.io/scrape: 'true'
  labels:
    name: carts
```

```yaml
  namespace: sock-shop
spec:
  ports:
    # the port that this service should serve on
  - port: 80
    targetPort: 80
  selector:
    name: carts
```

manifests/03-carts-db-dep.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: carts-db
  labels:
    name: carts-db
  namespace: sock-shop
spec:
  replicas: 2
  selector:
    matchLabels:
      name: carts-db
  template:
    metadata:
      labels:
        name: carts-db
    spec:
      containers:
      - name: carts-db
        image: mongo
        ports:
        - name: mongo
          containerPort: 27017
        securityContext:
          capabilities:
            drop:
              - all
            add:
              - CHOWN
              - SETGID
              - SETUID
          readOnlyRootFilesystem: true
        volumeMounts:
        - mountPath: /tmp
          name: tmp-volume
      volumes:
        - name: tmp-volume
          emptyDir:
            medium: Memory
      nodeSelector:
        beta.kubernetes.io/os: linux
```

manifests/04-carts-db-svc.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: carts-db
  labels:
    name: carts-db
  namespace: sock-shop
spec:
  ports:
    # the port that this service should serve on
  - port: 27017
    targetPort: 27017
  selector:
    name: carts-db
```

manifests/05-catalogue-dep.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: catalogue
  labels:
    name: catalogue
  namespace: sock-shop
spec:
  replicas: 2
  selector:
    matchLabels:
      name: catalogue
  template:
    metadata:
      labels:
        name: catalogue
    spec:
      containers:
      - name: catalogue
        image: weaveworksdemos/catalogue:0.3.5
        command: ["/app"]
        args:
        - -port=80
        resources:
          limits:
            cpu: 200m
            memory: 200Mi
          requests:
            cpu: 100m
            memory: 100Mi
        ports:
        - containerPort: 80
        securityContext:
          runAsNonRoot: true
          runAsUser: 10001
          capabilities:
            drop:
              - all
            add:
              - NET_BIND_SERVICE
          readOnlyRootFilesystem: true
        livenessProbe:
          httpGet:
            path: /health
            port: 80
          initialDelaySeconds: 300
          periodSeconds: 3
        readinessProbe:
          httpGet:
            path: /health
            port: 80
          initialDelaySeconds: 180
          periodSeconds: 3
      nodeSelector:
        beta.kubernetes.io/os: linux
```

manifests/06-catalogue-svc.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: catalogue
  annotations:
        prometheus.io/scrape: 'true'
  labels:
    name: catalogue
  namespace: sock-shop
spec:
  ports:
    # the port that this service should serve on
  - port: 80
    targetPort: 80
  selector:
```

```
    name: catalogue
```

manifests/06-catalogue-svc.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: catalogue
  annotations:
        prometheus.io/scrape: 'true'
  labels:
    name: catalogue
  namespace: sock-shop
spec:
  ports:
    # the port that this service should serve on
  - port: 80
    targetPort: 80
  selector:
    name: catalogue
```

manifests/07-catalogue-db-dep.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: catalogue-db
  labels:
    name: catalogue-db
  namespace: sock-shop
spec:
  replicas: 2
  selector:
    matchLabels:
      name: catalogue-db
  template:
    metadata:
      labels:
        name: catalogue-db
    spec:
      containers:
      - name: catalogue-db
        image: weaveworksdemos/catalogue-db:0.3.0
        env:
          - name: MYSQL_ROOT_PASSWORD
            value: fake_password
          - name: MYSQL_DATABASE
            value: socksdb
        ports:
        - name: mysql
          containerPort: 3306
      nodeSelector:
        beta.kubernetes.io/os: linux
```

manifests/08-catalogue-db-svc.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: catalogue-db
  labels:
    name: catalogue-db
  namespace: sock-shop
spec:
  ports:
    # the port that this service should serve on
  - port: 3306
    targetPort: 3306
```

```
  selector:
    name: catalogue-db
```

