**Лекція №23. Індустріальне виробництво програмних продуктів**

У даний час в інформаційному світі накопичено велику кількість якісних різноманітних програмних і інформаційних ресурсів у електронних бібліотеках, зокрема, наукових, діють фабрики програм системного і прикладного характеру та сформувалися базові основи індустріального виробництва програмних продуктів (ПП). Перехід до індустрії програм і систем пов'язаний також бурним розвитком високоефективної елементної бази, нової багатоядерної, процесорної конфігурації комп’ютерів, кластерів тощо.

**Сутність індустріального підходу**

Термін індустрія визначає виробництво різних видів продуктів масового застосування і засобів їх вироблення промисловими підприємствами, фірмами та корпораціями, а в випадку ПП – програмно-технологічного типу. Головним питанням будь-якої індустрії є не тільки випуск відповідної продукції, але й отримання прибутку від цього. Нині великі прибутки від випуску програмної продукції отримують такі світові фірми, як Microsoft, IBM, Sun, Unix, Intel та ін., а також індійські фірми по оновленню старіючих наслідуваних (legase) програмних систем. В Україні за останні десятиріччя розвиток індустрії ПП фактично відсутній, немає державної і наукової програми її підтримки. Одночасно діють невеличкі підприємства комерційного типу, які виконують роботи по виробленню не масових ПП на замовлення різних фірм, що фінансують їх.

Виробництво ПП виконується на технологічних лініях, за допомогою яких виконавці використовують відповідну теорію і інструментальні засоби підтримки для вироблення ПП масового застосування. Ці лінії включають необхідні процеси життєвого циклу, орієнтовані на розроблення відповідного ПП. Прикладом є технології – Engineering application (інженерія застосунків), family (сімейство), domain (домен), а також сформовані технологічні засоби оновлення застарілих систем в офшорних організаціях (наприклад, в Індії, Росії, Білорусії, України тощо). Нині для підтримки процесів виготовлення ПП використовуються системні сервісні засоби (ОС, загальносистемні сервіси горизонтального і вертикального типу, нові мови, транслятори, компоузери тощо), а також готові програмні ресурси, наприклад, компоненти повторного використання (КПВ) та методики проведення робіт з програмування та обчислювання.

Головна ідея КПВ полягає у накопиченні досвіду програмування, отриманого під час розробок різних програмних систем (ПС), і подання його у вигляді, придатному для використання при побудові майбутніх систем за конвеєрною технологією. Зараз накопичено велику кількість КПВ; це привело до формування таких напрямів їхнього практичного застосування у інженерній діяльності виробництва програмних продуктів:

1) *інженерія* КПВ – систематична і цілеспрямована діяльність з визначення можливостей готового компонента або повторно застосовуваного в ПС, або об’єднуваного у цільову їхню сукупність для виконання більш загальних напрямів роботи;

2) *інженерія застосувань* (application engineering), або прикладних систем — процес виробництва конкретних нових систем, застосувань із КПВ (модулів, програм, підпрограм та ін.), раніше створених як самостійні програмні продукти або як окремі елементи багаторазового використання в інженерії іншої предметної області;

3) *інженерія предметної області* або інженерія домену (domain engineering) містить у собі методи розробки, пошуку, класифікації, адаптації, збирання КПВ, а також одиночних програмних застосувань і створення з них або з готових частин прикладних систем сімейств систем. Системи сімейства зберігають напрацьований досвід з реалізації однієї прикладної проблеми предметної області для застосування його в даному або іншому, більшому сімействі. Необхідна умова цієї інженерії – системні інструментальні засоби підтримки методів накопичення КПВ і впровадження їх у системи нового сімейства.

4) *інженерія виробництва ПС* технологічними засобами та інструментами середовищ сучасних систем автоматизації.

У програмній інженерії під *доменом* розуміють предметну (проблемну) область, яка трактується як *сімейство систем,* призначених для розв’язання різних задач (проблем) цього домену відповідними користувачами. Домен визначається набором понять, які однаково розуміються усіма системами сімейства та користувачами.

Предметні області можна поділити на:

– спеціалізовані, що відбивають специфічні і виробничі інтереси груп фахівців у рамках деякої людської спільноти (наприклад, на підприємстві відображення інтересів бухгалтерів, плановиків, керівника кадрами тощо), певної галузі науки, фірми тощо;

– загальні, що відбивають процеси ЖЦ (визначення вимог, проектування, тестування, оцінювання показників якості тощо) або виробництво програмних систем за типом конвеєрної технології;

– універсальні, що відображають загальні функції, необхідні для використання у спеціалізованих і загальних предметних областях (наприклад, системи БД, захисту, ліцензування, редагування тощо).

У межах деякої предметної області розв’язуються задачі, які відображають спільні і варіантні аспекти її проблем, а також правила маніпулювання ними. Задачі представляються у моделі предметної області шляхом моделювання проблем області у просторі понять цієї предметної області або на перехресті декількох областей і входять до складу спільної мови між розробниками, користувачами і замовником системи.

Іншими словами, кожний домен (область) має систему знань, знайому відповідним професіоналам, з притаманними йому характерними властивостями (атрибутами), відношеннями та правилами поведінки. Посередником між професіоналами і розробниками різних систем, що містяться у домені (наприклад, бухгалтерська, кадрова системи тощо) є специфікована модель предметної області, яка повинна забезпечувати точність та однозначність трактування встановлених понять цієї області як фахівцям предметної області, так і різними категоріями її користувачів. Ця модель застосовується при комунікації між людьми і системами, анотуванні окремих її елементів щодо КПВ, а також при її трансформації у вихідну систему.

Перший і другій напрями інженерії фактично характеризують створення унікальних, одиночних ПС з різного роду КПВ, а третій ставить завдання створення програмних систем і їхніх сукупностей з виділенням окремих частин ПС, що мають загальні властивості і характеристики, які можна багаторазово використовувати в інших системах сімейства.

Четвертий напрям – це засоби і інструменти автоматизації виробництва програмних продуктів, які подаються для застосування сучасними закордонними фірмами (Microsoft, IBM, Rational Rose тощо).

**Фабрики програм.**

Під фабрикою програм розуміється інтегрована інфраструктура зі зборкою готових ресурсів у ПП, потрібних державним, науковим, комерційним й іншим замовникам. основу діяльності фабрики містять окремі лінії та готові КПВ і артефакти, що орієнтовані на конкретну предметну область і накопичені в репозитарію, для збирання з них, як на конвеєрі цільових ПС. Фабрика обладнається продуктовими лініями, набором засобів, інструментів і сервісів для автоматизованого виконання процесів на цих лініях в операційному середовищі.

Фабрика програмних продуктів – це погоджений набір процесів, засобів і інструментів для цілеспрямованого створення нових програмних об’єктів. Фабрика базується на середовище, що містить автоматизовані засоби і інструменти виробництва ПП за відповідними лініями. З погляду інформаційних технологій фабрика дає набір інструментів для переходу від ремесла до індустрії ПП із метою збільшення продуктивності розробки продукту на кожному процесі ЖЦ лінії із заданими функціями, архітектурою і якістю.

Фабрики містять лінії й відповідний набір засобів розробки простих і складних ПП. Лінії розробки простих елементів ПП, як правило, відповідає відповідний ЖЦ, наприклад, реалізований в середовищі MS .Net. Лінія розробки складних ПП на фабрики розглядаються збирального типу із готових програмних ресурсів бібліотек чи репозиторієв глобального типу.

**Базові основи індустрії програм**

Загальні елементи індустрії виробництва програм:

– готові програмні ресурси (артефакти, програми, системи, reuses, assets, КПВ тощо);

– інтерфейс, як специфікатор готових ресурсів, незалежно від мов програмування, в мові інтерфейсу (IDL, API, SIDL, WSDL, RAS тощо);

– технологічні лінії (ТЛ), продуктові лінії (Product Lines) з виробництва ПП;

– збірковий конвеєр (автоматизовані лінії) фабрики програм;

– методики з прийомів проведення робіт на лініях фабрики програм;

– середовище виробництва програм на фабриці.

