Basic Chaos Tool Development Guide With Python

By Shayan Saha

Table of Contents

1. Introduction

1.1. What is Chaos Engineering?

Chaos Engineering is a proactive approach to enhancing system resilience by intentionally injecting faults and observing the system's behavior under stress. This practice aims to identify and address potential vulnerabilities before they can cause real-world disruptions. By simulating failure scenarios, such as network latency, server crashes, or resource exhaustion, teams can gain insights into how their systems respond to unexpected conditions. This helps in building robust systems that can maintain operational integrity and recover quickly from failures, ultimately ensuring reliable service delivery.

Example:
Scenario: Inject high CPU usage on a critical service.
Purpose: Evaluate the system's performance and stability under heavy computational load, ensuring it can maintain service levels or recover gracefully.

1.2. Document goal

The goal of this document is to provide a comprehensive guide for developing a basic chaos engineering tool capable of generating three types of chaos experiments on target Linux servers.

1.3. About the utility we are going to develop?

We are set to develop a straightforward console application utilizing Python and its various libraries. This utility will first establish a connection with the target server and then execute chaos experiments. For testing our utility we will be using a CentOS machine which I already hosted in an Oracle Virtual Box.

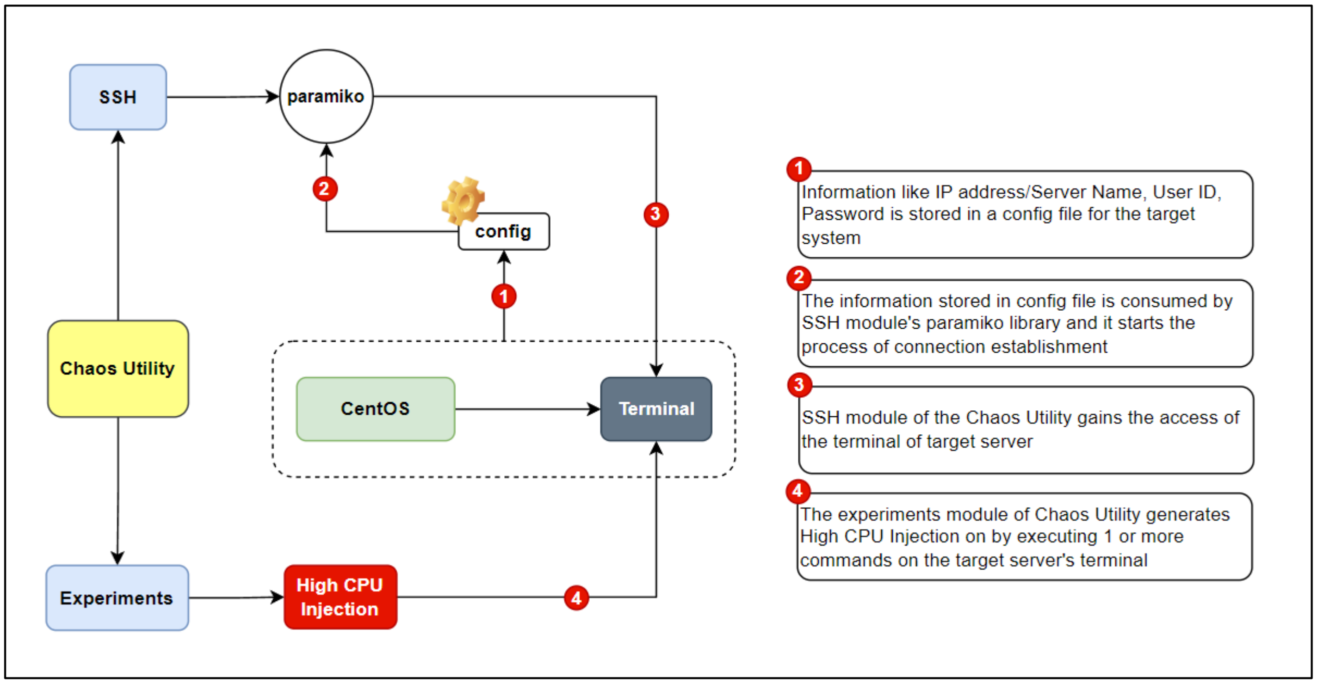
1.4. Requirements

For the development of this utility below requirements should be installed in the host machine

