Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

Кафедра математической
кибернетики и компьютерных наук

РАЗРАБОТКА ФРЕЙМВОРКА STEPLER ДЛЯ ТЕСТИРОВАНИЯ КОМПОНЕНТОВ OPENSTACK

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

Студентки 4 курса 411 группы
направления 02.03.02 — Фундаментальная информатика и информационные технологии
факультета КНиИТ
Нестеровой Алины Сергеевны

Научный руководитель
доцент, к. ф.-м. н. _____ 10.05.2017 В. Г. Самойлов

Заведующий кафедрой
к. ф.-м. н. _____ 10.05.2017 С. В. Миронов

Саратов 2017

ВВЕДЕНИЕ

Облачные вычисления — это мощный подход к проведению ресурсоемких вычислений. Он получает все большую популярность. Каждый пользователь хоть раз обращался к услугам сервисов, предоставляющих возможность работать с приложениями, не устанавливая их на компьютер.

Рынок программного обеспечения имел до недавнего времени достаточно простой вектор развития. Программисты разрабатывали приложения, которые потом распространялись традиционным образом — на носителях — и устанавливались на компьютер. Но в какой-то момент оказалось, что можно объединить вычислительные мощности для поддержки программных сервисов, аналогичных тем, которые задействуются обычными пользователями, — например, текстовые и табличные процессоры.

OpenStack [1] — это комплекс технологических проектов с открытым исходным кодом, пользующийся широкой спонсорской поддержкой обширной группы ведущих компаний отрасли. OpenStack предоставляет операционную платформу для оркестровки крупномасштабных облачных решений. Технологии OpenStack независимы от гипервизора и содержат программные средства для инициализации виртуальных машин на аппаратных средствах.

Целью проведенного исследования на производственной практике является разработка инструмента для обеспечения качества компонентов OpenStack. Для достижения поставленной цели необходимо выполнить следующие задачи:

- Разработать единый стандарт описания тестов.
- Перенести базовые проверки из различных компонентов в единый формат.
- Добавить возможность управлять процессом проверки из самого теста.
- Реализовать использование тестового фреймворка в трех последних стабильных релизах.
- Пакетирование фрэймворка в pip-пакет.
- Создать первые и единственные в своем роде тесты для проверки внешней оболочки OpenStack.
- Добавить возможность запуска тестов с помощью Docker-контейнера.

В работе две главы с теоретической и практической частями с соответствующими названиями.

1 Краткое содержание

1) Теоретическая часть

Весь первый раздел данной работы посвящен теоретической части. В первом подразделе рассматриваются виды и типы облаков, а также где они используются и как применяются на практике.

Облачные технологии — это технологии обработки данных, в которых компьютерные ресурсы предоставляются Интернет-пользователю как онлайн-сервис.

Как правило, используемый сегодня термин «облачные вычисления» (англ. cloud computing) применим для любых сервисов, которые предоставляются через сеть Интернет. Эти Интернет-услуги, также известные как «облачные сервисы», можно разделить на три основные категории:

- инфраструктура как сервис (Infrastructure as a Service, IaaS);
- платформа как сервис (Platform as a Service, PaaS);
- программное обеспечение как сервис (Software as a service, SaaS).

Следующий подраздел включает в себя основные понятия: что такое OpenStack, из каких основных компонентов он состоит, как используется и для чего применяется. Также объясняется, чем так удобна это ПО и кто его использует.

OpenStack [2] — open source (англ. с открытым исходным кодом) проект по разработке платформы, позволяющей строить частные и публичные «облака». Цель проекта — предоставление простых и удобных широкомасштабных и многофункциональных решений для любого типа «облака». Технология включает в себя серию взаимосвязанных проектов, обеспечивающих разработку многочисленных составляющих инфраструктурного решения для «облака».

