УДК 004.852

Кравчук Євгеній Сергійович
студент
Національний технічний університет України «Київський політехнічний інститут»
Магас Валентин Васильович
студент
Національний технічний університет України «Київський політехнічний інститут»

Кравчук Евгений Сергеевич
студент
Национальный технический университет Украины «Киевский политехнический институт»
Магас Валентин Васильевич
студент
Национальный технический университет Украины «Киевский политехнический институт»

Kravchuk E.
student
National Technical University of Ukraine «Kyiv Polytechnic Institute»
Mahas V.
student
National Technical University of Ukraine «Kyiv Polytechnic Institute»

МЕТОДОЛОГІЯ РОЗРОБКИ СУЧАСНИХ SAAS-ДОДАТКІВ

МЕТОДОЛОГИЯ РАЗРАБОТКИ СОВРЕМЕННЫХ SAAS-ПРИЛОЖЕНИЙ

METHODOLOGY FOR BUILDING MODERN SAAS-APPLICATIONS

Анотація. *Метою даної роботи є створення методології для розробки SaaS-додатків, яка б поєднувала у собі базові дизайн патерни cloud-native додатків та надавала середовище для їх легкого створення та підтримки.*

Ключові слова: *software-as-a-service, розгортання, паралелізм, прив'язка портів, масштабованість, хмарна платформа.*

Аннотация. *Целью данной работы является создание методологии для разработки SaaS-приложений, которая бы объединяла в себе базовые дизайн паттерны cloud-native приложений и предоставляла среду для их легкого создания и поддержки.*

Ключевые слова: *software-as-a-service, развертывание, параллелизм, привязка портов, масштабируемость, облачная платформа.*

Summary. *The aim of this research is to create a methodology for developing SaaS-applications, which would combine basic design patterns cloud-native applications and provide an environment for their easy creation and support.*

Key words: *software-as-a-service, deployment, concurrency, port binding, scalable, cloud platform.*

Вступ

Декілька десятків років тому способи та стилі розробки програмного забезпечення піддавалися впливу архітектурних стилів, популярних у ті часи. Сьогодні архітектури програмних додатків мають зовсім інший вигляд. Так, чверть століття тому, в індустрії програмного забезпечення відбувався бум вбудованого програмування. Розробка в основному велась мовою С чи, для задоволення потреб оборонних систем, мовою ADA. В таких умовах, ймовірно, найбільш комплексне завдання полягало у крос-компіляції вихідних кодів для вбудованих пристроїв і перенесенні створених бінарних файлів на них. Фактично мова про розподілені системи велася рідко, чого не можна сказати про наш час, де більшість систем є розподіленими. З огляду на це, можна сказати, що архітектури, які використовувалися у минулому, піддалися фундаментальним змінам, давши початок створенню хмарних архітектур. Наслідком цих змін є поява потреби у нових підходах до проектування, побудови, розгортання

сучасних додатків; гостра необхідність знайти рішення, котрі виходять за рамки існуючих загальноприйнятих методологій. Дана робота присвячена створенню методології, котра поєднує у собі базові патерни проектування cloud-native додатків і надає середовище, яке забезпечує їх легке створення та підтримку.

Вимоги до SaaS-додатків

У наш час програмне забезпечення зазвичай поширюється у вигляді сервісів, котрі називаються веб-додатками або *software-as-a-service* (SaaS). В процесі популяризації SaaS до цього типу додатків сформували Залежності вимог:

- Використовують декларативний формат опису процесу встановленні та налаштування, що зводить до мінімуму витрати часу та ресурсів для нових розробників, задіяних у проекті;
- Мають зв'язок з операційною системою, що передбачає максимальну переносимість між середовищами виконання;
- Узгоджуються з сучасними хмарними платформами, забезпечуючи легке розгортування, виключаючи необхідність в серверах та системному адмініструванні;
- Мінімізують розбіжність між середовищем розробки і середовищем виконання, що дозволяє використовувати неперервне розгортання для максимальної гнучкості;
- Можуть масштабуватися без суттєвих змін в інструментах, архітектурі та способі розробки.

