НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних систем

До захисту допущено

Завідувач кафедри

_____ Романкевич В.О.
(підпис) (ініціали, прізвище)

"___" червня 2023 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою

«Системне програмування та спеціалізовані комп'ютерні системи»

спеціальності **123 «Комп'ютерна інженерія»**

на тему: Інформаційна система на основі мікросервісної архітектури

Виконав:

студент IV курсу, групи КВ-91
(шифр групи)

Симонов Богдан Євгенович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник проф.каф.СПСКС, д.т.н. Романкевич В.О. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з нормоконтролю, доц.каф.СПСКС, к.т.н. Клятченко Я.М. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент ст. викл. каф. ОТ ФІОТ Виноградов Ю.М. _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2023 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність 123 «Комп'ютерна інженерія»

Освітньо-професійна програма

«Системне програмування та спеціалізовані комп'ютерні системи»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Віталій РОМАНКЕВИЧ
(підпис) (ініціали, прізвище)
«___» _____20___ р.

ЗАВДАННЯ
на дипломний проєкт студента
Симонова Богдана Євгеновича
(прізвище, ім'я, по батькові)

1. Тема проєкту *«Інформаційна система на основі мікросервісної архітектури»*,
керівник проєкту проф.каф.СПСКС, д.т.н. Романкевич Віталій Олексійович ,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «*31*» травня 2023 р. №2107-С

2. Термін подання студентом проєкту «*19*» *червня* 20*23* р.

3. Вихідні дані до проєкту *Технічне завдання*

4. Зміст пояснювальної записки

1) Аналіз існуючих рішень та обґрунтування теми дипломного проєкту

2) Опис інструментарію

3) Опис інформаційної системи. Розгортання на bare-metal інфраструктурі

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо)

1) Схема структурна

2) Kubernetes cluster. Схема структурна

3) База даних. Схема структурна

4) Процедура створення . Схема алгоритму

6. Консультанти розділів проєкту*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Клятченко Я. М., доцент		

7. Дата видачі завдання 07.11.2022

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Видача завдання на дипломне проектування	07.11.2022	
2	Вивчення літератури за тематикою роботи	21.11.2022	
3	Розроблення та узгодження технічного завдання	28.11.2022	
4	Аналіз існуючих рішень	20.01.2023	
5	Вибір середовища розробки	20.02.2023	
6	Розробка програмного продукту	01.03.2023	
7	Відлагодження програмного продукту	03.04.2023	
8	Підготовка матеріалів текстової частини проєкту	24.04.2023	
9	Підготовка матеріалів графічної частини проєкту	15.05.2023	
10	Оформлення технічної документації проєкту	31.05.2023	

Студент _____ Симонов Б.Є.
(підпис) (Ім'я та ПРІЗВИЩЕ)

Керівник проєкту _____ Романкевич В.О.
(підпис) (Ім'я та ПРІЗВИЩЕ)

* Консультантом не може бути зазначено керівника дипломного проєкту.

АНОТАЦІЯ

Дипломний проект виконаний на здобуття освітньо-кваліфікаційного рівня “Бакалавр” з напряму підготовки “Комп’ютерна інженерія” та включає пояснювальну записку 58 сторінок, 2 таблиці, 6 рисунків.

Об’єкт розробки – Інформаційна система на основі мікросервісної архітектури. Розгортання на bare-metal інфраструктурі..

Ціль розробки – створення методів розгортання інформаційної системи створеної на основі мікросервісної архітектури на bare-metal інфраструктурі. Система повинна дозволяти виконувати достатньо різнопланові завдання використовуючи певні інфраструктурні рішення визначенні замовником.

Система дозволяє дуже гнучке масштабування навантаження в розрізі окремих сервісів. Також система дозволяє використання інфраструктурних рішень визначених замовником, що збільшує рівень безпеки системи.

У роботі проаналізовані варіанти рішень розгортання кластера Kubernetes, проведений аналіз та тестування роботи системи зберігання даних для кластеру, протестовані різні варіанти розгортання БД.

У процесі розробки були використані мови програмування Python, Bash, можливості платформ Linux, Kubernetes.

Ключові слова: Інформаційна система, мікросервіси, bare-metal, Kubernetes, контейнери, скрипти.

ABSTRACT

The diploma project was completed to obtain the educational qualification level of "Bachelor" in the field of study "Computer Engineering" and includes an explanatory note of 58 pages, 2 tableы, and 6 figures.

The development object is an Information System based on a microservices architecture deployed on bare-metal infrastructure.

The development goal is to create deployment methods for an information system based on a microservices architecture on bare-metal infrastructure. The system should be capable of performing diverse tasks using specific infrastructure solutions defined by the client.

The system allows highly flexible workload scaling at the level of individual services. Additionally, the system enables the use of client-defined infrastructure solutions, enhancing the system's security level.

The work includes an analysis of Kubernetes cluster deployment options, an analysis and testing of the data storage system for the cluster, and testing of various database deployment options.

During the development process, programming languages such as Python and Bash were used, along with the capabilities of Linux platforms and Kubernetes.

Keywords: Information System, microservices, bare-metal, Kubernetes, containers, scripts.

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
			Документація загальна			
			Новорозроблена			
	А4	ІАЛЦ.045490.002 ТЗ	Інформаційна система на	4		
			основі мікросервісної			
			архітектури			
			Технічне завдання			
	А4	ІАЛЦ.045490.003 ТП	Інформаційна система на	2		
			основі мікросервісної			
			архітектури			
			Відомість технічного			
			проекту			
	А4	ІАЛЦ.045490.004 ПЗ	Інформаційна система на	58		
			основі мікросервісної			
			архітектури			
			Пояснювальна записка			
	А4	ІАЛЦ.045490.005 Д1	Інформаційна система на	1		
			основі мікросервісної			
			архітектури			
			Схема структурна			

					ІАЛЦ. 045490.001 ОА	
Змін.	Арк.	№ докум.	Підпис	Дата		
Розробив		Симонов Б.Є.			Інформаційна система на основі мікросервісної архітектури. Опис альбому	Літ. / Аркуш / Аркушів: 1 / 3
Перевірив		Романкевич В.О.				КПІ ім. Ігоря Сікорського, ФПМ КВ-91
Н. контроль		Клятченко Я.М.				
Затвердив		Романкевич В.О.				

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045490.006 Д2	Інформаційна система на	1		
			основі мікросервісної			
			архітектури			
			Kubernetes cluster			
			Схема структурна			
	А4	ІАЛЦ.045490.007 Д3	Інформаційна система на	1		
			основі мікросервісної			
			архітектури			
			База даних			
			Схема структурна			
	А4	ІАЛЦ.045490.008 Д4	Інформаційна система на	1		
			основі мікросервісної			
			архітектури			
			Процедура створення			
			Схема алгоритму			
		Диск CD-RW	Текст пояснювальної	1		
			записки			
			Архівований код програми			
			у форматі .zip			
			Презентація дипломного			
			проєкту			
			Графічний матеріал			

ТЕХНІЧНЕ ЗАВДАННЯ

До дипломного проєкту

на тему: «Інформаційна система на основі мікросервісної архітектури»

Київ – 2023

ЗМІСТ

1. НАЙМЕНУВАННЯ І ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання використовується для розробки методів розгортання інформаційної системи створеної на основі мікросервісної архітектури на bare-metal інфраструктурі. Область використання: виконання поставлених задач керування інформаційними процесами та виконання аналітичних чи стратегічних завдань планування.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставами для розробки є завдання на дипломне проектування на здобуття першого (бакалаврського) рівня вищої освіти з напряму підготовки 6.050102 “Комп’ютерна інженерія”, затверджене кафедрою системного програмування та спеціалізованих комп’ютерних систем Національного технічного університету України “Київський політехнічний інститут імені Ігоря Сікорського ”.

3. МЕТА РОЗРОБКИ

Метою даного проекту є створення методів розгортання інформаційної системи створеної на основі мікросервісної архітектури на bare-metal інфраструктурі. Система повинна дозволяти виконувати достатньо різнопланові завдання використовуючи певні інфраструктурні рішення визначенні замовником.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелами розробки є потреби отримання системи з високим рівнем масштабування та використанням інфраструктурних рішень визначених замовником.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до програмного продукту, що розробляється

Програмні засобі повинні забезпечувати такі основні функції:

1. Підготовка інфраструктури замовника для розгортання системи.
2. Розгортання системи, налаштування засобів моніторингу та керування.
3. Початкове налаштування ядра системи.

5.2. Вимоги до програмного забезпечення

Серверна частина: Кластер Kubernetes з відмовостійкістю та мінімум 3-ма нодами. Кластер або сервер БД Postgres. Сховище формату AWS S3.

Клієнтська частина: Пристрій з встановленим web-браузером. Спеціалізовані додатки під Android або iOS.

5.3. Вимоги до апаратного забезпечення

1. Визначена та погоджена з замовником конфігурація для роботи системи на серверах та клієнтах;
2. можливість мережевої взаємодії між серверами системи та клієнтами;
3. система резервування даних та апаратної частини;

6. ЕТАПИ РОЗРОБКИ

№ з/п	Назва етапів роботи та питань, які мають бути розроблені відповідно до завдання	Термін виконання
1.	Видача завдання на дипломне проектування	07.11.2022
2/	Вивчення літератури за тематикою роботи	21.11.2022
3.	Розроблення та узгодження технічного завдання	28.11.2022
4.	Аналіз існуючих рішень	20.01.2023
5.	Вибір середовища розробки	20.02.2023
6.	Розробка програмного продукту	01.03.2023
7.	Відлагодження програмного продукту	03.04.2023
8.	Підготовка матеріалів текстової частини проєкту	24.04.2023
9.	Підготовка матеріалів графічної частини проєкту	15.05.2023
10.	Оформлення технічної документації проєкту	31.05.2023

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
			Документація загальна			
			Новорозроблена			
	А4	ІАЛЦ.045490.004 ПЗ	Інформаційна система на	58		
			основі мікросервісної			
			архітектури			
			Пояснювальна записка			
	А4	ІАЛЦ.045490.005 Д1	Інформаційна система на	1		
			основі мікросервісної			
			архітектури			
			Схема структурна			
	А4	ІАЛЦ.045490.006 Д2	Інформаційна система на	1		
			основі мікросервісної			
			архітектури			
			Kubernetes cluster			
			Схема структурна			
	А4	ІАЛЦ.045490.007 Д3	Інформаційна система на	1		
			основі мікросервісної			
			архітектури			
			База даних			
			Схема структурна			

					ІАЛЦ. 045490.003 ТП	
Змін.	Арк.	№ докум.	Підпис	Дата		
Розробив		Симонов Б.Є.			Інформаційна система на основі мікросервісної архітектури. Відомість технічного проекту	Літ. / Аркуш / Аркушів: 1 / 3
Перевірив		Романкевич В.О.				
Н. контроль		Клятченко Я.М.				КПІ ім. Ігоря Сікорського, ФПМ КВ-91
Затвердив		Романкевич В.О.				

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045490.008 Д4	Інформаційна система на	1		
			основі мікросервісної			
			архітектури			
			Процедура створення			
			Схема алгоритму			
		Диск CD-RW	Текст пояснювальної	1		
			записки			
			Архівований код програми			
			у форматі .zip			
			Презентація дипломного			
			проєкту			
			Графічний матеріал			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ. 045490.003 ТП	Арк. 2

ПОЯСНЮВАЛЬНА ЗАПИСКА

До дипломного проєкту

на тему: «Інформаційна система на основі мікросервісної архітектури»

Київ – 2023

ЗМІСТ

ДОДАТКИ

ДОДАТОК 1. Копії графічних матеріалів

ІАЛЦ.045490.005 Д1	Інформаційна система на основі мікросервісної архітектури. Схема структурна.
ІАЛЦ.045490.006 Д2	Інформаційна система на основі мікросервісної архітектури. Kubernetes cluster. Схема структурна.
ІАЛЦ.045490.007 Д3	Інформаційна система на основі мікросервісної архітектури. База даних. Схема структурна.
ІАЛЦ.045490.008 Д4	Інформаційна система на основі мікросервісної архітектури. Процедура створення . Схема алгоритму.

ДОДАТОК 2. Лістинг програм. Скрипти розгортання та налаштування ІС.

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

БД – база даних

ОС – операційна система

ІС– Інформаційна система

СКБД – система керування базами даних

Скрипти – програмне забезпечення для інсталяції та розгортання ІС

ВСТУП

Сучасний світ – час розквіту комп'ютерних технологій, оскільки інформація є найважливішим чинником життя суспільства. З кожним роком потік інформації зростає, і її доводиться систематизувати, вибирати найпотрібнішу, витративши на це якнайменше часу. А це можливо за допомогою комп'ютерних засобів.

В даний час зручніше зберігати та обробляти інформацію в електронному вигляді, так як вона доступна для користувача в будь-який час, її можна змінювати та доповнювати, враховуючи потреби сучасності, вона займає мало місця, не доводиться працювати з великою кількістю паперових справ. З кожним днем з'являється безліч нових програмних програм, що дозволяють забезпечувати її якісне зберігання та обробку.

Зберігати інформацію будь-якої певної області можна шляхом використання інформаційної системи на основі бази даних. Вона зручна в роботі, її можна застосовувати для різних цілей, вона дозволяє контролювати дублювання даних, чим дозволяє скорочувати обсяг пам'яті, що займається.

База даних, або файл бази даних є набір даних з певної предметної області. База даних містить усі таблиці, запити, звіти, форми та інші структурні елементи. Замість зберігання даних на диску окремо, де їх можна втратити, забути або просто стерти, вони групуються і зберігаються в одному спільному файлі.

У багатьох великих підприємствах зберігається велика кількість інформації про товари та послуги, тому проектована інформаційна система стане незамінним помічником для полегшення роботи керівництва та працюючого персоналу. На даний момент використання такої системи є дуже актуальним.

Також, останнім часом, майже всі інформаційні системи мають web-інтерфейс, що полегшує роботу з ними за допомогою мобільних пристроїв та дозволяє зменшити затрати на підтримку робочих місць користувачів.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ОБГРУНТУВАННЯ ТЕМИ ДИПЛОМНОГО ПРОЕКТУ

1.1. Розвиток інформаційних систем. Процеси інформаційних систем.

Перші інформаційні системи з'явилися у 50-х роках минулого століття. У ці роки вони були призначені для обробки рахунків та розрахунку зарплати, а реалізовувалися на електромеханічних бухгалтерських рахункових машинах. Це призводило до деякого скорочення витрат та часу на підготовку паперових документів. Такі системи називаються системами обробки транзакцій. До транзакцій відносять такі операції: витяг рахунків, накладних, складання платіжних відомостей та інші операції бухгалтерського обліку.

У 60-ті роки. засоби обчислювальної техніки набули подальшого розвитку: з'являються операційні системи, дискова технологія, значно покращуються мови програмування. З'являються системи управлінських звітів (СУЗ), зорієнтовані на менеджерів, які приймають рішення.

У 70-ті роки. інформаційні системи продовжують активно розвиватись. У цей час з'являються перші мікропроцесори, інтерактивні дисплейні пристрої, технологія баз даних та дружнє, по відношенню до користувача, програмне забезпечення (засоби, що дозволяють працювати з програмою, не вивчаючи її опис). Ці досягнення створили умови появи систем підтримки прийняття рішень (СППР). На відміну від систем управлінських звітів, які надають інформацію щодо заздалегідь встановлених форм звітності, СППР надають її у міру виникнення необхідності.

Існують 3 стадії прийняття рішення: інформаційна, проектна та стадія вибору. На інформаційній стадії досліджується середовище, визначаються події та умови, що вимагають прийняття рішень. На проектній стадії розробляються та оцінюються можливі напрямки діяльності (альтернативи). На стадії вибору обґрунтовують та відбирають певну альтернативу, організуючи стеження за її

реалізацією. Найважливішою метою СППР є забезпечення технології формування інформації, а також технологічна підтримка прийняття рішення в цілому.

У 70–80-х роках. в офісах почали застосовувати різноманітні комп'ютерні та телекомунікаційні технології, які розширили сферу застосування інформаційних систем. До таких технологій належать: текстова обробка, настільне видавництво, електронна пошта та ін. Інтеграцію цих технологій в одному офісі називають офісною інформаційною системою. ІС починають широко використовуватися як засіб управлінського контролю, що підтримує та прискорює процес прийняття рішень.

1980-ті роки. характеризуються ще й тим, що інформаційні технології почали претендувати на нову роль організації: компанії відкрили собі, що інформаційні системи є стратегічним зброєю. Інформаційні системи цього періоду, надаючи вчасно потрібну інформацію, допомагають організації досягти успіху у своїй діяльності, створювати нові товари та послуги, знаходити нові ринки збуту, забезпечувати собі гідних партнерів, організовувати випуск продукції за низькою ціною та багато іншого.

1.2. Визначення та класифікація інформаційних систем.

