Министерство цифрового развития, связи и массовых коммуникаций
Федеральное государственное образовательное бюджетное учреждение

высшего профессионального образования

«Сибирский государственный университет телекоммуникаций и информатики» (СибГУТИ)

Лабораторная работа №6

«Разработка сценария нагрузочного тестирования с использованием Locust»

Выполнил студент:

Абрамов М.В.

группа ИП-311

Проверил:
Сентюров С.А.

Новосибирск 2025 г.

СОДЕРЖАНИЕ

1. Введение

Цель работы: Научиться разрабатывать сценарии нагрузочного тестирования с использованием **Locust**, тестируя **API OpenBMC** и открытое публичное **API**. Оценить производительность системы, проанализировать задержки и возможные точки отказа.

Задачи:

1. Установить и настроить инструмент нагрузочного тестирования **Locust**.
2. Разработать **Locust-скрипт** (locustfile.py) на **Python**, включающий сценарии для тестирования **OpenBMC API** (получение информации о системе и состояния питания).
3. Включить в сценарий тестирование публичного API (например, wttr.in).
4. Провести нагрузочное тестирование с заданными параметрами (пользователи, скорость запуска).
5. Проанализировать и представить полученные метрики (среднее время отклика, процент ошибок).

2. Теоретические сведения

Locust

Locust — это инструмент для нагрузочного тестирования, который позволяет моделировать поведение множества виртуальных пользователей (написанных на **Python**) и отправлять **HTTP-запросы** к тестируемому **API** или веб-приложению. Основные возможности включают: создание кастомных сценариев, запуск тестов с заданным числом пользователей и мониторинг времени отклика в реальном времени.

OpenBMC API (Redfish)

OpenBMC API предоставляет **Redfish-интерфейс** для удаленного управления серверным оборудованием. В рамках работы использовались запросы типа GET для получения данных о системе и ее состоянии питания.

3. Ход выполнения работы

3.1. Настройка окружения

1. Установлен инструмент **Locust** (версия 3.12.3 на Python).
2. Проверена доступность **OpenBMC API** (https://localhost:2443/redfish/v1/) с помощью **curl** с использованием учетных данных root:OpenBmc. Запрос выполнен успешно.
3. Для публичного тестирования выбран сервис **wttr.in**, доступность которого также подтверждена **curl**.
4. Проверка **JSONPlaceholder** завершилась ошибкой соединения, поэтому тестирование проводилось только с **OpenBMC** и **wttr.in**.

3.2. Разработка Locust-скрипта (locustfile.py)

Разработан скрипт с использованием двух классов для моделирования нагрузки:

1. **OpenBMCTest** (Host: https://localhost:2443):
    - **Задача:** Получение информации о системе (GET /redfish/v1/Systems/system).
    - **Задача:** Получение состояния питания (GET /redfish/v1/Systems/system), с извлечением поля PowerState из **JSON**-ответа.
    - ***Особенность:*** Для работы с **OpenBMC API** использовалась базовая аутентификация (auth=("root", "OpenBmc")) и отключение проверки **SSL-сертификата** (verify=False).
2. **WeatherTest** (Host: https://wttr.in):

- **Задача:** Получение прогноза погоды для Новосибирска (GET /Novosibirsk?format=j1).
- Ожидание между запросами (wait_time) для обоих классов установлено в диапазоне от **1 до 3 секунд**.

3.3. Запуск нагрузочного тестирования

Запуск **Locust** выполнен через команду locust, после чего тест инициирован через веб-интерфейс:

- **Количество пользователей (Number of users):** 10
- **Скорость запуска (Ramp Up):** 2 пользователя/сек
- **Общее количество пользователей:** 10, распределенных поровну (5 OpenBMCTest, 5 WeatherTest).

