ORES
Научно-практический электронный журнал
Оригинальные исследования (ОРИС), *ISSN 2222-0402*
www.ores.su
original-research.ru

УДК 004.05

СРАВНИТЕЛЬНЫЙ АНАЛИЗ ФРЕЙМВОРКОВ ДЛЯ РАЗРАБОТКИ И РАЗВЕРТЫВАНИЯ БЕССЕРВЕРНЫХ ПРИЛОЖЕНИЙ В ЭКОСИСТЕМЕ PYTHON

Мишина Владлена Игоревна,
студент университета ИТМО, Санкт-Петербург, электронная почта: vladaa905@gmail.com

Аннотация

В статье рассмотрены основные и наиболее популярные фреймворки, используемые при разработке бессерверных веб-приложений, а также проведен их сравнительный анализ с точки зрения разработки в экосистеме Python, в частности исследовано время холодного старта для приложений, разработанных с их использованием. Сформулированы рекомендации по выбору фреймворка для конкретных случаев в зависимости от задач разработчика.

Ключевые слова: serverless, веб-приложение, Python, функции, фреймворки, облачные сервисы, Serverless framework, Zappa, Chalice, Yappa.

COMPARATIVE ANALYSIS OF FRAMEWORKS FOR DEVELOPING AND DEPLOYING SERVERLESS APPLICATIONS IN THE PYTHON ECOSYSTEM

Vladlena I. Mishina,
student of ITMO University, St. Petersburg, e-mail: vladaa905@gmail.com

ABSTRACT

The article examines the main and most popular frameworks used in the development of serverless web applications, as well as a comparative analysis of them from the point of view of development in the Python ecosystem, in particular, the cold start time for applications developed using them was examined. Recommendations for choosing a framework for specific cases are formulated depending on the developer's tasks.

Keywords: serverless, web application, Python, functions, frameworks, cloud services, Serverless framework, Zappa, Chalice, Yappa.

В постоянном поиске более быстрых, простых и экономичных способов обеспечения функциональности приложений и разработки услуг организации обращаются к микросервисам, контейнерам и бессерверным вычислениям. Эти современные подходы помогают им эффективно создавать и запускать приложения в гибридных облачных

средах, ускоряя выход на рынок и получая конкурентные преимущества. Бессерверные вычисления обеспечивают чрезвычайно высокую производительность и гибкость при масштабировании, перекладывая все бремя администрирования серверов на облачных провайдеров. Освободившись от этих задач, разработчики могут сосредоточиться на оптимизации кода и добавлении функций для своих приложений, а не на управлении инфраструктурой. В связи с этим исследование инструментов для разработки serverless веб-приложений, в частности в экосистеме одного из наиболее распространенных языков программирования Python, необходимо для определения наиболее эффективного использования данной технологии, что обуславливает актуальность работы.

Целью исследования является проведение сравнительного анализа фреймворков для разработки и развертывания бессерверных приложений, в частности, в экосистеме одного из наиболее распространенных языков программирования Python.

В целом, все фреймворки для разных языков программирования имеют одну и ту же основную функцию — ускорение разработки. Фреймворки Python для бессерверной разработки ничем не отличаются, они автоматизируют общие процессы и реализацию. Однако степень автоматизации и то, насколько ускоряется и упрощается процесс зависит от конкретного фреймворка и его особенностей. Некоторые из этих них созданы для одного сервиса, такого как AWS Lambda, или одного языка, например Python, в то время как другие могут быть использованы для нескольких платформ и языков.

Для сравнения были отобраны наиболее известные и широкоиспользуемые фреймворки для создания и развертывания бессерверных приложений на языке Python, такие как Serverless framework, Zappa, Chalice. В дополнение рассмотрен не такой популярный фреймворк, но один из немногочисленных фреймворков для Python совместимых с отечественной платформой Yandex.Cloud – Yappa.

Serverless Framework

Serverless Framework — это набор инструментов с открытым исходным кодом для развертывания и эксплуатации бессерверных архитектур. Активно поддерживается и выпущен под лицензией MIT. Платформа предоставляет конфигурационный DSL, предназначенный для бессерверных приложений, а файлы, описывающие бессерверное приложение, представляют собой текстовые файлы в формате YAML или JSON. Платформа поддерживает таких облачных провайдеров как AWS, Microsoft Azure, Apache OpenWhisk, Google Cloud, Kubeless, Spotinst и Auth0 Webtasks.

Фреймворк полностью охватывает цикл разработки и предоставляет CLI для создания и развертывания нового сервиса, вызова функций, управления плагинами фреймворка и т.п.