Інформаційну систему можливо охарактеризувати наступними визначеннями :

Визначення 1. Інформаційна система – це сукупність взаємозалежних елементів, що становлять інформаційні, кадрові та матеріальні ресурси, процеси, які забезпечують збирання, обробку, перетворення, зберігання та передачу інформації в організації.

Визначення 2. Інформаційні технології – це сукупність методів, процедур та засобів, що реалізують процеси збирання, обробки, перетворення, зберігання та передачі інформації.

Інформація в сучасному світі перетворилася на один із найважливіших ресурсів, а інформаційні системи (ІС) стали необхідним інструментом практично у всіх сферах діяльності.

Різноманітність завдань, розв'язуваних за допомогою ІС, призвела до появи безлічі різнотипних систем, що відрізняються принципами побудови та закладеними в них правилами обробки інформації.

Інформаційні системи можна класифікувати за низкою різних ознак. В основу класифікації покладено найбільш суттєві ознаки, що визначають функціональні можливості та особливості побудови сучасних систем. Залежно від обсягу розв'язуваних завдань, використовуваних технічних засобів, організації функціонування, інформаційні системи діляться ряд груп (класів) (Рисунок 1).

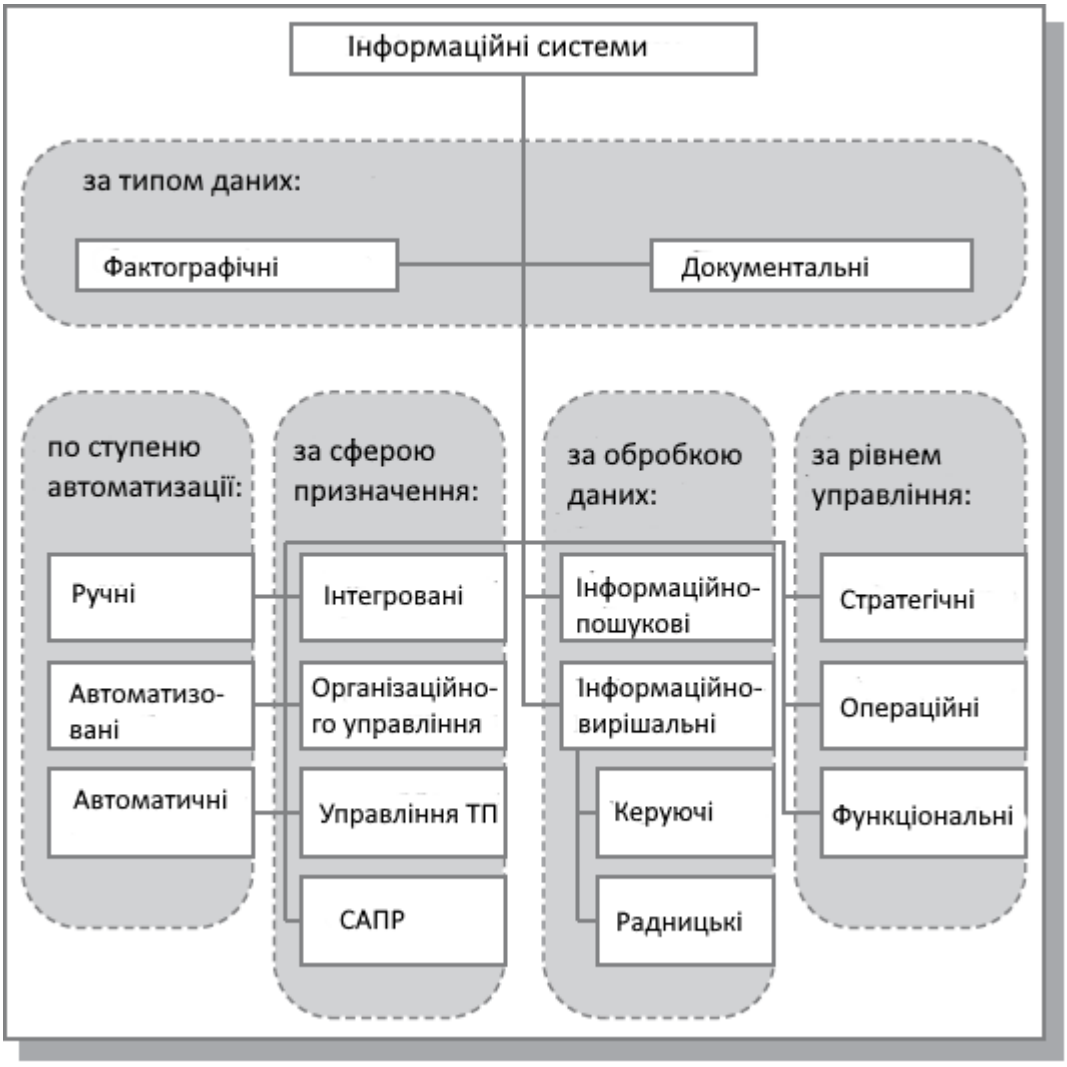

Рисунок 1- Класифікація інформаційних систем.

За типом даних ІС поділяються на фактографічні та документальні. Фактографічні системи призначені для зберігання та обробки структурованих даних у вигляді чисел та текстів. Над такими даними можна виконувати різні операції. У документальних системах інформація представлена у вигляді документів, що складаються з найменувань, описів, рефератів та текстів. Пошук за неструктурованими даними здійснюється з використанням семантичних ознак. Відібрані документи надаються користувачеві, а обробка даних у таких системах практично не виконується.

Ґрунтуючись на ступені автоматизації інформаційних процесів у системі управління фірмою, інформаційні системи діляться на ручні, автоматичні та автоматизовані.

Ручні ІС характеризуються відсутністю сучасних технічних засобів переробки інформації та виконання всіх операцій людиною.

В автоматичних ІС усі операції з переробки інформації виконуються без участі людини.

Автоматизовані ІС припускають участь у процесі обробки інформації та людини, і технічних засобів, причому головна роль у виконанні рутинних операцій обробки даних приділяється комп'ютеру. Саме цей клас систем відповідає сучасному уявленню поняття "інформаційна система".

Залежно від характеру обробки даних ІС діляться на інформаційно-пошукові та інформаційно-вирішальні.

Інформаційно-пошукові системи здійснюють введення, систематизацію, зберігання, видачу інформації на запит користувача без складних перетворень даних. (Наприклад, ІС бібліотечного обслуговування, резервування та продажу квитків на транспорті, бронювання місць у готелях та ін.)

Інформаційно-вирішальні системи здійснюють, крім того, операції переробки інформації за певним алгоритмом. За характером використання вихідної інформації такі системи прийнято ділити на керуючі та радницькі.

Результуюча інформація керуючих ІС безпосередньо трансформується в прийняті людиною рішення. Для цих систем характерні завдання розрахункового характеру та обробка великих обсягів даних. (Наприклад, ІС планування виробництва чи замовлень, бухгалтерського обліку)

Радницькі ІС створюють інформацію, яка приймається людиною до відома та враховується для формування управлінських рішень, а не ініціює конкретні дії. Ці системи імітують інтелектуальні процеси обробки знань, а не даних. (Наприклад, експертні системи.)

Залежно від сфери застосування розрізняють такі ІС класи.

Інформаційні системи організаційного управління – призначені для автоматизації функцій управлінського персоналу, як промислових підприємств, і непромислових об'єктів (готелів, банків, магазинів та інших.).

Основними функціями подібних систем є: оперативний контроль та регулювання, оперативний облік та аналіз, перспективне та оперативне планування, бухгалтерський облік, управління збутом, постачанням та інші економічні та організаційні завдання.

ІС управління технологічними процесами (ТП) – служать для автоматизації функцій виробничого персоналу з контролю та управління виробничими операціями. У таких системах зазвичай передбачається наявність розвинених засобів вимірювання параметрів технологічних процесів (температури, тиску, хімічного складу тощо), процедур контролю за допустимістю значень параметрів та регулювання технологічних процесів.

ІС автоматизованого проектування (САПР) – призначені для автоматизації функцій інженерів-проектувальників, конструкторів, архітекторів, дизайнерів під час створення нової техніки чи технології. Основними функціями таких систем є: інженерні розрахунки, створення графічної документації (креслень, схем, планів), створення проектної документації, моделювання об'єктів, що проектуються.

Інтегровані (корпоративні) ІС – використовуються для автоматизації всіх функцій фірми та охоплюють весь цикл робіт від планування діяльності до збуту

продукції. Вони включають ряд модулів (підсистем), що працюють в єдиному інформаційному просторі і виконують функції підтримки відповідних напрямів діяльності.

Аналіз сучасного стану ринку ІС показує стійку тенденцію зростання попиту на інформаційні системи організаційного управління. Причому попит продовжує зростати саме на інтегровані системи керування. Автоматизація окремої функції, наприклад, бухгалтерського обліку чи збуту готової продукції, вважається вже пройденим етапом багатьом підприємств.

Замовники ІС стали висувати все більше вимог, спрямованих на забезпечення можливості комплексного використання корпоративних даних в управлінні та плануванні своєї діяльності.

Таким чином виникла нагальна необхідність формування нової методології побудови інформаційних систем.

Мета такої методології полягає в регламентації процесу проектування ІС та забезпеченні управління цим процесом для того, щоб гарантувати виконання вимог як до самої ІС, так і до характеристик процесу розробки. Основними завданнями, вирішенню яких має сприяти методологія проектування корпоративних ІС, є:

- забезпечувати створення корпоративних ІС, що відповідають цілям і завданням організації, а також вимогам щодо автоматизації ділових процесів замовника;

- гарантувати створення системи із заданою якістю у задані терміни та в рамках встановленого бюджету проекту;

- підтримувати зручну дисципліну супроводу, модифікації та нарощування системи;

- забезпечувати наступність розробки, тобто. використання в ІС, що розробляється, існуючої інформаційної інфраструктури.

Впровадження методології має призводити до зниження складності процесу створення ІС за рахунок повного та точного опису цього процесу, а також

застосування сучасних методів та технологій створення ІС на ЇЇ всьому життєвому циклі – від задуму до реалізації.

Проектування ІС охоплює три основні сфери:

- проектування об'єктів даних, які будуть реалізовані у базі даних;

- проектування програм, екранних форм, звітів, які забезпечуватимуть виконання запитів до даних;

- вибір конкретного середовища чи технології, а саме: топології мережі, конфігурації апаратних засобів, використовуваної архітектури (файл-сервер або клієнт-сервер), паралельної обробки, розподіленої обробки даних тощо.

Проектування інформаційних систем завжди починається з визначення мети проекту. У загальному вигляді мета проекту можна визначити як рішення ряду взаємопов'язаних завдань, що включають забезпечення на момент запуску системи і протягом усього часу її експлуатації:

- необхідної функціональності системи та рівня її адаптивності до змінних умов функціонування;

- необхідної пропускної спроможності системи;

- необхідного часу реакції системи на запит;

- безвідмовної роботи системи;

- необхідного рівня безпеки;

- простоти експлуатації та підтримки системи.

Відповідно до сучасної методології, процес створення ІС є процес побудови та послідовного перетворення низки узгоджених моделей на всіх етапах життєвого циклу (ЖЦ) ІС. На кожному етапі ЖЦ створюються специфічні йому моделі – організації, вимог до ІС, проекту ІС, вимог до додатків тощо. Моделі формуються робочими групами команди проекту, зберігаються та накопичуються у репозиторії проекту. Створення моделей, їх контроль, перетворення та надання в колективне користування здійснюється за допомогою спеціальних програмних інструментів – CASE-засобів.

Процес створення ІС ділиться на ряд етапів (стадій), обмежених деякими часовими рамками і закінчуються випуском конкретного продукту (моделей, програмних продуктів, документації та ін.).

Зазвичай виділяють такі етапи створення ІС: формування вимог до системи, проектування, реалізація, тестування, введення в дію, експлуатація та супровід.

Початковим етапом процесу створення ІС є моделювання бізнес-процесів, що протікають в організації та реалізують її цілі та завдання. Модель організації, описана у термінах бізнес-процесів та бізнес-функцій, дозволяє сформулювати основні вимоги до ІС. Це фундаментальне становище методології забезпечує об'єктивність у виробленні вимог до проектування системи. Безліч моделей опису вимог до ІС потім перетворюється на систему моделей, що описують концептуальний проект ІС. Формуються моделі архітектури ІС, вимог до програмного забезпечення (ПЗ) та інформаційного забезпечення (ІЗ). Потім формується архітектура ПЗ та ІЗ, виділяються корпоративні БД та окремі додатки, формуються моделі вимог до додатків та проводиться їх розробка, тестування та інтеграція.

Метою початкових етапів створення ІС, виконуваних на стадії аналізу діяльності організації, є формування вимог до ІС, які коректно і точно відображають цілі та завдання організації-замовника. Щоб специфікувати процес створення ІС, що відповідає потребам організації, потрібно з'ясувати та чітко сформулювати, у чому полягають ці потреби. Для цього необхідно визначити вимоги замовників до ІС та відобразити їх мовою моделей у вимоги до розробки проекту ІС так, щоб забезпечити відповідність цілям та завданням організації.

Завдання формування вимог до ІС є однією з найбільш відповідальних, важко формалізованих і найдорожчих і найважчих для виправлення у разі помилки. Сучасні інструментальні засоби та програмні продукти дозволяють досить швидко створювати ІС за готовими вимогами. Але найчастіше ці системи не задовольняють замовників, вимагають численних доопрацювань, що призводить до різкого

здороження фактичної вартості ІС. Основною причиною цього є неправильне, неточне чи неповне визначення вимог до ІС на етапі аналізу.

На етапі проектування передусім формуються моделі даних. Проектувальники як вихідна інформація отримують результати аналізу. Побудова логічної та фізичної моделей даних є основною частиною проектування бази даних. Отримана у процесі аналізу інформаційна модель спочатку перетворюється на логічну, а потім у фізичну модель даних.

Паралельно з проектуванням схеми бази даних виконується проектування процесів, щоб отримати специфікацію (опис) всіх модулів ІС. Обидва ці процеси проектування тісно пов'язані, оскільки частина бізнес-логіки зазвичай реалізується в базі даних (обмеження, тригери, процедури, що зберігаються). Головна мета проектування процесів полягає у відображенні функцій, отриманих на етапі аналізу, у модулі інформаційної системи. Під час проектування модулів визначають інтерфейси програм: розмітку меню, вид вікон, гарячі клавіші та пов'язані з ними дзвінки.

Кінцевими продуктами етапу проектування є:

- схема бази даних (на підставі ER-моделі, розробленої на етапі аналізу);

- набір специфікацій модулів системи (вони будуються з урахуванням моделей функцій).

Крім того, на етапі проектування здійснюється також розробка архітектури ІС, що включає вибір платформи (платформ) і операційної системи (операційних систем). У неоднорідній ІС можуть працювати кілька комп'ютерів на різних апаратних платформах та під керуванням різних операційних систем. Крім вибору платформи, на етапі проектування визначаються такі характеристики архітектури:

- чи це буде архітектура «файл-сервер» або «клієнт-сервер»;

- чи буде це 3-рівнева архітектура з такими шарами: сервер, програмне забезпечення проміжного шару (сервер додатків), клієнтське програмне забезпечення;

- чи буде база даних централізованою чи розподіленою. Якщо база даних буде розподілена, то які механізми підтримки узгодженості та актуальності даних будуть використовуватись;

- чи буде база даних однорідною, тобто, чи будуть усі сервери баз даних продуктами одного й того ж виробника (наприклад, усі сервери лише Oracle або всі сервери лише MS SQL). Якщо база даних не буде однорідною, то яке програмне забезпечення буде використане для обміну даними між СУБД різних виробників (вже існуюче або розроблене спеціально як частина проекту);

- чи будуть для досягнення належної продуктивності використовуватися паралельні сервери баз даних (наприклад, Oracle Parallel Server, MS SQL Server Parallel Data Warehouse тощо).

Етап проектування завершується розробкою технічного проекту ІС.

На етапі реалізації здійснюється створення програмного забезпечення системи, встановлення технічних засобів, розробка експлуатаційної документації.

Етап тестування зазвичай виявляється розподіленим у часі.

Після завершення розробки окремого модуля системи виконують автономний тест, який має дві основні цілі:

- виявлення відмов модуля (жорстких збоїв);

- відповідність модуля специфікації (наявність всіх необхідних функцій, відсутність зайвих функцій).

Після того, як автономний тест успішно пройде, модуль включається до складу розробленої частини системи та група згенерованих модулів проходить тести зв'язків, які повинні відстежити їх взаємний вплив.

Далі група модулів тестується на надійність роботи, тобто проходять, по-перше, тести імітації відмови системи, а по-друге, тести напрацювання на відмову. Перша група тестів показує, як добре система відновлюється після збоїв програмного забезпечення, відмов апаратного забезпечення. Друга група тестів визначає ступінь стійкості системи при штатній роботі та дозволяє оцінити час

безвідмовної роботи системи. До комплекту тестів стійкості повинні входити тести, що імітують пікове навантаження на систему.