manifests/09-front-end-dep.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: front-end
  namespace: sock-shop
spec:
  replicas: 1
  selector:
    matchLabels:
      name: front-end
  template:
    metadata:
      labels:
        name: front-end
    spec:
      containers:
      - name: front-end
        image: weaveworksdemos/front-end:0.3.12
        resources:
          limits:
            cpu: 300m
            memory: 1000Mi
          requests:
            cpu: 100m
            memory: 300Mi
        ports:
        - containerPort: 8079
        env:
        - name: SESSION_REDIS
          value: "true"
        securityContext:
          runAsNonRoot: true
          runAsUser: 10001
          capabilities:
            drop:
              - all
          readOnlyRootFilesystem: true
        livenessProbe:
          httpGet:
            path: /
            port: 8079
          initialDelaySeconds: 300
          periodSeconds: 3
        readinessProbe:
          httpGet:
            path: /
            port: 8079
          initialDelaySeconds: 30
          periodSeconds: 3
      nodeSelector:
        beta.kubernetes.io/os: linux
```

manifests/10-front-end-svc.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: front-end
  annotations:
        prometheus.io/scrape: ’true’
  labels:
    name: front-end
  namespace: sock-shop
spec:
  type: NodePort
  ports:
  - port: 80
```

```
    targetPort: 8079
    nodePort: 30001
  selector:
    name: front-end
```

manifests/11-orders-dep.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: orders
  labels:
    name: orders
  namespace: sock-shop
spec:
  replicas: 2
  selector:
    matchLabels:
      name: orders
  template:
    metadata:
      labels:
        name: orders
    spec:
      containers:
      - name: orders
        image: weaveworksdemos/orders:0.4.7
        env:
         - name: JAVA_OPTS
           value: -Xms64m -Xmx128m -XX:+UseG1GC -Djava.security.egd=file:/dev/urandom -
           Dspring.zipkin.enabled=false
        resources:
          limits:
            cpu: 500m
            memory: 500Mi
          requests:
            cpu: 100m
            memory: 300Mi
        ports:
        - containerPort: 80
        securityContext:
          runAsNonRoot: true
          runAsUser: 10001
          capabilities:
            drop:
              - all
            add:
              - NET_BIND_SERVICE
          readOnlyRootFilesystem: true
        volumeMounts:
        - mountPath: /tmp
          name: tmp-volume
      volumes:
        - name: tmp-volume
          emptyDir:
            medium: Memory
      nodeSelector:
        beta.kubernetes.io/os: linux
```

manifests/12-orders-svc.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: orders
  annotations:
        prometheus.io/scrape: 'true'
  labels:
    name: orders
  namespace: sock-shop
spec:
  ports:
```

```yaml
    # the port that this service should serve on
  - port: 80
    targetPort: 80
  selector:
    name: orders
```

manifests/13-orders-db-dep.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: orders-db
  labels:
    name: orders-db
  namespace: sock-shop
spec:
  replicas: 2
  selector:
    matchLabels:
      name: orders-db
  template:
    metadata:
      labels:
        name: orders-db
    spec:
      containers:
      - name: orders-db
        image: mongo
        ports:
        - name: mongo
          containerPort: 27017
        securityContext:
          capabilities:
            drop:
              - all
            add:
              - CHOWN
              - SETGID
              - SETUID
          readOnlyRootFilesystem: true
        volumeMounts:
        - mountPath: /tmp
          name: tmp-volume
      volumes:
        - name: tmp-volume
          emptyDir:
            medium: Memory
      nodeSelector:
        beta.kubernetes.io/os: linux
```

manifests/14-orders-db-svc.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: orders-db
  labels:
    name: orders-db
  namespace: sock-shop
spec:
  ports:
    # the port that this service should serve on
  - port: 27017
    targetPort: 27017
  selector:
    name: orders-db
```