Опис складових елементів.

*Готові програмні ресурси*.

Програмні ресурси, які можуть використовуватися багаторазово, звуться reuse, assets, компоненти, КПВ, програми тощо. Вони відображають реалізацію різних прикладних або математичних функцій деякої наукової (фізика, математика, біологія тощо) або прикладної предметної області. Головна парадигма КПВ – «писати один раз, виконувати багато разів, де завгодно». Архітектура, в яку вбудовується готовий ресурс, використовує стандартні механізми роботи з ним, як із «будівельними» блоками. Щоб забезпечити високу ефективність повторного використання КПВ, вони мають бути якісними і надійними при обчисленні. Готові ресурси відображають деякі артефакти з діяльності розробників програм.

*Артефакт* – це реальна порція інформації з програмування, яка може створюватися, змінюватися і використовуватися при виготовленні ПП, їм може бути:

– модель предметної області зі своїми термінами, поняттями та лексикою;

– готові КПВ або окремі частини (фрагменти) елементи систем;

– проміжні продукти процесу розроблення (вимоги, завдання, моделі та ін.);

– специфікації (ресурсу, інтерфейсу і т. п.) окремих елементів, діаграм, даних і т. п.

Артефакти незалежні від платформ комп’ютерів мають опис інтерфейсів і параметрів для взаємодії з іншими готовими ресурсами. Усі ресурси і їх інтерфейси зберігаються у сховищах (бібліотеки, репозиторії) для подальшого їх пошуку іншими фахівцями з метою подальшого використання. Повторне використання є капіталомістким видом діяльності з ПП на фабрики.

**Інтерфейси готових програмних ресурсів***.*

Під *інтерфейсом* розуміється зв'язок двох окремих сутностей. Інтерфейси бувають програмні, апаратні, мовні, користувальницькі, цифрові й т. п. Програмний (API) і/або апаратний інтерфейс (port) є способом перетворення вхідних/вихідних даних під час зв’язку комп'ютера з периферійним устаткуванням.

У програмуванні інтерфейсом є програма або її частина, в якій визначаються дані (константи, змінні, параметри й структурні типи даних тощо) для передачі їх іншим програмам з завданням значень типів даних їх параметрів. У ролі інтерфейсу виступають оператори виклику до процедур і функцій у мовах програмування з завданням імен процедур або функцій і списку формальних параметрів з фактичними значеннями для виконання. Послідовність і число формальних параметрів відповідають фактичним параметрам.

Коли програма, процедура або функція, специфіковані різними мовами програмування, і вони розташовані на різних комп'ютерах, то вирішується проблема неоднорідності поданих їх типів даних, відмінності архітектур комп'ютерів або операційних середовищ та несумісності параметрів за їх кількістю та порядком розташування. Інтерфейсу фактично відповідає посередник або перехідник (stub, skeleton Corba) між двома модулями чи програмами. Він передає дані і виконує необхідне пряме й зворотне перетворення даних у випадку їх неоднорідності та невідповідності. Він визначається мовою IDL або APL для об'єкту-клієнта і об'єкту-сервера, має окрему реалізацію і доступний різномовним програмам.

Інтерфейсний посередник включає опис формальних і фактичних параметрів програм, їх типів і порядок завдання операцій передачі параметрів і отримання результатів після виконання. Іншими словами, такий опис є не що інше, як специфікація інтерфейсу двох різномовних програм, які взаємодіють між собою через виклик, який реалізований практично в різних загальних системах (наприклад, COM, CORBA, JAVA тощо).

У функції інтерфейсного посередника (stub у системі CORBA) для клієнта входить:

– підготовка зовнішніх параметрів клієнта для звернення до сервісу сервера;

– посилка параметрів серверу і його запуск для отримання результату або відомостей про помилки.

Загальні функції інтерфейсного посередника (skeleton) сервера:

– отримання повідомлення від клієнта, запуск видаленої процедури, обчислення результату і підготовка (кодування або перекодування) даних з формату клієнта;

– повернення результату клієнтові через параметри повідомлення з результату виконання) та ін.

Таким чином, інтерфейсні посередники задають зв'язок між клієнтом і сервером (stub – для клієнта і skeleton – для сервера), як це зроблено в системі СОРВА.

**Сучасні підходи до реалізації інтерфейсу**.

До підходів з вирішення проблеми інтерфейсу належать наступні:

1) зв'язок готових, різнорідних програм за допомогою інтерфейсу IDL, в якому визначені вхідні й вихідні дані цих програм;

2) спеціальний інтерфейс – JNI (Java Native Interface), що допускає звернення з Java-класів до функцій і бібліотек на інших мов програмування з пошуком прототипів звернень до функцій на C/C++, генерацією заголовкових файлів компіляторами C/C++ і звернення з Java-класів до COM-компонентам;

3) технологія Bridge2Java, що генерує оболонку для COM-компонентів у вигляді проксі-класа і забезпечує необхідне перетворення даних для різних мов програмування засобами стандартної бібліотеки перетворень типів;

4) зв'язок за допомогою мови CLR (Common Language Runtime) платформи .Net для будь-яких її мов програмування, в якій транслюються об'єкти (C#, Visual Basic, C++, Jscript) з використанням бібліотеки стандартних класів і засобів генерації в представлення .Net-компонентів;

5) стандартне рішення щодо специфікації типів даних незалежно від мов програмування за допомогою мови LI (Language Independent) для різномовних компонентів, що містить усі існуючі типи даних мов програмування або засоби їх конструювання. У мові LI описуються параметри виклику, як елементи інтерфейсу, вони перетворюються у типи даних конкретних мов програмування спеціальними правилами і операціями генерації складних типів даних цього стандарту до більш простих фундаментального типу;

6) XDL-стандарт опису структур даних довільної складності і перетворення форматів даних, що передаються з однієї платформи комп'ютера на іншу за допомогою спеціальних процедур та як стандарт забезпечення взаємозв'язків і перетворення типів даних мов програмування, зрозумілого багатьом розподіленим середовищам.

Рішення по конкретному перетворенню даних цими варіантами не вичерпуються, вони ще з'являтимуться при впровадженні нових платформ комп'ютерів і середовищ гетерогенного типу, особливо Cloud Computing.

Стандарт WSDL (*Web Services Description Language* — мова визначення інтерфейсу [вебсервісу](https://uk.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1%D1%81%D0%BB%D1%83%D0%B6%D0%B1%D0%B0) заснована на XML, що описує функціональність вебсервісу і спосіб доступу до нього ) пропонує специфікувати інтерфейс програмних об’єктів (як паспорт) виду:

– назва інтерфейсу;

– ID – ідентифікатор ресурсу;

– зміст програми (функції) з інтерфейсом;

– параметри виклику інших програм;

– інструменти підтримки виконання програми тощо;

– необов'язкові атрибути (дата, стан, версія, право доступу, автор, дата створення, термін придатності, правила придбання і т. п.).

Специфікації інтерфейсу зберігаються у бібліотеки інтерфейсів для майбутньої зборки відповідних об’єктів.

**Технологічні і продуктові лінії.**

*Технологічна лінія* – ТЛ будується з процесів ЖЦ після аналізу предметної області, виявлення її основних задач і функцій. Для процесів підбираються готові ресурси, а також засоби й інструменти перетворення функцій в програми; додається планування, контроль й керування процесами ТЛ з формуванням шаблонів і заготовок для фіксації проектних рішень, зміни станів і оцінювання якості ПП.

Набір процесів ТЛ будується з урахуванням міжнародних стандартів ISO/IEC 12207– 2007 і ДСТУ 9126–99,що підкріплюються відібраними методами, засобами й інструментами для здійснення змін станів проміжних об'єктів.