Потім весь комплект модулів проходить системний тест - тест внутрішнього приймання продукту, що показує рівень його якості. Сюди входять тести функціональності та тести надійності системи.

Останній тест інформаційної системи – приймально-здатні випробування. Такий тест передбачає показ інформаційної системи замовнику та має містити групу тестів, що моделюють реальні бізнес-процеси, щоб показати відповідність реалізації вимог замовника.

Необхідність контролювати процес створення ІС, гарантувати досягнення цілей розробки та дотримання різних обмежень (бюджетних, тимчасових тощо) призвело до широкого використання у цій сфері методів та засобів програмної інженерії: структурного аналізу, об'єктно-орієнтованого моделювання, CASE-систем.

1.3. Архітектурні рішення інформаційних систем. Сучасне бачення розвитку.

В даний час для розробки інформаційних систем використовуються переважно два архітектурні рішення – монолітна або мікросервісна архітектура.

Монолітна архітектура означає, що система має однорідну будову. Монолітний додаток описує однорівневу систему, в якій різні компоненти об'єднані в одну програму на єдиній платформі. Зазвичай монолітний додаток складається з бази даних, клієнтського користувальницького інтерфейсу та серверної частини. Усі частини програмного забезпечення уніфіковані, і його функції управляються одному місці.

Можна виділити такі переваги монолітної архітектури:

- Спрощена розробка та розгортання. Розробка монолітної програми відносно проста. Всі файли знаходяться в одному каталозі, що дозволяє швидко знаходити необхідний компонент і у разі помилки виправити його. У зв'язку з тим, що при монолітній розробці введеться робота в

основному на одному інструменті та розробляється однією мовою програмування, не потрібно розгортати зміни або оновлення окремо.

- Менше комунікаційних проблем та спрощене логування. Більшість програм залежать від взаємодії між компонентами, що ускладнює ведення контрольних журналів, логів, обмежує швидкість роботи і т. д. Монолітні програми легше виконують такі завдання завдяки єдиній кодовій структурі. До таких програм легше додавати нові компоненти, оскільки все працює однаково.
- Краща продуктивність. Монолітні програми надають досить швидкий зв'язок між компонентами завдяки єдиній кодовій структурі. Як правило, один запит з боку клієнта справді призведе до єдиної атомарної дії. Таким чином, не буде накладних витрат на взаємодію з іншими частинами системи.

Мінусами даної архітектури є:

- Кодова база з часом стає громіздкою. Складніше вносити зміни в таку велику і складну програму з дуже тісним зв'язком його компонентів. Будь-яка зміна коду впливає всю систему, тому його необхідно ретельно координувати. Це значно підвищує тривалість процесу розробки.
- Обмежена гнучкість. У монолітних програмах кожне невелике оновлення вимагає повного повторного розгортання. Таким чином, усі розробники зобов'язані чекати, доки це не буде зроблено. Коли кілька команд працюють над одним проектом, гнучкість може бути значно знижена.

Монолітна модель активно використовується та не викликає додаткових труднощів. Особливо корисною може бути така архітектура для невеликих проектів чи проектів на початковій стадії розробки, що досить важливо для перевірки нових гіпотез у програмних продуктах.

Мікросервіс – це тип сервісно-орієнтованої програмної архітектури (SOA), яка орієнтована створення серії автономних компонентів, складових додатоку. На

відміну від монолітних програм, побудованих як єдине неподільне ціле, мікросервісні програми складаються з декількох незалежних компонентів, склеєних разом за допомогою API. Порівняння двох архітектур представлено на Рисунок 2

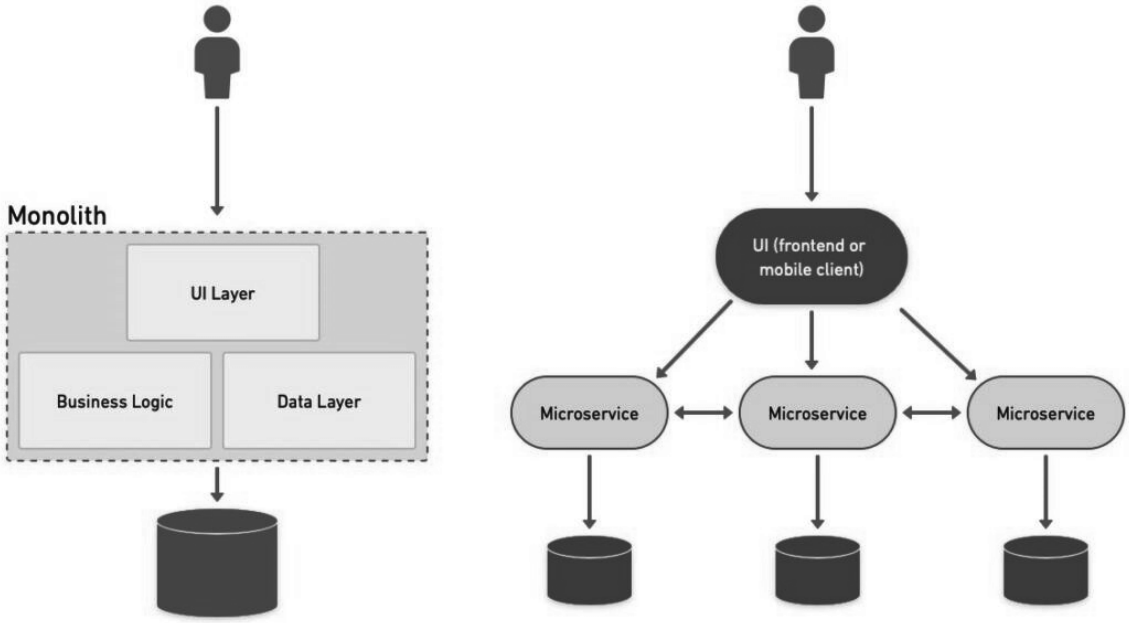

Рисунок 2 - Порівняння монолітної та мікросервісної архітектури

Підхід мікросервісів фокусується в основному на бізнес-пріоритетах і можливостях, тоді як монолітний підхід організований навколо технологічних рівнів, інтерфейсів і баз даних. Підхід з використанням мікросервісів став популярним, оскільки все більше і більше підприємств намагаються швидше виводити свої нові рішення на ринок та оптимізувати ресурси розробки.

До плюсів мікросервісної архітектури можна віднести наступне:

- Легко розробляти, тестувати та розгортати. Найбільшою перевагою мікросервісів перед іншими типами архітектур є те, що невеликі окремі сервіси можна створювати, тестувати та розгортати незалежно. Оскільки модуль розгортання невеликий, він спрощує та прискорює розробку та випуск у промислове середовище.
- Гнучкість. Усі сервіси можна розгортати та оновлювати незалежно, що надає розробникам велику гнучкість. Помилка в одному мікросервісі впливає тільки на конкретний сервіс і не впливає на додаток. Крім того,

набагато легше додавати нові функції в мікросервісний додаток, ніж монолітний.

- Можливість горизонтального масштабування. Кожен елемент можна масштабувати незалежно. Таким чином, весь процес більш рентабельний і ефективний за часом, ніж при використанні монолітів, коли весь додаток необхідно масштабувати, навіть якщо в цьому немає необхідності. Крім того, кожен моноліт має обмеження з точки зору масштабованості, існує низка обмежень за кількістю користувачів.

У мінуси цього підходу слід зарахувати такі пункти:

- Складність. Оскільки архітектура мікросервісів є розподіленою системою, необхідно вибирати та налаштовувати з'єднання між модулями та базами даних. Крім того, оскільки така програма включає набори окремих сервісів, всі вони повинні розгортатися незалежно один від одного..

Таким чином, мікросервісна архітектура вигідніша для складних і динамічно змінних додатків. Вона пропонує ефективні рішення для керування складною системою різних функцій та послуг в одному додатку. Мікросервіси ідеальні, коли йдеться про платформи, що охоплюють складні та пов'язані бізнес-процеси.

Інформаційна система сучасного підприємства є досить складним додатком, який потребує реалізації різних незв'язаних між собою функцій та послуг. Мікросервіси додають унікальну цінність за рахунок спрощення складності систем. Розбиваючи додаток на безліч дрібніших частин, зменшуючи дублювання та зв'язки між частинами, можна отримати більш зрозумілий, масштабований і змінюваний додаток в цілому.

Сьогодні організації, які прагнуть максимальної продуктивності, гнучкості та покращення якості обслуговування клієнтів, виходять за рамки монолітних додатків та використовують мікросервіси, чиї слабопов'язані архітектури

прискорюють розробку, тестування та впровадження, тим самим зменшуючи операційні витрати на розвиток та підтримку ІС підприємства.

Мікросервісні додатки можуть працювати як на фізичних так й на віртуальних серверах, але зазвичай використовуються контейнери, які містять лише прикладну програму, необхідні їй бібліотеки та файли, що виконуються. Це заощаджує дисковий простір і дозволяє ефективніше використовувати ресурси сервера.

Зараз найбільш популярним середовищем для автоматичного розгортання , масштабування та керування контейнерами є Kubernetes (K8s). Це фреймворк для гнучкої роботи розподілених систем. Він займається масштабуванням та обробкою помилок у додатку, надає шаблони розгортання та багато іншого.

1.4. Мікросервісна інформаційна система на bare-metal інфраструктурі.

Як ми зазначили вище основним середовищем мікросервісної ІС є Kubernetes.

Основні принципи Kubernetes:

- Надійність. Кластер Kubernetes повинен відповідати останнім рекомендаціям щодо безпеки.
- Простота у використанні. Кластер Kubernetes має керуватися за допомогою кількох простих команд.

Основні можливості Kubernetes:

- Моніторинг сервісів та розподіл навантаження. Kubernetes взаємодіє з контейнером, використовуючи ім'я DNS або IP-адресу. Також Kubernetes може збалансувати навантаження та розподілити мережевий трафік, щоб розгортання було більш стабільним.
- Оркестрація сховища. Kubernetes дозволяє автоматично змонтувати потрібну систему зберігання, наприклад, локальне сховище, провайдера хмарного сховища та багато іншого.

- Автоматичне розгортання та відкати. Використовуючи Kubernetes, можна змінювати фактичний стан розгорнутих контейнерів, а також описувати бажаний стан нових контейнерів.
- Автоматичне розподілення навантаження. На основі наданих для Kubernetes кластерів, вузлів, заданих даних CPU і пам'яті для кожного контейнера, Kubernetes може розмістити їх на вузлах так, щоб найбільш ефективно використовувати ресурси.
- Самоконтроль. Kubernetes автоматично перезапускає збійні контейнери. При необхідності він перезапускає та завершує роботу контейнерів, які не проходять певну задану користувачем перевірку працездатності та відключає доступ до них, доки вони не будуть готові до обслуговування.
- Управління конфіденційною інформацією та конфігурацією. Kubernetes може зберігати та керувати конфіденційною інформацією, такою як паролі, токени та ключі. При необхідності можна розгортати та оновлювати конфіденційну інформацію, конфігурацію програми без змін образів контейнерів і не розкриваючи конфіденційну інформацію.

Розгортання Kubernetes, що працює, називається кластером. Можна розділити Kubernetes як дві частини: площину керування (Control plane) та обчислювальні машини чи вузли (nodes). Кожен вузол є власним середовищем Linux і може бути фізичною або віртуальною машиною. На кожному вузлі працюють модулі, що складаються із контейнерів.

Багато хмарних сервісів пропонують платформу на основі Kubernetes або інфраструктуру як послугу, де Kubernetes може бути розгорнутий як сервіс. Багато постачальників також пропонують власні фірмові дистрибутиви Kubernetes. Наприклад VMware Tanzu, Redhat Openshift, Azure AKS, Amazon EKS, Google GKE та інші.

Також є можливість розгортання Kubernetes на власних фізичних серверах або віртуальних машинах (bare-metal варіант). Даний варіант, при всій своїй складності, може бути використаний у тому випадку коли треба розгорнути

мікросервісну систему в якій будуть оброблятись дані, які мають підвищений ступінь конфіденційності. Такими наприклад є дані органів державного управління та сфери критичної інфраструктури, наприклад енергетичних компаній, компаній газотранспортної та газовидобувної сфери. Навіть при наявності ліцензованих, за ступенем інформаційної безпеки, хмарних рішень не всі дані вони готові виносити за межі свого периметру внутрішньої мережі.

Виходячи з цього, є нагальною розробка алгоритму та програмних засобів для розгортання інформаційної системи на основі мікросервісів, розробленої для використання в цих сферах, саме з використанням власної (bare-metal) інфраструктури замовника.

2. ОПИС ІНСТРУМЕНТАРІЮ

Інструментарій, для розробки інформаційної системи, був частково регламентований вибором архітектури та вимогами замовників. Тип ОС чітко визначений вимогами до середовища Kubernetes та має лише можливість вибору дистрибутиву ОС. Вибір СКБД та системи зберігання файлів та об’єктів ІС частково залежить від завдань ІС та її кастомізації під конкретного замовника. API взаємодії мікросервісів та мова їх програмування було вибрано з прицілом на довгий строк життя технологій та зручність впровадження.

2.1 Операційна система та оркестратор мікросервісів

У якості базової ОС інформаційної системи було вибрано ЩС Linux: Debian Linux 11 та Oracle Linux 8. Такий вибір зумовлений наявністю, в відповідних дистрибутивах, необхідних компонентів для налаштування необхідних сервісів та частково внутрішнім регламентом одного з замовників.

Linux - вільно розповсюджуване ядро Unix-подібної системи, написане Линусом Торвальдсом за допомогою великого числа добровольців по всій Мережі. Linux має усі властивості сучасної Unix-системи, включаючи дійсну багатозадачність, розвиту підсистему керування пам'яттю і мережеву підсистему. Ядро Linux, що поставляється разом із вільно розповсюджуваними прикладними і системними програмами утворить повно функціональну універсальну операційну систему. Велику частину базових системних компонентів Linux успадкував від проекту GNU, ціллю якого є створення вільної мікроядерной операційної системи з обличчям Unix. На сьогоднішній день існує множина різноманітних Linux, дистрибутивів, що можна розділити на дистрибутиви загального призначення і спеціалізовані. До спеціалізованих дистрибутивів відносяться такі як Linux Router – урізани1 Linux для створення дешевого маршрутизатора на базі старого PC та ін. Незважаючи на розходження в дистрибутивах загального призначення, усі вони утворять обличчя ОС Linux таким, яким їх знають більшість користувачів ОС. На відміну від ядра, дистрибутиви можуть містити комерційні компоненти і тому їхнє

вільне поширення може бути обмежено. У такому випадку автори дистрибутивів роблять доступними усі вільні компоненти. Звичайно під словосполученням "ОС Linux" розуміють дистрибутиви Linux загального призначення.

Linux є високо масштабованою системою, спроможною задовольнити інформаційні потреби на великих і малих підприємствах. ОС Linux перетворилась в операційну систему для бізнесу, освіти та індивідуального користування. Це багатозадачна, багатокористувацька операційна система, тобто система, в якій одночасно можуть працювати декілька сотень користувачів і одночасно може бути запущено багато задач. Як і будь–яка інша операційна система, Linux постійно вдосконалюється. Будь-який користувач може взяти участь в розробці операційної системи Linux та її вдосконаленню. Звичайно - ж при відповідній кваліфікації і відповідних навичках роботи з ОС Linux.

У якості оркестратору мікросервісів був вибраний Kubernetes котрий на даний момент є основним стандартом в цій сфері.

2.2 Середовище розробки, платформа та мова програмування.

Середовищем розробки для мікросервісів ІС є Java, а середовищем розробки системи розгортання є командний інтерпретатор Shell, який виконує команди користувача, введені з терміналу. Він має вбудовану мову Shell-script, яка фактично являється мовою програмування високого рівня та є інтерфейсом між операційною системою та користувачем. Крім безпосереднього введення команд в командному рядку, можна використовувати так звані сценарії, які представляють собою набір команд, записаних в текстовому файлі. Сценарії - зручний спосіб для виконання часто використовуваних послідовностей команд.

Швидкість виконання шел-скриптів в багато разів менше за швидкість скомпільованої програми на С чи інших мовах. Проте шел надає можливість різко спростити рішення багатьох практичних задач, пов’язаних з управлінням ОС, обробкою текстових файлів і т.д. Це досягається за рахунок того, що шел-скрипти дозволяють використовувати так звані “крупні блоки”: складати нові програми

шляхом витонченого комбінування вже існуючих програм-утиліт, набір яких в будь-якій сучасній версії Linux вельми різноманітний.