manifests/15-payment-dep.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: payment
  labels:
    name: payment
  namespace: sock-shop
spec:
  replicas: 2
  selector:
    matchLabels:
      name: payment
  template:
    metadata:
      labels:
        name: payment
    spec:
      containers:
      - name: payment
        image: weaveworksdemos/payment:0.4.3
        resources:
          limits:
            cpu: 200m
            memory: 200Mi
          requests:
            cpu: 99m
            memory: 100Mi
        ports:
        - containerPort: 80
        securityContext:
          runAsNonRoot: true
          runAsUser: 10001
          capabilities:
            drop:
              - all
            add:
              - NET_BIND_SERVICE
          readOnlyRootFilesystem: true
        livenessProbe:
          httpGet:
            path: /health
            port: 80
          initialDelaySeconds: 300
          periodSeconds: 3
        readinessProbe:
          httpGet:
            path: /health
            port: 80
          initialDelaySeconds: 180
          periodSeconds: 3
      nodeSelector:
        beta.kubernetes.io/os: linux
```

manifests/16-payment-svc.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: payment
  annotations:
        prometheus.io/scrape: 'true'
  labels:
    name: payment
  namespace: sock-shop
spec:
  ports:
    # the port that this service should serve on
  - port: 80
    targetPort: 80
  selector:
    name: payment
```

manifests/17-queue-master-dep.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: queue-master
  labels:
    name: queue-master
  namespace: sock-shop
spec:
  replicas: 2
  selector:
    matchLabels:
      name: queue-master
  template:
    metadata:
      labels:
        name: queue-master
    spec:
      containers:
      - name: queue-master
        image: weaveworksdemos/queue-master:0.3.1
        env:
         - name: JAVA_OPTS
           value: -Xms64m -Xmx128m -XX:+UseG1GC -Djava.security.egd=file:/dev/urandom -
           Dspring.zipkin.enabled=false
        resources:
          limits:
            cpu: 300m
            memory: 500Mi
          requests:
            cpu: 100m
            memory: 300Mi
        ports:
        - containerPort: 80
      nodeSelector:
        beta.kubernetes.io/os: linux
```

manifests/18-queue-master-svc.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: queue-master
  annotations:
        prometheus.io/scrape: 'true'
  labels:
    name: queue-master
  namespace: sock-shop
spec:
  ports:
    # the port that this service should serve on
  - port: 80
    targetPort: 80
  selector:
    name: queue-master
```

manifests/19-rabbitmq-dep.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: rabbitmq
  labels:
    name: rabbitmq
  namespace: sock-shop
spec:
  replicas: 2
  selector:
    matchLabels:
      name: rabbitmq
```

```yaml
  template:
    metadata:
      labels:
        name: rabbitmq
      annotations:
        prometheus.io/scrape: "false"
    spec:
      containers:
      - name: rabbitmq
        image: rabbitmq:3.6.8-management
        ports:
        - containerPort: 15672
          name: management
        - containerPort: 5672
          name: rabbitmq
        securityContext:
          capabilities:
            drop:
              - all
            add:
              - CHOWN
              - SETGID
              - SETUID
              - DAC_OVERRIDE
          readOnlyRootFilesystem: true
      - name: rabbitmq-exporter
        image: kbudde/rabbitmq-exporter
        ports:
        - containerPort: 9090
          name: exporter
      nodeSelector:
        beta.kubernetes.io/os: linux
```

manifests/20-rabbitmq-svc.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: rabbitmq
  annotations:
        prometheus.io/scrape: 'true'
        prometheus.io/port: '9090'
  labels:
    name: rabbitmq
  namespace: sock-shop
spec:
  ports:
    # the port that this service should serve on
  - port: 5672
    name: rabbitmq
    targetPort: 5672
  - port: 9090
    name: exporter
    targetPort: exporter
    protocol: TCP
  selector:
    name: rabbitmq
```

manifests/21-session-db-dep.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: session-db
  labels:
    name: session-db
  namespace: sock-shop
spec:
  replicas: 2
  selector:
    matchLabels:
      name: session-db
```