Самым значительным преимуществом фреймворка Serverless является поддержка плагинов. Они написаны на Node.js и получили широкое распространение в бессерверном сообществе. Самый популярный плагин serverless-offline, который локально эмулирует AWS lambda и API Gateway, был скачан более 570 тысяч раз[1].

Zappa

Zappa — опенсорсный фреймворк для Python, который поддерживает Python WSGI приложения, такие как Flask или Django, упрощая их развертывание в AWS Lambda.

Zappa выполняет множество функций в качестве инструмента развертывания, несколько наиболее заметных его преимуществ:

- упаковка проектов в zip-файлы с поддержкой Lambda и загрузка их в Amazon S3;
- настройка необходимых ролей и разрешений AWS Identity and Access Management (IAM);

- развертывание приложения на разных этапах;
- автоматическая настройка маршрутов, методов и ответов API Gateway проекта;
- преобразование запросов API Gateway проекта в WSGI и возвращение HTTP-ответов, совместимых с API Gateway[2].

Chalice

Chalice — это микрофреймворк Python для бессерверных приложений, созданный командой AWS, распространяемый под лицензией Apache License 2.0. Единственный поддерживаемый язык — Python, а AWS — единственный поддерживаемый облачный провайдер. Chalice CLI имеет функции для создания, развертывания и удаления приложения, а также для их локального запуска.

Фреймворк в том числе предлагает:

- инструмент командной строки для создания, развертывания и управления приложением;
- API на основе декоратора для интеграции с Amazon API Gateway, Amazon S3, Amazon SNS, Amazon SQS и другими сервисами AWS;
- автоматическое создание политики IAM.

Главной отличительной чертой этого микрофреймворка является то, что конфигурация определяется с помощью аннотаций в коде. Это может быть удобно для небольших и простых приложений, однако аннотации тесно связывают фреймворк с кодом[3].

Yappa

Yappa фреймворк для Python, вдохновленный Zappa. Так же как и Zappa он поддерживает WSGI/ASGI приложения (в том числе Flask, Django, FastAPI), позволяя переводить их в бессерверную среду и разворачивать наYandex Cloud (единственной поддерживаемой платформе). При развертывании Yappa:

- упаковывает проект: собирает в архив код приложения, хендлеры и зависимости из requirements.txt, загружает в s3;
- генерирует файл yappa.yaml, содержащий необходимые нстройки;
- создает версию функции, указывая на нужный хендлер;

сохраняет файл настроек API Gateway в yappa_gw.yaml, создает gateway [4].

В таблице 1 представлено сравнение вышеуказанных фреймворков.

Таблица 1 - Сравнительная таблица фреймворков для разработки и развертывания бессерверных приложений

Критерий	Serverless framework	Zappa	Chalice	Yappa
Платформы для бессерверной разработки и развертывания, которые совместимы с фреймворком	AWS, Microsoft Azure, Google Cloud Platform, Apache OpenWhisk, Cloudflare Workers, Kubeless	AWS	AWS	Yandex.cloud

Документация и поддержка сообщества разработчиков	Страница на Github (45,7 тыс. звезд) с описанием фреймворка и примерами использования, документация с руководствами и туториалами для поддерживаемых языков программирования, обширная база статей в сообществе разработчиков.	Страница на Github (10,2 тыс. звезд) с описанием фреймворка и примерами использования, документация с руководствами и туториалами, обширная база статей в сообществе разработчиков.	Страница на Github (2,9 тыс. звезд) с описанием фреймворка и примерами использования, обширная база статей в сообществе разработчиков.	Страница на Github (68 звезд) с описанием фреймворка и примерами использования, несколько статей в сообществе разработчиков, описывающих фреймворк. Был заброшен, но на данный момент поддерживается энтузиастами.
Поддержка WSGI приложений (в частности flask и django)	Есть	Есть, позволяет перенести уже готовое WSGI приложение в бессерверную среду	Есть	Есть, позволяет перенести уже готовое WSGI приложение в бессерверную среду
Функциональные возможности	Полностью охватывает цикл разработки и предоставляет CLI для создания и развертывания нового сервиса, вызова функций, управления плагинами фреймворка и т.п	Функции в качестве инструмента развертывания, в том числе упаковка проектов в zip-файлы с поддержкой Lambda и загрузка их в Amazon S3; настройка необходимых ролей и разрешений AWS Identity and Access Management	Функции для создания, развертывания, удаления, локального запуска приложения.	При развертывании упаковывает проект (архивирует код приложения, файл зависимостей и обработчики), загружает его в S3, создает версию функции, указывая нужный обработчик,