Існує декілька різновидів шел:

- Bourne Shell - самий старший з шелів, який був написаний Стефаном Борном в Лабораторії Белла. Цей шелл є шелом за замовчуванням для HP-UX користувачів і довгий час був стандартом де факто. Bourne Shell не має у своєму арсеналі ні інтерактивних можливостей ні складних програмних конструкцій на відміну від C та Korn Шелл.
- C Shell. Цей шел був розроблений Біллом Джоєм в Каліфорнійському Університеті Берклі. Його синтаксис має схожість з мовою програмування C. Він також має інтерактивний інтерфейс наприклад історію команд та розкриття імен файлів.
- Korn Shell. Він є відносно новим шолом розробленим Девідом Корном в Лабораторії Белла і є догори сумісним з більшістю можливостей Bourne Shell. Так само як і C shell він має інтерактивні можливості, але виконується швидше має розширені можливості редагування командного рядка.
- POSIX shell. Цей шел базується на стандарті визначеному в Portable Operation System Interface (POSIX) - IEEE P1003.2. Цей стандарт був розроблений для прикладних і системних програмістів. Він фактично визначає стандарт на інтерфейс операційної системи. Більшість можливостей POSIX Shell дуже сильно схожі з аналогічними можливостями Korn Shell-a. POSIX Shell має теж ім'я що і Bourne Shell тому він поміщений в/usr/bin/posix директорію на відміну від Bourne Shell, який знаходиться в директорії/usr/bin.
- Key Shell. Це оболонка для Kourn Shell-a розроблена фірмою Hewlett-Packard. Вона дозволяє використовувати меню і онлайн допомогу допомагаючи в побудові команд та виконанні ряду завдань, які часто зустрічаються, таких як перегляд, редагування та друк файлів, перегляд

вмісту директорії т.п. Побудована вона таким чином що користувач може сам надалі розширювати її можливості.

- Майже у всіх дистрибутивах Linux для користувачів за замовчуванням призначається командна оболонка Bash. Назва цього шела розшифровується як Bourne Again Shell. Він би розроблений консорціумом Free Software Foundation і незважаючи на те що за замовчуванням він відсутній у стандартній поставці HP-UX 10.20, в наслідок своїх потужних функціональних можливостей він користується величезною популярністю серед користувачів і адміністраторів HP-UX. Його інтерпретатор команд сумісний з Bourne Shell. Також він вмістив в собі корисні можливості Korn С Шел. Він розроблявся у відповідності зі специфікаціями IEEE POSIX Shell.

Також використовуються Ansible ролі для розгортання кластерів Kubernetes та PostgreSQL, за необхідністю.

Ansible — це інструмент оркестрації та автоматизації з відкритим вихідним кодом, який використовується для підготовки програмного забезпечення, керування конфігурацією та розгортання програм. Вперше Ansible був розроблений 2012 року Майклом ДеХааном, творцем Cobbler and Func. Компанія, що фінансує Ansible, було придбано RedHat, Inc. 2015 року. RedHat була придбана IBM у 2019 році, тому тепер Ansible живе під егідою IBM.

Ansible працює у Windows та Unix-подібних операційних системах, надаючи інфраструктуру у вигляді коду. Має власну декларативну мову програмування для управління та налаштування системи. Він може підключатися до хмарних середовищ, таких як Amazon AWS та Microsoft Azure, щоб допомогти вам керувати інфраструктурою та розгортанням коду та автоматизувати їх.

Ansible простий в установці, легко підключається до клієнтів і містить багато функцій. Він заснований на push та підключається до клієнтів через SSH, тому не вимагає наявності агента на клієнті. Відправляючи модулі клієнтам, модулі клієнта виконуються локально, а вихідні дані відправляються назад на сервер

Ansible. Він використовує SSH-ключ для спрощення процесу підключення клієнтів. Імена хостів, IP-адреси та порти SSH зберігаються у файлах інвентаризації. Як тільки файл інвентаризації створено та заповнено, Ansible може його використовувати.

2.3 Система керування базами даних

У якості СКБД використовується PostgreSQl.. Це безкоштовна СКБД з відкритим кодом. На даний момент вона займає четверте місце в рейтингу популярності в своєму класі. Має наступні переваги:

1. Вільний доступ

Будь-який фахівець може безкоштовно завантажити, встановити СУБД та одразу розпочати роботу з базами даних. Якщо розмістити базу та СУБД у хмарному сховищі, доведеться платити.

2. Можна встановити будь-яку платформу

PostgreSQL підходить для роботи в будь-якій операційній системі: Linux, MacOS, Windows. Користувач отримує систему "з коробки" - щоб встановити та використовувати програму, не потрібні додаткові інструменти.

3. Підтримує різні формати даних

PostgreSQL підтримує багато різних типів і структур даних, у тому числі мережеві адреси, дані в текстовому форматі JSON та геометричні дані для координат геопозицій. Всі ці формати можна зберігати та обробляти у СУБД.

Працюючи з PostgreSQL можна створювати власні типи даних, їх називають користувальницькими. Типи даних користувача потрібні, щоб спростити роботу з базою або встановити обмеження. Допустимо, є прилад, який показує лише цілі числа від 1 до 5. Показання цього приладу потрібно вносити до бази. Можна створити тип даних, який складається тільки з чисел 1, 2, 3, 4, 5. Тоді введення інших значень приведе до помилки, а значить, не порушить дані.

Запит PostgreSQL для створення такого типу даних виглядає так:

```
CREATE TYPE entity_source AS enum (1, 2, 3, 4, 5);
```

4. Дозволяє працювати з великими розмірами даних

Розмір бази даних PostgreSQL не обмежений і залежить від того, скільки вільної пам'яті є в місці зберігання: на сервері, локальному комп'ютері або в хмарі.

Максимальний розмір таблиці - 32 терабайти. Цього цілком достатньо для зберігання даних компаній типу Amazon. Один рядок у базі даних не може перевищувати 1,6 терабайт, а максимальний розмір одного осередку - 1 гігабайт. До такого осередку можна додати навіть відео.

PostgreSQL дозволяє працювати з базами даних та елементами в них таких розмірів, які на практиці здебільшого не потрібні. Тому ці обмеження можна назвати умовними.

5. Відповідає вимогам ACID

Абревіатура ACID розшифровується так:

- атомарність (від англ. atomicity),
- узгодженість (від англ. consistency),
- ізольованість (від англ. isolation),
- стійкість (від англ. durability).

Це чотири вимоги до надійної роботи систем, які обробляють дані в режимі реального часу. Якщо всі вимоги виконуються, дані не губляться через технічні помилки або збої в роботі обладнання.

Атомарність системи – можливість здійснювати транзакції. Транзакції — це операції чи дії, які відбуваються повністю чи взагалі відбуваються.

На відміну від атомарності, узгодженість означає, що транзакція буде виконана лише в тому випадку, якщо вона не порушує узгодженість даних у базі.

Наприклад, потрібно транзакцією перевести гроші з однієї таблиці до іншої. У одній таблиці гроші зберігаються у вигляді числових значень. В іншій таблиці є поля з датами. Дати - це також числа. Під час транзакції числові дані можуть потрапити до поля з датами. Це не суперечить атомарності, але узгодженість буде порушена: дані про гроші не можуть зберігатися у просторі з датами. За узгодженості системи така транзакція виконуватися не повинна.

Ізольованість системи означає, що паралельні дії не впливають одна на одну. Наприклад, грошовий переказ між двома рахунками не повинен впливати на третій рахунок.

Стійкість системи означає, що вже виконана транзакція не скасується через технічні проблеми, наприклад, якщо відключать світло в серверній.

PostgreSQL відповідає всім чотирьом вимогам ACID та забезпечує збереження даних при виконанні транзакцій та інших робіт.

6. Підтримує всі функції, які є у сучасних базах

Наприклад, у PostgreSQL є віконні функції, вкладені транзакції та тригери.

Віконні функції дозволяють вибрати певні записи в таблиці та робити обчислення з ними в окремому стовпці. Наприклад, можна додати в таблицю з даними інтернет-магазину стовпець з датою першого відвідування користувачем сайту. Цей стовпець стане в нагоді, якщо знадобиться розрахувати LTV (від англ. customer lifetime value).

Вкладеними називають транзакції всередині інших. Наприклад, виконання серії переказів траншами у межах одного договору. Припустимо, було виконано п'ять транзакцій, а на шостій виникли проблеми. Відкотитися мають усі попередні транзакції, які були всередині однієї великої. Відкат транзакції – це скасування всіх змін даних, викликаних цією транзакцією.

У PostgreSQL можна створювати тригери - функції, які автоматично запускаються за певних умов. Наприклад, можна створити тригер, який запускається при видаленні даних закритої компанії з бази. Створений тригер автоматично додасть у потрібному полі іншої таблиці запис: «дані про компанію видалено, компанія закрита».

7. Є власний діалект мови SQL

Приблизно 80% операцій у СКБД виконується за допомогою запитів класичною мовою SQL. Для деяких дій у PostgreSQL є свої запити. Вони значно коротші, тому працювати з ними простіше.

Наприклад, потрібно перевести рядок '365' та цілий тип (int). Запит на класичному SQL виглядає так:

CAST('365' AS int)

У PostgreSQL запит можна скоротити:

'365'::int

8. Можна настроїти синхронне дублювання даних

Робота з PostgreSQL підтримує логічну реплікацію. Реплікація – це збереження копії бази даних. Копія може знаходитись на іншому сервері. У разі логічної реплікації будь-які зміни синхронізуються у всіх копіях бази даних незалежно від місця їх зберігання. Це означає, що скрізь зберігатиметься одна версія бази даних.

9. Можна без втрат перенести дані з іншої СКБД

Обсяг даних великих компаній може бути розміром 10 терабайт. Їхнє перенесення займе час і призупинить роботу. Невеликі компанії або стартапи зможуть «переїхати» в PostgreSQL з іншої СКБД швидко, не втративши нічого в процесі. Всі дані можна перенести за допомогою спеціальних інструментів.

2.4 Система обміну повідомленнями

Для взаємодії між мікросервісами в нашій ІС використовується такий message-broker як Apache Kafka. Це розподілена система обміну повідомленнями між серверними програмами в режимі реального часу. Завдяки високій пропускній здатності, масштабованості та надійності застосовується в компаніях, що працюють з великими обсягами даних. Написана мовами Java та Scala.

Apache Kafka— ефективний інструмент для роботи серверних проектів будь-якого рівня. Завдяки гнучкості, масштабованості та стійкості до відмови використовується в різних напрямках IT-індустрії, від сервісів потокових відео до аналітики Big Data.

Стисло архітектуру системи повідомлень можна охарактеризувати так:

розподіл - окремі вузли системи розташовуються на декількох апаратних платформах (кластерах). Це забезпечує їй високу стійкість до відмови;

масштабованість - систему можна нарощувати за рахунок простого додавання нових вузлів (брокерів повідомлень).

В архітектурі Apache Kafka ключовими є концепції:

продюсер (producer) - додаток або процес, що генерує і посилає дані (що публікує повідомлення);

споживач (consumer) - додаток або процес, який приймає згенероване продюсером повідомлення;

повідомлення - пакет даних, необхідний для здійснення будь-якої операції (наприклад, авторизації, оформлення покупки або підписки);

брокер - вузол (диспетчер) передачі повідомлення від процесу-продюсера додатку-споживачеві;

топік (тема) — віртуальне сховище повідомлень (журнал записів) однакового чи схожого змісту, з якого додаток-споживач отримує необхідну йому інформацію.

У спрощеному вигляді робота Apache Kafka виглядає так:

Додаток-продюсер створює повідомлення та відправляє його на вузол Kafka.

Брокер зберігає повідомлення у топіці, на який підписані додатки-споживачі.

Споживач у разі потреби робить запит у топік і отримує з нього потрібні дані.

Повідомлення зберігаються в Kafka у вигляді журналу коммітів - записів, розміщених у суворій послідовності. Їх можна лише додавати. Видаляти чи коригувати не можна. Повідомлення зберігаються в тій послідовності, в якій надійшли, їхнє зчитування ведеться зліва направо, а відстеження за зміною порядкового номера. Брокери Kafka не обробляють записи - тільки поміщають їх у

тему на кластері. Зберігання може тривати протягом певного періоду або досягнення заданого порога.

Якщо тема надто розростається, для спрощення та прискорення процесу вона поділяється на секції. Кожна секція містить повідомлення, що згруповані за об'єднуючою ознакою. Наприклад, масив запитів користувача можна згрупувати за першою літерою імені користувачів. Так додатку-споживачеві не доведеться переглядати весь топік — лише потрібну тему, що прискорює процес обміну повідомленнями.

3. ОПИС ІНФОРМАЦІЙНОЇ СИСТЕМИ. РОЗГОРТАННЯ НА BARE-METAL ІНФРАСТРУКТУРІ

3.1. Загальна інформація про інформаційну систему

Головною метою вказаної ІС є надання можливості обробки геоінформаційних даних, пов’язаних з основними даними ІС. Дана можливість надає зручності в аналізі та керуванні даними, які розподілені географічно. Основними замовниками є великі фірми, які мають декілька регіональних підрозділів.

ІС забезпечує виконання наступних основних функцій:

- Обслуговування запитів інформаційно-аналітичних служб інших систем та користувачів замовника;
- Формування та надання інформації необхідної для прийняття рішень в управлінських сферах;
- Отримання відомостей по узгодженими з замовником параметрам;
- Надання довідкової інформації для підрозділів замовника;
- Оперативна реалізація аналітичних та управлінських потреб за переліком атрибутів ШС.

Загальна структура інформаційної системи зображена на Рисунок 3 та складається з восьми складових частин, на які вона поділяється за функціональними ознаками:

1. Kubernetes кластер – складає ядро системи, забезпечує взаємодію всіх її частин та є середовищем виконання мікросервісів
2. Сховище медіаданих S3 Minio – об’єктно-орієнтоване сховище для об’єктів ІС
3. Кластер бази даних PostgreSQL – основна СКБД ІС
4. Термінальна ферма геододатків – термінальне середовище для розгортання десктопних геододпатків

5. Сервіс даних OpenStreetMap – локальний варіант відкритого картографічного сервісу
6. Картографічний сервіс ArcGIS – серверна частина картографічного сервісу
7. Сервіс взаємодії з зовнішніми системами – базове налащтування серверу для взаємодії з іншими ІС замовника
8. Сервіс обміну повідомленнями Apache Kafka – середовище для обміну повідомленнями та об’єктами між мікросервісами в межах ІС

Рисунок 3 - Структурна схема інформаційної системи

Основним завданням ІС є збереження та надання користувачам повної, достовірної та структурованої інформації про виробничі об'єкти підприємства, їх технічні, діагностичні, експлуатаційні та просторові характеристики, а також інформації про технічний стан обладнання.. Робота з просторовими даними здійснюється за допомогою розробленого набору інструментів і функціональних властивостей з обробки різнотипних і різномасштабних картографічних ресурсів, даних дистанційного зондування, матеріалів спеціалізованих графічних побудов та

паспортної інформації нафтогазових об'єктів, просторово описаних в базі даних системи.

До основних можливостей ШС належить :

- Облік та ведення технічної інформації про виробничі об'єкти;
- Формування логічних багатокритеріальних запитів до технологічних об'єктів;
- Облік споживачів та контрагентів;
- Ведення виробничої нормативно-технічної документації;
- Організація доступу до проектно-кошторисної документації (ПКД);
- Супровід метаданих;
- Супровід довідників нормативно-довідкової інформації (НДІ);
- Ведення оргструктури підприємств;
- Управління просторовими даними;
- Робота з технологічною графікою;
- Побудова однолінійних діаграм протяжних об'єктів;
- Побудова поздовжнього профілю лінійного об'єкта;
- Побудова та візуалізація даних на тривимірних моделях;
- Побудова тематичних карт;
- Робота із земельно-кадастровою інформацією;
- Побудова та облік зон впливу вражаючих факторів на картографічних матеріалах;
- Порівняння проектної та виконавчої документації;
- Редагування графічних матеріалів;
- Редагування картографічних матеріалів і зміна їх положення;
- Імпорт кадастрової інформації;

Розроблена система розгортання та первинного налаштування ІС, є невід’ємною частиною ІС розробленою відповідно вимог замовника.

Схема розроблених базових структур даних у відповідності до бази даних зображена на схемі ІАЛЦ.467200.006 Д3.

3.2. Розгортання та налаштування ІС

Виходячи з основної мети та вимог замовників ІС має використовувати мінімальну кількість з’єднань з зовнішнім світом та мережею Internet. Для цього як вказано на Рис. 3 є один з компонентів й відповідає за взаємодію з зовнішніми системами. Також для зменшення залежності від мережі Internet буде розгортатись локальний сервіс даних OpenStreetMap.

Ще одним аспектом локального використання необхідних ресурсів є створення станції адміністрування ІС, на якій розташовані програмні засоби для адміністрування системи та локальний репозиторій образів контейнерів Harbor. Даний репозиторій, використовується для контейнерів компонентів системи та опціонально може бути використаний для зберігання системних контейнерів Kubernetes та іншого системного ПЗ, яке використовується в системі, при вимогах замовника строгої ізоляції ІС при роботі.

3.2.1. Розгортання та налаштування основних структурних компонентів ІС

Для уніфікації системи, під різних замовників, був розроблений шаблон дистрибутиву для розгортання. Він складається з переліку директорій та файлів дистрибутиву, файлу конфігурації розгортання під певного замовника та скриптів запуску конфігурації та розгортання.