Поняття *продуктової лінії* (product line (лінія продуктів) і product family (сімейство продуктів) визначені у словнику ISO/IEC FDIS 24765:2009(E) – Systems and Software Engineering Vocabulary як «група продуктів або послуг, які мають спільну керовану множину властивостей, що задовольняють потреби певного сегмента ринку або виду діяльності»). Тобто сім‘я продуктів або лінія продуктів для деякої предметної області – це множина ПС, розроблених з використанням набору готових програмних ресурсів, які мають спільний набір властивостей і рис для деякої предметної області.

Під ресурсами розуміються артефакти, КПВ, проектні рішення, коди, вимоги тощо. Кожна лінія ініціює первинний процес або зборочний конвеєр з виробу ПП з готових програмних ресурсів з метою задовольняння потреб деякого ринка ПП. Деяка фабрика може бути зорієнтована на спеціальні лінії функціонального типу, наприклад, на створення програм із класу задач статистичної обробки, чисельних методів тощо.

**Інженерія продуктових ліній.**

Створення ліній виробництва продуктів з систем, каркасів, готових програм і КПВ, з яких формується кінцевий продукт, що задовольняє певним потребам ринку програмної продукції, запропоноване інститутом SEI *Software Engineering Institute* - американський науково - дослідницький центр Карнеги-Меллон).

Поняття фреймворка для лінії програмних продуктів (Framework for Product Line Practice) сформувалося як деяка автоматизована реалізація інженерії предметної області, в завдання якої входить побудова різних видів програмних продуктів для ринку за допомогою методів і засобів ліній виробництва продуктів. Для побудови ліній виробництва вони досліджували ринок і потреби майбутніх покупців, створювали виробничий план, визначали процеси та організацію їх виконання. На основі аналізу потреб ринку і інтересу до певного виду продукту вони побудували лінію продукції, в яку включаються необхідні методи розробки, тестування і оцінки процесів, продуктів ліній. У інфраструктуру розробки лінії продуктів, окрім необхідних методів і засобів побудови, експлуатації ліній продуктів, входять матеріали і методики по керівництву.

Хоча лінії і сімейства продуктів ототожнюють, лінії продуктів (на ринок) може бути побудована на базі певного представника сімейства продуктів (що розробляється, наприклад, для певного замовлення). При побудові конкретної лінії для деякого представника сімейства у предметні області визначаються:

– технологічні і виробничі обмеження, властиві продуктової лінії;

– зразки і каркаси, які можуть використовуватися на лінії;

– набір засобів, методів і інструментів для розробки продукту на лінії.

Ці лінії ґрунтуються на використанні КПВ, які відбираються з репозиторію для предметні області. Вони підтримують процес складання готових функцій, компонентів і КПВ в єдиний продукт на лінії виробництва.

**Середовище виробництва програм на фабриці**

Середовище виробництва – це засоби, інструменти і методики забезпечення роботи фабрики програм при виробництві ПП методом зборки, функції яких забезпечують їх віртуалізацію, синхронізацію, виконання та підтримку продуктових та технологічних ліній з виробництва різнорідних програм на мовах програмування.

Засоби – це різні мови програмування, які використовуються для опису програм і інструменти, типу транслятори, редактори обов’язково мають бути на фабриці, як необхідний засіб виробництва програм.

Інструменти – це системні, прикладні програмні компоненти (Eclipse, Protégé, Eclipse-DSL тощо) підтримки процесів побудови прикладних ПС чи СПС та сучасні фреймворки (Jaspect, Jbeans, Ant, WCF, Web-service, Amazon, Sky-Driven тощо), які мають комплекс засобів для побудови окремих програмних об’єктів і даних в технології виробництва окремих особливих елементів, та загальносистемні засоби (VS.Net, Corba, Java, IBM Vsphere тощо), які самі як фабрики з великим набором різних засобів (Tools), у т.ч. засобів, що підтримують командну розробку складних програм і проектів. Методики розроблення і виконання процесів виробництва ПП включають документацію і інструкції по організації ТЛ і ТП та принципам проектування на них різних окремих елементів. Сюди відноситься нормативи і регламентовані методи виконання різних робіт з розробки, зборки, керування експертизами, вимірами і оцінками артефактів. Таким чином, ці базові елементи індустрії програм створюють процес виробництва ПП.

Головним базисом фабрики є лінії виробництва спеціальних програм, які будуються засобами мета технології для відображення:

* сфери застосування продукту, що виготовляється;
* функціональної повноти процесів для побудови програмного продукту;
* здатності до внесення змін у набір функцій процесів та і продукту.
* розширення структури лінії «точками варіантності» для поповнення функціональності ліній та відповідних продуктів, а також внесення змін в них тощо.

*Розробка лінії* продукту складається з завдання загального аналізу вимог, побудови процесів проектування і розробки гнучкої архітектури сімейства продуктів на заданому наборі КПВ, активів і reuses для реалізації деякої функціональності продукту, що будується.

Усі програмні ресурси та їхні інтерфейси зберігаються у сховищах (бібліотеках, репозиторіях) для подальшого пошуку іншими фахівцями з метою подальшого застосування. Тобто, повторне використання стає капіталомістким видом діяльності на фабрики ПП.

**Різновиди ресурсів для фабрики**

Пропонується такий набір технічних, технологічних і загальних ресурсів фабрики, необхідних для виконання підпроцесів базового процесу (БП) програмної інженерії і спрямованих на виконання договорів щодо ПП із замовником.

*Технічні ресурси* – платформи, процесори (Intel, IBM, Apple, MS; комунікації) OSI, TCP/IP; комп’ютери користувачів; файли й сервери; локальні та глобальні комп’ютерні мережі; електронна пошта; техніка налагодження, тестери тощо.

*Технологічні ресурси* – бібліотеки, репозитарії готових ПП (КПВ, reuses, аssеts, applications, domains, systems); методики методів програмування збіркового типу (модульного, компонентного, сервісного, UML і ін.); керівництва й методики з мов інтерфейсів (IDL, API, DII, SIDL, XML, RDF і ін.,); стандарти (каркаси, шаблони, контейнери, процеси, програмні проекти, системи й ін.).

*Загальносистемні ресурси* – ОС, клієнт/серверні технології, інструменти; офісні системи (рідери/райтери форматів pdf, ps, html тощо); системи документообігу; утиліти (архіватори, записувачі інформації тощо); засоби захисту інформації (антивірусні, парольні тощо); CASE-інструменти, транслятори; графічні інструменти; СКБД, тощо.

*Людські ресурси фабрики* – це групи розробників і служб керування проектними роботами на фабриці, виконанням планів, керування якістю, ризиком, конфігурацією та перевіркою правильності реалізації вимог проекту, тощо.

В інфраструктуру людських ресурсів згідно стандарту ISO/IEC 12207 включені ряд груп за таким призначенням (рис.1):

* техніко-технологічної підтримки (вивчення ринку, придбання Case, ПП, консультації співробітникам тощо);
* захисту інформації (паролі, ключі захисту, перевірки тощо );
* технологічної служби (супроводження, підтримки ЖЦ, контролю дій тощо);
* якості (SQA-група) із функціями планування та виконання ЖЦ, перевірки робіт, контроль якості робочих продуктів і документів ПП тощо;
* верифікації, валідації і тестування компонентів чи ПП на правильність завдання вимог, координування планів робіт з менеджером, перевірка правильності ПП у тестовому середовищі системи;
* керівник проекту, що відповідає за фінансові та технічні ресурси проекту, виконання проектних угод перед замовником та керування розробкою ПП;
* менеджер проекту, відповідальний за розробку проекту фабрики згідно вимог, проектні рішення й плани робіт по їх реалізації;
* проектувальники і програмісти, що відповідають за розробку проектних рішень і їх реалізацію у вигляді програм, документів і інших вихідних результатів;
* керівник конфігурацією, який реєструє версії ПП, зберігає тверді копії й версії з розмежуванням доступу до них.

