УДК 004.046: 004.896: 004.942: 004.021:042
DOI: 10.20535/SRIT.2308-8893.2017.3.04

ОСОБЛИВОСТІ РЕАЛІЗАЦІЇ СЕРВІС-ОРІЄНТОВАНИХ ДОДАТКІВ У ХМАРІ

О.О. ПЕТРЕНКО

Анотація: Найбільший європейський проект зі створення Європейської відкритої науково-дослідницької хмари (European Open Science Cloud for Research, EOSC), що розпочався в 2017 р. і базується на сервіс-орієнтованому підході, мотивує дослідження технологій розміщення сервіс-орієнтованих структур (SOA) прикладних додатків у хмарі. Досліджено базові відмінності традиційних SOA першого покоління (на основі веб-сервісів з уніфікованими протоколами зв'язку) і хмарних SOA нового покоління (на основі мікросервісів з контейнерами), які необхідно враховувати під час переміщення SOA застосувань у хмару.

Ключові слова: SOA, мікросервіс, контейнер, хмара, API, ESB, Workflows, EOSC.

ВСТУП

Підвищення складності сучасних інформаційних систем, включаючи грід (хмарні) інфраструктури, зумовило поширення модульного підходу до розроблення їх програмного забезпечення з використанням стандартизованих по можливості інтерфейсів між частинами, як це передбачено концепцією SOA. Ця архітектура має вигляд набору сервісів і процесів, які можна комбінувати, а також змінювати з часом відповідно до змін вимог за допомогою планувальників потоку завдань (workflows). У широкому сенсі SOA — це підхід до розроблення додатків, відповідно до якого додаток розщеплюється на окремі частини. Ці частини, як правило, розподілені по всій системі і спілкуються одна з одною через мережу. Функціональність SOA найпростіше реалізується за веб-сервісами (службовими) з використанням стандартів ***WSDL*** (Web Services Description Language) і ***SOAP*** *(*Simple Object Access Protocol) для опису інтерфейсів та формату повідомлень, що приймаються або посилаються [1–3]. Для забезпечення комунікації та інтеграції великомасштабних гетерогенних прикладних процесів найбільш зручним є використання сервісної шини підприємства ***ESB*** (Enterprise Service Bus), яка діє як проміжний шар (або посередник) і використовується для інтеграції, оркестровки, маршрутизації, моніторингу оброблення подій і кореляції ділової активності [5]. Веб-сервіси — це технологічні специфікації, тоді як SOA є принципом проектування архітектури програмних систем, а ESB — архітектурним шаблоном.

Додатки, які мають бути розміщеними в хмарі у зв'язку зі створенням Європейської відкритої науково-дослідницької хмари EOS [6], значно змінюють традиційні правила, за якими побудовано SOA першого покоління. Щоб максимізувати переваги хмарних технологій завдяки використанню віртуалізації апаратних і загальносистемних ресурсів, необхідно змінити

моделі сервісних додатків і оптимізувати хмару відповідно до цих змін. Починаючи з 2017 р., ***мікросервіси*** в поєднанні ***з контейнерами*** дозволили пришвидшити розроблення сервісних хмарних додатків і підвищити ефективність їх розгортання; забезпечити переміщення сервісів і їх перезапуск в умовах відмови, а також масштабування сервісів зі зміною навантаження [7].

Мікросервіси є додатками з однією функцією, як правило, невеликими за розміром, набагато меншими, ніж традиційні компоненти SOA програмних додатків, і доступними за допомогою простих *RESTful HTTP* або *JSON* інтерфейсів. Це ідеальний варіант, особливо для мобільних пристроїв та інтернету речей (IoT), двох з основних джерел і драйверів для мікросервісів. ***API Swagger***, імовірно, стає стандартом за замовчуванням для визначення, реалізації, виявлення і тестування REST сервісів.

Контейнер (докер) є програмним забезпеченням для автоматизації розгортання і керування додатками в середовищі віртуалізації на рівні операційної системи (ОС). Він дозволяє «упакувати» додаток з усім його оточенням і залежностями в контейнер, який може бути перенесений на будь-яку Linux-систему, а також надає середовище для керування контейнерами. Контейнеризація, по суті, реалізується на рівні віртуалізації ОС (на відміну від віртуальних машин), кожна з яких оснащена повною вбудованою ОС). Кілька контейнерів можуть бути розміщені в одній віртуальній машині. Контейнери та мікросервіси не одне й те саме. Мікросервіс може працювати в контейнері, але він також може працювати і на виділеній віртуальній машині. Проте контейнери є надійним способом розроблення і розгортання мікросервісів, згрупованих в певні композиції, а інструменти і платформи для запуску контейнерів є надійним способом для керування мікросервісними додатками.

У роботі наведено результати досліджень базових технологій і напрацювань хмарних SOA, розглянуто мікросервіси і API, контейнери і SOA, а також описано дійсний стан підтримання сервісного забезпечення європейської відкритої наукової хмари і сформовано висновки та передбачення.