Ключові аспекти методології

Кодова база

Хорошим тоном вважається відслідковування додатку в системі контролю версій, такій як Git, Mercurial чи Subversion. Копію бази коду, що відслідковується прийнято називати репозиторієм коду.

Кодова база — це один (в централізованих системах контролю версій, як Subversion) чи кілька репозиторіїв, що мають спільні початкові коміти (в децентралізованих системах контролю версій, як Git).

Характерно, що завжди є чітка відповідність між кодовою базою та додатком:

- Якщо є декілька кодових баз, то це не додаток, а розподілена система. Кожний компонент в розподіленій системі є додатком і може індивідуально відповідати ключовим аспектам методології, що пропонується.
- Факт того, що декілька додатків спільно використовують той же код, є порушенням принципів методології. Вирішенням даної ситуації є виділення спільного коду в бібліотеки, які можуть бути підключені через менеджер залежностей.

Таким чином, кодова база повинна бути єдиною для всіх розгортувань, проте різні версії однієї кодової бази можуть виконуватися в кожному з розгортувань. Наприклад, розробник може мати деякі зміни, які ще не додані в проміжне розгортування; проміжне розсортування може мати деякі зміни, які ще не були додані в кінцеве розгортування.

Однак, всі ці розгортування використовують одну і ту ж кодову базу, таким чином можна їх ідентифікувати як різні розгортування одного і того ж додатку.

Залежності та інструменти системи

Додаток ніколи не повинен мати неявні залежності. Він декларує всі свої залежності повністю і точно за допомогою маніфест декларації залежностей. Крім цього, додаток використовує інструмент ізоляції залежностей під час виконання для забезпечення того, що неявні залежності не «проникли» з оточуючої системи. Повна і явна специфікація залежностей застосовується рівним чином як при розробці, так і при виконанні додатку.

Наприклад, Gem Bundler в Ruby використовує Gemfile як формат маніфеста для оголошення залежностей і bundle exec — для ізоляції залежностей. Python має два різних інструмента для цих задач: Pip використовується для оголошення і Virtualenv — для ізоляції.

Незалежно від того, який інструмент використовується, декларування та ізоляція залежностей повинні завжди використовуватися разом — тільки одного з них недостатньо.

Однією з переваг явного оголошення залежностей є те, що це спрощує налаштування додатку для нових розробників.

Конфігурація

Конфігурація додатку — це все, що може змінюватися між розгортуваннями (середовище розробки, проміжні та робочі розгортування). Конфігурація включає в себе:

- Ідентифікатори підключення до ресурсів типу бази даних, кеш-пам'яті чи другим стороннім службам;
- Реєстраційні дані для підключення до зовнішніх сервісів, наприклад Amazon S3 чи Twitter;
- Значення залежні від середовища розгортування так, як канонічне ім'я хоста;

Дана методологія потребує строгого розділення конфігурації та коду. Конфігурація може відрізнятися між розгортуваннями, код — не повинен. Це дозволить в будь-який час відкрити доступ до кодової бази без компрометації будь-яких приватних даних.

Прив'язка портів

Інколи веб-додатки запускаються всередині контейнера веб-сервера. Наприклад, PHP-додаток може

бути запущений як модуль всередині Apache HTTPD, чи Java-додаток може бути запущений всередині Tomcat.

Додаток, побудований за пропонованою методологією, є повністю самодостатнім і не покладається на ін'єкцію від веб-сервера під час виконання для того, щоб створити веб-сервіс. Веб-додаток експортує HTTP-сервіс шляхом прив'язки до порту і прослуховування запитів, що на нього надходять.

Під час локальної розробки розробник переходить по URL-адресі вигляду http://localhost:5000/, щоб отримати доступ до сервера, що надається додатком.

При розгортуванні шар маршрутизації опрацьовує запити до загальнодоступного хосту і перенапрявляє їх до прив'язаного до порту веб-додатку. Це переважно реалізується за допомогою оголошення залежностей для додавання бібліотеки веб-сервера до додатку.