4. Результаты тестирования

4.1. Сводная таблица результатов (Пример запуска)

Сводные результаты, полученные в ходе нагрузочного тестирования (фрагмент статистики):

Тип	**Имя запроса**	**Запросов**	**Ошибок**	**% Ошибок**	**Медиана (мс)**	**95% персентиль (мс)**	**Среднее (мс)**
GET	/Novosibirsk?format=j1	7	7	**100%**	73	110	72.67
GET	/redfish/v1/Systems/system	6	0	**0%**	410	660	443.42
Агрегировано	**-**	13	7	53.8%	110	660	243.78
38							

4.2. Анализ результатов

1. **Тестирование OpenBMC API (https://localhost:2443)**
    - Показана **отличная стабильность** с **0% ошибок**[39]. Все 6 запросов на получение информации о системе были выполнены успешно.
    - Среднее время отклика составило **443.42 мс**[40]. **Медиана (50% запросов)** составила **410 мс**, а **95% запросов** были обработаны быстрее, чем за **660 мс**[41]. Такие показатели свидетельствуют о приемлемой производительности системы под данной нагрузкой.
2. **Тестирование публичного API (https://wttr.in)**
    - Зафиксирован **100% процент ошибок** (7 из 7 запросов)[42].
    - Низкое время отклика (**72.67 мс** в среднем) [43] при полном отсутствии успешных запросов указывает на то, что ошибка, скорее всего, связана не с производительностью, а с **некорректной обработкой ответа** или его формата скриптом **Locust**, который ожидает успешный статус-код, отличный от полученного, либо некорректно парсит ответ, не являющийся **JSON**.
3. **Общий вывод по нагрузке**
    - Из-за проблем с публичным API, общий процент ошибок составил **53.8%**[44].
    - Однако целевой объект тестирования (**OpenBMC API**) полностью справился с заданной нагрузкой (10 пользователей, 2 пользователя/сек) с **0% ошибок**, продемонстрировав хорошую доступность и скорость отклика.

5. Заключение

В ходе лабораторной работы №6 были успешно освоены навыки разработки сценариев нагрузочного тестирования с использованием фреймворка **Locust**.

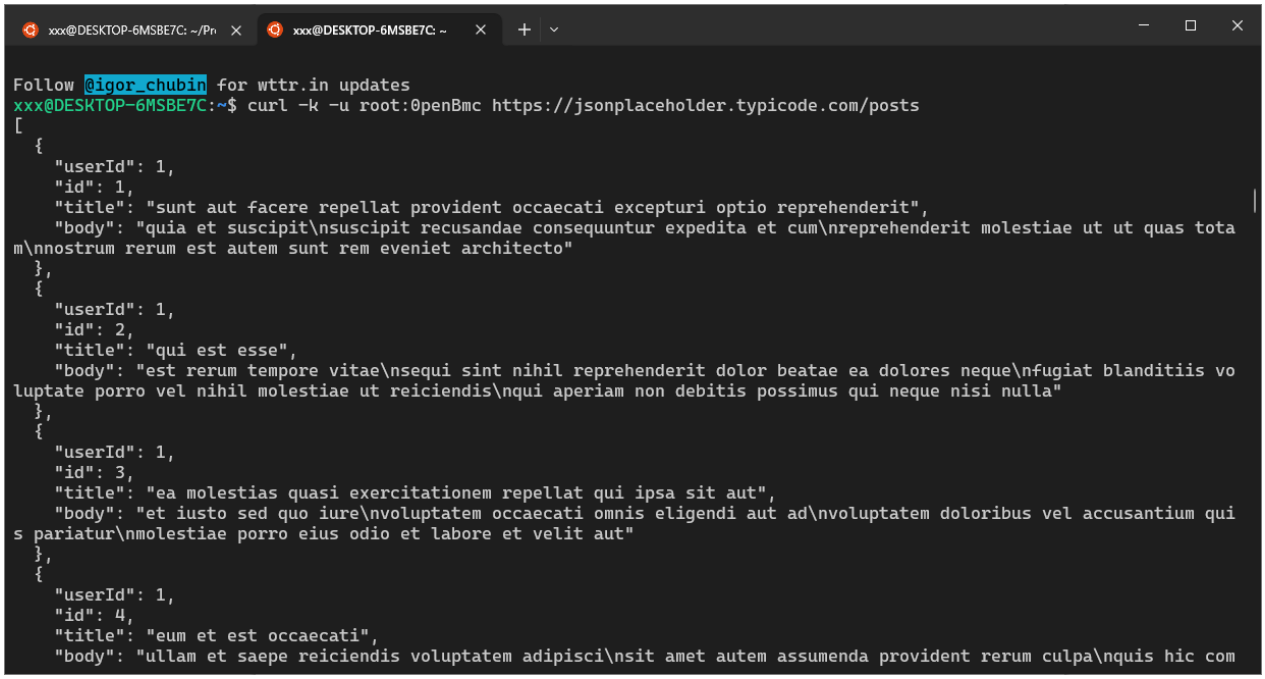
1. Был разработан скрипт на **Python** (locustfile.py), имитирующий нагрузку на две различные системы: **OpenBMC API** и открытый сервис **wttr.in**.
2. Нагрузочное тестирование **OpenBMC API** показало **высокую надежность (0% ошибок)** и **приемлемое среднее время отклика (443.42 мс)**, что подтверждает его готовность к эксплуатации.
3. Была выявлена проблема при тестировании публичного API (wttr.in), что привело к **100% ошибок** в этом сегменте.