```
  template:
    metadata:
      labels:
        name: session-db
      annotations:
        prometheus.io.scrape: "false"
    spec:
      containers:
      - name: session-db
        image: redis:alpine
        ports:
        - name: redis
          containerPort: 6379
        securityContext:
          capabilities:
            drop:
              - all
            add:
              - CHOWN
              - SETGID
              - SETUID
          readOnlyRootFilesystem: true
      nodeSelector:
        beta.kubernetes.io/os: linux
```

manifests/22-session-db-svc.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: session-db
  labels:
    name: session-db
  namespace: sock-shop
spec:
  ports:
    # the port that this service should serve on
  - port: 6379
    targetPort: 6379
  selector:
    name: session-db
```

manifests/23-shipping-dep.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: shipping
  labels:
    name: shipping
  namespace: sock-shop
spec:
  replicas: 2
  selector:
    matchLabels:
      name: shipping
  template:
    metadata:
      labels:
        name: shipping
    spec:
      containers:
      - name: shipping
        image: weaveworksdemos/shipping:0.4.8
        env:
         - name: ZIPKIN
           value: zipkin.jaeger.svc.cluster.local
         - name: JAVA_OPTS
           value: -Xms64m -Xmx128m -XX:+UseG1GC -Djava.security.egd=file:/dev/urandom -
           Dspring.zipkin.enabled=false
        resources:
          limits:
```

```
            cpu: 300m
            memory: 500Mi
          requests:
            cpu: 100m
            memory: 300Mi
        ports:
        - containerPort: 80
        securityContext:
          runAsNonRoot: true
          runAsUser: 10001
          capabilities:
            drop:
              - all
            add:
              - NET_BIND_SERVICE
          readOnlyRootFilesystem: true
        volumeMounts:
        - mountPath: /tmp
          name: tmp-volume
      volumes:
        - name: tmp-volume
          emptyDir:
            medium: Memory
      nodeSelector:
        beta.kubernetes.io/os: linux
```

manifests/24-shipping-svc.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: shipping
  annotations:
        prometheus.io/scrape: 'true'
  labels:
    name: shipping
  namespace: sock-shop
spec:
  ports:
    # the port that this service should serve on
  - port: 80
    targetPort: 80
  selector:
    name: shipping
```

manifests/25-user-dep.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: user
  labels:
    name: user
  namespace: sock-shop
spec:
  replicas: 2
  selector:
    matchLabels:
      name: user
  template:
    metadata:
      labels:
        name: user
    spec:
      containers:
      - name: user
        image: weaveworksdemos/user:0.4.7
        resources:
          limits:
            cpu: 300m
            memory: 200Mi
          requests:
```

```yaml
            cpu: 100m
            memory: 100Mi
        ports:
        - containerPort: 80
        env:
        - name: mongo
          value: user-db:27017
        securityContext:
          runAsNonRoot: true
          runAsUser: 10001
          capabilities:
            drop:
              - all
            add:
              - NET_BIND_SERVICE
          readOnlyRootFilesystem: true
        livenessProbe:
          httpGet:
            path: /health
            port: 80
          initialDelaySeconds: 300
          periodSeconds: 3
        readinessProbe:
          httpGet:
            path: /health
            port: 80
          initialDelaySeconds: 180
          periodSeconds: 3
      nodeSelector:
        beta.kubernetes.io/os: linux
```

manifests/26-user-svc.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: user
  annotations:
        prometheus.io/scrape: 'true'
  labels:
    name: user
  namespace: sock-shop
spec:
  ports:
    # the port that this service should serve on
  - port: 80
    targetPort: 80
  selector:
    name: user
```

manifests/27-user-db-dep.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: user-db
  labels:
    name: user-db
  namespace: sock-shop
spec:
  replicas: 2
  selector:
    matchLabels:
      name: user-db
  template:
    metadata:
      labels:
        name: user-db
    spec:
      containers:
      - name: user-db
        image: weaveworksdemos/user-db:0.3.0
```

```
        ports:
        - name: mongo
          containerPort: 27017
        securityContext:
          capabilities:
            drop:
              - all
            add:
              - CHOWN
              - SETGID
              - SETUID
          readOnlyRootFilesystem: true
        volumeMounts:
        - mountPath: /tmp
          name: tmp-volume
      volumes:
        - name: tmp-volume
          emptyDir:
            medium: Memory
      nodeSelector:
        beta.kubernetes.io/os: linux
```

manifests/28-user-db-svc.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: user-db
  labels:
    name: user-db
  namespace: sock-shop
spec:
  ports:
    # the port that this service should serve on
  - port: 27017
    targetPort: 27017
  selector:
    name: user-db
```