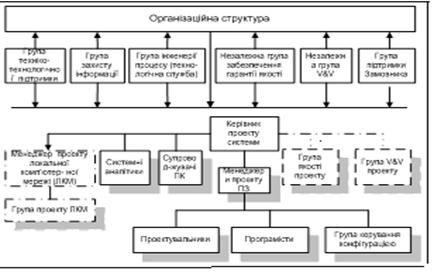


Рис. 1. Взаємодія ресурсів на фабрики

Наведені ресурси є необхідними й достатніми для будь-якого індустріального колективу виробників ПП в подальшому. Роль і призначення різних фахівців наведені також у ряді інших стандартів.

**Індустрія обчислень програм і даних**

Нові індустріальні підходи до обчислення програм і даних з’явилися у зв’язку з бурним розвитком високоефективної елементної бази, нової багатоядерної, процесорної конфігурації комп’ютерів тощо. Це сприяє великомасштабним обчисленням дуже складних задач сучасності в e-sciences (геології, біології, фізики, математики та ін.), АСУ й інших галузях промисловості. Комп’ютерна індустрія більш ніж на порядок опереджає розвиток як з теоретичної, так і з практичної точки зору інші види індустрії, а саме, індустрію обчислень систем і програм. Індустрія обчислень у теоретичному плані іде по шляху застосування нових комп’ютерних можливостей щодо швидкості дій, розподілення пам’яті для обчислення задач з великими обсягами даних та інтероперабельності.

Головна проблема організації обчислень полягає в потребі визначення нових методів, таких як координація, кооперація та взаємодія різних сервісів і інших готових ресурсів через конфігураційний файл для виконання обчислень відповідних задач. Накопичення готових ресурсів у різних глобальних сховищах потребує розвитку індустріальних методів і моделей з урахуванням специфіки наукових задач, готових звичайних компонентах (артефактах, reuses, assets і даних) багаторазового використання, що знаходяться у різних бібліотеках, репозиторіях та нових «хмарних» сховищ Інтернету. Головним методом індустрії наукових продуктів є удосконалений метод зборки наукових різнорідних ресурсів, що належить до напрямків e–science в структури ПП, після виготовлення вони забезпечуються моделлю взаємодії для організації обчислень на різних глобальних серверах. Запити до виконання використовується віртуальний сервер з метою віртуализації і балансування навантаженнями засобами розподілу віртуальних задач по організації обчислень на реальних платформах і великих комп’ютерах чи кластерах.

**Індустрія даних**

Сучасний банк сервісів для збереження даних on-lin e за останні п’ять років є прозорим і вбудований в продукти IBM , Microsoft, що забезпечують синхронне подання даних і передачі даних на віддаленні сервери. Сервіси взаємодіють зі хмарними сховищами для отримання необхідних даних, дозволяють виконувати обчислення і створювати резервні копії даних.

Поряд з таким механізмом доступу до хмарних даних з’являються нові безпровідної технології типу WiMAX подібно безпровідному Інтернету. Сервіс даних on-line сховищ включає:

– синхронізацію даних засобами Windows Live Mech, SkyDriven тощо;

– передачу даних на віддаленні сервери і резервування різних копій даних (Office W ebApps );

– використання нових веб-технології (Flash, Silverlight, Amazon);

– компоузер завдань, робота з БД (Windows Azure, MS SQL, Services, Live Services);

– бібліотеки примітивів з перетворення типів даних із бібліотек. Бібліотека примітивів для перетворення типів даних GDT (примітивних, агрегатних і генерованих) до FDT типів даних (простих, структурних і складних) мов програмування використовується при передачі даних між різномовними компонентами, підсистемами і проектами. Бібліотека складних типів даних GDT (контракт, портфель тощо) використовується для генерації простих типів даних FDT і накопичення у базі даних. Бібліотека функцій для перебудови форматів не релевантних даних інтерфейсних посередників (stub, skeleton), що передають їх на інші платформи взаємодіючим компонентам і зворотно.

**Стандартні ресурси**

Міжнародний комітет по стандартизації на даний часу розробив стандарти програмної інженерії, що регламентують порядок розробки ПП із керованими методами для деякої фабрики програм. Ці стандарти є ресурсом фабрики, головні серед них такі.

Базовий процес призначений для забезпечення «процесного продукування» ПП, як виду інженерної діяльності з виготовлення ПП з операціями оцінки, вимірювання, керування змінами та вдосконалення самого БП відповідно до стандарту ДСТУ ISO/IEC 15504–7 («Оцінювання процесів ЖЦ ПЗ. Настанови з удосконалення процесу»). Оцінка зрілості організації або фабрики програм задається моделлю зрілості CMM (Capability Maturity Models) інституту SEI США, а також моделі Bootstrap, Trillium тощо. Рівень зрілості визначається наявністю фінансових ресурсів, стандартів і методик та здібностей (зрілості) членів колективу фабрики, здатних виготовляти ПП в задані строки та вартість.

Життєвий цикл (ЖЦ) у стандарті ISO/IEC 12207 "Процеси ЖЦ ПЗ” регламентований різними напрямами діяльності з розробки, проектування та керування ПП, організації процесів (планування, керування і супроводження), вимірювання, оцінювання продуктів і процесів. Найбільш важливими серед них є серія стандартів: ДСТУ ISO/IEC 14598 «Оцінювання програмного продукту», стандарт ДСТУ ISO 15939 «Процес вимірювання», серія стандартів ISO/IEC 15504 «Оцінювання процесів ЖЦ ПЗ», базові стандарти з якості – ISO 9001 «Системи керування якістю. Вимоги», ДСТУ 2844–94, ДСТУ 2850–94, що регламентують різні аспекти забезпечення якості ПП, а також стандарт ISO/IEC TR 9126 «Програмна інженерія. Якість продукту».

Ядро знань SWEBOK – це стандарт опису 10 розділів (knowledge areas) і за двома категоріями. Перша категорія – це методи і засоби розробки (формування вимог, проектування, конструювання, тестування, супровід), друга категорія – методи керування проектом, конфігурацією і якістю та базовим процесом. Методи ядра знань відповідають стандартним процесам ЖЦ для наповнення бізнес-процесів з урахуванням потреб конкретної фабрики програм з регламентованою послідовністю розробки і супроводу ПП, починаючи з вимог, виробки проектних рішень, каркасів майбутнього продукту та вибору готових компонентів для «наповнення» цього каркасу відповідним змістом.

Ядро знань менеджменту проекту – це стандарт з керування проектом РМВОК, розроблений інститутом РМІ, що містить у собі опис лексики, структури процесів та три області знань: *керування змістом проекту (*планування з розподілом робіт); *керування якістю* з контролем результатів на відповідність стандартам якості; *керування людськими ресурсами* організації відповідно до їх кваліфікації й професіоналізму. На теперішній час *–* це стандарт IEEE Std.1490 «IEEE Guide adoption of PMI Standard. A Guide to the Project Management Body of Knowledge).

Крім цих головних стандартів, є багато інших, які повинні використовуватися при виготовленні ПП в організаціях або фабриках виробництва програм.

Kubernetes (K8s) — це платформа з відкритим вихідним кодом, яка не залежить від постачальників і допомагає масштабувати, автоматично розгортати та керувати програмами, створеними за допомогою контейнерної технології. Kubernetes упорядковує контейнери вашої програми та забезпечує ефективне керування ними та відстеження. Спочатку розроблений групою інженерів Google і випущений у 2015 році Kubernetes став провідною системою оркестровки контейнерів. Kubernetes дозволяє командам керувати кількома контейнерами одночасно, заощаджуючи ваш час і зусилля.

Попередній підхід (до Kubernetes) до розробки програмного забезпечення полягав у тому, щоб об’єднати функції, логіку та різні сервіси, які складають програму, в одну програму. Однак ця монолітна архітектура не відповідала вимогам сучасної індустрії програмного забезпечення, які постійно змінювалися.

Оскільки програми з часом розширюються, може бути складно вносити точні зміни в код. Якщо його потрібно оновити, доведеться його повторно розгорнути, що призведе до простою та втрати грошей. Якщо в службі є помилка, її виправлення може залишити програму недоступною для користувачів. Масштабування однієї служби може конфліктувати з іншими.