МІКРОСЕРВІСИ ТА API

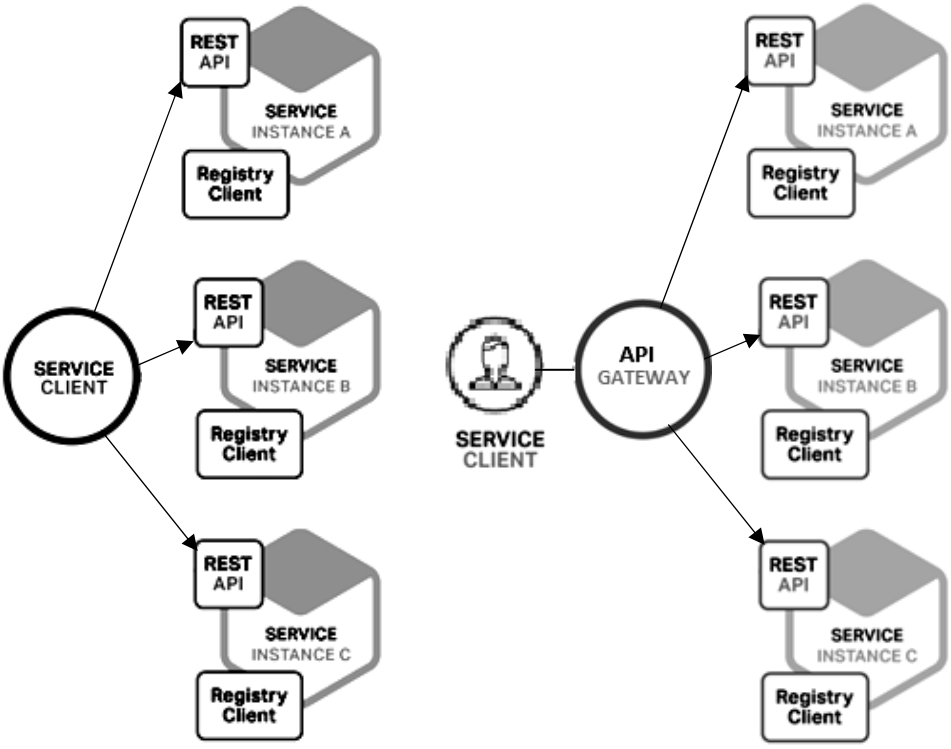
Нині розпочалася епоха мікросервісів. Сервіси реалізують обмежений набір функцій, вони розробляються, розгортаються і масштабуються незалежно, мають власний життєвий цикл, не ставлять жодних умов до засобів їх опису або моделювання (наприклад, форми веб-сервісів). Як результат менше часу витрачається на досягнення результатів і підвищення гнучкості.

Крім переваг мікросервісів, вони створюють також деякі складності, а саме: потребують інтеграції, засобів автоматизації розгортання і конфігурації, реєстрації та моніторингу, додають значно збільшеного обсягу комунікаційних процедур.

Мікросервісний архітектурний стиль являє собою підхід до розроблення єдиного додатка як набору невеликих мікросервісів, кожний з яких працює у власному процесі і спілкується з іншими через сервісний шлюз ***API Gateway*** за допомогою HTTP ресурсів API. Кожен мікросервіс зазвичай розташований в окремій віртуальній машині або Docker-контейнері для забезпечення необхідної ізоляції і самостійно розгортається за допомогою повні-

стю автоматизованого механізму розгортання. Мікросервіси можуть бути написані різними мовами програмування і використовувати різні типи технологій зберігання даних. Щоб збільшити доступність до них, можна використати кілька екземплярів мікросервісу разом з балансувальником навантаження. У цій ситуації кожен мікросервіс матиме власну базу даних або можна зробити всі екземпляри мікросервісу одного типу, зв'язаного із загальною базою даних. Мікросервіси зберігають свій внутрішній стан у власних базах даних. Для зміни мікросервісу може знадобитися оновлення схеми бази даних: таблиці мають бути оновлені для нової схеми.

Є два можливі способи організації взаємодії клієнта з цим мікросервісом: прямий зв'язок клієнт–мікросервіс і зв'язок через API Gateway (рис. 1). У першому випадку (рис. 1, *а*) клієнт зв'язується з кожним мікросервісом окремо, тобто клієнт має бути запрограмований на реалізацію індивідуального механізму зв'язку з кожним мікросервісом. Це може зробити програму клієнта більш громіздкою і складною в обслуговуванні. Такі поцедури, як аутентифікація, моніторинг та інші також потрібно обробляти кожним мікросервісом окремо, підвищуючи їх складність. Оновлення API мікросервісу потребує поновлення клієнтської програми та підтримання синхронізації між API. Оскільки кожен мікросервіс має бути оприлюднений, то ускладнюється підтримання безпеки.



а *б*

Рис. 1. Взаємодія клієнта з мікросервісом: пряма (*а*) і через API Gateway (*б*)

У другому випадку (рис. 1, *б*) API шлюз є єдиною точкою входу в систему, побудовану на базі мікросервісів. Замість того, щоб клієнт відправляв запити кожному мікросервісу, він спілкується з API Gateway. API шлюз, у свою чергу, переформовує і пересилає кожен запит на відповідний мікро-

сервіс та повертає його відповідь клієнту. Використання цього підходу дозволяє зробити публічно відкритим тільки API шлюз.

API керування стає важливим у багатьох галузях, незалежно від того, чи то комунікації бізнес для бізнесу (B2B), чи бізнес для клієнта (B2C). Широко відомі такі засоби API керування: *TIBCO API Exchange, IBM, Apigee, 3scale, WSO2, MuleSoft, Mashery, Layer 7, Vordel* [10]. Для API потрібні складні портали, на яких розробники додатків могли би знаходити й експериментувати з цими програмними інтерфейсами. Оскільки програмні інтерфейси виставляються публічно, то шлюзи, через які надається доступ, повинні забезпечувати найвищий рівень безпеки.

Сервісний шлюз (service gateway) використовується для забезпечення безпеки, дотримання політик, подання мікросервісів через API зовнішнім споживачам і статичного оброблення відповідей. З ним зв'язані засоби, побудовані для аутентифікації, моніторингу, балансування навантаження, кешування, формування і керування запитами, а також інтеграції, оркестровки, маршрутизації, моніторингу, оброблення і кореляції подій, ділової активності. Можна назвати ці засоби псевдо-ESB, або платформою інтеграції чи мікросервісною платформою. Наявність шлюзу API дозволяє кожному мікросервісу використовувати механізм зв'язку, який підходить до нього, без внесення будь-яких змін до клієнта. Різні механізми зв'язку можуть бути змішані і підібрані для мікросервісів відповідно до вимог його функціональності, наприклад, для синхронного зв'язку «один-до-одного» чи асинхронного зв'язку «один-до-багатьох».