C.2.2 OUTPUTS

VaC script for the steady state #1 (carts-db-dep.yaml): At least 1 ready replica 100% of the time and 2 ready replicas 80% of the time during the monitoring period

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

class TestCartsDBReplicas(K8sAPIBase):
    def __init__(self, namespace='sock-shop', deployment_name='carts-db', duration=5):
        super().__init__()
        self.namespace = namespace
        self.deployment_name = deployment_name
        self.duration = duration
        # Use the correct API client for deployments
        self.apps_v1 = client.AppsV1Api()

    def test_replicas_threshold(self):
        # Initialize counters for ready replicas
        total_checks = 0
        ready_replicas_count = 0
        fully_ready_replicas_count = 0

        for _ in range(self.duration):
            try:
                # Read the deployment status using the correct API client
                deployment = self.apps_v1.read_namespaced_deployment(self.
                deployment_name, self.namespace)
```

```python
                replicas = deployment.status.replicas
                ready_replicas = deployment.status.ready_replicas
                print(f"Total replicas: {replicas}, Ready replicas: {ready_replicas}")

                # Increment the total checks
                total_checks += 1

                # Check if at least 1 replica is ready
                if ready_replicas >= 1:
                    ready_replicas_count += 1

                # Check if both replicas are ready
                if ready_replicas == 2:
                    fully_ready_replicas_count += 1

            except client.exceptions.ApiException as e:
                print(f"Exception when calling AppsV1Api->read_namespaced_deployment: {
                e}")

            # Wait for 1 second before the next check
            time.sleep(1)

        # Calculate the percentage of time conditions are met
        one_ready_percentage = (ready_replicas_count / total_checks) * 100
        two_ready_percentage = (fully_ready_replicas_count / total_checks) * 100

        # Assert the threshold conditions
        assert one_ready_percentage == 100, "At least 1 ready replica was not available
         100% of the time."
        assert two_ready_percentage >= 80, "2 ready replicas were not available at
        least 80% of the time."

        print("Test passed: Steady state conditions are satisfied.")


def main():
    parser = argparse.ArgumentParser(description='Test carts-db replicas threshold')
    parser.add_argument('--duration', type=int, default=5, help='Duration to check the
    replicas')
    args = parser.parse_args()

    # Create an instance of the test class with the specified duration
    test = TestCartsDBReplicas(duration=args.duration)
    # Run the test
    test.test_replicas_threshold()


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

VaC script for the steady state #2 (front-end-dep.yaml): At least 1 ready replica must be present 100% of the time during the monitoring period

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

class TestFrontEndReplica(K8sAPIBase):
    def __init__(self, namespace, deployment_name, duration):
        super().__init__()
        self.namespace = namespace
        self.deployment_name = deployment_name
        self.duration = duration
        # Use AppsV1Api to interact with deployments
        self.apps_v1 = client.AppsV1Api()

    def test_steady_state(self):
        ready_replicas_count = 0

        # Loop for the specified duration
        for _ in range(self.duration):
            # Get the deployment status
```

```python
            resp = self.apps_v1.read_namespaced_deployment_status(self.deployment_name, self.namespace)
            ready_replicas = resp.status.ready_replicas or 0
            print(f"Ready replicas for {self.deployment_name}: {ready_replicas}")

            # Check if the number of ready replicas is at least 1
            if ready_replicas >= 1:
                ready_replicas_count += 1

            # Wait for 1 second before the next check
            time.sleep(1)

        # Calculate the percentage of time the deployment was ready
        readiness_percentage = (ready_replicas_count / self.duration) * 100
        print(f"{self.deployment_name} was ready {ready_replicas_count}/{self.duration} times.")