		(IAM); развертывание приложения на разных этапах; автоматическая настройка маршрутов, методов и ответов API Gateway проекта; преобразование запросов API Gateway проекта в WSGI и возвращение HTTP-ответов, совместимых с API Gateway		обновляет API Gateway. Поддерживается только подключение только к YDB.
Развертывание	Имеется поддержка автоматизации загрузки кода, упаковки и развертывания [1].	Имеется поддержка автоматизации загрузки кода, упаковки и развертывания м [2].	Имеется поддержка автоматизации загрузки кода, упаковки и развертывания [3].	Имеется поддержка автоматизации загрузки кода, упаковки и развертывания [4].

Таким образом, в результате сравнения можно сделать вывод о том, что все фреймворки имеют базовый функционал для разработки приложения, подготовки проекта и его развертывания, а также поддерживают свежие версии Python. Самым универсальным фреймворком является Serverless framework, он поддерживается большинством платформ для разработки, имеет наиболее широкий спектр функций и обширную документацию, в том числе и практические руководства от сообщества разработчиков. Zappa фреймворк и вдохновленный им российский Yappa наиболее удобны для разработки WSGI-приложений (в частности flask и django), а Chalice для быстрого входа в бессерверную разработку и небольших приложений.

В частности, был проведен эксперимент по сравнению времени «холодного» и «теплого» старта для приложений, разработанных с использованием вышеописанных фреймворков.

В контексте бессерверных вычислений «теплый старт» относится к состоянию, когда экземпляр бессерверной функции повторно используется для выполнения последующих вызовов, что значительно снижает задержку при запуске и повышает эффективность использования ресурсов. Теплый старт отличается от «холодного старта», когда для обработки входящего запроса создается новый экземпляр бессерверной функции, что приводит к более высокой задержке и потреблению ресурсов из-за процесса инициализации.

Время холодного старта является важным показателем для бессерверных приложений, поскольку оно влияет на удобство работы пользователей, производительность

и экономическую эффективность. На время холодного старта может влиять множество параметров, но в данном случае рассмотрена зависимость именно от фреймворка, использованного для разработки и развертывания.

Было разработано и развернуто 3 flask-приложения одинаковой функциональности с использованием бессерверных фреймворков.

Использовались следующие версии фреймворов: flask 3.0.0, yappa 0.4.29, zappa 0.58.0, chalice 1.2.0, serverless framework 3.0.

Разработанные приложения включали 2 пути, а функционал состоял из выдачи клиенту html-страницы с заголовком и изображением по основному пути и кодирование небольшого объекта в JSON и возвращение клиенту.

Приложения были развернуты на платформе AWS, а для мониторинга времени инициализации и холодного старта использовался сервис AWS Cloudwatch. Усредненные результаты по вызову функции представлены в таблице 2.

Таблица 2 – Сравнение времени инициализации

Использованный фреймворк	Время «холодного старта»	Время «теплого старта»
Serverless framework	512,7 мс	13,15 мс
Zappa	486,20 мс	36,8 мс
Chalice	153,70 мс	84,70 мс

Таким образом, исходя из полученных данных, можно сделать вывод о том, что наиболее эффективным с точки зрения холодного старта является приложение, написанное с использованием Chalice. Однако время инициализации для такого приложения при «теплом» старте значительно превышает показатели, полученные для других приложений. Serverless framework имеет наивысшее время холодного старта, однако и наименьшее время инициализации при «теплом» страте. Сравнимые показатели получены и для Zappa фреймворка, в сравнении с исследуемыми фреймворками он имеет среднее время как «холодного», так и «теплого» старта.

Поскольку yappa фреймворк не поддерживается AWS, а zappa, chalice, serverless framework не поддерживаются Yandex.Cloud провести с ним объективное сравнение не представляется возможным. Однако в информационных целях было измерено время «холодного» и «теплого» стартов для приложения, разработанного с использованием yappa, развернутого на Yandex.Cloud. Усредненное значение при «холодном» старте составило 967,34 мс, при «теплом» - 25,12 мс. Это наивысший показатель при «холодном» старте среди всех исследуемых фреймворков, но сравнимое время инициализации при «теплом» старте. Следует подчеркнуть, что в данном случае не известно, что и в какой степени влияет на полученные результаты, исследуемый фреймворк или платформа, на котором развернуто приложение.

В заключение, можно по результатам сравнения можно отметить следующие моменты, на которые следует обращать при выборе фреймворка для бессерверной разработки.