Сервісний шлюз керує сервісами інтеграції (побудованими за допомогою псевдо-ESB), прикладними сервісами (побудованими за допомогою псевдо-ESB або будь-якої іншої технології), а також зовнішніми хмарними сервісами (не важливо, як вони побудовані, важливо, що є домовленість про постачання сервісів).

Щоб корелювати події, які відбуваються в різних мікросервісах, необхідно зберігати ці події в пам'яті, зробити їх видимими в режимі моніторингу в реальному часі, доступними для аналітики та активних прогнозованих дій. Підтримувати синхронізацію даних між мікросервісами стає набагато складніше у випадку мікросервісних SOA додатків, оскільки кожен мікросервіс має власну базу даних, яка може бути доступна або модифікована тільки через його API, тобто кінцеву точку. Для вирішення цієї проблеми можна використати подієво-керовану архітектуру [4], у якій мікросервіс публікує подію, коли відбувається подія, наприклад, коли він оновлює запис у базі даних. Інші мікросервіси підписуються на ці події. Коли мікросервіс отримує інформацію про подію, він може оновлювати відповідні записи у власній базі даних, що може викликати появу більшої кількості подій, які публікуються і споживаються іншими сервісами.

Подіями можна керувати через використання:

- черг повідомлень (messaging queues), таких як RabbitMQ або ZeroMQ;
- бази даних як проміжного шару між сервісами, коли один сервіс створює і записує подію в базу даних. Інші сервіси постійно запитують цю базу даних і починають діяти, коли виявляють запис про подію.

Варто відзначити, що подієво-орієнтована архітектура при переході до мікросервісів стає базовим типом архітектури додатків на відміну від традиційних SOA [4], що означає більш високу складність таких додатків. Це призводить до значного зростання комунікаційного навантаження порівняно з транзакціями в пам'яті традиційних додатків. Мікросервісам потрібна така мова опису інтерфейсу, щоб за її допомогою можна було легко реалізовувати клієнтів, з одного боку, а з другого, була типізованою звичною мовою (Swagger, proto, WSDL). Навіть якщо мікросервіси розробляються за різними технологіями (наприклад, мов Java, Scala, Python, або графічного інструментарію), всі вони керуватимуться і контролюватимуться за допомогою єдиного призначеного для користувача інтерфейсу.

Перехід на контейнерне керування сервісами за технологією API є по суті альтернативною реалізацією сервісної шини підприємства, що не тільки спрощує виклик сервісів додатками (або їх частинами), але й допомагає їм передавати дані і поширювати повідомлення про події з позицій оброблення складних подій.

КОНТЕЙНЕРИ І SOA

Контейнери, імовірно, є найбільш швидко зростаючою і найбільш захопливою галуззю розвитку віртуалізації і хмарних обчислень. Зв'язок між контейнерами та мікросервісами не є простим. Контейнери не потрібні для розгортання мікросервісів, але водночас мікросервіси потрібні для обґрунтування доцільності використання контейнера. Мікросервіси є одним зі способів підвищення маневреності додатків і повторного використання коду. Контейнери — форма віртуалізації, яка дозволяє уникнути дублювання ОС в машинному образі, забезпечуючи загальну платформу, яка поділяється між всіма компонентами додатка (рис. 2) [8].

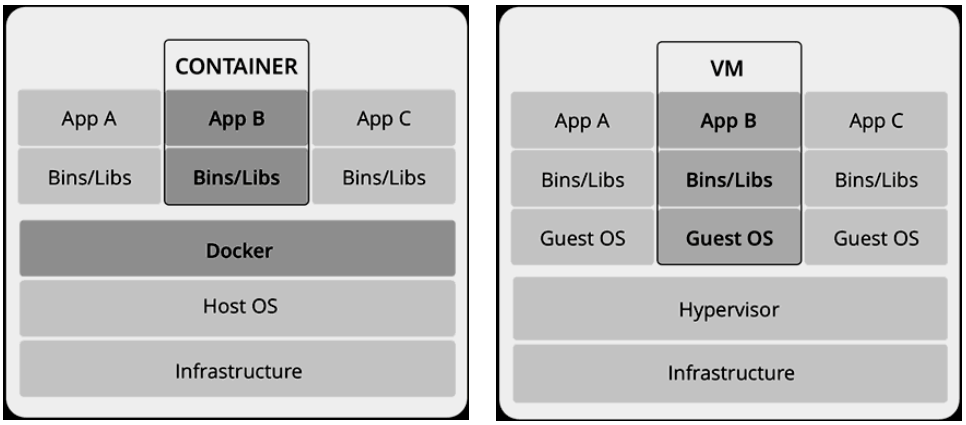


Рис. 2. Порівняння структур контейнерів і віртуальних машин [8]

Усунення дублювання цих великих програмних елементів дозволяє набагато більшій кількості мікросервісів працювати на одному сервері: від п'яти до десяти разів більше, ніж у звичайних провадженнях. Деякі користувачі повідомляють, що вони можуть запускати мікросервіси з контейнерами в 30 разів більшу кількість порівняно з кількістю віртуальних машин.