Ці перешкоди усуваються за допомогою архітектури мікросервісів. За допомогою мікросервісів служби програми слабо пов’язані, і можна віртуалізувати їх за допомогою контейнерів. Розробник може самостійно кодувати, розгортати, оновлювати та підтримувати різні служби; також може збільшити або зменшити масштаб залежно від потреб своэї програми. Однак це означає, що в доведеться стежити за сотнями або навіть тисячами контейнерів.

Найпопулярнішим рішенням впорядкування цього божевілля є Kubernetes, який дає змогу командам керувати кількома контейнерами одночасно, заощаджуючи час і зусилля.

**Від віртуальних машин до контейнерів**

Раніше компанії розгортали програми на фізичному сервері з операційною системою та іншими розгорнутими програмами на додаток до цього. Якби програма вимагала більше ресурсів, інші програми не мали б того, що їм потрібно для коректної роботи.

Віртуальними машинами зручно керувати — зовсім не важливо, на якому залізі запущено програму, головне, **щоб збігався набір інструкцій процесора і вистачало ресурсів (оперативної пам'яті, обсягу жорсткого диска та ін.).** Віртуальний сервер можна легко переносити з одного фізичного сервера на інший. До того ж часто це навіть не вимагає зупинки програм.

Віртуалізоване розгортання, яке дозволяє кільком віртуальним машинам працювати на одному фізичному сервері, мало вирішити цю проблему. Як і раніше, на сервері є ОС. На додаток до цього є гіпервізор, який запускає кілька віртуальних машин із власними ОС для розгортання потрібної програми. Це забезпечує більш ефективне використання ресурсів, кращу масштабованість і зниження витрат на обладнання. Але для роботи ОС віртуальної машини потрібні цінні ресурси, а програма не повністю оптимізована.

Контейнери вирішують цю проблему. Вони схожі на віртуальні машини, але мають однакову операційну систему. Вони не покладаються на базову інфраструктуру, тому ними можна керувати та розгортати їх на будь-якому апаратному забезпеченні — у вашій локальній системі, локальній системі, з «голим металом» або в хмарі.

У кожній віртуальній машині запускається ядро операційної системи (наприклад, Linux чи Windows). Для окремої програми запускати своє ядро операційки, з погляду споживаних ресурсів, досить невигідно. Розв'язанням цієї проблеми стала поява механізму використання контейнерів .

Суть контейнера та сама, що й у віртуальної машини: **ми запускаємо програми ізольовано одна від одної**. Але у разі застосування контейнерів **ми не запускаємо ядро операційної системи на кожен із них.** Ізоляція контейнерів забезпечується вбудованими інструментами вже наявної операційної системи.

Розгортання контейнерів є найефективнішим методом, який дає багато переваг, але вам все одно потрібно підтримувати свої контейнери та стежити, щоб не було простоїв. Саме цього Kubernetes може допомогти вам досягти.

Розглянемо детальніше віртуалізацію та контейнерізацію.

Віртуалізація - надання набору обчислювальних ресурсів або їх логічного об'єднання, абстраговані від апаратної реалізації, і забезпечує при цьому логічну ізоляцію один від одного обчислювальних процесів, які виконуються на одному фізичному ресурсі. Прикладом використання віртуалізації є можливість запуску декількох операційних систем на одному комп'ютері: при тому кожен з примірників таких гостьових операційних систем працює зі своїм набором логічних ресурсів (процесорних, оперативної пам'яті, пристроїв зберігання), наданням яких із загального пулу, доступного на рівні обладнання, управляє хостова операційна система - гіпервізор. Якщо говорити про технології, то віртуалізація - це в широкому сенсі відділення уявлення системи від її реалізації. Тобто віртуалізація присутня в будь-якому аспекті ІТ, включаючи операційні системи, сховища даних, веб-браузери та багато іншого.

Існує декілька основних варіантів віртуалізаії, а саме два підходи за допомогою яких ми маємо змогу створювати незалежні простори, які в свою чергу будуть ізольовані один від одного на одному фізичному сервері.

Першим є створення віртуальних машин, для роботи яких потрібен гіпервізор, а другим підходом є створення віртуальних контейнерів. У випадку з віртуальними машинами для кожної використовується власна гостьова ОС, а у випадку з віртуальними контейнерами застосовується ядро хостової ОС. Завдяки цьому, перший підхід дозволяє створювати неоднорідні середовища для обчислень на одному комп'ютері, другий підхід – лише однорідні.

Але оскільки віртуальні машини включають в себе операційну систему, їх розмір буває занадто великим і може досягати кілька гігабайт. Ще одним недоліком віртуальних машин можна вважати те, що для завантаження операційної системи і встановлення програми, яке в ній розміщено, потрібно доволі багато часу в порівнянні з контейнерами. В свою чергу, контейнери більш легкі і, в більшості випадків, їх розмір вимірюється в мегабайтах. В порівнянні продуктивності контейнерів з віртуальними машинами, контейнери можна запускати майже миттєво. Тому при виборі між контейнерами та віртуальними машинами слід враховувати цілі, які намагаєтеся досягти.