Характерно, що підхід прив'язки до порту означає, що один додаток може виступати сторонньою службою для іншого шляхом надання URL-адреси стороннього додатку як ідентифікатор ресурсу в конфігурації споживаючого додатку.

Паралелізм

Будь-яка комп'ютерна програма після запуску являє собою один чи декілька працюючих процесів. Історично веб-додатки приймали різноманітні форми виконання процесів. PHP-процеси виконуються як дочірні процеси Apache і запускаються при потребі в необхідній для обслуговування отриманих запитів кількості.

Java-процеси використовують протилежний підхід, JVM являє собою один монолітний мета-процес, який резервує великий об'єм системних ресурсів при запуску і керує паралельністю всередині себе за допомогою потоків. В обох випадках запущені процеси тільки мінімально видимі для розробника додатку.

В додатку, створеному за даною методологією, процеси є сутностями першого класу. Процеси взяли сильні сторони з моделі процесів Unix для запуску демонів. За допомогою цієї моделі розробник може спроектувати свій додаток таким чином, що для обробки різного робочого навантаження необхідно призначити кожному типу роботи свій тип процесу.

Таким чином, процеси в додатках створених за пропонованою методологією ніколи не повинні демонізуватися и записувати PID файли. Натомість, вони повинні покладатися на менеджер процесів операційної системи (наприклад, Upstart, розподілений менеджер процесів на хмарній платформі) для керування потоками виведення чи реагування на падіння процесу.

Збірка, реліз, виконання

Кодова база трансформується в розгортування (без врахування розгортування для розробки) за три етапи:

- Етап збірки — це трансформація, яка перетворює репозиторій коду у виконуваний пакет, який називають збіркою. Використовуючи версію коду за вказаним процесом розгортання коміту, етап збірки завантажує сторонні залежності і компілює двійкові файли і ресурси.
- Етап релізу — приймає збірку, отриману на етапі збірки, і об'єднує її з поточною конфігурацією розгортування. Отриманий реліз містить збірку і конфігурацію та готовий до негайного запуску в середі виконання.
- Етап виконання (також відомий як «runtime») запускає додаток в середі виконання шляхом запуску деякого набору процесів додатку з певного релізу.

Важливим є те, що додаток, побудований за даною методологією, використовує строгий розподіл між етапами збірки, релізу та виконання. Наприклад, неможливо внести зміни в код під час виконання, адже немає способу поширити ці зміни назад на етапі збірки.

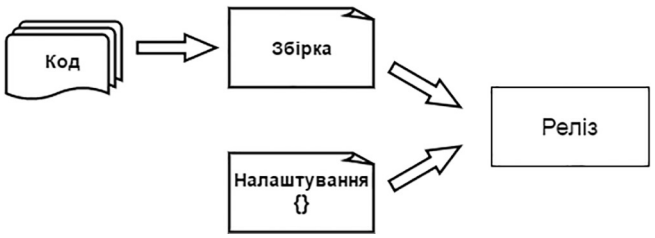

Рисунок 1.

Варто наголосити, що збірка ініціалізується розробником додатку кожен раз, коли розгортається новий код. Запуск етапу виконання, навпаки, може відбуватися автоматично в таких випадках, як перезавантаження сервера, чи перезапуск впавшого процесу менеджером процесів. Таким чином, етап виконання повинен бути якомога більш технічно простим, так як проблеми, які можуть завадити додатку завантажитись, можуть виникнути в час, коли немає доступних розробників.

Висновок

В даній роботі була запропонована методологія, ключові аспекти якої дозволили б створювати SaaS-додатки з урахуванням основних вимог до їх побудови у контексті сучасних вимог, котрі диктує індустрія. Характерно, що запропонована методологія може бути застосована для додатків, написаних будь-якою мовою програмування з використанням різних комбінацій сторонніх служб, як-то бази даних, черги повідомлень, механізми кешування тощо. Запропонована методологія охоплює загальноприйняті підходи до розробки програмного забезпечення, враховуючи сучасні тенденції, притаманні кожному етапу життєвого циклу додатку.