Перелік скриптів та їх опис наведені в Таблиці 1.

Таблиця 1

Перелік каталогів та скриптів

Шлях до каталогу	Файли, що розташовані в каталозі	Опис файлу
/srv/ITTEAM/drawio		Каталог с скриптами розгортання системи візуалізації блок-схем та технологічної графіки

	drawio-set-localtime.yaml	Скрипт налаштування часу та часової зони
	eam-drawio-deployment.yaml	Скрипт розгортання поду сервісу
	eam-drawio-ingress.yaml	Скрипт конфігурації ingress для поду сервісу
	eam-drawio-svc.yaml	Скрипт створення мережевого сервісу
	eam-local-ca-configmap.yaml	Скрипт додавання кастомних сертифікатів SSL
/srv/ITTEAM/geoserver		Каталог з скриптами розгортання сервісу роботи з просторовими даними Geoserver та сервісу керування файлами для нього
	geoserver-context-configmap.yaml	Скрипт створення configmap для поду сервісу Geoserver
	geoserver-deployment.yaml	Скрипт розгортання поду сервісу Geoserver
	geoserver-filemanager-deployment.yaml	Скрипт розгортання поду сервісу керування файлами
	geoserver-filemanager-ingress.yaml	Скрипт конфігурації ingress для поду сервісу керування файлами
	geoserver-filemanager-svc.yaml	Скрипт створення мережевого сервісу керування файлами
	geoserver-geodata.yaml	Скрипт створення persistent volume для поду Geoserver
	geoserver-ingress.yaml	Скрипт конфігурації ingress для поду сервісу Geoserver

	geoserver-set-localtime.yaml	Скрипт налаштування часу та часової зони
	geoserver-svc.yaml	Скрипт створення мережевого сервісу Geoserver
/srv/ITTEAM/ingress		Каталог з скриптами розгортання сервісу Ingress Controller на базі Nginx
/srv/ITTEAM/integr		Каталог розгортання серверу інтеграції з зовнішніми системами
	integr-server-create.sh	Загальний скрипт створення
	server-install.sh	Локальний скрипт налаштування серверу
	server-setup	Каталог програмного забезпечення для розгортання на сервері
	setup-server.sh	Скрипт налаштування та переносу локального скрипта на сервер
	vm_create	Каталог з налаштуваннями створення віртуальної машини
/srv/ITTEAM/k8s-create		Каталог з налаштуваннями створення віртуальних машин кластеру Kubernetes
	create_vm_cluster.sh	Скрипт розгортання та налаштування віртуальних машин кластера Kubernetes
	fix_node.sh	Системний скрипт для роботи worker нод

	main.tf vm.tf	Загальний файл конфігурації розгортання Terraform Скрипт опису конфігурацій розгортання віртуальних машин Terraform
/srv/ITTEAM/kafka	 kafka-ui-ingress.yaml kafka-ui-values.yaml	Каталог з скриптами роз гортання серверу обміну повідомлення ми Apache Kafka та допоміжними утилітами до нього Скрипт конфігурації ingress для утиліти Kafka-ui Скрипт кастомізації розгортання утиліти Kafka-ui
/srv/ITTEAM/kubespray		Набір Ansible-ролей для встановлення та конфігурації Kubernetes
/srv/ITTEAM/loki	 loki-stack-values.yaml loki-store-pvc.yaml	Каталог з скриптами розгортання сервісу агрегації логів Grafana Loki Скрипт кастомізації розгортання сервісу агрегації логів Grafana Loki Скрипт створення мережевого сервісу
/srv/ITTEAM/minio	 minio-install.sh minio-server-create.sh	Каталог розгортання серверу сховища медіаданих S3 Minio Локальний скрипт налаштування серверу Загальний скрипт створення

	server	Каталог програмного забезпечення для розгортання на сервері
	setup-server.sh	Скрипт налаштування та переносу локального скрипта на сервер
	vm_create	Каталог з налаштуваннями створення віртуальної машини
/srv/ITTEAM/monitoring		Каталог з скриптами розгортання сервісу моніторингу Prometheus та візуалізації Grafana
	grafana-service.yaml	Скрипт створення мережевого сервісу Grafana
	grafana-store-pvc.yaml	Скрипт створення persistent volume для сервісу Grafana
	kube-prometheus-stack-values.yaml	Скрипт кастомізації розгортання стеку сервісу моніторингу та візуалізації
	monitoring-grafana-ingress.yaml	Скрипт конфігурації ingress для поду сервісу Grafana
	monitoring-prometheus-ingress.yaml	Скрипт конфігурації ingress для поду сервісу Prometheus
	prometheus-service.yaml	Скрипт створення мережевого сервісу Prometheus
/srv/ITTEAM/postgres		Каталог з скриптами розгортання сервісу БД PostgreSQL та утилітами до нього
	pg-create	Каталог з скриптами створення початкової БД

	prometheus-postgres-exporter-values.yaml	Скрипт кастомізації розгортання сервісу адаптеру моніторингу БД PostgreSQL
	server-install.sh	Локальний скрипт налаштування серверу
	setup-server.sh	Скрипт налаштування та переносу локального скрипта на сервер
	stand-db-server-create.sh	Загальний скрипт створення одиночного серверу БД PostgreSQL
	vm_create	Каталог з налаштуваннями створення віртуальної машини
/srv/ITTEAM/storage		Каталог з скриптами розгортання сервісу створення persistence volume для використання в кластері Kubernetes
/srv/ITTEAM/system	metallb-config-crd.yaml	Скрипт конфігурації для локальної підсистеми балансування навантаження кластеру Kubernetes
	podpreset.yaml	Скрипт створення та конфігурації сервісу додавання налаштувань в існуючі поди
/srv/ITTEAM/		Основний каталог шаблону розгортання
	config_itteam.sh	Скрипт конфігурації шаблону під налаштування замовника

	config_loader.sh	Скрипт конфігурації шаблону завантаження даних під налаштування замовника
	create_itteam.sh	Основний скрипт створення системи та її налаштування
	itteam.config	Файл з основними налаштуваннями під замовника
	stack.config	Файл налаштування списку компонентів системи, що будуть встановлюватись

Перед запуском основного скрипта створення та налаштування системи, необхідно внести , узгоджені з замовником, параметри для налаштування шаблону до файлу itteam.config. Після цього необхідно запустити на виконання скрипти конфігурації шаблонів розгортання config_itteam.sh та шаблону завантаження даних config_loader.sh.

Після Їх успішного виконання, необхідно налаштувати список компонентів, які будуть розгортатись та необхідність завантаження первинних даних в файлі stack.config

Після всього запускаємо основний файл розгортання create_itteam.sh. При його виконанні на консоль будуть виводитись логи процесу.

3.2.2. Розгортання та налаштування базових компонентів ІС

Для розгортання базового набору сервісів ІС був також створений шаблон розгортання, який знаходиться в загальному каталозі шаблону системи /srv/ITTEAM/eam

Також розроблений шаблон первинного наповнення системи, якмй знаходиться в каталозі /srv/DATA

Перелік скриптів та їх опис наведені в Таблиці 2.

Таблиця 2

Шлях до каталогу	Файли, що розташовані в каталозі	Опис файлу
/srv/DATA		Каталог первинного наповнення системи
	db	Каталог з файлом дампу БД POstgreSQL
	geoserver	Каталог з преналаштованими файлами конфігурації сервісу роботи з просторовими даними Geoserver
	loader.sh	Основний скрипт завантаження даних
	minio	Каталог з файлами для завантаження на сервер сховища медіаданих S3 Minio
	recr_db.sql	SQL скрипт перестворення БД PostgreSQL
/srv/ITTEAM/eam/dashboard		Каталог з файлами конфігурації для кастомних панелей для сервісу візуалізації \Grafana
	eam-cluster-dashboard-configmap.yaml	Палень загального моніторингу стану кластера Kubernetes
	eam-ingress-dashboard-configmap.yaml	Панель моніторингу стану Ingress
	eam-postgres-dashboard-configmap.yaml	Панель моніторингу стану БД PostgeSQL

	eam-service-external-logs-dashboard-configmap.yml	Панель перегляду логів кластерних сервісів ІС
	eam-service-internal-logs-dashboard-configmap.yaml	Панель перегляду логів зовнішніх сервісів системи
	eam-services-http-throughput-dashboard-configmap.yaml	Панель перегляду метрик сервісів системи
	eam-services-metrics-dashboard-configmap.yaml	Панель перегляду метрик сервісів системи
/srv/ITTEAM/eam/datasource		Каталог конфігурації шляхів даних для системи візуалізації Grafana
	eam-grafana-datasource.yaml	Файл конфігурації шляхів даних для системи візуалізації Grafana
/srv/ITTEAM/eam/eam-service		Каталог з загальним Helm chart для деплою сервісів системи на кластер Kubernetes
/srv/ITTEAM/eam/env		Каталог з файлами додаткових скриптів створення змінних для сервісів та налаштування проксювання зовнішніх сервісів
/srv/ITTEAM/eam/metrics		Каталог скриптів створення кастомних метрик для системи моніторингу Prometheus
	metadata-jsonapi-topic-lag-prometheusrule.yaml	Скрипт опису кастомних метрик для автомаштабування сервісів системи

/srv/ITTEAM/eam/service-conf		Каталог з скриптами конфігурацій для використання з загальним Helm chart сервісів системи
/srv/ITTEAM/eam/ssl		Каталог з сертифікатами SSL для налаштування HTTPS з'єднань в Ingress системи
/srv/ITTEAM/eam/	set-eam-service.sh	Загальний каталог базових компонентів ІС Скрипт розгортання базових компонентів ІС

Виконання даних скриптів активується в файлі налаштування списку компонентів системи, що будуть встановлюватись stack.config

3.3. Алгоритми роботи.

Для виконання поставленого технічного завдання було розроблено відповідні алгоритми. Порядок їх виконання викладений на схемі ІАЛЦ.467200.008 Д4. Для зручності оптсу розділили даний алгоритм на два етапи: розгортання та наповнення ІС

3.3.1. Процедура розгортання ІС.

Першим етапом виконується налагодження станції керування ІС. На цій станції розташовується шаблон розгортання та завантаження даних , локальний репозиторій образів контейнерів Harbor та утіліти моніторингу талбслуговування ІС, наприклад PgAdmin – ПЗ для роботи з СКБД PostgreSQL, Lens – ПЗ для керування кластером Kubernetes, браузери та офісне ПЗ для роботи з даними системи. Також, по вимогам замовника, дана станція може виконувати роль шлюзу, DNS та NTP сервера, при конфігурації з ізольованою мережею. При даних умовах весь мережевий трафік буде проходити крізь дану стацію, що обмежує зв'язки мережі з зовнішнім світом.

Після налагодження станції керування та мережи для ІС., на неї переноситеся шаблон розгортання та завантаження даних, пакет з образами контейнерів системи та за потреби стартове наповнення. Шаблони розгортання та завантаження переносяться в у відповідне місце – папку /srv. Спочатку завантажуються образи контейнерів в локальний репозиторій образів та перевіряється їх цілісність за допомогою контрольних сум.

Наступним етапом є приведення файлу з основними налаштуваннями системи itteam.config , до стану узгодженого х замовником. Приклад наповнення приведено нижче:

```
NETWORK=172.16.77.0
SUBNET=24
GATEWAY=172.16.77.1
NTP=172.16.77.1
DNS=172.16.77.1

VSPHERE_SERVER=10.10.20.22
VSPHERE_USER=administrator@vcenter7.it-transit
VSPHERE_PASSWD=1q2w3e4r
VSPHERE_FOLDER="/EAM"
VSPHERE_DC=it-transit
VSPHERE_CLUSTER=it-transit
VSPHERE_DATASTORE=datastore_27_6
VSPHERE_NETWORK="VM Network"

DOMAIN=it-transit.com
SEGMENT=eam
WWW_URI=www
WWW_IP=172.16.77.201
```

```
CONTROL_NODE_URI=control
MASTER1_URI=master1
MASTER1_IP=172.16.77.210
MASTER1_VMNAME=77_210_master1
MASTER2_URI=master2
MASTER2_IP=172.16.77.211
MASTER2_VMNAME=77_211_master2
MASTER3_URI=master3
MASTER3_IP=172.16.77.212
MASTER3_VMNAME=77_212_master3
WORKER1_URI=node1
WORKER1_IP=172.16.77.214
WORKER1_VMNAME=77_214_node1
WORKER2_URI=node2
WORKER2_IP=172.16.77.215
WORKER2_VMNAME=77_215_node2
WORKER3_URI=node3
WORKER3_IP=172.16.77.216
WORKER3_VMNAME=77_216_node3
FLANEL_INTERFACE_REGEXP="172\\\\\\\\.16\\\\\\\\.77\\\\\\\\.\\\\\\\\d{1,3}"

MINIO_URI=minio
MINIO_IP=172.16.77.220
MINIO_VMNAME=77_220_minio
MINIO_DISK_SIZE=400
MINIO_ADMIN_PASSWD=1q2w3e4r
MINIO_CONSOLE_PASSWD=1q2w3e4r
MINIO_MEDIADATA_USER=mediadata
MINIO_MEDIADATA_PASSWD=1q2w3e4r
```

MINIO_MEDIADATA_BUCKET=eammedia

POSTGRE_URI=postgre
POSTGRE_IP=172.16.77.225
POSTGRE_VMNAME=77_225_pgdb
POSTGRE_PORT=5432
POSTGRE_DISK_SIZE=250
POSTGRE_PASSWD=1q2w3e4r

KAFKA_DISK_SIZE=60
PROMETHEUS_ENDPOINT_URI=prometheus
PROMETHEUS_DISK_SIZE=70
GRAFANA_ADMIN_PASSWD=1q2w3e4r
GRAFANA_DISK_SIZE=10
LOKI_DISK_SIZE=60

INTEGR_URI=integr
INTEGR_IP=172.16.77.219
INTEGR_VMNAME=77_219_integr

ITTGIS_DB_USER=services
ITTGIS_DB_PASSWD=1q2w3e4r
ITTGIS_GEOSERVER_ADMIN_PASSWD=geoserver
ITTGIS_GEOSERVER_DATA_DISK_SIZE=120

В даному прикладі мережа системи не є ізольованою.

Після цього етапу виконуються скрипти конфігурації шаблонів розгортання config_itteam.sh та шаблону завантаження даних config_loader.sh. Вони перетворюють відповідні шаблони на вже налаштовану систему розгортання та наповнення ІС. Далі у файлі stack.config налаштовується перелік компонентів, які

будуть розгорнуті та необхідність в завантаженні первинних даних. Приклад файлу наведено нижче:

POSTGRESQL=true

MINIO=true

INTEGR=true

K8S=true

ITTEAM=false

LOAD_DATA=false

За допомогою налаштувань цього файлу є можливість розгортати та завантажувати дані в систему в декілька етапів.

Коли шаблони вже налаштовані та заданий перелік компонентів то запуск основного процесу виконується командою:

./create_ittpims.sh

Розпочнеться процес з виводом логів на екран як показано на Рисунок 4-6.

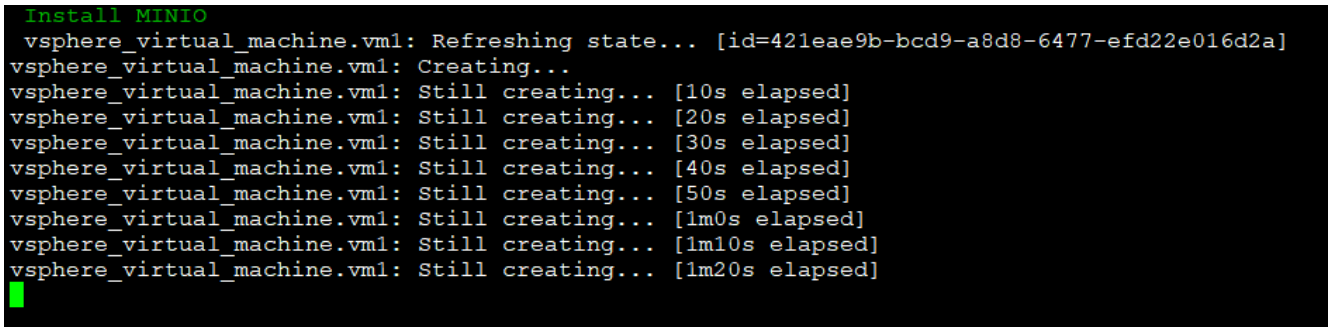

Рисунок 4 - Вигляд консолі з логами процесу розгортання. Створення ВМ

```
TASK [kubernetes/preinstall : Update package management cache (APT)] ***********
skipping: [node4]
skipping: [node5]
skipping: [node6]
ok: [node1]
ok: [node2]
ok: [node3]

TASK [kubernetes/preinstall : Remove legacy docker repo file] ******************
skipping: [node1]
skipping: [node2]
skipping: [node3]
ok: [node4]
ok: [node5]
ok: [node6]

TASK [kubernetes/preinstall : Install python3-dnf for latest RedHat versions] ***
skipping: [node1]
skipping: [node2]
skipping: [node3]
skipping: [node4]
skipping: [node5]
skipping: [node6]
```

Рисунок 5 - Вигляд консолі з логами процесу розгортання. Створення кластеру Kubernetes

```
 Install MetalLB
 namespace/metallb created
NAME: metallb
LAST DEPLOYED: Sun Jun 11 21:47:48 2023
NAMESPACE: metallb
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
MetalLB is now running in the cluster.

Now you can configure it via its CRs. Please refer to the metallb official docs
on how to use the CRs.
ipaddresspool.metallb.io/eam created
l2advertisement.metallb.io/eam-l2advertisement created

Press any key to continue or Ctrl+C to exit...

 Install NFS Datastore
 NAME: system-nfs-datastore
LAST DEPLOYED: Sun Jun 11 21:48:26 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
storageclass.storage.k8s.io/data-single-store patched

Press any key to continue or Ctrl+C to exit...
```

Рисунок 6 - Вигляд консолі з логами процесу розгортання. Розгортання основних структурних компонентів

При необхідності мати логи, для подальшого аналізу, проводимо запуск основного процесу командою:

./create_ittpims.sh 2>&1 | tee setup.log

Результатом буде файл setup.log, в каталозі запуску, в якому буде повна копія того що виводилось на екран під час роботи скрипта.