        # Assert that the deployment was ready 100% of the time
        assert readiness_percentage == 100, f"{self.deployment_name} readiness was {readiness_percentage}%, expected 100%."


def main():
    parser = argparse.ArgumentParser(description='Test front-end replica readiness')
    parser.add_argument('--duration', type=int, default=5, help='Duration to check the replicas')
    args = parser.parse_args()

    # Create a test instance and run the test
    test = TestFrontEndReplica('sock-shop', 'front-end', args.duration)
    test.test_steady_state()


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

Chaos Mesh workflow manifest

```yaml
apiVersion: chaos-mesh.org/v1alpha1
kind: Workflow
metadata:
  name: chaos-experiment-20241127-045539
spec:
  entry: the-entry
  templates:
    #-------------------------------
    # entry point of whole workflow
    #-------------------------------
    - name: the-entry
      templateType: Serial
      deadline: 30m45s
      children:
        - pre-validation-phase
        - fault-injection-phase
        - post-validation-phase

    #-------------------------------------
    # Entry point of pre-validation-phase
    #-------------------------------------
    - name: pre-validation-phase
      templateType: Serial
      deadline: 10m20s
      children:
        - pre-validation-parallel-workflows

    - name: pre-validation-parallel-workflows
      templateType: Parallel
      deadline: 5m20s
      children:
        - pre-unittest-carts-db-replicas
        - pre-unittest-front-end-replica

    # Definitions of children of pre-validation-phase
    - name: pre-unittest-carts-db-replicas
      templateType: Task
```

```
      deadline: 5m20s
      task:
        container:
          name: pre-unittest-carts-db-replicas-container
          image: chaos-eater/k8sapi:1.0
          imagePullPolicy: IfNotPresent
          command: ["/bin/bash", "-c"]
          args: ["python /chaos-eater/sandbox/cycle_20241127_043136/unittest_carts-db-
          replicas_mod0.py --duration 20"]
          volumeMounts:
            - name: pvc-volume
              mountPath: /chaos-eater
        volumes:
          - name: pvc-volume
            persistentVolumeClaim:
              claimName: pvc

    - name: pre-unittest-front-end-replica
      templateType: Task
      deadline: 5m20s
      task:
        container:
          name: pre-unittest-front-end-replica-container
          image: chaos-eater/k8sapi:1.0
          imagePullPolicy: IfNotPresent
          command: ["/bin/bash", "-c"]
          args: ["python /chaos-eater/sandbox/cycle_20241127_043136/unittest_front-end-
          replica_mod0.py --duration 20"]
          volumeMounts:
            - name: pvc-volume
              mountPath: /chaos-eater
        volumes:
          - name: pvc-volume
            persistentVolumeClaim:
              claimName: pvc

    #-------------------------------------
    # Entry point of fault-injection-phase
    #-------------------------------------
    - name: fault-injection-phase
      templateType: Serial
      deadline: 10m15s
      children:
        - fault-injection-overlapped-workflows

    - name: fault-injection-parallel-workflow
      templateType: Parallel
      deadline: 5m10s
      children:
        - fault-unittest-carts-db-replicas
        - fault-stresschaos

    - name: fault-injection-suspend-workflow
      templateType: Serial
      deadline: 5m15s
      children:
        - fault-injection-suspend
        - fault-injection-parallel-workflows

    - name: fault-injection-suspend
      templateType: Suspend
      deadline: 10s

    - name: fault-injection-parallel-workflows
      templateType: Parallel
      deadline: 5m5s
      children:
        - fault-unittest-front-end-replica
        - fault-podchaos

    - name: fault-injection-overlapped-workflows
      templateType: Parallel
      deadline: 5m15s
      children:
        - fault-injection-parallel-workflow
        - fault-injection-suspend-workflow

    # Definitions of children of pre-validation-phase
```

```
# unit tests
- name: fault-unittest-carts-db-replicas
  templateType: Task
  deadline: 5m10s
  task:
    container:
      name: fault-unittest-carts-db-replicas-container
      image: chaos-eater/k8sapi:1.0
      imagePullPolicy: IfNotPresent
      command: ["/bin/bash", "-c"]
