МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра математической кибернетики и компьютерных наук

ИНСТРУМЕНТ ДЛЯ НЕФУНКЦИОНАЛЬНОГО ТЕСТИРОВАНИЯ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 411 группы
направления 02.03.02 — Фундаментальная информатика и информационные технологии
факультета КНиИТ
Салыгина Ильи Юрьевича

Научный руководитель
зав. каф. техн. прогр., к. ф.-м. н. _____ И. А. Батраева

Заведующий кафедрой
к. ф.-м. н., доцент _____ С. В. Миронов

Саратов 2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Нефункциональное тестирование является важной частью процесса обеспечения качества программного обеспечения. Оно имеет ряд особенностей, которые не позволяют организовывать процесс тестирования тем же способом, что и при функциональном тестировании:

- определение исхода теста определяется путем анализа метрик (зачастую путем анализа графиков и таблиц), а не путем определения работоспособности того или иного функционала;
- анализ требует исторических данных для сравнения текущих показателей системы с предыдущими;
- особенности некоторых видов тестирования (например нагрузочного) не позволяют проводить тесты локально на машинах разработчиков.

Целью настоящей работы является создание фреймворка для нефункционального тестирования, который призван решить ряд проблем, связанных с автоматизацией процесса нефункционального тестирования. В рамках практического применения данного инструмента необходимо провести работу по его адаптации для проведения нефункционального тестирования платформы Th2 компании Exactpro.

В частности нужно решить следующие задачи:

- выбор основы для будущего инструмента,
- адаптация под задачи нефункционального тестирования,
- параметризация тестов должна осуществляться из конфигурационных файлов, а тестовые наборы зависеть от содержимого этих файлов,
- система отчетности,
- реализация механизма параллелизации процесса тестирования,

В рамках адаптации данного фреймворка для задач тестирования Th2 необходимо найти решение для следующих проблем:

- осуществление автоматического развертывания, удаления и модификации компонентов,
- параллелизация должна быть на уровне окружений Kubernetes кластера,
- должна быть возможность запуска тестов в автоматическом режиме (при выпуске новой версии того или иного компонента).

Немаловажным аспектом нефункционального тестирования является сбор, обработка и хранение информации полученной при тестировании компонен-

тов, поэтому необходимо реализовать хранилище артефактов, которые собираются в процессе тестирования. К таким артефактам относятся:

— метрики (использование памяти, загрузка центрального процессора и т. д.),
— логи, которые возникли в процессе работы компонентов, чтобы имелась возможность проанализировать их на предмет ошибок.

1 Теоретические аспекты

1.1 Процесс тестирования системы

Для того, чтобы провести нефункциональное тестирование системы необходимо:

- развернуть тестовое окружение;
- произвести манипуляции с компонентами (нагрузить, выключить и т. д.), попутно снимая нужные метрики;
- провести ручной или автоматический анализ метрик и сравнить с предыдущими тестами на предмет изменений показателей;
- сохранить результаты в хранилище.

Поскольку нефункциональные тесты включают в себя нагрузочные, то использовать для тестирования машины разработчиков или специалистов по тестированию не совсем корректно. Для того, чтобы произвести наиболее объективные тесты необходимо осуществлять тестирование в условиях, которые являются наиболее приближенными к реальной системе.

Поскольку количество тестовых окружений конечно, а количество тестов может быть большим, то для ускорения процесса тестирования необходимо иметь возможность запускать тесты параллельно. Для этих целей был разработан компонент под названием Gatekeeper. Принцип его работы заключается в организации очередности запросов на выделение изолированного тестового окружения, путем получения заявок из очереди RabbitMQ. Немаловажной особенностью данного компонента является. то, что осуществляется временная аренда окружений с необходимостью продления путем отправки сообщений о том, что тестовый процесс еще не окончен.

Поскольку с помощью данного тестового инструмента предполагается тестировать системы, которые используют Kubernetes в качестве средства для организации инфраструктуры, и на текущий момент набирает популярность подход управления конфигурацией кластера с помощью git репозитория, то было принято решение реализовать модуль управления конфигурацией через изменение содержимого репозитория и отслеживания изменений при помощи клиентской библиотеки для Kubernetes.

Поскольку современные подходы для организации мониторинга приложений (Kubernetes кластеров в частности) очень часто используется Prometheus, то было принято решение не собирать метрики путем запросов в Kubernetes

кластер, а осуществлять выгрузку метрик после проведения тестов, путем осуществления запросов на языке PromQL. Поскольку для реализации инструмента используется язык Python, то выгрузка осуществляется в pandas dataframe, что дает следующие преимущества:

- сохранение метрик от удаления Prometheus в рамках реализации retention policy;
- возможность осуществлять примитивный анализ при помощи встроенных в pandas методов (поиск максимума, минимума и т. д.);
- специалисты по машинному обучению и анализу данных очень часто используют данный формат в своей работе.