Мікросервіси також розгортаються швидше в контейнерах, ніж на віртуальних машинах. Це може бути дуже корисним під час горизонтального масштабування сервісів зі зміною навантаження або перерозподілу мікросервісів у зв'язку з відмовою мережевого сервера. Якщо кожен мікросервіс розмістити в окремій віртуальній машині з незалежною ОС, то витрачатиметься багато пам'яті й інших ресурсів.

Продукція фірми Docker домінує на ринку, задовольняючи на 40% існуючий попит. Google давно використовує контейнери. Компанія є одним з основних вкладників у різні контейнеро-орієнтовані проекти з відкритим вихідним кодом, включаючи *Kubernetes оркестратор* і *Google Container Engine*, що функціонують у Google Cloud Platform [11]. Microsoft підтримує контейнеризацію за допомогою *Windows Server Containers*, що дозволяє спільне використання ядра ОС між хостом і контейнерами. *Hyper-V* контейнери розширюють цю послугу, запускаючи кожен контейнер в оптимізованій віртуальній машині. Для хмари є також *Azure Container Сервіс* (ACS), розроблений спільно з Docker, який може керувати підтриманням інших інструментів оркестровки, таких як Kubernetes. Клієнти AWS можуть розгортати контейнери на своїй EC2 платформі, забезпечуючи керування кластерами і планування двигуном Docker за допомогою *EC2 Container Service* (ECS). Ця можливість підтримується *EC2 Container Registry (*ECR) шляхом зберігання. Компанія VMware випустила *vSphere Integrated Containers* для перетворення віртуальних машин у Docker-подібні контейнери для створення платформи відкритого керування системою сервісів для реалізації бізнес-процесів у середовищі PhotonOS. Інші приклади включають IBM контейнери для *Bluemix, Rackspace Carina* (на основі OpenStack Magnum убудоване підтримання для контейнерів та оркестровка). Ще є одна ініціатива з відкритим вихідним кодом Deis, платформа-як-сервіс (PaaS) на основі CoreOS. Слід визнати, що ринок контейнеризації стрімко змінюється [12].

Hyper-V контейнери пропонують новий спосіб реалізації SOA, який є більш ефективним у багатьох випадках. За допомогою контейнера можна запустити кожен сервіс (частину програми) всередині контейнера і далі використовувати контейнерний оркестрант (orchestrator) для керування зв'язком між контейнерами та гарантувати, що кожний сервіс усередині програми працює правильно.

Контейнери є кращими будівельними блоками для SOA, ніж традиційні веб-сервіси через такі їх якості [8]:

- *Контейнери легко переміщуються між хостами (хмарами, вузлами*). У використанні традиційних SOA сервіси залежать від конкретних умов хостів. Наприклад, якщо мережевий файл файлової системи встановлюється на сервері CentOS Linux для надання сервісу зберігання даних для SOA, зміна хоста NFS на сервер Ubuntu потребує значних зусиль. На відміну від цього переміщення контейнера від одного хоста до іншого значно простіше, оскільки контейнерне середовище хоста завжди однакове.

- *Контейнери мають високу масштабованість*. Оскільки контейнери додатків можуть легко переміщуватися між вузлами, контейнерна інфраструктура також легко масштабується. Якщо необхідні додаткові екземпляри певного сервісу, то більше контейнерів можуть бути розгорнуті. Масштабування в традиційних розподілених системах не реалізується так просто.

- *Контейнери легко оновлюються*. Щоб оновити додаток, який працює в контейнері, треба відновити зображення контейнера, на якому заснований

додаток, а потім перезапустити контейнер. Цей процес є більш простим і безвідмовним порівняно з традиційним оновленням сервісу в розподіленій системі, тому нескладно відмінити зміни, якщо це необхідно.

- *Контейнери можуть бути використані повторно*. Однією з основних цілей SOA є повторне використання сервісів, їх розділення і зменшення кількості надлишкових сервісів. Контейнери легко використовувати повторно, оскільки зображення контейнерів можна скопіювати і контейнери можуть швидко переміщатися між вузлами.
- *Контейнери спричиняють менше навантаження*. Контейнери забезпечують гнучкість роботи віртуальних сервісів усередині програмного середовища без необхідності запуску повної віртуальної машини для розміщення сервісів. Замість цього контейнери поділяють обчислювальні ресурси з хостом контейнера. Це означає, що є переваги традиційних програмно-орієнтованих сервісів без істотних накладних витрат ресурсів. Для розроблення і впровадження нових додатків вибираються контейнери для їх розміщення. Необхідно переконатися, що додатки написані для роботи всередині контейнерів, потім установити Docker, розгорнути додатки і використовувати контейнерний оркестрант для керування ними.

Сервер забезпечує повну ізоляцію контейнерів, що запускаються на вузлі, *на рівні файлової системи* (у кожного контейнера власна коренева файлова система), *на рівні процесів* (процеси мають доступ тільки до власної файлової системи контейнера, а ресурси розділені засобами), *на рівні мережі* (кожен контейнер має доступ тільки до цього мережевого простору імен і до відповідних віртуальних мережевих інтерфейсів).

Набір клієнтських засобів дозволяє запускати процеси в нових контейнерах, зупиняти і запускати контейнери, припиняти та відновлювати процеси в контейнерах. Серія команд дає змогу здійснювати моніторинг запущених процесів. Нові зображення контейнера можна створювати зі спеціального сценарного файла, записувати всі зміни, зроблені в контейнері, в нове зображення. Крім того, в інтерфейсі командного рядка вбудовані можливості взаємодії з публічним репозиторієм Docker Hub, у якому розміщені попередньо зібрані зображення контейнерів [12]. Кілька контейнерів розгортаються у кластерах, багато з яких попередньо вбудовуються як компоненти, які можуть бути розміщені поруч для створення зображення додатків.

