Чи є подібні системи, які вже реалізовані, так на даний момент вже є ряд систем, які вже працюють, проходять бетатестування чи тільки анансовані:

- Playstation Now;
- Geforce Now;
- Google Stadia;
- Microsoft Project xCloud (робоча назва);
- Drova;
- Shadow;
- Vortex.

ПЕРЕЛІК ПОСИЛАНЬ:

1. Nintendo Famdom, Transfer Pak //URL: https://nintendo. fandom.com/wiki/Transfer_Pak .
2. Wikipedia, Сьоме покоління ігрових систем //URL: https://uk.wikipedia.org/wiki/Сьоме_покоління_ігрових_систем (або https://uk.wikipedia.org/wiki/%D0%A1%D1%8C%D0%BE%D0%BC%D0%B5_%D0%BF%D0%BE%D0%BA%D0%BE%D0%BB%D1%96%D0%BD%D0%BD%D1%8F_%D1%96%D0%B3%D1%80%D0%BE%D0%B2%D0%B8%D1%85_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC).
3. Wikipedia, PowerPC //URL: https://uk.wikipedia.org/wiki/PowerPC
4. Wikipedia, Cell //URL: https://uk.wikipedia.org/wiki/Cell_(процесор) (або https://uk.wikipedia.org/wiki/Cell_(%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D0%BE%D1%80))

УДК 004.3

DEPENDENCY INJECTION КОНТЕЙНЕР ТА ВПРОВАДЖЕННЯ ЙОГО В СУЧАСНІ WEB-ДОДАТКИ

К.Ю. Островська, О.В. Захарченко
(Україна, Дніпро, Національна металургійна академія України)

Постановка проблеми. Впровадження залежності (Dependency injection, DI) - процес надання зовнішньої залежності програмного компоненту. Є специфічною формою «інверсії управління» (Inversion of control, IoC), коли вона застосовується до управління залежностями. У повній відповідності з принципом єдиних обов'язків об'єкт віддає піклуватися про побудову необхідних йому залежностей зовнішньому, спеціально призначеному для цього спільного механізму.

При цьому впровадження залежностей об'єкт пасивний і не вживає взагалі ніяких кроків для з'ясування залежностей, а надає для цього сеттери і / або приймає своїм конструктором аргументи, за допомогою яких впроваджуються залежності.

Принцип роботи. Робота фреймворка, що забезпечує впровадження залежності, описується наступним чином. Додаток, незалежно від оформлення, виповнюється всередині контейнера IoC, що надається фреймворком. Частина об'єктів в програмі як і раніше створюється звичайним способом мови програмування, частина створюється контейнером на основі наданої йому конфігурації.

Умовно, якщо об'єкту потрібно отримати доступ до певного сервісу, об'єкт бере на себе обов'язок щодо доступу до цього сервісу: він або отримує пряме посилання на місцезнаходження сервісу, або звертається до відомого «сервіс-локатор» і запитує посилання на реалізацію певного типу сервісу. Використовуючи ж впровадження залежності, об'єкт просто надає властивість, яке в змозі зберігати посилання на потрібний тип сервісу; і коли об'єкт створюється, посилання на реалізацію потрібного типу сервісу автоматично вставляється в це властивість (поле), використовуючи кошти середовища.

Впровадження залежності більш гнучко, тому що стає легше створювати альтернативні реалізації даного типу сервісу, а потім вказувати, яка саме реалізація повинна бути використана, наприклад, файли конфігурації, без змін в об'єктах, які цей сервіс використовують. Це особливо корисно в юніт-тестуванні, тому що вставити реалізацію «заглушки» сервісу в тестований об'єкт дуже просто.

З іншого боку, зайве використання впровадження залежностей може зробити програми більш складними і важкими в супроводі: так як для розуміння поведінки програми програмісту необхідно дивитися не тільки в вихідний код, а ще й в конфігурацію, а конфігурація, як правило, невидима для IDE, які підтримують аналіз посилань і рефакторинг, якщо явно не зазначена підтримка фреймворків з впровадженнями залежностей.

У web-додатках JavaScript практично безальтернативно займає своє місце на фронті, в браузері. На серверній стороні щільно окопалися Java, PHP, .Net, Ruby, Python. Але з появою nodejs JavaScript також проник і на сервер. А технології, які використовуються в інших мовах, в тому числі і DI, почали проникати в серверний JavaScript.

Розвиток JavaScript обумовлено асинхронні роботи коду в браузері. Асинхронність не є винятковою особливістю JavaScript, швидше за вродженої. Зараз наявність JavaScript і на сервері, і на фронті вже нікого не дивує, а скоріше, стимулює до використання одних і тих же підходів на обох "кінцях" web-додатку. І одного і того ж коду.

Висновки. Обробка залежностей без DI можлива, але це може привести до збоїв роботи програми. DI - це просто ефективна ідея, згідно з якою можливо обробляти залежності поза залежного класу. Найефективніше використовувати DI в певних частинах програми. Багато фреймворків цьому сприяють. Фреймворки і бібліотеки не потрібні для DI, але можуть багато в чому допомогти.

ПЕРЕЛІК ПОСИЛАНЬ:

1. Inversion of Control Containers and the Dependency Injection pattern (https://www.martinfowler.com/articles/injection.html).

2. Java for fun: Что такое Dependency injection, Inversion of Control и почему это возникло. Часть #1(http://www.apofig.com/2010/08/dependency-injection-inversion-of.html).

УДК 004.94

ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ З ВИКОРИСТАННЯМ VREALIZE AUTOMATION

С.О. Смолянов, І.С. Дмитрієва
(Україна, Дніпро, Національна металургійна академія України)

Постановка проблеми. Хмарні рішення пропонують компаніям ряд переваг, такі як: безпека, гнучкість на рівні ПО і апаратного забезпечення, надійність, автоматизація роботи IT персоналу, легкість доступу. vRealize Suite, платформа управління VMWare - це абсолютно новий спосіб розширення центру обробки даних до хмарного сховища за допомогою уніфікованих засобів управління:

1. Automation - автоматичне управління життєвим циклом інфраструктури, послуг і сервісів;
2. Orchestrator - це інструмент для управління IT інфраструктурою, її адміністрування;
3. Operations - попереджуюче управління продуктивністю, використання ресурсів і журналами подій;
4. Business insight - узгодження витрат на IT, забезпечення повної прозорості витрат;
5. Unified management - використання єдиної платформи для надання та управління додатками та інфраструктурою.

Мета роботи полягала у вирішенні таких IT завдань:

1. Автоматизація процесу створення, налаштування і реконфігурації віртуальних машин під потреби різних додатків;
2. Надання більшості хмарних сервісів і рішень за заниженою вартістю (в порівнянні з AWS і Microsoft Azure).

Було розгорнуто кластер vRealize Automation, реалізовано підключення декількох vCenter як endpoint, налагодження процесу створення і кастомізації ОС. Були створені сервіси по розвороту готових додатків і баз даних. Автоматизований процес реєстрації віртуальних машин в Identity Manager і створення політик доступів за допомогою API. Так само було створено ряд сервісів по реконфігурації і донастройки віртуальних хостів. Підключений VMware vRealize Business for Cloud для розрахунку вартості обчислювальних