Для хранения результатов проведения тестирования было реализовано хранилище на базе MongoDB. Туда сохраняются все тестовые артефакты, которые были собраны в процессе проведения тестирования (метрики, логи и т. д.).

1.2 Технологический стек

Реализация практической части выпускной квалификационной работы была осуществлена на языке Python с использованием ряда библиотек, наиболее значимые из которых показаны ниже.

1.2.1 transitions

Данная библиотека позволяет превратить любой Python объект в конечный детерминированный автомат путем внедрения внутреннего состояния и специальных методов для управления. На ее основе был реализован компонент под названием Gatekeeper. Данный процесс производит несколько дочерних процессов, которые представляют собой конечные детерминированные автоматы, которые реагируют на сообщения из очереди RabbitMQ. Общение с брокером осуществляется при помощи протокола, основанного на AMQP.

1.2.2 kombu

Для взаимодействия с брокером RabbitMQ была использована библиотека kombu. В отличие от библиотек, которые позволяют работать с AMQP (например pika), данная библиотека предоставляет возможность сменить брокера сообщений, а также предоставляет удобный интерфейс для взаимодействия с очередями, путем реализации интерфейса SimpleQueue, который берет на

себя почти все проблемы, которые могут возникнуть в процессе использования брокеров очередей (однако из-за этого теряется гибкость и данный подход нельзя использовать там, где процесс взаимодействия с брокером является нетривиальным).

1.2.3 Kubernetes

Kubernetes предоставляет удобный API для взаимодействия с кластером, а также официальную библиотеку, которая является оберткой най REST API. На ее основе была реализована обертка, которая имитирует работу утилиты kubectl, но позволяет взаимодействовать с кластером из Python кода.

1.2.4 Pytest

Для того, чтобы не реализовывать тестовый фреймворк с нуля был взят тестовый фреймворк pytest. С его помощью были реализованы следующие механизмы:

- генерация множества тестов по базовой конфигурации (у одного теста может быть несколько наборов значений);
- автоматическое обнаружение кода тестов;
- путем подключения плагина pytest-xdist был реализован механизм параллелизации тестов в рамках одного запуска;
- динамическое подключение ресурсов для тестов (через фикстуры).

С помощью плагина pytest-json-report был реализован механизм сбора дополнительной информации из тестов (метрик и событий).

Часть функционала данного фреймворка не была использована, поскольку он предназначен прежде всего для проведения автоматизированного функционального тестирования.

1.2.5 yamldataclassconfig

Данная библиотека были использована для того, чтобы можно было гибко настраивать поведение тестов. С ее помощью был реализован механизм проверки конфигурационных файлов на корректность, а также удобный доступ при помощи атрибутов классов, а не через скобочную нотацию, которую предоставляет библиотека yaml.

1.2.6 pygit2

Данная библиотека представляет собой обертку над библиотекой libgit2, которая является легковесной реализацией утилиты git. Использование этой библиотеки позволяет запускать тестовый инструмент даже тогда, когда в системе не установлен git. Недостатком данной библиотеки можно считать то, что в данной библиотеке отсутствует реализация некоторых команд, которые есть в полноценной версии git.

2 Практическая часть

2.1 Управление тестовым окружением

Для осуществления действий с git репозиторием производится клонирование, модификация и отправка изменений обратно. Поскольку предполагается совместное использование тестового окружения несколькими тестовыми процессами, то необходимо зафиксировать некоторое базовое состояние ветки репозитория, которая относится к тестовому окружению. Для этих целей репозиторий приводится к нужному виду и зафиксированные изменения помечаются специальным тегом вида `base-branch`. Перед тем, как внести новые изменения в инфраструктурный репозиторий производится его откат до базового состояния путем последовательной нейтрализации фиксаций, которые были произведены после базовой фиксации. Данные операции производятся при помощи git revert. После нейтрализации тег переносится на новое базовое состояние.

Поскольку изменения конфигурации применяются не сразу, а с некоторой задержкой, то после отправки фиксации в удаленный репозиторий необходимо подождать нужных изменений. Набор этих изменений определяется перед отправкой в удаленный репозиторий путем получения разницы между `refs/heads/branch_name` и `refs/remotes/origin/branch_name`.

2.2 Конфигурация тестов

Одной из целей при построении тестового фрейворка была максимизация декларативного подхода к тестированию. Для этого была реализована гибкая система конфигурации, которая реализует гибридную систему конфигурирования. Процесс тестирования может быть сконфигурирован через:

- yaml файлы,
- переменные окружения,
- аргументы командной строки.

Каждый тест, помимо кода имеет еще и конфигурацию, которая определяет следующую информацию:

- наборы значений, которые используются при параметризации тестов;
- собираемые метрики,
- модификаторы для метрик.

Помимо конфигурации, которая относится непосредственно к тестам, в

конфигурационном файле также находится информация, которая используется для коммуникации с Gatekeeper, Prometheus, информация, которая необходима для подключения к инфраструктурному репозиторию и кластеру Kubernetes, а также параметры для подключения к базе MongoDB.