Є чотири положення, які необхідно враховувати:

1. Операційні системи контейнерів. Навіть якщо контейнери не мають вбудованих ОС, одна спільна ОС, як і раніше, потрібна. Будь-яка стандартна ОС буде працювати, у тому числі Linux або Windows. Проте фактичні потрібні ресурси ОС, як правило, обмежені, тому ОС може бути багато. Це спонукало до розроблення спеціальних контейнерних ОС, таких як Rancher OS, CoreOS, VMware Photon, Ubuntu Snappy, Red Hat Project Atomic і Microsoft Nano сервер [13].

2. Двигун (рушій) контейнера. Тут домінує Docker, але є конкуренти, такі як CoreOS Rocket (Rkt). Двигуни поставляються з допоміжними засобами, наприклад, Docker Toolbox, який спрощує налаштування Docker для розробників, і Docker Trusted Registry для зображення керування.

3. Оркестровка контейнерів. Контейнери повинні бути розумно згруповані для забезпечення функціонування додатків, і це потребує оркестровки.

Двигуни виконують основне підтримання для визначення простих мультиконтейнерних застосувань, наприклад, Docker Compose. Проте повна оркестровка включає в себе планування того, як і коли контейнери повинні працювати, керування кластером і надання додаткових ресурсів, часто інших хостів. Інструменти оркестровки включають Docker Swarm, Google Kubernetes і Apache Mesos [14]. Можна використовувати інструменти конфігурації загального призначення, такі як Chef and Puppet (обидва з відкритим вихідним кодом) або комерційні пропозиції, такі як HashiCorp Atlas або Electric Cloud ElectricFlow.

4. Переміщення додатка з однієї хмарної платформи на іншу. У цьому випадку постачальники програмного забезпечення повинні послідовно оновити свої додатки для розгортання їх у користувачів. Потрібні контейнери будуть копіюватися і додатки будуватимуться з цих контейнерів, оскільки повне робоче середовище відтворене, у тому числі і самі контейнери, баланси навантаження, мережі і т. ін. У 2015 р. компанія Docker випустила *Docker Networking* для включення віртуальних з'єднань між контейнерами. Англійська фірма *Weaveworks* також створила *WeaveNet*, а *Micro-Software-Calico* збільшує безпеку контейнерної мережі через динамічну побудову firewalls (брандмауерів). Docker теж розробляє нові інструменти для підтримання життєвого циклу контейнерних додатків. *Docker Universal Control Plane (UCP)* забезпечує можливості створення, розгортання та керування додатками на вимогу [15].

СУЧАСНИЙ СТАН СЕРВІСНОГО ЗАБЕЗПЕЧЕННЯ EOSC

Із 2017 р. розпочався справжній бум зі створення SOA додатків другого покоління. Найкращими підходами було б розбиття існуючих додатків (legacy) на мікросервіси відповідно до потреб науки чи створення абсолютно нових сервісів (рис. 3).

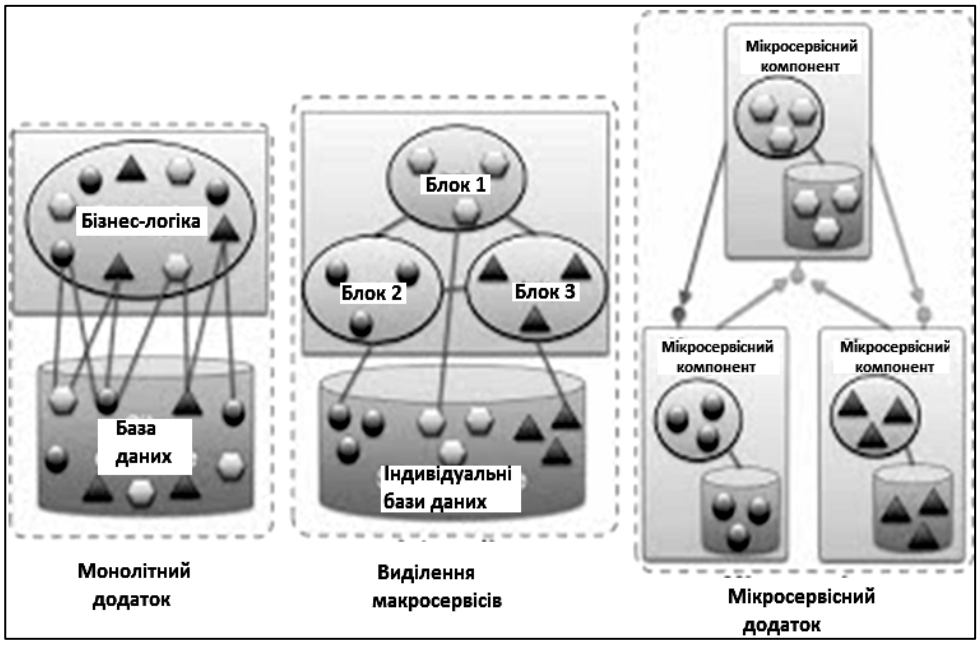


Рис. 3. Розкладання монолітних прикладних додатків на компоненти

Згідно з планами Європейського Союзу в 2020 р. має бути створена Європейська відкрита наукова хмара (EOSC) як екосистема сервісів, які надаються різними постачальниками сервісів, для обслуговування 1,7 млн учених і 80 млн професіоналів з різних галузей науки і технологій (рис. 4).