Для створення сервісів системи використовується загальний Helm chart для якого при запуску використовуються кастомізовані файли значень, написані під конкретний сервіс розташовані в каталозі /srv/ITTEAM/eam/service-conf. Приклад такого файлу:

```
# Default values for eam-service.
# This is a YAML-formatted file.
# Declare variables to be passed into your templates.

AppVersion: "09.02"
namespace: eam
fullnameOverride: ""
replicaCount: 1
stackName: ""

image:
  repository: "#CONTROL_NODE_URI#.#SEGMENT#.#DOMAIN#/eam"
  imagePrefix: "eam"
  NameOverride: ""
  pullPolicy: Always
  tag: "pims"

imagePullSecrets: []

podAnnotations: {}
```

```
ports: {}
#  - name: http
#    containerPort: 80
#    protocol: TCP

envFrom:
  create: true
  mainEnv: true
  serviceEnv: true

livenessProbe: #{}
  initialDelaySeconds: 10
  periodSeconds: 10

readinessProbe: #{}
  initialDelaySeconds: 20
  periodSeconds: 10

resources: {}
  # We usually recommend not to specify default resources and to leave this as a conscious
  # choice for the user. This also increases chances charts run on environments with little
  # resources, such as Minikube. If you do want to specify resources, uncomment the following
  # lines, adjust them as necessary, and remove the curly braces after 'resources:'.
  # limits:
  #   cpu: 100m
```

```
  #   memory: 128Mi
  # requests:
  #   cpu: 100m
  #   memory: 128Mi

nodeSelector: {}

tolerations: []

affinity: {}

podSecurityContext: {}
  # fsGroup: 2000

securityContext: {}
  # capabilities:
  #   drop:
  #   - ALL
  # readOnlyRootFilesystem: true
  # runAsNonRoot: true
  # runAsUser: 1000

envConfigmap:
  create: true
  data:
    ITT_EAM_AUTH_DB_SCHEMA: auth
    ITT_EAM_AUTH_DB_APP_CON_NAME: authService
    ITT_LDAP_URL: "ldap://172.16.77.240:389"
    ITT_LDAP_BASE: "cn=users,dc=tsoua,dc=local"
```

```
    ITT_LDAP_USERDN: "cn=Alex Smith,cn=users,dc=tsoua,dc=local"
    ITT_LDAP_PASSWORD: "!Q2w3e4r%T"
    ITT_EAM_USE_AD: "false"

serviceAccount:
  # Specifies whether a service account should be created
  create: false
  # Annotations to add to the service account
  annotations: {}
  # The name of the service account to use.
  # If not set and create is true, a name is generated using the fullname template
  name: ""

service:
#  type: ClusterIP
  port: 8081
  targetPort: 8081

ingress:
  enabled: true
  annotations: #{}
    nginx.ingress.kubernetes.io/rewrite-target: /$1
    ingress.kubernetes.io/enable-cors: "true"
    kubernetes.io/ingress.class: "nginx"
    # kubernetes.io/ingress.class: nginx
    # kubernetes.io/tls-acme: "true"
  hosts:
    - host: #WWW_URI#.#SEGMENT#.#DOMAIN#
      paths:
```

```
      - path: /auth/(.*)
        pathType: Prefix
    tls: []
    #  - secretName: chart-example-tls
    #    hosts:
    #      - chart-example.local


  serviceMonitor:
    # Specifies whether a service account should be created
    create: true
    # Annotations to add to the service account
    annotations: {}
    # The name of the service account to use.
    # If not set and create is true, a name is generated using the fullname template
    additionalLabels:
      prometheus: prometheus
      release: prometheus
    endpoints:
      interval: 5s
      port: http
      path: /actuator/prometheus
```

3.3.2. Процедура наповнення

Окремою частиною процесу розгортання ІС є необов'язкова можливість первинного завантаження даних в систему. Для цього в налаштований шаблон завантаження необхідно додати відповідні дані:

- dump БД PostgreSQL

- преналаштовані файли конфігурації сервісу роботи з просторовими даними Geoserver
- файли для завантаження на сервер сховища медіаданих S3 Minio

Ці дані можна додавати в будь-якій необхідній комбінації.

Запуск процесу може відбуватись разом з розгортанням системи при вказаному параметрі LOAD_DATA=true в файлі налаштувань компонентів системи stack.config або окремо від розгортання командою:

/srv/LOAD/loader.sh restore all 2>&1 | tee load.log

Можливі варіанти окремого завантаження конфігурації Geoserver чи медіаданих. Також цім скриптом є можливість робити резервні копії цих даних командою:

/srv/LOAD/loader.sh backup all 2>&1 | tee backup.log

Для цього режиму варіанти аналогічні режиму завантаження.

Після виконання ми отримуємо відповідний лог-файл в каталозі запуску

ВИСНОВКИ

Метою дипломного проекту було створення методів розгортання інформаційної системи створеної на основі мікросервісної архітектури на bare-metal інфраструктурі. Система повинна дозволяти виконувати достатньо різнопланові завдання використовуючи певні інфраструктурні рішення визначенні замовником.

В ході виконання дипломного проекту було:

- проаналізовано існуючі рішення інформаційних систем;
- проведено аналіз предметної області;
- розроблено алгоритм розгортання інформаційної системи з подальшим її первинним налагодженням;
- розроблені відповідні додаткові структури даних для подальшої роботи;

Розроблені методи розгортання були використані в роботі з впровадження ІС у декількох замовників. Були використані пізні варіанти розгортання виходячи з умов замовника.

Пріоритетом подальшого розвитку ІС є створення засобів доступу до ІС з використанням мобільних технологій, що на даний час є актуальним питанням в даній сфері.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ТА ЛІТЕРАТУРИ

1. Гагарина Л.Г., Киселев Д.В. и Федотова Е.Л. Разработка и эксплуатация автоматизированных информационных систем. – М.: ИД «Форум» – ИНФРА – М, 2009. – 384 с.
2. Гвоздева В.А., Лаврентьева И.Ю. Основы построения автоматизированных информационных систем. – М, ИД «ФОРУМ»; ИНФРА-М, 2010. – 320 с
3. Гвоздева Т.В., Баллод Б.А. Проектирование информационных систем. – Ростов-на-Дону, «Феникс», 2009. – 509 с
4. Martin L. Abbott. The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise. – Addison-Wesley Professional, 2015. – 624 с.
5. Хабрахабр, Микросервисы (Microservices). – https://habrahabr.ru/post/249183/
6. Martin Fowler, Microservices. – https://martinfowler.com/articles/microservices.html
7. Vikram Murugesan, Microservices Deployment Cookbook. – Packt Publishing, 2017. – 378 с.
8. Sam Newman. Building Microservices: Designing Fine-Grained Systems. O'Reilly Media, 2015. – 280 с.
9. Rubygarage - https://rubygarage.org/blog/monolith-soamicroservices-serverless
10. Docker, Docker Documentation – https://docs.docker.com/

ДОДАТОК 1

До дипломного проєкту

на тему: «Інформаційна система на основі мікросервісної архітектури»

Копії графічних матеріалів

Схема структурна.
ІАЛЦ.045490.005 Д1

Kubernetes cluster. Схема структурна.
ІАЛЦ.045490.006 Д2

База даних. Схема структурна.
ІАЛЦ.045490.007 Д3

Процедура створення . Схема алгоритму.
ІАЛЦ.045490.008 Д4

Аркушів 4

Київ – 2023

ІАЛЦ.045490.005 Д1

					ІАЛЦ.045490.005 Д1	
Зм.	Арк.	№ докум.	Підпис	Дата		
Розроб.		Симонов Б.Є.			Інформаційна система на основі мікросервісної архітектури. Схема структурна	Літ. / Маса / Масштаб
Перев.		Романкевич В.О				Аркуш 1 / Аркушів 1
Т.контр.						КПІ ім. Ігоря Сікорського, ФПМ КВ-91
Н.контр.		Клятченко Я.М.				
Затв.		Романкевич В.О				

ІАЛЦ.045490.006 Д2

Зм.	Арк.	№ докум.	Підпис	Дата
Розроб.		Симонов Б.Є.		
Перев.		Романкевич В.О.		
Т.контр.				
Н.контр.		Клятченко Я.М.		
Затв.		Романкевич В.О		

Інформаційна система на основі мікросервісної архітектури.

Kubernetes cluster

Схема структурна.

Літ.	Маса	Масштаб

Аркуш 1	Аркушів 1

КПІ
ім. Ігоря Сікорського,
ФПМ КВ-91

					ІАЛЦ.467200.007 Д3	
Зм.	Арк.	№ докум.	Підпис	Дата	Інформаційна система на основі мікросервісної архітектури. База даних. Схема структурна	Літ. / Маса / Масштаб
Розроб.		Симонов Б.Є.				
Перев.		Романкевич В.О.				Аркуш 1 / Аркушів 1
						КПІ ім. Ігоря Сікорського, ФПМ КВ-91
Т.контр.						
Н.контр.		Клятченко Я.М.				
Затв.		Романкевич В.О.				

ІАЛЦ.045490.008 Д4

					ІАЛЦ.045490.008 Д4	
Зм.	Арк.	№ докум.	Підпис	Дата	Інформаційна система на основі мікросервісної архітектури. Процедура створення . Схема алгоритму	Літ. / Маса / Масштаб
Розроб.		Симонов Б.Є.				
Перев.		Романкевич В.О				Аркуш 1 / Аркушів 1
Т.контр.						КПІ ім. Ігоря Сікорського, ФПМ КВ-91
Н.контр.		Клятченко Я.М.				
Затв.		Романкевич В.О				

ДОДАТОК 2

До дипломного проєкту

на тему: «Інформаційна система на основі мікросервісної архітектури»

Лістинг програм. Скрипти розгортання та налаштування ІС

Аркушів 20

Київ – 2023

/srv/ITTEAM/eam/set-eam-service.sh

```
#!/bin/bash
LIST=$(ls ./service-conf -p | grep -v / | grep values | rev | cut -c 13- | rev | xargs)
NAMESPACE=eam
echo ====== Install writer ======
helm install writer --wait  -n $NAMESPACE ./eam-service/ -f ./service-conf/writer-values.yaml
for SRV in $LIST
do
   if [[ $SRV != *writer* ]]; then
          echo ====== Install $SRV ======
          helm install $SRV --wait  -n $NAMESPACE ./eam-service/ -f ./service-conf/$SRV-values.yaml
#    helm template $SRV -n $NAMESPACE ./eam-service/ -f ./service-conf/$SRV-values.yaml > $SRV-service.yaml
   fi
done
echo ====== Install frontend ======
kubectl apply -f ./service-conf/eam-frontend/
```

/srv/ITTEAM/k8s-create/create_vm_cluster.sh

```
#!/bin/bash
export GOVC_URL='#VSPHERE_SERVER#'
export GOVC_USERNAME='#VSPHERE_USER#'
export GOVC_PASSWORD='#VSPHERE_PASSWD#'
export GOVC_INSECURE=1
if [ "$1" = "--recreate" ]
then
# Destroy VM
govc vm.destroy "#MASTER1_VMNAME#"
govc vm.destroy "#MASTER2_VMNAME#"
govc vm.destroy "#MASTER3_VMNAME#"
govc vm.destroy "#WORKER1_VMNAME#"
govc vm.destroy "#WORKER2_VMNAME#"
govc vm.destroy "#WORKER3_VMNAME#"
#terraform destroy
fi
# Create VM
terraform apply -auto-approve
basedir="`dirname ${PWD}`"
CA=$basedir/eam/ssl/ca.crt
if [ "$1" = "--recreate" ]
then
# Set NTP Server
MASTERS="#MASTER1_IP# #MASTER2_IP# #MASTER3_IP#"
for MASTER in $MASTERS
```

```
do
  ssh-keygen -R $MASTER
  ssh-keyscan -H $MASTER >> ~/.ssh/known_hosts
  ssh root@$MASTER "echo 'NTP=#NTP#' >> /etc/systemd/timesyncd.conf && systemctl restart systemd-timesyncd"
  if test -f "$CA"; then
    scp ../eam/ssl/ca.crt root@$MASTER:/usr/local/share/ca-certificates
    ssh root@$MASTER "update-ca-certificates"
  fi
  ssh root@$MASTER "wget https://github.com/kubernetes-sigs/cri-tools/releases/download/v1.23.0/critest-v1.23.0-linux-amd64.tar.gz && tar zxvf critest-v1.23.0-linux-amd64.tar.gz -C /usr/local/bin && rm -f critest-v1.23.0-linux-amd64.tar.gz"
done
WORKERS="#WORKER1_IP# #WORKER2_IP# #WORKER3_IP#"
for WORKER in $WORKERS
do
  ssh-keygen -R $WORKER
  ssh-keyscan -H $WORKER >> ~/.ssh/known_hosts
  ssh root@$WORKER "echo 'server #NTP# iburst prefer' >> /etc/chrony.conf && systemctl restart chronyd && timedatectl | awk '/Time zone:/ {print $3}' >> /etc/timezone"
  if test -f "$CA"; then
    scp ../eam/ssl/ca.crt root@$WORKER:/etc/pki/ca-trust/source/anchors
    ssh root@$WORKER "update-ca-trust extract"
  fi
  scp $PWD/fix_node.sh root@$WORKER:/root/fix_node.sh
  ssh root@$WORKER "echo '@reboot /root/fix_node.sh' >> /tmp/c1 && crontab /tmp/c1"
  ssh root@$WORKER "wget https://github.com/kubernetes-sigs/cri-tools/releases/download/v1.23.0/critest-v1.23.0-linux-amd64.tar.gz && tar zxvf critest-v1.23.0-linux-amd64.tar.gz -C /usr/local/bin && rm -f critest-v1.23.0-linux-amd64.tar.gz"
  ssh root@$WORKER "yum install -y s3fs-fuse nfs-utils python36"
done
fi
```

/srv/ITTEAM/k8s-create/main.tf

```
provider "vsphere" {
  user           = "#VSPHERE_USER#"
  password       = "#VSPHERE_PASSWD#"
  vsphere_server = "#VSPHERE_SERVER#"
  allow_unverified_ssl = true
}
data "vsphere_datacenter" "dc" {
  name = "#VSPHERE_DC#"
}
data "vsphere_datastore" "datastore" {
  name          = "#VSPHERE_DATASTORE#"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"
}
```

```
 data "vsphere_compute_cluster" "cluster" {
  name          = "#VSPHERE_CLUSTER#"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"
}
 data "vsphere_network" "network" {
  name          = "#VSPHERE_NETWORK#"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"
}

data "vsphere_virtual_machine" "template1" {
  name          = "kube-masters-new"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"
}

data "vsphere_virtual_machine" "template2" {
  name          = "kube-workers-new"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"
}
```