Модуль OpenStack Compute (Nova) управляет «фабрикой» облачных вычислений (это базовый компонент инфраструктурных сервисов). OpenStack Compute обеспечивает активное управление виртуальными машинами с применением таких функций, как запуск, изменение размеров, приостановка, остановка и перезагрузка, посредством интеграции с набором поддерживаемых гипервизоров.

Модуль Networking (Neutron), прежде носивший имя Quan-tum, обеспечивает управление локальными сетями с поддержкой виртуальных сетей (VLAN), протокола DHCP и IP v6. Пользователи могут определять сети, под-

сети и маршрутизаторы с целью конфигурирования их внутренней топологии, а затем назначать этим сетям IP-адреса и виртуальные сети.

Модуль OpenStack Identity Management (Keystone) управляет каталогом пользователей, а также каталогом сервисов OpenStack, к которым эти пользователи могут обращаться.

Модуль OpenStack Block Storage (Cinder) управляет хранилищем блочного уровня, которое используют экземпляры Compute. Блочное хранилище хорошо подходит для сценариев со строгими требованиями к производительности (базы данных и файловые системы).

Компонент Horizon предоставляет пользовательский интерфейс для взаимодействия пользователя с другими компонентами OpenStack. На рисунке 1 представлена последняя версия этого интерфейса.