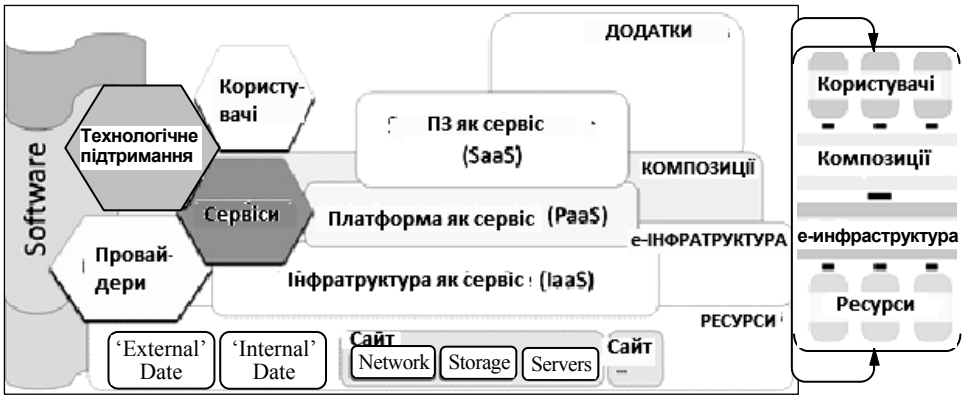


Рис. 4. Європейська відкрита наукова хмара [16]

Жорсткі терміни введення її в дію змушують відомі європейські організації та інфраструктури (EUDAT, CERN, OpenAIRE, GÉANT, EGI, ESFRI, e-IRG, ESA, INDIGO та ін.) базуватися на своїх існуючих монолітних програмних рішеннях, перетворюючи їх на декілька десятків *макросервісів* (замість мікросервісів). Тому, розміщуючи такі макросервіси в контейнери, тимчасово відмовляються від віртуалізації ОС і вимушено в кожний контейнер включають повний екземпляр ОС і власну базу даних. Приклади сервісів, які запропонували EGI, INDIGO, EUDAT, наведено в табл.1–3 [17–19].

Таблиця 1. EGI (European Grid Infrastructure) сервіси [17]

№ з/п	Назва сервісу	Сервіс забезпечення або підтримання
		Обчислювальні сервіси
1	2	3
1	***Cloud Compute***	Виконання обчислень і оброблення даних. Підтримання тривалих сервісів (наприклад, веб-сервісів або баз даних). Створення середовища розроблення і виконання тестування. Вибір конфігурації віртуальної машини відповідно до вимог користувача. Керування ресурсами хмари на гнучкій основі з комплексним моніторингом і можливостями обліку
2	***Cloud Container Compute***	Підготовка додатка на вимогу. Середовище для максимального продуктивності. Стандартний інтерфейс для розгортання сервісів від кількох постачальників. Виконання Docker контейнеів у віртуальному середовищі. Сумісність (interoperable) і прозорість. Усунення конфліктів між середовищами розроблення та експлуатації

Продовження табл. 1

1	2	3
3	***High-Throughput Compute***	Доступ до високоякісних обчислювальних ресурсів. Інтегровані засоби моніторингу та обліку для надання інформації про доступність і споживання ресурсів. Інструменти для робочого навантаження і керування даними для всіх обчислювальних задач. Велика кількість перероблених завдань протягом тривалого періоду часу
Сервіси зберігання		
4	***Online Storage***	Призначення глобальних ідентифікаторів файлам. Доступ до високомасштабованих сховищ у будь-якій точці світу. Контроль за даними, що використовуються зумисно. Організація даних з використанням гнучкої ієрархічної структури
5	***Data Transfer***	Сумісність із надвеликими файлами. Здатність оброблення великих обсягів файлів. Підтримання процесу передавання з автоматичною повторною спробою
6	***Archive Storage***	Збереження даних на тривалий термін. Збереження великої кількості даних. Звільнення власного інтернет-сховища
Сервіси навчання і тренувань		
7	**FitSM*Training***	Збільшення досвіду в галузі керування IT-сервісами. Підвищення професійного профілю визнаної сертифікації
8	***Training Infrastructure***	Цільові конкретні курси та додана вартість для наукових співтовариств. Легке розгортання курсів і їх повторне використання

Таблиця 2. INDIGO (**IN**tegrating **D**istributed data **I**nfrastructures for **G**lobal Expl**O**itation) сервіси [18]

№ з/п	**Назва сервісу**	**Сервіс забезпечення або підтримання**
1	2	3
1	***IAM*** (Identity and Access Management)	Реєстрація сервісу. Менеджмент спільних рішень VO (віртуальних організацій). Постачання сертифікованих OIDC / сервер авторизації OAuth. Надання APIs, базованих на стандарті SCIM. Забезпечення ідентифікації користувачів і надання інформації про політики сервісам для прийняття послідовних рішень щодо авторизації в умовах розподілених сервісів
2	***Dynpart*** (Partition Director Service for Batch and Cloud resources)	Керування гібридним центром оброблення даних, який може надавати сервіси як для пакетного оброблення даних, так і для хмарно розподілених сервісів
3	***Cloud Provider Ranker***	Автономний WEB REST сервіс, який виконує ранжування постачальників хмар за пріоритетами користувачів з вибору ресурсів. Використання двигуна системи керування бізнес-правилами (BRMS, Business Rules Management System)