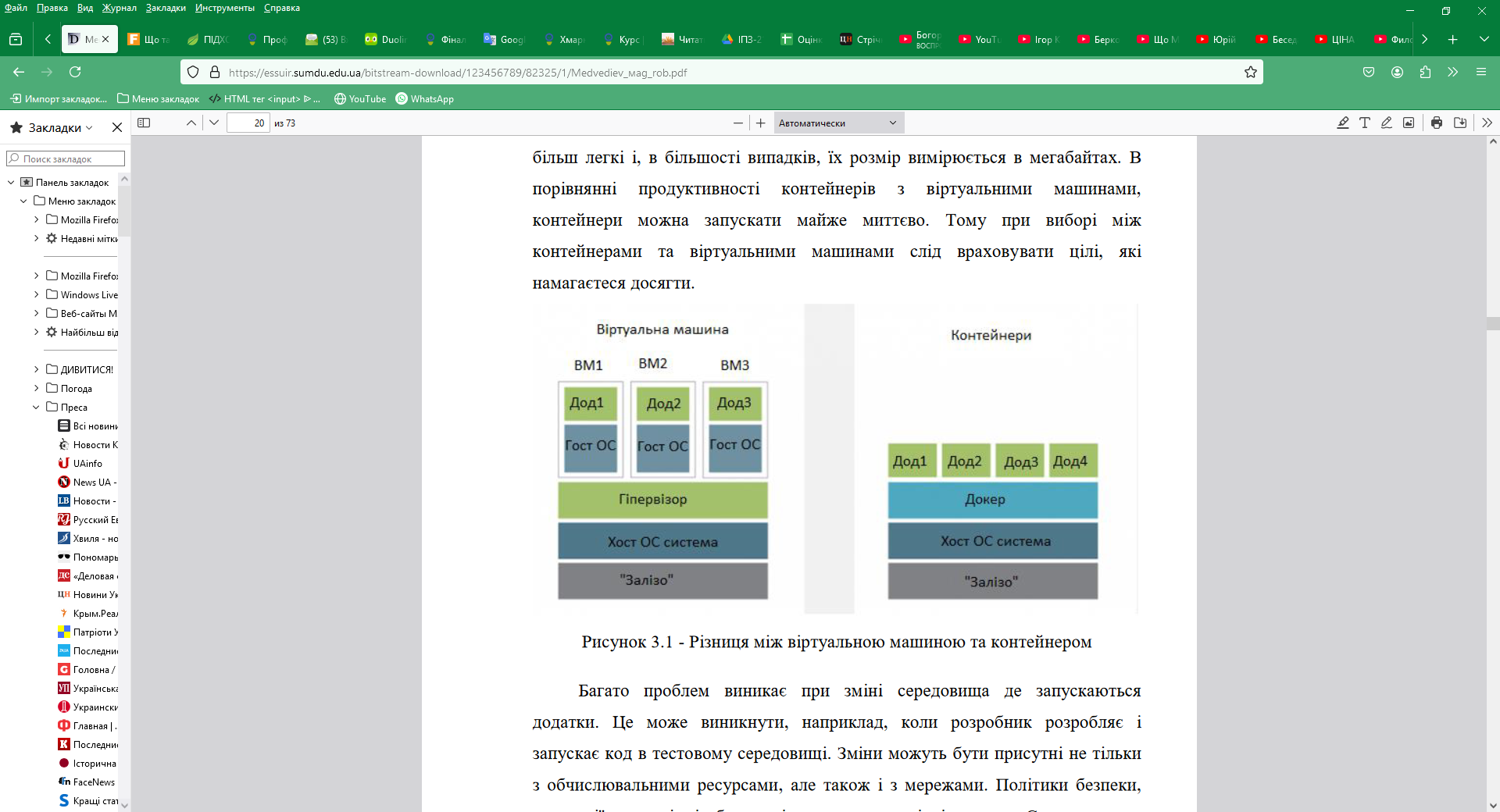


Рисунок 1 - Різниця між віртуальною машиною та контейнером

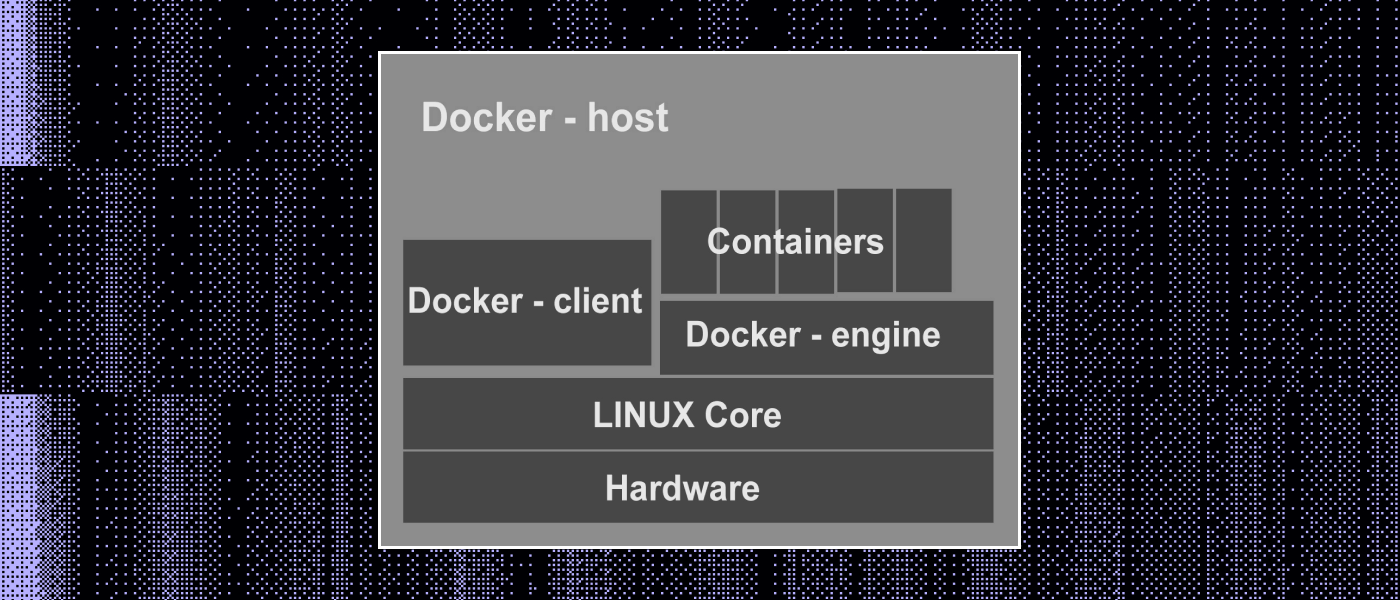
**Управління інфраструктурою, що містить безліч контейнерів, називають**оркестрацією .

**Найпопулярніше рішення для роботи з контейнерами — це** [**Docker**](https://www.docker.com/), контейнеризатор застосунків, що дає змогу запакувати їх з усім оточенням та залежностями.

Контейнери в Docker створюються на основі образів, які по суті є шаблонами. **Образ — це набори файлів програми та інсталяційних команд.** Щоб запустити свою програму через докер, формують Dockerfile, у якому є покроковий опис установки на комп'ютер: на основі цієї інформації створюється образ із програмою.

Образ можна встановити на сервер та на його базі запустити контейнер. Він описується одним файлом, що дуже зручно, тому що його можна записати в Git-репозиторій. Файл версіонується, всі зміни в ньому фіксуються, і можна робити збирання нових версій контейнера через систему Continuous Integration (наприклад, GitLab CI).

Docker запускається як системна служба на сервері під керуванням Windows чи Linux. Ця служба відповідає за запуск контейнера, зупинку та інше керування.

У певному сенсі контейнер можна розглядати як «пісочницю» для застосунків, якесь ізольоване середовище, у якому можна запускати застосунки з обмеженими правами доступу до ресурсів системи. У контейнері міститься ізольоване оточення — файлова система, бібліотеки, залежності та навіть операційна система (у випадку контейнера Docker).

При цьому **контейнери залучають загальне ядро ​​ОС-хоста, тому вони більш легкі, якщо порівнювати з віртуальними машинами**, де кожна має власне ядро. Контейнери також дають змогу упаковувати застосунок та його залежності в одну одиницю, що забезпечує цілісність даних та переносність між різними середовищами.

Концепція контейнерів виявилася настільки успішною, що великі компанії, як-от Google, Amazon, Microsoft та Docker, об'єдналися, щоб розробити загальний стандарт формату образів контейнерів. Це спричинило створення проєкту OCI (Open Container Initiative), який визначає стандарти для контейнерів та образів, базуючись на ідеях Docker.

При цьому **контейнери залучають загальне ядро ОС-хоста, тому вони більш легкі, якщо порівнювати з віртуальними машинами**, де кожна має власне ядро. Контейнери також дають змогу упаковувати застосунок та його залежності в одну одиницю, що забезпечує цілісність даних та переносність між різними середовищами.

Концепція контейнерів виявилася настільки успішною, що великі компанії, як-от Google, Amazon, Microsoft та Docker, об'єдналися, щоб розробити загальний стандарт формату образів контейнерів. Це спричинило створення проєкту OCI (Open Container Initiative), який визначає стандарти для контейнерів та образів, базуючись на ідеях Docker.

Переваги контейнерів:

* Гнучке середовище. Основна перевага у використанні контейнерів полягає в їх швидкості створення, на відміну екземплярів віртуальних машин.
* Підвищена продуктивність. Швидкість розробки досягається за рахунок того, що контейнери усувають мережеві залежності й конфлікти. Кожен контейнер може бути незалежно оновлений без будь-яких проблем тому їх можна розглядатися як окремий мікросервіс.
* Версіонування. Управління версіями дозволяє стежити за відмінностями.
* Переносимість. Контейнери інкапсулюють всі необхідні залежності, такі як бібліотеки, необхідні для запуску програми. Це дозволяє переносити контейнера з одного середовища в інше.
* Стандартизація. Контейнери засновані на відкритих стандартах і можуть працювати в основних дистрибутивах Linux, Microsoft.
* Безпека. Контейнери ізолюють процеси один від іншого і від базової операційної системи. Таким чином, будь-яке оновлення не впливає на роботу іншого контейнера.

Недоліки контейнерів:

* Підвищена складність: при великій кількості контейнерів, які працюють з додатком, збільшується складність. Управління безліччю контейнерів досить складне завданням у виробничому середовищі. Такі інструменти, як Kubernetes і Mesos, полегшують управління великою кількістю працюючих контейнерів.
* Також до мінусів можна віднести складність, яка полягає в тому, що зазвичай в контейнер додається більше ресурсів, ніж потрібно – це призводить до розростання образу і великому розмірі контейнера.
* Підтримка Native Linux: основна кількість контейнерних технологій, таких як Docker, основані на Linux-контейнерах. Тому виконання цих контейнерів в середовищі Microsoft – досить складний процес, а їх щоденне використання викликає складності в порівнянні з початковим запуском цих додатків на Linux.
* Незрілість: контейнери - доволі нова технологія на ринку.