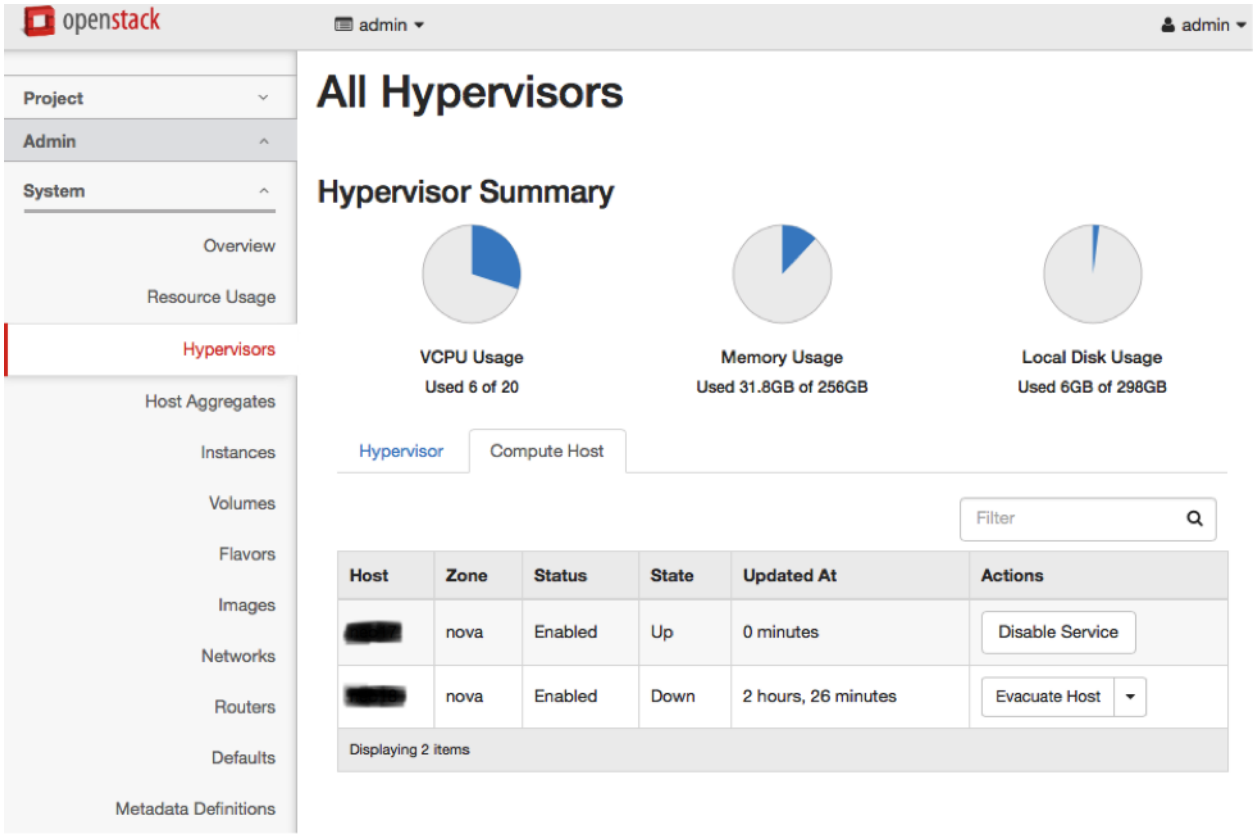

Рисунок 1 – Пользовательский интерфейс Horizon

Модуль OpenStack Image Service (Glance) обеспечивает поддержку образов виртуальных машин, в частности, системных дисков, предназначенных для использования при запуске экземпляров виртуальных машин. Помимо сервисов обнаружения, регистрации и активации данный модуль поддерживает возможности создания моментальных снимков и резервных копий.

В третьем подразделе рассказывается, что такое Docker и как он относится к разрабатываемому фреймворку. Docker [3] — программное обеспечение для автоматизации развёртывания и управления приложениями в среде виртуализации на уровне операционной системы, например LXC. Позволяет «упаковать» приложение со всем его окружением и зависимостями в контейнер, который может быть перенесен на любой Linux–системе с поддержкой cgroups в ядре, а также предоставляет среду по управлению контейнерами.

В следующем теоретическом подразделе описывается технология Ci/Cd, что она из себя представляет и как используется в фирмах с Agile-методологией.

Непрерывная интеграция (англ. Continuous Integration, далее CI) [4] — это практика разработки программного обеспечения, которая заключается в выполнении частых автоматизированных сборок проекта для скорейшего выявления и решения проблем интеграции результатов работы нескольких разработчиков. Так можно защититься от «плохого кода», часто повторяющихся багов, «кривых» слияний. CI увеличивает возможности обратной связи потому, что она позволяет следить за состоянием проекта в течение дня.

Непрерывная поставка (англ. Continuous Delivery, далее CD) [5] — это автоматизированный процесс сборки, развертывания, тестирования и выпуска. При использовании CD программное обеспечение всегда находится в готовом к выпуску состоянии. При каждом изменении в любой части системы программного обеспечения — инфраструктуре, коде приложения, конфигурации или данных — предвыпускная версия проходит через конвейер поставки, в котором создается программное обеспечение, строятся среды, составляются базы данных, выполняются тесты, и это программное обеспечение развертывается.

Следующий подраздел посвящен утилитам для работы с тестовым фреймворком: pip, pytest, tox, venv. Описаны основные команды по работе с каждой из этих утилит, и как они могут быть применены в тестовых фреймворках. Pip является установщиком пакетов Python написанный Яном Бикингом. Он может устанавливать, выводить список установленных пакетов, обновлять и удалять пакеты. Pip — это замена для easy_install.

PyTest [6] — фреймворк для запуска тестов. Он находит тесты, глядя на имена метода, класса и файла. Каждый из них должен содержать в себе слово

«test», так, например, класс должен выглядеть как TestClassName, метод — test_method_name, а файл иметь вид test_file_name.

Virtualenv [7] — инструмент, предназначенный для создания виртуального окружения для конкретного проекта. В одном проекте может быть несколько различных окружений. В нем можно изменять, устанавливать, удалять пакеты, и они не повлияют на другие проекты или системное окружение.

Tox [8] — средство автоматизации, которое позволяет запустить тесты с помощью одной команды. Чтобы работать с tox, создается конфигурационный файл с названием «tox.ini», где можно указать версии питона, что сделать перед запуском тестов, как и какие тесты запускать и что делать после запуска теста.