Продовження табл. 2

1	2	3
5	***Orchestrator*** (PaaS Orchestrator)	Основний компонент прошарку PaaS. Збирання запитів на розгортання з прошарку (рівня) програмного забезпечення. Координація ресурсів або сервісів розгортання динамічних Mesos кластерів або безпосередньо на базі IaaS платформ
6	***udocker*** (Userspace Container Support)	Інструмент для виконання простих контейнерів Docker у просторі користувача. Можливість завантаження і виконання повністю кінцевим користувачем. *Bdocker* і *udocker*: два взаємодоповнювальні підходи для виконання контейнерів в пакетних системах оброблення даних
7	***Ophidia*** (Data Mining and Analytics for eScience Server)	Фреймворк (структура, яка підтримує аналіз даних, інтенсивно використовуючи паралельні методи обчислень і смарт-методи розподілу даних. Використання ієрархічної організації зберігання для поділу і розподілу багатовимірних масивів наукових даних на декілька вузлів
8	***IM*** (Infrastructure Manager)	Розгортання складних і індивідуальних віртуальних ресурсів на різних платформах IaaS хмара (таких як AWS, OpenStack). Надання оркестранту обчислювальних ресурсів можливості використання стандартних мов OASIS (наприклад, протоколу TOSCA)
9	***Kubernetes Monitoring Accounting***	Забезпечення основних функціональних можливостей та інструментів, що підтримують роботу всіх севісів PaaS, доступних в інфраструктурі
10	***fgAPIServer, fgPortal-Setup, fgTools*** **(**Future Gateways (Programmable Scientific Portal)	Програмований інтерфейс з RESTful API сервера: містить набір програмних компонентів, здатних створити або допомогти існуючим веб-порталам стати первісними шлюзами; забезпечує доступ до розподілених обчислювальних ресурсів, таким як грід, хмари і високопродуктивні обчислення
11	***indigoclient, indigoKepler*** (INDIGO Plug-ins for scientific workflow systems)	Система керування робочими потоками (Workflow). Забезпечення інфраструктури установленням, запуском і моніторингом певної послідовності завдань, що створює робочий потік з використанням ресурсів та сервісів, наданих постачальниками й іншими спільнотами
12	***NOW*** (Network Orchestration Wrappercomponent for OpenNebula-based clouds).	Приймання запитів для маніпуляції віртуальної мережі. Глобальна попередня авторизація і виконання запитів через API OpenNebula
13	***Orchent*** (the orchestrator client)	Командний рядок додатка для керування його розгортанням і вибору ресурсів (оркестратор) у швидкий і простий спосіб
14	***Mobile Toolkit***	Надання набору засобів розроблення програмного забезпечення (SDK, Software Development Kit), необхідного для створення мобільних додатків, здатних експлуатувати севіси INDIGO PaaS. Надання можливості менеджерам і розробникам взаємодіяти з усіма сервісами і ресурсами з використанням мобільних пристроїв. Інструментарій, призначений для Android

Таблиця 3. EUDAT (European Data Infrastructure) сервіси [19]

№ з/п	Назва сервісу	Сервіс забезпечення або підтримання
1	***B2DROP*** (Sync and Exchange Research Data)	Рішення для персональної хмари, засноване в довіреному EUDAT CDI домені для зберігання та спільного використання наборів даних на початку дослідження життєвого циклу даних
2	***B2SHARE*** (Store and Share Research Data)	Зручний, надійний сервіс, що заслуговує довіри дослідницьких спільнот, для зберігання та обміну дрібномасштабними обсягами даних, що надходять з різних контекстів
3	***B2FIND*** (Find Research Data)	Простий, зручний портал для пошуку колекцій наукових даних, що зберігаються в EUDAT центрах оброблення даних та інших сховищах даних
4	***B2SAFE*** (Replicate Research Data Safely)	Надійний, безпечний і доступний для керування даними і їх реплікаціями сервісів, що дозволяє спільнотам і відомчим рекозитаріям дублювати та зберігати наукові дані у вузлах EUDAT
5	***B2STAGE*** Get Data to Computation)	Надійний, ефективний, простий у використанні сервіс для переміщення великих обсягів наукових даних між вузлами EUDAT і робочим простором обчислювальних систем високої продуктивності
6	***B2ACCESS*** (Indentity and Authoresation)	Підтримання декількох методів аутентифікації ідентичності користувачів (OpenID, SAML, x.509). Використання як первинного постачальника посвідчень; у разі потреби може бути інтегрований з EduGain і, отже, підтримувати ідентичність користувачів по всьому світу. Забезпечення унікальних ідентифікаторів EUDAT
7	***B2HANDLE*** (Register your research data)	Забезпечення доступу до даних, довговічних посилань на дані і полегшення публікації даних

Відмінності наведених сервісів у різних реєстрах є істотними згідно з різними стандартами і фрейворками, які доступні для керування сервісами, а також через різні процеси керування портфелем сервісів у різних місцях е-інфраструктури на різних рівнях реалізації практики керування сервісами, відсутність стандартизованого способу визначення і опису сервісної е-інфраструктури сервісів, різні пропозиції сервісів від різних постачальників, що потенційно частково перекриваються.

Тому існує насушне завдання сьогодення, яке полягає в необхідності об'єднання існуючих сервісів хоч би у загальний каталог сервісів (не кажучи вже про загальний репозитарій). Роботи зі складання загального каталогу сервісів розпочато в межах проекту ***eInfraCentral*** зусиллями дев'яти організацій (січень 2017 – червень 2019) [20].

Життя (особливо потреби сервісного сектору економіки) потребує засобів створення хмарних мікросервісних систем і адаптації для цілей існуючого інструментарію провідних фірм (*ASP.NET, SAP Enterprise SOA, Oracle SOA Suite 12c, HP SOA Center* та ін.) і для створення мікросервісних реєстрів та забезпечення тим самим реалізації у хмарних SOA віртуалізації як серве-

рів, так і операційних систем [21–24]. Видається доцільним поєднати для цього веб-сервіси з контейнерними технологіями, що дає змогу збільшити розміри хмарних репозитаріїв сервісів (до кількох тисяч компонентів замість десятка тепер) і організації в них ***автоматичних семантичних методів пошуку*** (відкриття) необхідних сервісів (замість ручних, що практикуються в EOSC) [25] .