Цели работы достигнуты: сценарий нагрузочного тестирования разработан, запуск произведен, а результаты проанализированы и представлены в отчете.

6. Приложение (Скриншоты)

(В данном разделе должны быть представлены скриншоты, подтверждающие выполнение работы)

```
(venv) xxx@DESKTOP-6MSBE7C:~/Projects/QTStudy/testPO/lab6$ locust -V
locust 2.42.4 from /home/xxx/Projects/QTStudy/testPO/lab6/venv/lib/python3.12/site-packages/locust (Python 3.12.3)
```



1. **Установка Locust и проверка доступности API** (Curl запросы к OpenBMC и wttr.in).

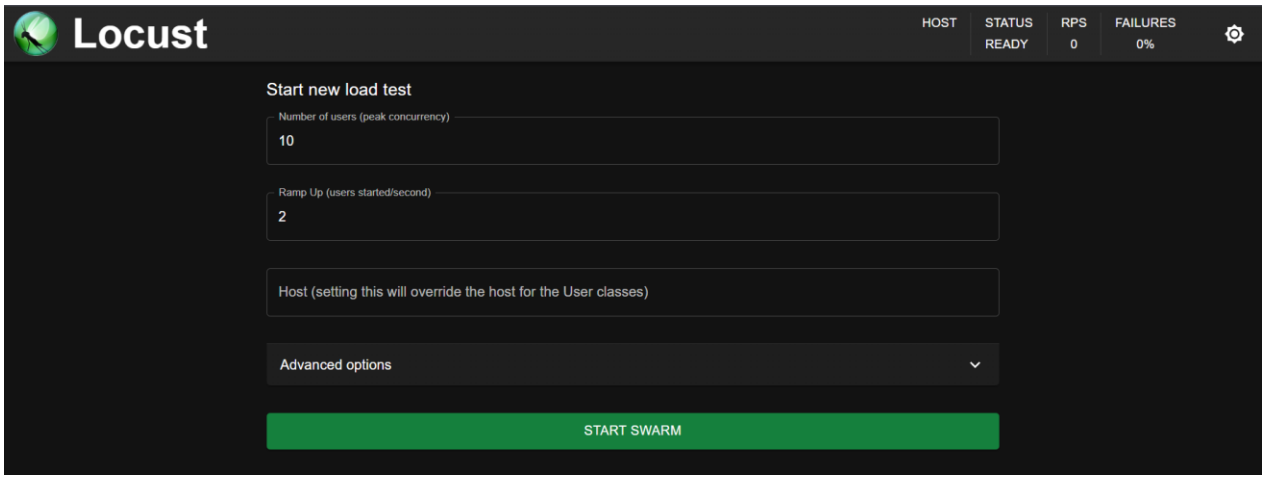
locustfile.py

Ubuntu > home > xxx > Projects > QTStudy > testPO > lab6 > locustfile.py > OpenBMCTest > get_system_info_and_power_state