Віртуалізація на рівні операційної системи - це спосіб віртуалізації, при якому операційна система допускає роботу декількох ізольованих контейнерів, а не тільки одного. Вони виглядають як справжні комп'ютери з точки зору програм, запущених в них. Підхід корисний, коли необхідно налаштувати декілька операційних систем з ідентичними параметрами. Різні програми можуть бути встановлені і працювати ідентично, як якщо ми запускаємо додаток на операційній системі хоста. Ресурси, призначені контейнеру, доступні лише йому. Для створення контейнерів операційної системи ми можемо використовувати такі контейнерні технології, як VServer, Jails ,LXC, OpenVZ, Solaris , Linux, BSD.

Контейнери додатків: віртуалізація додатків - це програмна технологія, яка інкапсулює комп'ютерні програми з базової операційної системи, на якій вона виконана. Повністю віртуалізований додаток не встановлюється в традиційному сенсі, хоча він все одно виконується так, як якщо б він був встановлений. Додаток поводиться під час виконання, так як ніби він безпосередньо взаємодіє з вихідної операційною системою і всіма ресурсами, якими він управляє, але може бути ізольованим в різному ступені. Контейнери додатків призначені для упаковки і запуску служб як одного процесу, тоді як в контейнерах ОС можуть виконуватися кілька сервісів і процесів. Контейнерні технології, такі як Docker і Rocket, є прикладами контейнерів для додатків.

***Оркестрація контейнерів***

Контейнерні платформи, такі як Docker, зараз є дуже популярними для упаковки додатків, які базуються на мікросервісній архітектурі. Оркестрації - це координація взаємодії декількох контейнерів. Звичайно, можна працювати і без оркестрації - ніхто не забороняє створити контейнер, в якому будуть запущені всі необхідні процеси. Однак в цьому випадку ви будете позбавлені гнучкості, масштабованості, а також виникнуть питання безпеки, оскільки запущені в одному контейнері процеси не будуть ізольовані і зможуть впливати друг на друга. Оркестрація дозволяє створювати інформаційні системи з безлічі контейнерів, кожен з яких відповідає тільки за одну певну задачу, а спілкування здійснюється через мережеві порти і загальні каталоги. При необхідності кожен такий контейнер можна замінити іншим, що дозволяє, наприклад, швидко перейти на іншу версію бази даних при необхідності.

Існують різні платформи для оркестрації контейнерів. Вони дозволяють реалізувати зручні та ефективні засоби розгортання контейнерних систем, побудови єдиної централізованої консолі для застосування політик управління.

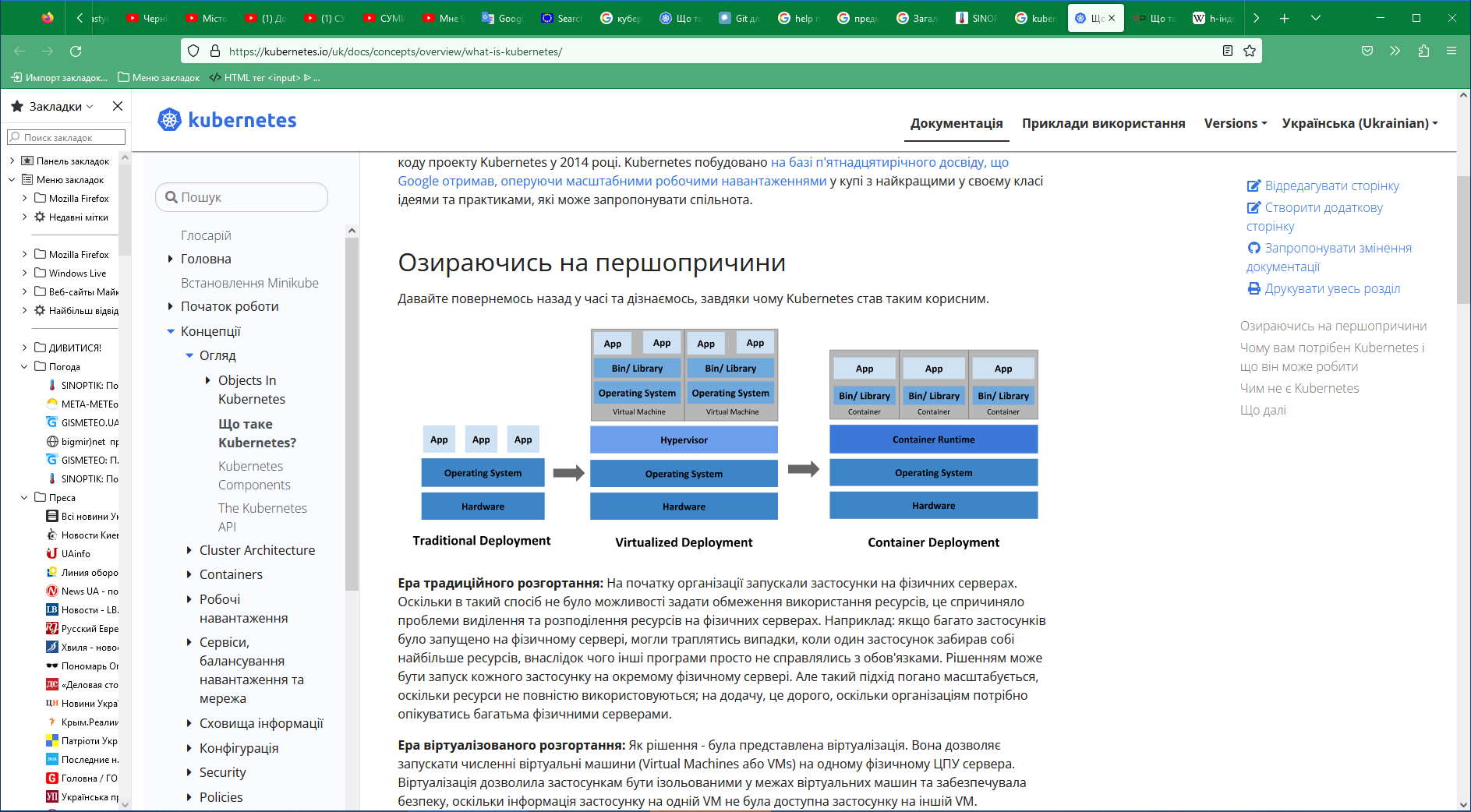
Найбільш популярним є Kubernetes - платформа з відкритим вихідним кодом, спочатку розроблена Google і в даний час підтримувана Cloud Native Computing Foundation. Kubernetes підтримує як декларативну конфігурацію, так і автоматизацію. Це може допомогти автоматизувати розгортання, масштабування і управління контейнерної робочим навантаженням і послугами. API Kubernetes допомагає встановити зв'язок між користувачами, компонентами кластеру і зовнішніми компонентами сторонніх виробників.

Рівень управління Kubernetes і самі вузли виконуються на групі вузлів, які разом утворюють кластер. Робоче навантаження додатка складається з одного або декількох модулів, які виконуються на вузлі (вузлах) Worker.

Рівень управління контролює групи контейнерів (Pod-и) і робочі вузли. До особливостей можна віднести :

* служби виявлення і балансування навантаження;
* оркестрації системи зберігання даних;
* автоматизовані розгортання і відкати;
* горизонтальне масштабування;
* управління секретом і конфігурацією;
* самовідновлення;
* пакетне виконання;
* подвійний стек IPv4 / IPv6.

**Kubernetes**  це платформа з відкритим вихідним кодом для управління контейнеризованими робочими навантаженнями та супутніми службами. Її основні характеристики - кросплатформенність, розширюваність, успішне використання декларативної конфігурації та автоматизації. Вона має гігантську, швидко прогресуючу екосистему.