ВИСНОВКИ

Ключ до успіху SOA лежить у повторному використанні існуючих IT-ресурсів, наприклад, успадкованих додатків. Однак хмарні SOA мають відмінності від SOA, побудованих на ізольованих е-інфраструктурах, передусім у використанні переваг віртуалізації як апаратних (серверів), так і програмних (ОС) ресурсів.

Натепер наявні два напрями: о*рієнтація на потреби наукових спільнот*, мотивована початком створення з 2016 р. Європейської відкритої наукової хмари, і *орієнтація на потреби бізнесу*, зумовлена стрімким становленням сервісного сектору економіки. У першому випадку сервіси, необхідні для створення додатків, переважно вимушено формуються на базі успадкованих наукових застосувань. Таких макросервісів у реєстрах EGI, INDIGO, EUDAT, GIANT поки небагато, і користувач зможе в разі потреби їх знаходити досить просто. Утім тепер стоїть завдання складання з них хоч би каталогу. У другому випадку сервіси, необхідні для створення додатків, розробляються переважно самими користувачами за допомогою ефективного і різноманітного інструментарію провідних фірм, таких як Microsoft, Oracle, HP, SAP тощо. Сервіси, як правило, створюються у вигляді веб-сервісів із семантичним описом (повним або тільки анотованим), що дозволяє впровадити методи автоматичного відкриття необхідних сервісів у надвеликих за обсягом репозитаріях сервісів. Принаймні, веб-сервіси можна створювати і без розглянутого інструментарію провідних фірм, користуючись лише відкритими технологіями і мовами семантичного веб.

Для умов хмарного середовища найбільш притаманне з точки зору досягнення максимальної віртуалізації ресурсів використання мікросервісів і контейнерів для організації їх взаємодій. Тому видається ***доцільним рекомендувати надалі (як для наукових, так бізнесових додатків) застосовувати веб-технології для формування та опису мікросервісів і їх композицій, а також контейнерне керування ними.***

ЛІТЕРАТУРА

1. *Maglio P.* Handbook of Service Science (Service Science: Research and Innovations in the Service Economy) / P. Maglio, C.A. Kieliszewski, J. Spohrer // Springer, NewYork, 2010. — 82 p.
2. *Service* Oriented Architecture: SOA Features and Benefits. — Available at: https://www.opengroup.org/soa/ source-book/soa/soa_features.htm
3. *Петренко А.А.* Объекты и методы науки о сервісах / А.А. Петренко // Системні дослідження і інформаційні технології. —№ 2. — 2015. — С. 75–83.
4. *Петренко О.О.* Порівняння типів архітектури систем сервісів / О.О. Петренко // Системні дослідження і інформаційні технології. — № 4. — 2015. — С. 48–62.

5. *Enterprise* service bus. — Available at: http://www.service-architecture.com/articles/web-services/enterprise_service_bus_esb.html
6. *Tiziana Ferrari.* EGI towards the European Open Science Cloud. — Available at: https://indico.egi.eu/indico/event/3249/session/24/contribution/10
7. *Ньюмен С.* Создание микросервисов / С. Ньюмен. — СПб.: Питер, 2016. — 304 с.
8. *Picking* The Right Cloud Container Platform. — Available at: http://www.communicationstoday.co.in/images/reports/20170501-container-deployment-g6gc442244-report.pdf
9. *Microservices* and containers present a new deployment model in 2017. — Available at: http://searchmicroservices.techtarget.com/opinion/Microservices-and-containers- present-a-new-deployment-model-in-2017
10. *Wilson Mar*. API Management Evaluation. — Available at: https://wilsonmar.github.io/api-management-evaluation/
11. *Containers* as a Service: Comparing Providers and Evaluating the State of the Market. — Available at: http://sandhill.com/article/containers-as-a-service-comparing-providers-and-evaluating-the-state-of-the-market/
12. *Repositories* on Docker Hub. — Available at: https://docs.docker.com/docker-hub/repos/
13. *Operating* System Containers vs. Application Containers. — Available at: https://blog.risingstack.com/operating-system-containers-vs-application-containers/
14. *Kubernetes* is an open-source system for automating deployment, scaling, and management of containerized applications. — Available at: https://kubernetes.io/
15. *Universal* Control Plane overview. — Available at: https://docs.docker.com/datacenter/ucp/2.1/guides/
16. *Realising* the European Open Science Cloud. — Available at: https://ec.europa.eu/research/openscience/pdf/realising_the_european_open_science_cloud_ 2016.pdf
17. *Online* EGI Service Catalogue. — Available at: https://www.egi.eu/services/
18. *INDIGO* сервіси. — Available at: https://www.indigo-datacloud.eu/service-component
19. *EUDAT* site. — Available at: www.eudat.eu
20. *eInfraCentra*. — Available at: http://einfracentral.eu/
21. *.NET* Framework. — Available at: https://en.wikipedia.org/wiki/.NET_Framework
22. *Enterprise* soa development handbook. — Available at: https://archive.sap.com/kmuuid2/40db4735-02f9-2a10-b198-a888a056bb67/Enterprise%20SOA%20Development%20Handbook.pdf
23. *Oracle* SOA. — Available at: https://www.oracle.com/middleware/application-integration/products.html/
24. *HP* SOA Center: Concepts, Technology and Architecture. — Available at: http://www.hp.com/hpinfo/analystrelations/wp_cloudcomputing_soa_capgemini_hp.pdf
25. *Петренко І.А.* Автоматизовані методи пошуку і відкриття необхідних сервісів / І.А. Петренко, О.О. Петренко // Вісник Університету «Україна», Серія «Інформатика, обчислювальна техніка та кібернетика». — №1(17). — 2015. — С. 55–64.
26. *Kunal Joshi.* Introduction to Microservices. Available at: https://techblog.xavient.com/introduction-to-microservices/

Надійшла 21.06.2017