/srv/ITTEAM/k8s-create/vm.tf

```
resource "vsphere_virtual_machine" "vm1" {
  name             = "#MASTER1_VMNAME#"
  resource_pool_id = "${data.vsphere_compute_cluster.cluster.resource_pool_id}"
  datastore_id     = "${data.vsphere_datastore.datastore.id}"
  folder = "#VSPHERE_FOLDER#"
   num_cpus = 4
  memory   = 6144
  guest_id = "${data.vsphere_virtual_machine.template1.guest_id}"
   scsi_type = "${data.vsphere_virtual_machine.template1.scsi_type}"
  network_interface {
    network_id   = "${data.vsphere_network.network.id}"
    adapter_type = "${data.vsphere_virtual_machine.template1.network_interface_types[0]}"
  }
  disk {
    label            = "disk0"
    size             = "${data.vsphere_virtual_machine.template1.disks.0.size}"
    eagerly_scrub    = "${data.vsphere_virtual_machine.template1.disks.0.eagerly_scrub}"
    thin_provisioned = "${data.vsphere_virtual_machine.template1.disks.0.thin_provisioned}"
  }
   clone {
    template_uuid = "${data.vsphere_virtual_machine.template1.id}"
    customize {
      linux_options {
        host_name = "#MASTER1_URI#"
        domain    = "#SEGMENT#.#DOMAIN#"
```

```
    }
    dns_server_list     = ["#DNS#"]
    network_interface {
     ipv4_address = "#MASTER1_IP#"
     ipv4_netmask = #SUBNET#
    }
    ipv4_gateway = "#GATEWAY#"
   }
  }
}
resource "vsphere_virtual_machine" "vm2" {
  name             = "#MASTER2_VMNAME#"
  resource_pool_id = "${data.vsphere_compute_cluster.cluster.resource_pool_id}"
  datastore_id     = "${data.vsphere_datastore.datastore.id}"
  folder = "#VSPHERE_FOLDER#"
  num_cpus = 4
  memory   = 6144
  guest_id = "${data.vsphere_virtual_machine.template1.guest_id}"
   scsi_type = "${data.vsphere_virtual_machine.template1.scsi_type}"
  network_interface {
   network_id   = "${data.vsphere_network.network.id}"
   adapter_type = "${data.vsphere_virtual_machine.template1.network_interface_types[0]}"
  }
   disk {
   label            = "disk0"
   size             = "${data.vsphere_virtual_machine.template1.disks.0.size}"
   eagerly_scrub    = "${data.vsphere_virtual_machine.template1.disks.0.eagerly_scrub}"
   thin_provisioned = "${data.vsphere_virtual_machine.template1.disks.0.thin_provisioned}"
  }
   clone {
   template_uuid = "${data.vsphere_virtual_machine.template1.id}"
   customize {
    linux_options {
     host_name = "#MASTER2_URI#"
     domain    = "#SEGMENT#.#DOMAIN#"
    }
    dns_server_list     = ["#DNS#"]
    network_interface {
     ipv4_address = "#MASTER2_IP#"
     ipv4_netmask = #SUBNET#
    }
    ipv4_gateway = "#GATEWAY#"
   }
  }
}
```

```
resource "vsphere_virtual_machine" "vm3" {
  name             = "#MASTER3_VMNAME#"
  resource_pool_id = "${data.vsphere_compute_cluster.cluster.resource_pool_id}"
  datastore_id     = "${data.vsphere_datastore.datastore.id}"
  folder = "#VSPHERE_FOLDER#"
  num_cpus = 4
  memory   = 6144
  guest_id = "${data.vsphere_virtual_machine.template1.guest_id}"
   scsi_type = "${data.vsphere_virtual_machine.template1.scsi_type}"
   network_interface {
    network_id   = "${data.vsphere_network.network.id}"
    adapter_type = "${data.vsphere_virtual_machine.template1.network_interface_types[0]}"
  }
   disk {
    label            = "disk0"
    size             = "${data.vsphere_virtual_machine.template1.disks.0.size}"
    eagerly_scrub    = "${data.vsphere_virtual_machine.template1.disks.0.eagerly_scrub}"
    thin_provisioned = "${data.vsphere_virtual_machine.template1.disks.0.thin_provisioned}"
  }
  clone {
    template_uuid = "${data.vsphere_virtual_machine.template1.id}"
    customize {
      linux_options {
        host_name = "#MASTER3_URI#"
        domain    = "#SEGMENT#.#DOMAIN#"
      }
      dns_server_list     = ["#DNS#"]
      network_interface {
        ipv4_address = "#MASTER3_IP#"
        ipv4_netmask = #SUBNET#
      }
      ipv4_gateway = "#GATEWAY#"
    }
  }
}
resource "vsphere_virtual_machine" "vm4" {
  name             = "#WORKER1_VMNAME#"
  resource_pool_id = "${data.vsphere_compute_cluster.cluster.resource_pool_id}"
  datastore_id     = "${data.vsphere_datastore.datastore.id}"
  folder = "#VSPHERE_FOLDER#"
  num_cpus = 8
  memory   = 24576
  guest_id = "${data.vsphere_virtual_machine.template2.guest_id}"
  firmware  = "efi"
  scsi_type = "${data.vsphere_virtual_machine.template2.scsi_type}"
```

```
  network_interface {
   network_id   = "${data.vsphere_network.network.id}"
   adapter_type = "${data.vsphere_virtual_machine.template2.network_interface_types[0]}"
  }
   disk {
   label            = "disk0"
   size             = "${data.vsphere_virtual_machine.template2.disks.0.size}"
   eagerly_scrub    = "${data.vsphere_virtual_machine.template2.disks.0.eagerly_scrub}"
   thin_provisioned = "${data.vsphere_virtual_machine.template2.disks.0.thin_provisioned}"
   unit_number      = 0
  }
  clone {
   template_uuid = "${data.vsphere_virtual_machine.template2.id}"
   customize {
    linux_options {
     host_name = "#WORKER1_URI#"
     domain    = "#SEGMENT#.#DOMAIN#"
    }
    dns_server_list     = ["#DNS#"]
    network_interface {
     ipv4_address = "#WORKER1_IP#"
     ipv4_netmask = #SUBNET#
    }
    ipv4_gateway = "#GATEWAY#"
   }
  }
}
resource "vsphere_virtual_machine" "vm5" {
  name             = "#WORKER2_VMNAME#"
  resource_pool_id = "${data.vsphere_compute_cluster.cluster.resource_pool_id}"
  datastore_id     = "${data.vsphere_datastore.datastore.id}"
  folder = "#VSPHERE_FOLDER#"
  num_cpus = 8
  memory   = 24576
  guest_id = "${data.vsphere_virtual_machine.template2.guest_id}"
  firmware  = "efi"
  scsi_type = "${data.vsphere_virtual_machine.template2.scsi_type}"
   network_interface {
   network_id   = "${data.vsphere_network.network.id}"
   adapter_type = "${data.vsphere_virtual_machine.template2.network_interface_types[0]}"
  }
   disk {
   label            = "disk0"
   size             = "${data.vsphere_virtual_machine.template2.disks.0.size}"
   eagerly_scrub    = "${data.vsphere_virtual_machine.template2.disks.0.eagerly_scrub}"
```

```
    thin_provisioned = "${data.vsphere_virtual_machine.template2.disks.0.thin_provisioned}"
    unit_number      = 0
  }
  clone {
    template_uuid = "${data.vsphere_virtual_machine.template2.id}"
    customize {
      linux_options {
        host_name = "#WORKER2_URI#"
        domain    = "#SEGMENT#.#DOMAIN#"
      }
      dns_server_list     = ["#DNS#"]
      network_interface {
        ipv4_address = "#WORKER2_IP#"
        ipv4_netmask = #SUBNET#
      }
      ipv4_gateway = "#GATEWAY#"
    }
  }
}
resource "vsphere_virtual_machine" "vm6" {
  name             = "#WORKER3_VMNAME#"
  resource_pool_id = "${data.vsphere_compute_cluster.cluster.resource_pool_id}"
  datastore_id     = "${data.vsphere_datastore.datastore.id}"
  folder = "#VSPHERE_FOLDER#"
  num_cpus = 8
  memory   = 24576
  guest_id = "${data.vsphere_virtual_machine.template2.guest_id}"
  firmware  = "efi"
  scsi_type = "${data.vsphere_virtual_machine.template2.scsi_type}"
   network_interface {
    network_id   = "${data.vsphere_network.network.id}"
    adapter_type = "${data.vsphere_virtual_machine.template2.network_interface_types[0]}"
  }
   disk {
    label            = "disk0"
    size             = "${data.vsphere_virtual_machine.template2.disks.0.size}"
    eagerly_scrub    = "${data.vsphere_virtual_machine.template2.disks.0.eagerly_scrub}"
    thin_provisioned = "${data.vsphere_virtual_machine.template2.disks.0.thin_provisioned}"
    unit_number      = 0
  }
  clone {
    template_uuid = "${data.vsphere_virtual_machine.template2.id}"
    customize {
      linux_options {
        host_name = "#WORKER3_URI#"
```

```
      domain    = "#SEGMENT#.#DOMAIN#"
    }
    dns_server_list     = ["#DNS#"]
    network_interface {
      ipv4_address = "#WORKER3_IP#"
      ipv4_netmask = #SUBNET#
    }
    ipv4_gateway = "#GATEWAY#"
  }
 }
}
```

/srv/ITTEAM/monitoring/grafana-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: grafana
  namespace: monitoring
spec:
#  type: LoadBalancer
  ports:
    - name: http
      port: 3000
      targetPort: 3000
      protocol: TCP
  selector:
    app.kubernetes.io/name: grafana
```

/srv/ITTEAM/monitoring/grafana-store-pvc.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: grafana-store-pv
  namespace: monitoring
  labels:
    name: grafana-store-pv
    app: eam-grafana
spec:
  storageClassName: manual
  capacity:
    storage: #GRAFANA_DISK_SIZE#Gi
  accessModes:
    - ReadWriteMany
```

```
  nfs:
    server: #MINIO_IP#
    path: "/srv/data/monitoring/grafana"
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: grafana-store-pvc
  namespace: monitoring
  labels:
    name: grafana-store-pvc
    app: eam-grafana
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: #GRAFANA_DISK_SIZE#Gi
```

/srv/ITTEAM/monitoring/monitoring-grafana-ingress.yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: monitoring-grafana-ingress
  namespace: monitoring
  annotations:
    kubernetes.io/ingress.class: "nginx"
    nginx.ingress.kubernetes.io/rewrite-target: /$1
    nginx.ingress.kubernetes.io/use-regex: "true"
spec:
  rules:
  - host: #WWW_URI#.#SEGMENT#.#DOMAIN#
    http:
      paths:
        - path: /grafana/?(.*)
          pathType: Prefix
          backend:
            service:
              name: grafana
              port:
                number: 3000
```

/srv/ITTEAM/monitoring/monitoring-prometheus-ingress.yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: monitoring-prometheus-ingress
  namespace: monitoring
  annotations:
    kubernetes.io/ingress.class: "nginx"
    nginx.ingress.kubernetes.io/rewrite-target: /$2
    nginx.ingress.kubernetes.io/use-regex: "true"
spec:
  rules:
  - host: #WWW_URI#.#SEGMENT#.#DOMAIN#
    http:
      paths:
        - path: /prometheus(/|$)(.*)
          pathType: Prefix
          backend:
            service:
              name: prometheus
              port:
                number: 9090
```

/srv/ITTEAM/monitoring/prometheus-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: prometheus
  namespace: monitoring
spec:
#  type: LoadBalancer
  ports:
    - name: http
      port: 9090
      targetPort: 9090
      protocol: TCP
  selector:
    app.kubernetes.io/name: prometheus
    prometheus: prometheus-kube-prometheus-prometheus
```

/srv/ITTEAM/postgres/server-install.sh

```
#!/bin/bash
# Add global mount point
```

```
mkdir /srv/data
# Prepare disk
pvcreate /dev/sdb
vgcreate data /dev/sdb
lvcreate -l 100%FREE -n data data
mkfs.xfs /dev/data/data
mount /dev/data/data /srv/data/
echo "/dev/data/data                  /srv/data       xfs     defaults        0 0">> /etc/fstab
# PosygreSQL 12 Install
dnf -y install https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-x86_64/pgdg-redhat-repo-latest.noarch.rpm
dnf -y install epel-release
dnf module disable postgresql -y
dnf --repo pgdg12 -y install postgresql12-server postgresql12
mkdir /srv/data/postgresql
cp -r /var/lib/pgsql/* /srv/data/postgresql
chown -R postgres:postgres /srv/data/postgresql
sed -i "s%/var/lib/pgsql/%/srv/data/postgresql/%g" /var/lib/pgsql/.bash_profile
sed -i "s%/var/lib/pgsql/%/srv/data/postgresql/%g" /usr/lib/systemd/system/postgresql-12.service
systemctl enable postgresql-12
/usr/pgsql-12/bin/postgresql-12-setup initdb
echo host all all all password>>/srv/data/postgresql/12/data/pg_hba.conf
sed -i "s%max_connections = 100%max_connections = 2000%g" /srv/data/postgresql/12/data/postgresql.conf
sed -i "s%#listen_addresses = 'localhost'%listen_addresses = '*'%g" /srv/data/postgresql/12/data/postgresql.conf
systemctl start postgresql-12
yum-config-manager --enable ol8_codeready_builder
dnf -y install postgis30_12 pgrouting_12-3.0.5
# Initial PostgreSQL Server for EAM
mkdir /srv/data/postgresql/12/create
cp ./pg-create/* /srv/data/postgresql/12/create
chown -R postgres:postgres /srv/data/postgresql/12/create
chmod +x /srv/data/postgresql/12/create/add_base.sh
cd /srv/data/postgresql/12/create
./add_base.sh
```

/srv/ITTEAM/postgres/setup-server.sh

```
#!/bin/bash
PGDB="#POSTGRE_IP#"
# Set NTP Server
ssh-keygen -R $PGDB
ssh-keyscan -H $PGDB >> ~/.ssh/known_hosts
#ssh root@$PGDB "yum install -y chrony && systemctl enable --now chronyd"
ssh root@$PGDB "echo 'server #NTP# iburst prefer' >> /etc/chrony.conf && systemctl restart chronyd && timedatectl | awk '/Time zone:/ {print $3}' >> /etc/timezone"
ssh root@$PGDB "mkdir /root/pg-create"
```

```
scp ./pg-create/* root@$PGDB:/root/pg-create
scp ./server-install.sh root@$PGDB:/root
ssh root@$PGDB "./server-install.sh"
```

/srv/ITTEAM/postgres/stand-db-server-create.sh

```
#!/bin/bash
cd vm_create
./create_vm_cluster.sh --recreate
cd ..
./setup-server.sh
```

/srv/ITTEAM/system/metallb-config-crd.yaml

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: eam
  namespace: metallb
spec:
  addresses:
  - #WWW_IP#/32
---
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: eam-l2advertisement
  namespace: metallb
spec:
  ipAddressPools:
  - eam
---
```

/srv/ITTEAM/config_itteam.sh

```
#!/bin/bash
source ./itteam.config

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i "s/#SUBNET#/$SUBNET/g" {} +
find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i "s/#GATEWAY#/$GATEWAY/g" {} +
find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i "s/#NTP#/$NTP/g" {} +
find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i "s/#DNS#/$DNS/g" {} +
find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i "s/#VSPHERE_SERVER#/$VSPHERE_SERVER/g" {} +
```

```
find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#VSPHERE_USER#/$VSPHERE_USER/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#VSPHERE_PASSWD#/$VSPHERE_PASSWD/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s%#VSPHERE_FOLDER#%$VSPHERE_FOLDER%g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#VSPHERE_DC#/$VSPHERE_DC/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#VSPHERE_CLUSTER#/$VSPHERE_CLUSTER/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#VSPHERE_DATASTORE#/$VSPHERE_DATASTORE/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#VSPHERE_NETWORK#/$VSPHERE_NETWORK/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i "s/#DOMAIN#/$DOMAIN/g"
{} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#SEGMENT#/$SEGMENT/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#WWW_URI#/$WWW_URI/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i "s/#WWW_IP#/$WWW_IP/g"
{} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#MASTER1_URI#/$MASTER1_URI/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#MASTER1_IP#/$MASTER1_IP/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#MASTER1_VMNAME#/$MASTER1_VMNAME/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#MASTER2_URI#/$MASTER2_URI/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#MASTER2_IP#/$MASTER2_IP/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#MASTER2_VMNAME#/$MASTER2_VMNAME/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#MASTER3_URI#/$MASTER3_URI/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#MASTER3_IP#/$MASTER3_IP/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#MASTER3_VMNAME#/$MASTER3_VMNAME/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#WORKER1_URI#/$WORKER1_URI/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#WORKER1_IP#/$WORKER1_IP/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#WORKER1_VMNAME#/$WORKER1_VMNAME/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#WORKER2_URI#/$WORKER2_URI/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#WORKER2_IP#/$WORKER2_IP/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#WORKER2_VMNAME#/$WORKER2_VMNAME/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#WORKER3_URI#/$WORKER3_URI/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#WORKER3_IP#/$WORKER3_IP/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#WORKER3_VMNAME#/$WORKER3_VMNAME/g" {} +
```

```
find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s%#FLANEL_INTERFACE_REGEXP#%$FLANEL_INTERFACE_REGEXP%g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#MINIO_URI#/$MINIO_URI/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i "s/#MINIO_IP#/$MINIO_IP/g"
{} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#MINIO_VMNAME#/$MINIO_VMNAME/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#MINIO_DISK_SIZE#/$MINIO_DISK_SIZE/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#MINIO_ADMIN_PASSWD#/$MINIO_ADMIN_PASSWD/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#MINIO_CONSOLE_PASSWD#/$MINIO_CONSOLE_PASSWD/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#MINIO_MEDIADATA_USER#/$MINIO_MEDIADATA_USER/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#MINIO_MEDIADATA_PASSWD#/$MINIO_MEDIADATA_PASSWD/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#MINIO_MEDIADATA_BUCKET#/$MINIO_MEDIADATA_BUCKET/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#POSTGRE_URI#/$POSTGRE_URI/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#POSTGRE_IP#/$POSTGRE_IP/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#POSTGRE_PORT#/$POSTGRE_PORT/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#POSTGRE_VMNAME#/$POSTGRE_VMNAME/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#POSTGRE_DISK_SIZE#/$POSTGRE_DISK_SIZE/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#POSTGRE_PASSWD#/$POSTGRE_PASSWD/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#KAFKA_DISK_SIZE#/$KAFKA_DISK_SIZE/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#PROMETHEUS_DISK_SIZE#/$PROMETHEUS_DISK_SIZE/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#PROMETHEUS_ENDPOINT_URI#/$PROMETHEUS_ENDPOINT_URI/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#GRAFANA_ADMIN_PASSWD#/$GRAFANA_ADMIN_PASSWD/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#GRAFANA_DISK_SIZE#/$GRAFANA_DISK_SIZE/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#LOKI_DISK_SIZE#/$LOKI_DISK_SIZE/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#CONTROL_NODE_URI#/$CONTROL_NODE_URI/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#INTEGR_URI#/$INTEGR_URI/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#INTEGR_IP#/$INTEGR_IP/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#INTEGR_VMNAME#/$INTEGR_VMNAME/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i "s/#OSM_URI#/$OSM_URI/g"
{} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i "s/#OSM_IP#/$OSM_IP/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#OSM_VMNAME#/$OSM_VMNAME/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#ITTGIS_DB_USER#/$ITTGIS_DB_USER/g" {} +
```

```
find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#ITTGIS_DB_PASSWD#/$ITTGIS_DB_PASSWD/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#ITTGIS_GEOSERVER_ADMIN_PASSWD#/$ITTGIS_GEOSERVER_ADMIN_PASSWD/g" {} +

find . -type f -not -name "config_itteam.sh" -not -name "config_loader.sh" -exec sed -i
"s/#ITTGIS_GEOSERVER_DATA_DISK_SIZE#/$ITTGIS_GEOSERVER_DATA_DISK_SIZE/g" {} +
```