**Ера традиційного розгортання:** На початку організації запускали застосунки на фізичних серверах. Оскільки в такий спосіб не було можливості задати обмеження використання ресурсів, це спричиняло проблеми виділення та розподілення ресурсів на фізичних серверах. Наприклад: якщо багато застосунків було запущено на фізичному сервері, могли траплятись випадки, коли один застосунок забирав собі найбільше ресурсів, внаслідок чого інші програми просто не справлялись з обов'язками. Рішенням може бути запуск кожного застосунку на окремому фізичному сервері. Але такий підхід погано масштабується, оскільки ресурси не повністю використовуються; на додачу, це дорого, оскільки організаціям потрібно опікуватись багатьма фізичними серверами.

**Ера віртуалізованого розгортання:** Як рішення - була представлена віртуалізація. Вона дозволяє запускати численні віртуальні машини (Virtual Machines або VMs) на одному фізичному ЦПУ сервера. Віртуалізація дозволила застосункам бути ізольованими у межах віртуальних машин та забезпечувала безпеку, оскільки інформація застосунку на одній VM не була доступна застосунку на іншій VM.

Віртуалізація забезпечує краще використання ресурсів на фізичному сервері та кращу масштабованість, оскільки дозволяє легко додавати та оновлювати застосунки, зменшує витрати на фізичне обладнання тощо. З віртуалізацією ви можете представити ресурси у вигляді одноразових віртуальних машин.

Кожна VM є повноцінною машиною з усіма компонентами, включно з власною операційною системою, що запущені поверх віртуалізованого апаратного забезпечення.

**Ера розгортання контейнерів:** Контейнери схожі на VM, але мають спрощений варіант ізоляції і використовують спільну операційну систему для усіх застосунків. Саме тому контейнери вважаються "легкими", в порівнянні з віртуалками. Подібно до VM, контейнер має власну файлову систему, ЦПУ, пам'ять, простір процесів тощо. Оскільки контейнери вивільнені від підпорядкованої інфраструктури, їх можна легко переміщати між хмарними провайдерами чи дистрибутивами операційних систем.

Контейнери стали популярними, бо надавали додаткові переваги, такі як:

* Створення та розгортання застосунків за методологією Agile: спрощене та більш ефективне створення образів контейнерів у порівнянні до використання образів віртуальних машин.
* Безперервна розробка, інтеграція та розгортання: забезпечення надійних та безперервних збирань образів контейнерів, їх швидке розгортання та легкі відкатування (за рахунок незмінності образів).
* Розподіл відповідальності команд розробки та експлуатації: створення образів контейнерів застосунків під час збирання/релізу на противагу часу розгортання, і як наслідок, вивільнення застосунків із інфраструктури.
* Спостереження не лише за інформацією та метриками на рівні операційної системи, але й за станом застосунку та іншими сигналами.
* Однорідність середовища для розробки, тестування та робочого навантаження: запускається так само як на робочому комп'ютері, так і у хмарного провайдера.
* ОС та хмарна кросплатформність: запускається на Ubuntu, RHEL, CoreOS, у власному дата-центрі, у Google Kubernetes Engine і взагалі будь-де.
* Керування орієнтоване на застосунки: підвищення рівня абстракції від запуску операційної системи у віртуальному апаратному забезпеченні до запуску застосунку в операційній системі, використовуючи логічні ресурси.
* Нещільно зв'язані, розподілені, еластичні, вивільнені мікросервіси: застосунки розбиваються на менші, незалежні частини для динамічного розгортання та управління, на відміну від монолітної архітектури, що працює на одній великій виділеній машині.
* Ізоляція ресурсів: передбачувана продуктивність застосунку.
* Використання ресурсів: висока ефективність та щільність.

Kubernetes надає каркас для еластичного запуску розподілених систем. Він опікується масштабуванням та аварійним відновленням вашого застосунку, пропонує шаблони розгортань тощо. Наприклад, Kubernetes дозволяє легко створювати розгортання за стратегією canary (розгортання нових версій застосунку поряд зі стабільними, робочими версіями.) у системі.

Kubernetes надає:

* **Виявлення сервісів та балансування навантаження** Kubernetes може надавати доступ до контейнера, використовуючи DNS-ім'я або його власну IP-адресу. Якщо контейнер зазнає завеликого мережевого навантаження, Kubernetes здатний збалансувати та розподілити його таким чином, щоб якість обслуговування залишалась стабільною.
* **Оркестрація сховища інформації** Kubernetes дозволяє вам автоматично монтувати системи збереження інформації на ваш вибір: локальні сховища, рішення від хмарних провайдерів тощо.
* **Автоматичне розгортання та відкатування** За допомогою Kubernetes ви можете описати бажаний стан контейнерів, що розгортаються, і він регульовано простежить за виконанням цього стану. Наприклад, ви можете автоматизувати в Kubernetes процеси створення нових контейнерів для розгортання, видалення існуючих контейнерів і передачу їхніх ресурсів на новостворені контейнери.
* **Автоматичне розміщення задач** Ви надаєте Kubernetes кластер для запуску контейнеризованих задач і вказуєте, скільки ресурсів ЦПУ та пам'яті (RAM) необхідно для роботи кожного контейнера. Kubernetes розподіляє контейнери по вузлах кластера для максимально ефективного використання ресурсів.
* **Самозцілення** Kubernetes перезапускає контейнери, що відмовили; заміняє контейнери; зупиняє роботу контейнерів, що не відповідають на задану користувачем перевірку стану, і не повідомляє про них клієнтам, допоки ці контейнери не будуть у стані робочої готовності.
* **Управління секретами та конфігурацією** Kubernetes дозволяє вам зберігати та керувати чутливою інформацією, такою як паролі, OAuth токени та SSH ключі. Ви можете розгортати та оновлювати секрети та конфігурацію без перезбирання образів ваших контейнерів, не розкриваючи секрети в конфігурацію стека.

***Контрольні запитання для самоперевірки***.

1. В чому сутність індустріального підходу?
2. В чому полягає ідея компонентів повторного використання?
3. Що розуміється під фабрикою програм?
4. Визначте загальні елементи індустрії виробництва програм.
5. Як будуються технологічні і продуктові лінії?
6. Визначте ресурси фабрики ПП.
7. В чому полягає головна проблема організації обчислень?
8. Які стандарти є ресурсом фабрики?

**Література**

1. Що таке Kubernetes? | Короткий вступ простою мовою. URL: <https://nt.ua/blog/what-is-kubernetes> (дата звернення 11.07.2024)

# Чому не варто використовувати Kubernetes. URL: https://dou.ua/forums/topic/44569/(дата звернення 11.07.2024)

# Серверний диригент: що таке Kubernetes і як він працює. URL: https://robotdreams.cc/uk/blog/397-serverniy-dirigent-shcho-take-kubernetes-i-yak-vin-pracyuye (дата звернення 11.07.2024)

# Документація Kubernetes. URL: https://kubernetes.io/uk/docs/home/ (дата звернення 11.07.2024)

# Підручник Kubernetes для початківців: основи, функції, Archiтектура. URL: <https://www.guru99.com/uk/kubernetes-tutorial.html> (дата звернення 11.07.2024)

1. Бабенко Л.П., Лавріщева К.М*.* Основи програмної інженерії.– Навч. посібник.–К.: Знання, 2001.– 269с.
2. И. Соммервиль. Инженерия программного обеспечения,  
    6 изд. – И.д. "Вильямс", 2002.
3. Лавріщева К.М. Програмна інженерія. Електронний підручник: http://csc.knu.ua/uk/library/books/lavrishcheva-6.pdf
4. Лавріщева К.М. Програмна інженерія. – Підручник.–К.:Академперіодика, 2008.–415с.

***Для самостійного вивчення***: Поглибити матеріал лекції за наданою літературою. Вивчення лекційного матеріалу та додаткових джерел. Розгляд запитань і виконання завдань для самостійної роботи, запропонованих на лекції.