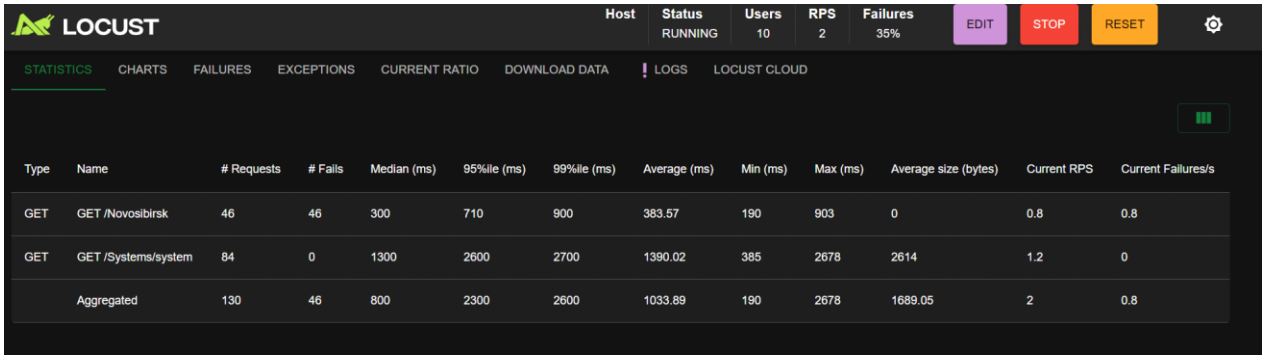
```python
from locust import HttpUser, task, between
import json


OPENBMC_HOST = "https://localhost:2443"
OPENBMC_AUTH = ("root", "0penBmc")


class OpenBMCTest(HttpUser):
    host = OPENBMC_HOST
    wait_time = between(1, 3)
    disable_known_hosts = True

    def on_start(self):
        self.client.auth = OPENBMC_AUTH
        self.client.verify = False

    @task(1)
    def get_system_info_and_power_state(self):
        """Запрос информации о системе и проверка состояния питания."""
        response = self.client.get(
            "/redfish/v1/Systems/system", name="GET /Systems/system"
        )

        response.raise_for_status()

        try:
            system_info = response.json()

            power_state = system_info.get("PowerState", "Unknown")
            print(f"System info: {response.status_code}. Power state: {power_state}")

            valid_power_states = ["On", "Off", "PoweringOn", "PoweringOff", "Unknown"]

            if power_state not in valid_power_states:

                raise Exception(
                    f"Получено недопустимое состояние питания: {power_state}"
                )

        except json.JSONDecodeError:

            raise Exception("Ошибка декодирования JSON в ответе OpenBMC.")


class WeatherTest(HttpUser):
    host = "https://wttr.in"
    wait_time = between(1, 3)

    @task(1)
    def get_novosibirsk_weather(self):
        response = self.client.get("/Novosibirsk?format=j1", name="GET /Novosibirsk")

        response.raise_for_status()

        try:
            weather_data = response.json()
            current_temp = weather_data["current_condition"][0]["temp_C"]
            print(f"Weather: {response.status_code}. Temp: {current_temp}°C")
        except (KeyError, json.JSONDecodeError) as e:

            raise Exception(f"Ошибка парсинга или структуры ответа wttr.in: {e}")


class MyLoadTest(OpenBMCTest, WeatherTest):
    pass
```