2) Практическая часть

В этой части рассматривается проблематика этой работы: насколько она актуальна и почему вообще возникла необходимость в написании нового фреймворка для тестирования. Также приведены основные типы тестов, которые были реализованы. В следующем подразделе рассказывается о выбранной архитектуре фреймворка и откуда, собственно, произошло его название. В третьем подразделе описываются различные способы его установки: через venv и через Docker. В четвертом подразделе рассказывается о портативности данного фреймворка: используя его, можно легко написать свой фреймворк для тестирования какого-то компонента. После установки тесты необходимо запустить — об этом следующий подраздел данной части. В последнем разделе рассматриваются отличительные особенности этого фреймворка, как происходит добавление новых тестов, изменение старых, описан процесс code review.

В следующих восьми подразделах рассказано, как были написаны и как пишутся тесты для конкретного компонента. Рассмотрены основные особенности каждого из них и приведены примеры для каждого компонента.

В последнем разделе рассматриваются другие фреймворки для тестирования OpenStack. Приведено обоснование, почему они не подходят для новых целей, и почему они тоже нужны.

Для обеспечения качества работы всех компонентов OpenStack используется различные виды тестов: модульное (или юнит), функциональное, интеграционное. В независимости от того, какой вид тестирования будет использоваться, у тестов есть недостаток — негибкость. Если программный код будет

изменен, то потребуется изменение тестов.

У каждого компонента существуют отдельные тест-планы, тест-кейсы, с помощью которых продукт становится полностью покрытым и которые гарантируют, что он работает. Но они не всегда полностью покрывают интеграцию различных сервисов. Не существует таких тестов ни в одном проекте, который бы полностью проверял всю работоспособность сервера с развернутым OpenStack на адекватность работы с точки зрения функционала, но и различные другие тесты, например, UI или destructive. Поэтому было решено создать новый фреймворк для тестирования OpenStack под названием Stepler.

Фреймворк Stepler предназначен для предоставления множества различных тестов, которые способны выполнять расширенные, нетривиальные и деструктивные сценарии, например, миграция Nova instances, перезапуск служб и сервисов, выполнение различных HA-кейсов.

Stepler предназначен не только для проверки OpenStack API; помимо этого, он представляет собой мощный, но довольно легкий в разработке и изменению кода инструмент для программиста. Он является простым в использовании и для проверки усложненных тестовых сценариев для конечных пользователей.

Архитектура основана на методологии STEPS, которая рассматривает тест как последовательность шагов, каждый из которых заканчивается проверкой, что этот шаг был завершен с ожидаемым результатом. Это позволяет составлять множество тестов с минимально возможным количеством кода. Таким образом, был разработан единый стандарт описания тестов, что являлось первой задачей разработки фреймворка. Stepler использует py.test в качестве инструмента для запуска тестов и tox для запуска более рутинных операций (например, для проверки соблюдения кодом всех правил pep8).

Архитектура фреймворка имеет следующие уровни абстракции, где расположен код (от большего к меньшему):

1. клиенты (clients) могут манипулировать ресурсами (пользователями, ролями, серверами и т.д.). Например, keystone-client, nova-client, node-client;
2. шаги (steps) — это действия, которые нужно сделать над ресурсами через клиентов: создать, удалить, обновить, показать информацию, мигрировать и т.д. Каждый шаг должен быть закончен проверкой, что он был завершен без ошибок и правильно, с ожидаемым результатом;

3. фикстуры (fixtures) управляют конструированием ресурсов, а в дальнейшем их разрушением посредством шагов;
4. тесты (tests) объединяют шаги и фикстуры в соответствии со сценарием (или тест-кейсом).

Данный тестовый фреймворк может запускаться против трех последних релизов OpenStack (Mitaka, Newton, Ocata). Помимо этого, фреймворк будет поддерживать будущие релизы, которые намечены в ближайшем будущем, и все особенности, которые будут в них присутствовать (Pike и Queens).