Requirements	How to download/install?
Python 3.6 or above	https://www.python.org/downloads/
pip 24 or above	Comes pre-installed with Python
paramiko library	Execute this command in host machine: pip install paramiko
Oracle VirtualBox	https://www.virtualbox.org/wiki/Downloads
CentOS 7 or above	https://www.centos.org/download/
stress module in the target machine (CentOS)	Execute this command in the terminal of CentOS: sudo yum install stress
Any IDE (Preferred : Visual Studio Code)	https://code.visualstudio.com/download
configparser	Comes pre-installed with Python

2. Development Approach

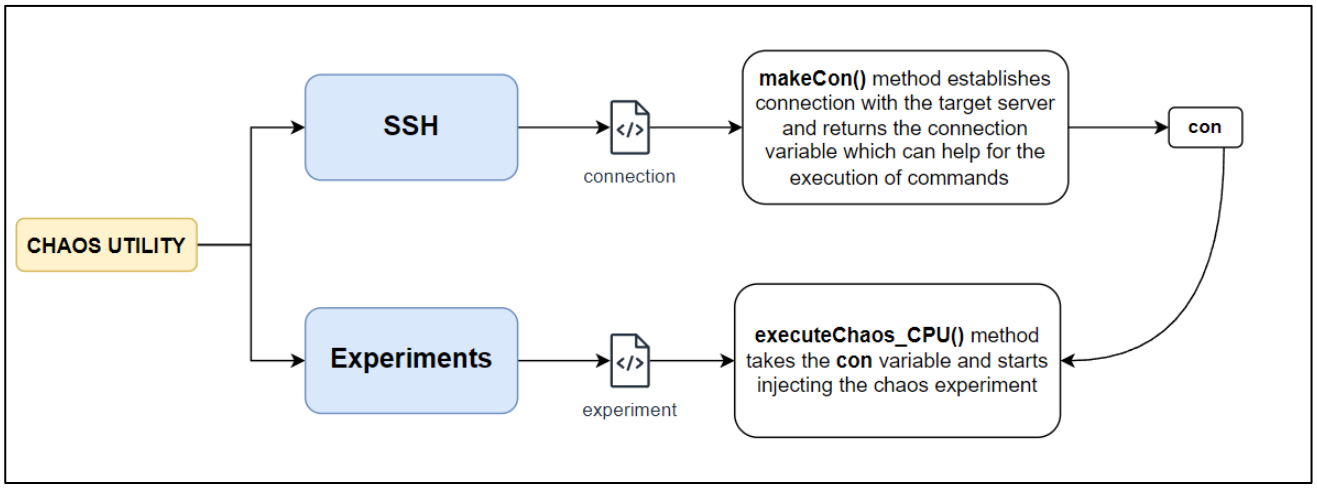
2.1. Architecture Diagram


SSH
paramiko
config
Chaos Utility
CentOS
Terminal
Experiments
High CPU Injection
1
2
3
4
Information like IP address/Server Name, User ID, Password is stored in a config file for the target system
The information stored in config file is consumed by SSH module's paramiko library and it starts the process of connection establishment
SSH module of the Chaos Utility gains the access of the terminal of target server
The experiments module of Chaos Utility generates High CPU Injection on by executing 1 or more commands on the target server's terminal

2.2. Step by step approach

2.2.1. Architecture For Developing Codebase

The entire code will be developed according to the below structure:


CHAOS UTILITY
SSH
connection
makeCon() method establishes connection with the target server and returns the connection variable which can help for the execution of commands
con
Experiments
experiment
executeChaos_CPU() method takes the con variable and starts injecting the chaos experiment

2.2.2. Files Structure For Development

For the development we would need to create below files for our codebase (Following above diagram)

File name	What we will store inside?
connection.py	Method for establishing connection with target server
experiment.py	Method (s) for generating stress on the target server
data.config	Storing all necessary information of the server in order to perform SSH
main.py	For collating all methods and executing them sequentially in order to generate the chaos experiment