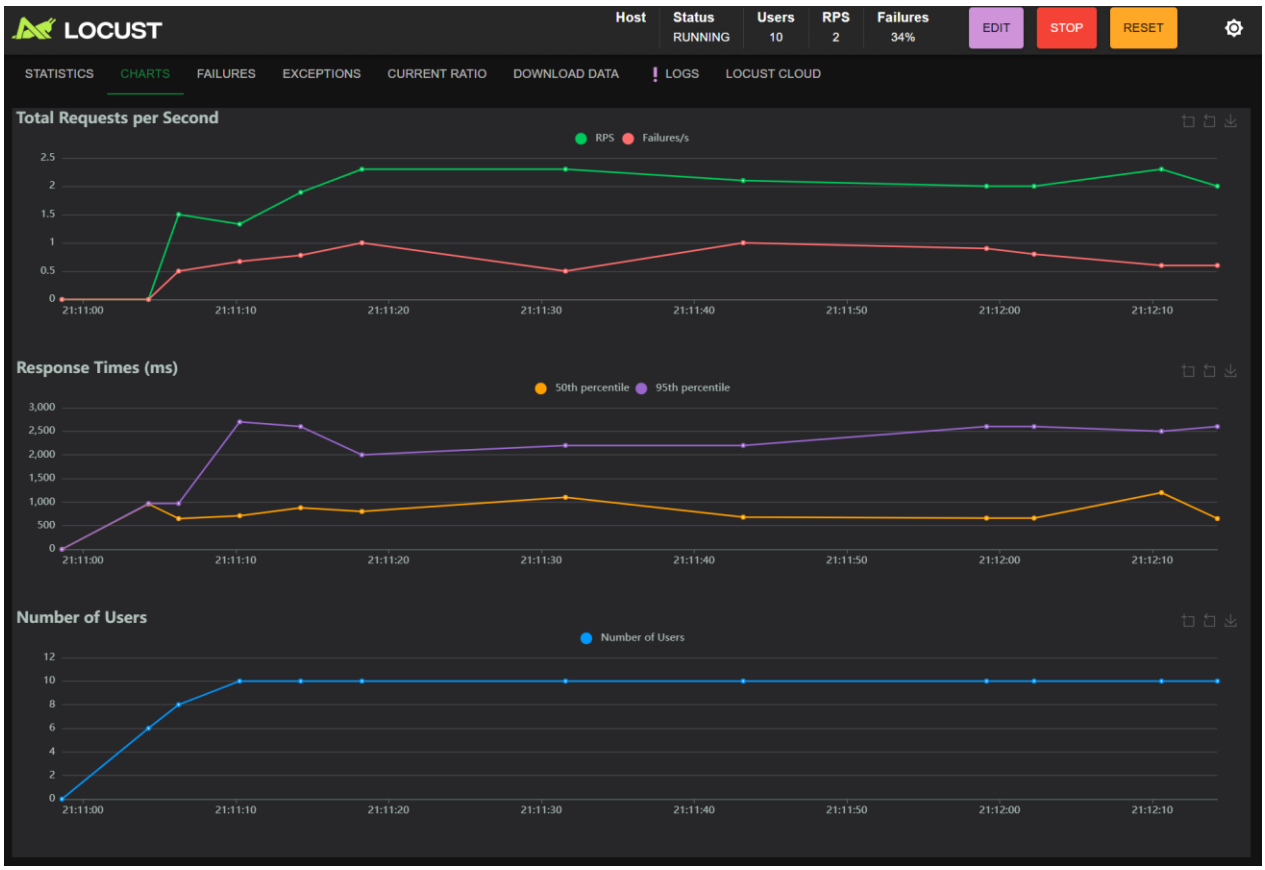
```
xxx@DESKTOP-6MSBE7C:~/Projects/QTStudy/testPO/lab6$ locust
[2025-11-20 20:57:40,403] DESKTOP-6MSBE7C/INFO/locust.main: Starting web interface at http://0.0.0.0:8089
[2025-11-20 20:57:40,414] DESKTOP-6MSBE7C/INFO/locust.main: Starting Locust
```

2. **Листинг кода locustfile.py** (Определение классов OpenBMCTest и WeatherTest).

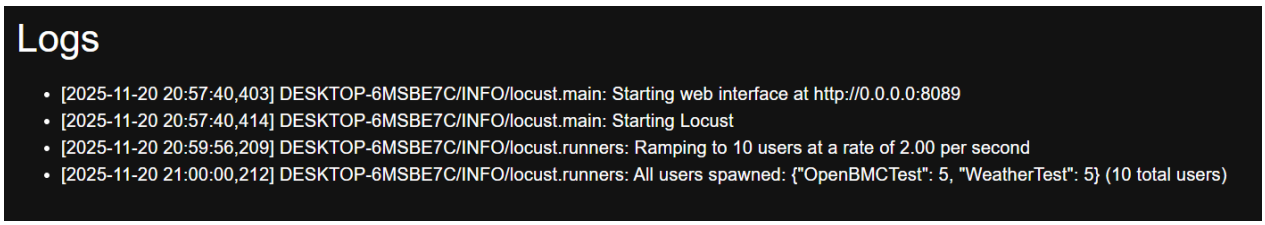


3. **Веб-интерфейс Locust** (Начало теста: 10 пользователей, Ramp Up 2).





4. **Сводная статистика Locust** (Таблица с метриками RPS, Fails, Median, Average).



5. **Логи работы Locust** (Вывод в терминале с System info: 200 и Power state: Off).