Проект разделен на компоненты (физически, на директории), которые минимально коммуницируют друг с другом. Главная папка `stepler` содержит в себе файл `conftest.py` и папку с фикстурами для подключения всех фикстур, описанных для всех проектов. Папка каждого компонента имеет файл `init.py` для запуска тестов, а также содержит в себе фикстуры пакетов, шаги, тесты и, возможно, `config.py`. Модули называются в соответствии с его содержимым без префикса и суффикса. То есть, фикстура с названием `cinder.py` корректна, а вот с `fixtures_cinder.py`. Все файлы с тестами находятся внутри папки `tests` с префиксом test, как это было сказано ранее.

Для сборки фреймворка Stepler лучше всего использовать Ubuntu 14.04 или 16.04. Затем необходимо скачать сам Stepler, который находится в открытом доступе на GitHub и установить необходимые требуемые пакеты. Их огромное количество, но, к счастью, все эти пакеты с требуемыми версиями можно перечислить в файле `requirements.txt`, а затем установить с помощью pip. Весь проект был имплементирован в pip-пакет, что делает его установку гораздо проще и быстрее.

После клонирования репозитория с программным кодом, необходимо перейти в только что скачанную директорию, создать виртуальное окружение, в котором будут запускаться тесты и куда нужно поставить требуемые пакеты.

Доступные компоненты для тестирования: Cinder, Glance, Heat, Horizon, Keystone, Neutron, Nova, Swift.

Если другой разработчик хочет создать свой собственный фреймворк для тестирования, но используя уже описанные в Stepler фикстуры и шаги, он может использовать его как плагин и как необходимую библиотеку в своем проекте.

Основной класс для работы с шагами находится в файле `base.py`. Класс

BaseSteps является базовым шаг-классом. Шаги-классы группируются по одному ресурсу. Это означает, что для каждого ресурса существует много унаследованных шагов-классов.

Всего существует 3 типа шагов:

1. get-steps. Цель этих шагов — получить информацию о ресурсе и вернуть ее. Если информация недоступна, то шаг должен вызвать исключение. Имя пошагового метода имеет префикс get (например, get_ips, get_flavor, get_network). Step должен вернуть что-то в последней строке кода.
2. check-steps. Цель — проверка входного ресурса и вызов исключения, если произошла какая-то ошибка. Имя метода должно иметь префикс check. Шаг не должен ничего возвращать.
3. change-steps. Их цель — сменить ресурс и проверить успешность изменений. Имя метода должно начинаться с глагола, обозначающего действие. Каждый step имеет аргумент check=True. Шаг должен производить действие над ресурсом перед проверкой. Внутри проверки step должен использовать код для возбуждения исключения, как и в get-steps. Возвращать ничего не должен, его раздел предназначен только для проверки.

Иногда (в очень редких случаях) нужно выполнить шаг без проверки check=True, например, в отрицательных тестах, когда необходимо создать сервер без имени и проверить, выводится ли ошибка. Но чаще всего используется проверка, которая гарантирует, что шаг теста был успешно завершен, и тест может переходить к следующему шагу.

Порядок автоматизации тест-кейса:

1. получить тест-кейс для автоматизации;
2. убедиться, что он еще не был автоматизирован;
3. разработать логику его работы, исходя из возможностей автоматизации и актуальности;
4. написать прототип теста с использованием шагов и фикстур;
5. внедрить или реализовать недостающие фикстуры или шаги;
6. отладить тест до рабочего состояния;
7. отправить на проверку остальным программистам.

Для того, чтобы избежать двойной работы, если пишутся различные те-

сты с одинаковой логикой, инженеры должны общаться между собой и знать, что делает каждый член его команды. Для этого используется процесс code review. Каждое изменение в коде программист должен отправить с commit-message. Оно должно включать в себя несколько строк: в первой строке пишется краткое описание того, что было сделано, а затем — подробное описание проделанной работы, чтобы все коллеги поняли, зачем данные изменения должны быть внесены. После одобрения всеми данного изменения, происходит «заливание» кода в репозиторий на Github. Ветка мастера закрыта от обычных людей, свои изменения в мастер сразу выложить нельзя.