1. Опыт в сфере бессерверной разработки

Для новичков в области serverless разработки хорошим стартом станет использование фреймворка Chalice. Он имеет простой и лаконичный интерфейс, требует меньше кода для настройки и развертывания бессерверного приложения, автоматически создает и настраивает все необходимые ресурсы на основе конфигурации приложения.

2. Уровень сложности и специфичные особенности приложения

При выборе фреймворка следует учитывать особенности разрабтываемого приложения. Если задача состоит в переносе уже готового приложения в бессерверную

среду, то самым быстрым и эффективным решением станет Zappa (в случае использоввния платформы AWS) или Yappa (в случае развертывания в Yandex.Cloud).

В случае, когда планируется разработка приложения с нуля, следует определить насколько моментов касаемо функционала, сложности и масштаба приложения.

Для того чтобы быстро разработать и развернуть небольшое и простое приложение хорошим выбором является Chalice.

В свою очередь Zappa обеспечивает большую гибкость, настройки и расширенные функции для более сложных приложений, позволяя разработчикам точно настроить процесс развертывания в соответствии с конкретными требованиями.

Для больших и сложных проектов и обеспечения максимальной гибкости стоит обратиться к Serverless framework. Он предоставляет максимальную свободу действий и возможности по настройке параметров. Serverless framework имеет более обширную экосистему и предоставляет богатый набор плагинов и интеграций, что делает его подходящим для более сложных случаев использования, обеспечивая комплексную среду, которая позволяет разработчикам создавать сложные приложения, предлагая более высокий уровень абстракции.

3. Документация и поддержка сообщества

Serverless framework более зрелый и стабильный по сравнению, например, с Zappa. Он был разработан намного раньше и имеет больше пользователей и большую пользовательскую активность, из чего следует что доступно больше информационных источников описывающих как с ним работать, в частности уроков, обучающих программ, дискуссий, что длает его более надежным выбором.

Chalice активно поддерживается AWS, получает регулярные обновления , улучшения и имеет довольно сильную поддержку сообществом разработчиков.

В свою очередь Zappa и Yappa более специализированные фреймворки с открытым исходным кодом, имеющие меньшую базу пользователей и поддерживаемые исключительно сообществом. Ввиду этого частота обновлений, исправления ошибок зависит лишь от активности пользователей, поэтому не стабильна и не регулрна.

4. Платформа для развертывания

Необходимо определить насколько важно и в какой степени для приложения иметь возможность смены поставщика облачных услуг.

В частности, Chalice и Zappa тесно интегрированы и совместимы только с AWS и в случае перехода на другую платформу придется полностью перписывать код приложения. В свою очередь Yappa совместим только с Yandex.cloud.

Serverless framework наоборот совместим с несколькими провайдерами такими как AWS Lambda, Azure Functions, and Google Cloud Functions, что позволяет не привязываться к одной провайдеру и сохранять возможность в случае необходимости сменить провайдера, что делает приложение более портативным.

Развитием работы может стать более подробное исследование факторов, влияющих на время холодного старта, например, в зависимости от масштаба приложения, при использовании вышеописанных фреймворков.

Список литературы:

1. Serverless Framework Documentation [Электронный ресурс]. – 2023. – URL: https://www.serverless.com/framework/docs (Дата обращения: 21.06.2023)
2. Zappa - Serverless Python [Электронный ресурс]. – 2023. – URL: https://github.com/zappa/Zappa (Дата обращения: 21.06.2023)

3. Chalice [Электронный ресурс]. – 2023. – URL: https://aws.github.io/chalice/ (Дата обращения: 21.06.2023)
4. Yappa: запускаем python web-приложения. Просто. Бессерверно. В Яндекс Облаке [Электронный ресурс]. – 2024. – URL: (Дата обращения: https://habr.com/ru/articles/569674/ (21.04.2024)

References:

1. Serverless Framework Documentation [Electronic resource]. – 2024. – URL: https://www.serverless.com/framework/docs (Accessed: 21.04.2024)
2. Zappa - Serverless Python [Electronic resource]. – 2024. – URL: https://github.com/zappa/Zappa (Accessed: 21.04.2024)
3. Chalice [Electronic resource]. – 2024. – URL: https://aws.github.io/chalice/ (Accessed: 21.04.2024)
4. Yappa: start up python web-application. Simple. Serverless. In yandex cloud [Electronic resource]. – 2024. – URL: (Accessed: https://habr.com/ru/articles/569674/ (21.04.2024)