2.2.3. Development Steps

- Open Visual Studio Code and create a project, and create above mentioned files.
- Open terminal and create a virtual environment using the below command :

```
python -m venv cpuChaos
```

- It generates a virtual environment for you. All the pre-requisites of the project will refer to this directory and it keeps a separate dependency list dedicated for the project
- Activate the virtual environment by running the bash script : **cpuChaos\Scripts\activate**
- Install the paramiko library using the below command :

```
pip install paramiko
```

- Add your target server information and chaos experiment information (e.g. servername/ip, username, password) in the **data.config** file like below:

```
[centos_server]
server_name = 192.168.56.102
username = root
password = root
cores = 4
duration = 60
```

- Below script is for the **connection.py** file which will help us for doing SSH and gaining access for the target server terminal

```
import paramiko

def makeCon(server_name, username, password):
    ssh = paramiko.SSHClient()
    ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh.connect(server_name, username=username, password=password)
    return ssh
```

- Below script is for **experiment.py** file which will help for generating CPU hog/stress on the target server

```
import connection


def executeChaos_CPU(numberOfCores, duration, server_name, username, password):
    ssh = connection.makeCon(server_name, username, password)
    command = f"stress --cpu {numberOfCores} --timeout {duration}"
    print("**** CHAOS EXPERIMENT STARTED ****")
    try:
        print(f"EXECUTING CPU HOG WITH {numberOfCores} CORES FOR {duration} SECONDS")
        stdin, stdout, stderr = ssh.exec_command(str(command))
        output = stdout.read().decode()
    except:
        print("ERROR WHILE PERFORMING CPU HOG")
    finally:
        print("**** CHAOS EXPERIMENT STOPPED ****")
```

- Below script is for **main.py** file where we are reading the **data.config** file, and feeding all information to the helper methods in order to generate the CPU chaos experiment

```
import configparser
import experiment

config = configparser.ConfigParser()

config.read('data.config')

server_name = config['centos_server']['server_name']
username = config['centos_server']['username']
password = config['centos_server']['password']
cores = config['centos_server'].getint('cores')
duration = config['centos_server'].getint('duration')

experiment.executeChaos_CPU(cores, duration, server_name, username, password)
```

- Now the Chaos Experiment can be started using `python main.py` command, and it can be visualized in the target server

```
  1  [||||||||||||||||||||||||||||||||||||||||||||100.0%]   Tasks: 40, 70 thr; 4 running
  2  [||||||||||||||||||||||||||||||||||||||||||||100.0%]   Load average: 1.06 0.39 0.35
  3  [||||||||||||||||||||||||||||||||||||||||||||100.0%]   Uptime: 00:46:38
  4  [||||||||||||||||||||||||||||||||||||||||||||100.0%]
  Mem[|||||||||||||||||                    457M/3.70G]
  Swp[                                         0K/2.00G]
```

***Note**: For getting this view, **htop** can be installed in the CentOS target server. A simple view of the stress can be seen with preinstalled **top** command also.*

3. Usage

3.1. Who can use?

This CPU chaos generation utility is designed for IT professionals, system administrators, and DevOps engineers who are responsible for ensuring the robustness and reliability of their systems. Specifically, it is intended for:

- **Site Reliability Engineers (SREs):** To test and improve the resilience of their infrastructure by identifying potential weaknesses and mitigating risks associated with CPU resource exhaustion.
- **Quality Assurance (QA) Teams:** To perform stress testing and validate the performance and stability of applications under high CPU load conditions.
- **Development Teams:** To simulate real-world scenarios and evaluate the impact of CPU stress on application behavior, allowing for proactive optimization and bug fixing.
- **IT Operations Teams:** To enhance disaster recovery plans by understanding how their systems react under extreme CPU usage, ensuring that they can maintain service continuity.
- **Cybersecurity Teams:** To simulate denial-of-service attacks and evaluate the system's response and defensive mechanisms under such conditions.
- **Academic Researchers:** To study system performance, resilience, and behavior under various stress conditions for educational and research purposes.