Каждый компонент OpenStack имеет свою директорию в фреймворке, где расположены все тесты для данного компонента.

Так как данный фрейворк содержит в себе деструктивные тесты, аналогов как таковых не существует, есть инструменты, которые позволяют проверить функционал компонентов в общем потоке проверок. Среди существующих фреймворков можно выделить нескольких лидеров — Rally и Tempest.

Rally [9] — инструмент для нагрузочного тестирования OpenStack. Для запуска необходимо установить фрэймворк на мастер–ноду, девстек–хост или независимый хост с доступом к облаку.

Tempest — инструмент для валидации API, развертывания компонентов в OpenStack кластере. Все проверки включены в *.py файлы, что позволяет работать с ними, как с интеграционными тестами в продукте.

Stepler — инструмент для тестирования компонентов OpenStack. Этот фреймворк тестирует интеграцию между различными компонентами, запускает деструктивные тесты против них. Также это первый фреймворк, который тестирует внешнюю оболочку OpenStack. Он независим от хоста, на котором установлен, можно запускать на контроллере, мастер-ноде или на хосте с виртуальными машинами. Для запуска необходимо:

1. Установить фреймворк.
2. Указать значения переменных для тестов.
3. Запустить тесты.

ЗАКЛЮЧЕНИЕ

В результате выполнения данной выпускной квалификационной работы были закреплены знания о разработке инструмента для тестирования продукта, выполнен обзор существующих аналогов и технологий, и, в конечном итоге, был разработан фреймворк Stepler, который позволяет контролировать качество продукта OpenStack во всех поддерживаемых стабильных релизах.

Структура Stepler предназначена для выполнения деструктивных тестов, таких как перезапуск сервисом и различных случаев, характерных для распределенных систем, и предоставляет простой инструмент по созданию тестов для проверки различных расширенных сценариев конечных пользователей.

Были достигнуты все цели, которые были поставлены перед началом данной выпускной квалификационной работы, и получился готовый продукт, который используется в Mirantis как один из основных фреймворков для тестирования.

Разработанный фреймворк бы имплементирован на CI для проверки изменений. Использование нового инструмента позволило значительно упростить процесс написания новых тестовых сценариев и обеспечить максимальное покрытие кода продуктов тестами.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 *Foundation, O.* OpenStack Operations Guide / O. Foundation. — New York: O'Reily, 2016.

2 *Nurmi, D.* Continuous Integration / D. Nurmi. — California: The Eucalyptus Open Source Private Cloud, 2012.

3 *VMware,*. VMware vCloud Director Infrastructure Resiliency Case Study / VMware. — California: VMware, 2013.

4 Build-Deploy-Test. Непрерывная интеграция [Электронный ресурс]. — URL: https://habrahabr.ru/company/icl_services/blog/262173/ (Дата обращения 12.06.2017). Загл. с экр. Яз. рус.

5 Agile DevOps ? гибкая разработка и эксплуатация ПО: Непрерывная поставка программного обеспечения в облаке [Электронный ресурс]. — URL: https://www.ibm.com/developerworks/ru/library/a-devops8/ (Дата обращения 12.06.2017). Загл. с экр. Яз. рус.

6 Pytest Introduction [Электронный ресурс]. — URL: http://pythontesting.net/framework/pytest/pytest-introduction/ (Дата обращения 19.05.2017). Загл. с экр. Яз. англ.

7 Virtualenv Introduction [Электронный ресурс]. — URL: https://virtualenv.pypa.io/en/stable/ (Дата обращения 19.05.2017). Загл. с экр. Яз. англ.

8 What is Tox? [Электронный ресурс]. — URL: https://tox.readthedocs.io/en/latest/ (Дата обращения 18.05.2017). Загл. с экр. Яз. англ.

9 *Kurup, L.* Comparative Study of Eucalyptus, Open Stack and Nimbus / L. Kurup. — New York: International Journal of Soft Computing and Engineering, 2015.

10.06.2017