      args: ["python /chaos-eater/sandbox/cycle_20241127_043136/unittest_carts-db-
      replicas_mod0.py --duration 10"]
      volumeMounts:
        - name: pvc-volume
          mountPath: /chaos-eater
    volumes:
      - name: pvc-volume
        persistentVolumeClaim:
          claimName: pvc

- name: fault-unittest-front-end-replica
  templateType: Task
  deadline: 5m5s
  task:
    container:
      name: fault-unittest-front-end-replica-container
      image: chaos-eater/k8sapi:1.0
      imagePullPolicy: IfNotPresent
      command: ["/bin/bash", "-c"]
      args: ["python /chaos-eater/sandbox/cycle_20241127_043136/unittest_front-end-
      replica_mod0.py --duration 5"]
      volumeMounts:
        - name: pvc-volume
          mountPath: /chaos-eater
    volumes:
      - name: pvc-volume
        persistentVolumeClaim:
          claimName: pvc

# fault_injections
- name: fault-stresschaos
  templateType: StressChaos
  deadline: 10s
  stressChaos:
    containerNames:
      - carts-db
    mode: all
    selector:
      labelSelectors:
        name: carts-db
      namespaces:
        - sock-shop
    stressors:
      cpu:
        load: 80
        workers: 2


- name: fault-podchaos
  templateType: PodChaos
  deadline: 5s
  podChaos:
    action: pod-kill
    mode: one
    selector:
      labelSelectors:
        name: front-end
      namespaces:
        - sock-shop
    value: '1'


#-------------------------------------
# Entry point of post-validation phase
#-------------------------------------
- name: post-validation-phase
  templateType: Serial
  deadline: 10m10s
```

```yaml
      children:
        - post-validation-parallel-workflows

    - name: post-validation-parallel-workflows
      templateType: Parallel
      deadline: 5m10s
      children:
        - post-unittest-carts-db-replicas
        - post-unittest-front-end-replica

    # Definitions of children of pre-validation-phase
    - name: post-unittest-carts-db-replicas
      templateType: Task
      deadline: 5m10s
      task:
        container:
          name: post-unittest-carts-db-replicas-container
          image: chaos-eater/k8sapi:1.0
          imagePullPolicy: IfNotPresent
          command: ["/bin/bash", "-c"]
          args: ["python /chaos-eater/sandbox/cycle_20241127_043136/unittest_carts-db-replicas_mod0.py --duration 10"]
          volumeMounts:
            - name: pvc-volume
              mountPath: /chaos-eater
        volumes:
          - name: pvc-volume
            persistentVolumeClaim:
              claimName: pvc

    - name: post-unittest-front-end-replica
      templateType: Task
      deadline: 5m10s
      task:
        container:
          name: post-unittest-front-end-replica-container
          image: chaos-eater/k8sapi:1.0
          imagePullPolicy: IfNotPresent
          command: ["/bin/bash", "-c"]
          args: ["python /chaos-eater/sandbox/cycle_20241127_043136/unittest_front-end-replica_mod0.py --duration 10"]
          volumeMounts:
            - name: pvc-volume
              mountPath: /chaos-eater
        volumes:
          - name: pvc-volume
            persistentVolumeClaim:
              claimName: pvc
```

Reconfigured 09-front-end-dep.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: front-end
  namespace: sock-shop
spec:
  replicas: 2
  selector:
    matchLabels:
      name: front-end
  template:
    metadata:
      labels:
        name: front-end
    spec:
      containers:
      - name: front-end
        image: weaveworksdemos/front-end:0.3.12
        resources:
          limits:
            cpu: 300m
            memory: 1000Mi
          requests:
            cpu: 100m
```

```
            memory: 300Mi
        ports:
        - containerPort: 8079
        env:
        - name: SESSION_REDIS
          value: "true"
        securityContext:
          runAsNonRoot: true
          runAsUser: 10001
          capabilities:
            drop:
              - all
          readOnlyRootFilesystem: true
        livenessProbe:
          httpGet:
            path: /
            port: 8079
          initialDelaySeconds: 300
          periodSeconds: 3
        readinessProbe:
          httpGet:
            path: /
            port: 8079
          initialDelaySeconds: 30
          periodSeconds: 3
      nodeSelector:
        beta.kubernetes.io/os: linux
```