4. Conclusion

In today's dynamic and demanding IT environments, ensuring the resilience and robustness of systems is paramount. The CPU chaos generation utility is a critical tool for IT professionals, enabling them to simulate high-stress scenarios and identify potential vulnerabilities in their infrastructure. By leveraging this utility, teams can proactively enhance their systems' performance and reliability, ensuring uninterrupted service delivery even under extreme conditions.

Whether it is for stress testing, performance validation, disaster recovery planning, or security assessments, this utility provides a comprehensive solution for a wide range of applications. By integrating this tool into their regular testing and maintenance routines, organizations can build more resilient, reliable, and robust systems, ultimately leading to greater operational efficiency and customer satisfaction.

5. Bonus

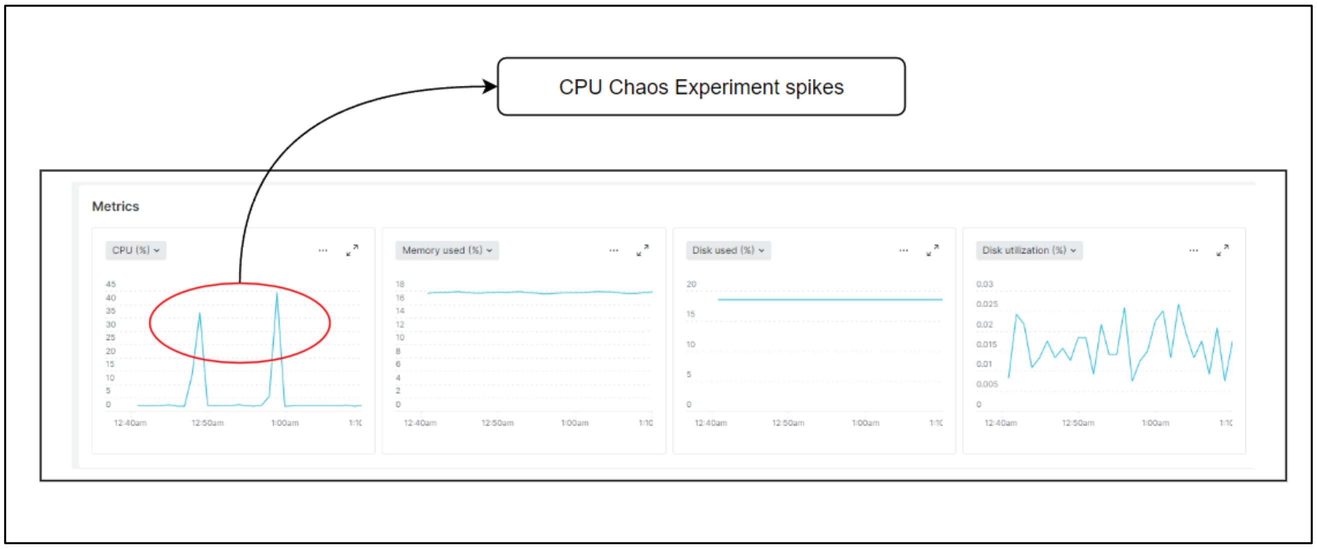
For monitoring the experiments, the **htop** or **top** utilities can be utilized. However, for enhanced interactivity and detailed insights, integrating a New Relic infrastructure agent, or any other APM tool's infrastructure agent, into the server is recommended. This allows for comprehensive monitoring and analysis of the system performance during the experiments.

For the installation of New Relic Infrastructure agent, the user needs to execute the below command in the terminal of the target server:

```
curl -Ls https://download.newrelic.com/install/newrelic-cli/scripts/install.sh | bash &&
sudo NEW_RELIC_API_KEY=<YOUR_API_KEY> NEW_RELIC_ACCOUNT_ID=<YOUR_ACCOUNT_ID>
/usr/local/bin/newrelic install
```

Replace **<YOUR_API_KEY>** and **<YOUR_ACCOUNT_ID>**, with your own.

After performing these steps, the same chaos experiments can be monitored in the New Relic's **Infrastructure** capability.


CPU Chaos Experiment spikes
Metrics
CPU (%)
Memory used (%)
Disk used (%)
Disk utilization (%)
12:40am
12:50am
1:00am