2.3 Сбор метрик

Как уже было сказано, сбор метрик осуществляется при помощи выгрузки данных из Prometheus, однако помимо выгрузки метрик тестовый инструмент осуществляет их обработку.

В конфигурации к тесту можно указать какие метрики необходимо собирать, а также какие манипуляции необходимо провести с ними после получения (например определить максимум, минимум и среднее значение).

После выгрузки метрик из Prometheus производится их сохранение в локальное хранилище (папку со всеми артефактами, которые были получены в ходе тестирования).

2.4 Параллелизация

Применение плагина pytest-xdist, а также компонента под названием Gatekeeper позволяет осуществить параллельное тестирование, которое обеспечивает изоляцию тестирующих процессов на всех уровнях (в процессе тестирования окружения справедливо распределяются между всеми участниками тестирования).

Параллельно тестирование может проводится несколькими специалистами по тестированию, а также путем запусков в системах непрерывной интеграции, причем каждый участник процесса тестирования также может запускать свой набор тестов в параллельном режиме, если количество окружений больше 1.

Каждый дочерний процесс, который был запущен при помощи pytest является независимым и рассматривается компонентом Gatekeeper как самостоятельная единица. Балансировка заявок по тестовым окружениям осуществляется средствами RabbitMQ, поскольку заявки скапливаются в одной очереди, а чтение производится несколькими процессами (каждый отвечает за свое окружение).

2.5 Отчетность

Для того, чтобы выводить результаты тестирования, а также историческую справку, которая хранится в репозитории с результатами была реализована система отчетности, которая отображает результаты в виде таблицы (крайний правый столбец содержит информацию о текущем запуске, а те, что левее показывают исторические значения).

Для построения таблицы используется библиотек tabulate, которая строит таблице в стиле Github Markdown. В таблице содержатся идентификаторы предыдущих запусков теста, а также сравнение полученных результатов, выраженное в процентах.

На рисунке 1 изображен пример отчета для одного из тестов.

```
--------------------------------------------------------------------------------
Meta:
  environment_id: '1'
  environment_type: Exclusive
  namespace: th2-nft

Values:
  a: 10

Events:
2021-06-10 16:38:20.612766: kuber start
2021-06-10 16:38:20.627550: env_dict: {'namespace': 'th2-nft', 'host': '127.0.0.1', 'id': '1'}
2021-06-10 16:38:25.041201: Environment acquired (id: 1)
2021-06-10 16:38:30.680167: Environment released
2021-06-10 16:38:30.680199: kuber stop

Total test runs: 13

tr-0: 60c20685b012a9a3d742acf6
tr-1: 60c207796b23ac423c9f6796
tr-2: 60c207a98b380bebea349cef

| history             |          P tr-0 |         P tr-1 |          P tr-2 | P local                |
|---------------------|-----------------|----------------|-----------------|------------------------|
| cpu_before_test     |      0.002      |      0.002     |      0.002      | 0.002 (-22.14%)        |
| ram_before_test, KB | 145888          | 146016         | 145900          | 146668.000 (+0.53%)    |
| min_ram, KB         | 145888          | 146016         | 145900          | 146668.000 (+0.53%)    |
| avg_ram             |      1.49389e+08 |      1.4952e+08 |      1.49402e+08 | 150188032.000 (+0.53%) |
| max_ram             |      1.49389e+08 |      1.4952e+08 |      1.49402e+08 | 150188032.000 (+0.53%) |
| min_cpu             |      0.002      |      0.002     |      0.002      | 0.002 (-22.14%)        |
| avg_cpu             |      0.002      |      0.002     |      0.002      | 0.002 (-21.63%)        |
| max_cpu             |      0.002      |      0.002     |      0.002      | 0.002 (-21.38%)        |

Test id: metrics-0
Test digest: 1531e72ced79135e376ba90d8af12dc9dcb125fd8a36158032b6ac0524c65236

--------------------------------------------------------------------------------
```

Рисунок 1 – Отчет

Помимо таблицы в отчет включается следующая информация:

— список событий, которые произошли в процессе проведения тестирования;

— информация о тесте (значения, которые были использованы для параметризации, а также идентификаторы);
— информация об окружении, которое было использовано.

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы был построен инструмент для нефункционального тестирования приложений на базе Kubernetes. В ходе работы были решены следующие задачи:

- сбор и анализ метрик,
- управление конфигурацией тестового окружения,
- изоляция тестирующих процессов друг от друга,
- отчетность,
- хранилище тестовых результатов.

В ходе выполнения работы были изучены следующие технологии:

- Kubernetes,
- RabbitMQ,
- Kombu,
- transitions,
- mongoengine.

Результат выпускной квалификационной работы был успешно применен для реализации процесса нефункционального тестирования платформы Th2 компании Exactpro.