/srv/ITTEAM/create_itteam.sh

```
#!/bin/bash
ource ./stack.config
WDir=$PWD
if $MINIO; then
   echo -en "\033[32m Install MINIO \n \033[0m"
   cd minio
   ./minio-server-create.sh
   cd ..
   echo -e "\033[32m";
   read -n1 -rsp $'Press any key to continue or Ctrl+C to exit...\n'
   echo -e "\033[0m";
fi
if $POSTGRESQL; then
   echo -en "\033[32m Install PostgreSQL Cluster \n\033[0m"
   cd postgres
   ./stand-db-server-create.sh
   cd ..
   echo -e "\033[32m";
   read -n1 -rsp $'Press any key to continue or Ctrl+C to exit...\n'
   echo -e "\033[0m";
fi
if $K8S; then
   echo -en "\033[32m Install Kubernetes Cluster VM \n\033[0m"
   cd k8s-create
   ./create_vm_cluster.sh --recreate
   cd ..
   echo -e "\033[32m";
   read -n1 -rsp $'Press any key to continue or Ctrl+C to exit...\n'
   echo -e "\033[0m";
   echo -en "\033[32m Install Kubernetes Cluster \n\033[0m"
   export ANSIBLE_HOST_KEY_CHECKING=False
   ansible-playbook -i ./kubespray/inventory/itteam/inventory.ini --become-user=root ./kubespray/cluster.yml
   echo -e "\033[32m";
   read -n1 -rsp $'Press any key to continue or Ctrl+C to exit...\n'
   echo -e "\033[0m";
   rm -rf ~/.kube
   mkdir ~/.kube
```

```
    cp ./kubespray/inventory/itteam/artifacts/admin.conf ~/.kube/config
    chmod 600 ~/.kube/config
    kubectl config view --minify --raw > ./kubespray/inventory/itteam/artifacts/lens_config_file
    rm -rf ./kubespray/inventory/itteam/credentials/etcd-ssl
    mkdir ./kubespray/inventory/itteam/credentials/etcd-ssl
    scp root@#MASTER1_IP#:/etc/ssl/etcd/ssl/* ./kubespray/inventory/itteam/credentials/etcd-ssl/
    echo -e "\033[32m";
    read -n1 -rsp $'Press any key to continue or Ctrl+C to exit...\n'
    echo -e "\033[0m";
    echo -en "\033[32m Install MetalLB \n \033[0m"
    kubectl create ns metallb
    helm install metallb --wait --namespace metallb ./system/metallb
    kubectl apply -f ./system/metallb-config-crd.yaml
    echo -e "\033[32m";
    read -n1 -rsp $'Press any key to continue or Ctrl+C to exit...\n'
    echo -e "\033[0m";
#    echo -en "\033[32m Install Linstor \n \033[0m"
#    kubectl create ns linstor
#    helm install linstor-etcd --wait --namespace linstor ./storage/pv-hostpath #--set "nodes={node1,node2,node3}"
#    helm install piraeus-op --wait --namespace linstor ./storage/piraeus -f ./storage/piraeus-values.yaml -f ./storage/resource-requirements.yaml -f ./storage/storage-create.yaml
#    kubectl apply -f ./storage/storage-class.yaml
#    kubectl patch storageclass data-single-store -p '{"metadata": {"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
#    echo -e "\033[32m";
#    read -n1 -rsp $'Press any key to continue or Ctrl+C to exit...\n'
#    echo -e "\033[0m";
    echo -en "\033[32m Install NFS Datastore \n \033[0m"
    helm install system-nfs-datastore --wait --namespace=default ./storage/nfs-subdir-external-provisioner/ \
    --set nfs.server=#MINIO_IP#\
    --set nfs.path=/srv/data/cluster \
    --set storageClass.name=data-single-store
    kubectl patch storageclass data-single-store -p '{"metadata": {"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
    echo -e "\033[32m";
    read -n1 -rsp $'Press any key to continue or Ctrl+C to exit...\n'
    echo -e "\033[0m";
    echo -en "\033[32m Install Prometheus Stack \n \033[0m"
    kubectl create namespace monitoring
    kubectl -n monitoring apply -f ./eam/datasource/
    kubectl -n monitoring create secret generic kube-etcd-client-certs \
    --from-file=etcd-client-ca.crt=./kubespray/inventory/ittpims/credentials/etcd-ssl/ca.pem \
    --from-file=etcd-client.crt=./kubespray/inventory/ittpims/credentials/etcd-ssl/node-node1.pem \
    --from-file=etcd-client.key=./kubespray/inventory/ittpims/credentials/etcd-ssl/node-node1-key.pem
    kubectl apply -f ./monitoring/grafana-store-pvc.yaml
    helm install prometheus --wait --namespace=monitoring -f ./monitoring/kube-prometheus-stack-values.yaml ./monitoring/kube-prometheus-stack/
```

```
echo -e "\033[32m";
read -n1 -rsp $'Press any key to continue or Ctrl+C to exit...\n'
echo -e "\033[0m";
echo -en "\033[32m Install Nginx Ingress \n \033[0m"
kubectl create ns ingress-nginx
kubectl -n ingress-nginx create secret tls --cert ./eam/ssl/fullchain.pem --key ./eam/ssl/privkey.pem ingress-nginx-tls
helm install ingress-nginx --namespace=ingress-nginx -f ./ingress/ingress-nginx-values.yaml ./ingress/ingress-nginx/
sleep 30
echo -e "\033[32m";
read -n1 -rsp $'Press any key to continue or Ctrl+C to exit...\n'
echo -e "\033[0m";
echo -en "\033[32m Configure Prometheus Stack \n \033[0m"
kubectl apply -f ./monitoring/grafana-service.yaml
kubectl apply -f ./monitoring/prometheus-service.yaml
kubectl apply -f ./monitoring/monitoring-grafana-ingress.yaml
kubectl apply -f ./monitoring/monitoring-prometheus-ingress.yaml
##release: prometheus
helm install prometheus-adapter --wait --namespace monitoring ./monitoring/prometheus-adapter/  --set metricsRelistInterval=30s,prometheus.url=http://prometheus.monitoring.svc.cluster.local
echo -e "\033[32m";
read -n1 -rsp $'Press any key to continue or Ctrl+C to exit...\n'
echo -e "\033[0m";
echo -en "\033[32m Install Loki Stack \n \033[0m"
kubectl create namespace loki
#loki.loki.svc.cluster.local
kubectl apply -f ./loki/loki-store-pvc.yaml
helm upgrade --install loki --wait --namespace=loki -f ./loki/loki-stack-values.yaml ./loki/loki-stack
echo -e "\033[32m";
read -n1 -rsp $'Press any key to continue or Ctrl+C to exit...\n'
echo -e "\033[0m";
echo -en "\033[32m Install DB Utils \n \033[0m"
helm install prometheus-postgres-exporter --wait --namespace=monitoring -f ./postgres/prometheus-postgres-exporter-values.yaml ./postgres/prometheus-postgres-exporter/
echo -e "\033[32m";
read -n1 -rsp $'Press any key to continue or Ctrl+C to exit...\n'
echo -e "\033[0m";
echo -en "\033[32m Install Kafka Cluster \n \033[0m"
kubectl create ns kafka
helm install kafka --wait --namespace=kafka ./kafka/kafka/ \
--set defaultReplicationFactor=3 \
--set offsetsTopicReplicationFactor=3 \
--set transactionStateLogReplicationFactor=3 \
--set transactionStateLogMinIsr=3 \
--set numPartitions=10 \
--set replicaCount=3 \
--set persistence.storageClass="data-single-store" \
```

```
    --set persistence.size=#KAFKA_DISK_SIZE#Gi \
    --set metrics.kafka.enabled=true \
    --set metrics.jmx.enabled=true \
    --set metrics.serviceMonitor.enabled=true \
    --set metrics.serviceMonitor.namespace=monitoring \
    --set metrics.serviceMonitor.interval=10s \
    --set metrics.serviceMonitor.scrapeTimeout=10s \
    --set metrics.serviceMonitor.selector.prometheus=prometheus \
    --set metrics.serviceMonitor.selector.release=prometheus \
    --set zookeeper.replicaCount=3 \
    --set zookeeper.persistence.storageClass="data-single-store" \
    --set zookeeper.metrics.enabled=true \
    --set zookeeper.metrics.serviceMonitor.enabled=false \
    --set zookeeper.metrics.serviceMonitor.namespace=monitoring \
    --set zookeeper.metrics.serviceMonitor.interval=10s \
    --set zookeeper.metrics.serviceMonitor.scrapeTimeout=10s \
    --set zookeeper.metrics.serviceMonitor.selector.prometheus=prometheus \
    --set zookeeper.metrics.serviceMonitor.selector.release=prometheus
    ##kafka.kafka.svc.cluster.local
    ##kafka-zookeeper.kafka.svc.cluster.local:2181
    helm install kafka-lag-exporter --wait ./kafka/kafka-lag-exporter/ \
    --namespace kafka \
    --set clusters\[0\].name=pims \
    --set clusters\[0\].bootstrapBrokers=kafka.kafka.svc.cluster.local:9092 \
    --set prometheus.serviceMonitor.enabled=true \
    --set prometheus.serviceMonitor.interval="15s" \
    --set prometheus.serviceMonitor.additionalLabels.release=prometheus \
    --set prometheus.serviceMonitor.additionalLabels.prometheus=prometheus
    helm install kafka-kminion --wait --namespace=kafka -f ./kafka/kminion-values.yaml ./kafka/kminion/
    echo -e "\033[32m";
    read -n1 -rsp $'Press any key to continue or Ctrl+C to exit...\n'
    echo -e "\033[0m";
    echo -en "\033[32m Set CertManager And PodPreset \n \033[0m"
    helm install cert-manager ./system/cert-manager --namespace cert-manager --create-namespace --version v1.7.1 --set
installCRDs=true
    kubectl apply -f ./system/podpreset.yaml
    echo -e "\033[32m";
    read -n1 -rsp $'Press any key to continue or Ctrl+C to exit...\n'
    echo -e "\033[0m";
    echo -en "\033[32m Set KafkaUI \n \033[0m"
    helm install kafka-ui --wait ./kafka/kafka-ui/ --namespace default -f ./kafka/kafka-ui-values.yaml
    kubectl apply -f ./kafka/kafka-ui-ingress.yaml
    echo -e "\033[32m";
    read -n1 -rsp $'Press any key to continue or Ctrl+C to exit...\n'
    echo -e "\033[0m";
```

```
fi
if $INTEGR; then
   echo -en "\033[32m Install Integration server \n \033[0m"
   cd integr
   ./integr-server-create.sh
   cd ..
   echo -e "\033[32m";
   read -n1 -rsp $'Press any key to continue or Ctrl+C to exit...\n'
   echo -e "\033[0m";
fi
if $LOAD_DATA; then
   echo -en "\033[32m Load PostgreSQL DB \n \033[0m"
   cd ..
   cd DATA
   ./loader.sh restore db
   if [[ -e $PWD/db/migration.sql ]]; then
         export PGPASSWORD="#POSTGRE_PASSWD#"
      echo -en "\033[33m Execute migration script \n \033[0m"
      kubectl scale deploy -n eam  --replicas=0 --all
      kubectl scale deploy -n geoserver  --replicas=0 --all
      sleep 40
      psql -U postgres -h #POSTGRE_IP# -p #POSTGRE_PORT# eam -f "$PWD/db/migration.sql"
      kubectl scale deploy -n geoserver  --replicas=1 --all
      kubectl scale deploy -n eam  --replicas=1 --all
   fi
   cd $WDir
   echo -e "\033[32m";
   read -n1 -rsp $'Press any key to continue or Ctrl+C to exit...\n'
   echo -e "\033[0m";
fi
if $ITTEAM; then
   echo -en "\033[32m Init ITTGIS \n \033[0m"
   kubectl create namespace eam
   kubectl apply -f ./eam/env/
   echo -en "\033[32m Install Reloader \n \033[0m"
   helm install reloader --wait  --namespace eam ./system/reloader --set reloader.watchGlobally=false
   echo -e "\033[32m";
   read -n1 -rsp $'Press any key to continue or Ctrl+C to exit...\n'
   echo -e "\033[0m";
   echo -en "\033[32m Install ITTGIS Geoserver \n \033[0m"
   kubectl create namespace geoserver
   kubectl apply -f ./geoserver/geoserver-set-localtime.yaml
   kubectl apply -f ./geoserver/geoserver-context-configmap.yaml
   kubectl apply -f ./geoserver/geoserver-geodata.yaml
   kubectl apply -f ./geoserver/geoserver-deployment.yaml
```

```
    kubectl apply -f ./geoserver/geoserver-svc.yaml
    kubectl apply -f ./geoserver/geoserver-ingress.yaml
    kubectl apply -f ./geoserver/geoserver-filemanager-deployment.yaml
    kubectl apply -f ./geoserver/geoserver-filemanager-ingress.yaml
    kubectl apply -f ./geoserver/geoserver-filemanager-svc.yaml
    echo -e "\033[32m";
    read -n1 -rsp $'Press any key to continue or Ctrl+C to exit...\n'
    echo -e "\033[0m";
    echo -en "\033[32m Install ITTGIS Draw.io \n \033[0m"
    kubectl create namespace drawio
    kubectl apply -f ./drawio/drawio-set-localtime.yaml
    kubectl apply -f ./drawio/eam-local-ca-configmap.yaml
    kubectl apply -f ./drawio/eam-drawio-deployment.yaml
    kubectl apply -f ./drawio/eam-drawio-svc.yaml
    kubectl apply -f ./drawio/eam-drawio-ingress.yaml
    echo -e "\033[32m";
    read -n1 -rsp $'Press any key to continue or Ctrl+C to exit...\n'
    echo -e "\033[0m";
    echo -en "\033[32m Install ITTGIS Services \n \033[0m"
    kubectl apply -f ./eam/metrics/
    kubectl apply -f ./eam/dashboard/
    cd $WDir/eam/
    ./set-eam-service.sh
    cd $WDir
    echo -e "\033[32m";
    read -n1 -rsp $'Press any key to continue or Ctrl+C to exit...\n'
    echo -e "\033[0m";
fi
if $ITTPIMS && $LOAD_DATA; then
    echo -en "\033[32m Load Geoserver & Minio data \n \033[0m"
    cd ..
    cd DATA
    ./loader.sh restore geoserver,minio
    cd $WDir
    echo -e "\033[32m";
    read -n1 -rsp $'Press any key to continue or Ctrl+C to exit...\n'
    echo -e "\033[0m";
fi
```