DOI: 10.18372/2410-7840.24.16852

УДК 004.42

ЗАБЕЗПЕЧЕННЯ КОНФІДЕНЦІЙНОСТІ БАЗ ДАНИХ В DEVOPS

Михайло Коломицев, Світлана Носок

Інтеграція розробки баз даних в конвеєр доставки DevOps є актуальним завданням в силу зростаючої популярності цієї методології розробки додатків. Метою інтеграції змін бази даних як частини процесу DevOps є підвищення швидкості доставки будь-яких змін бази даних. Практика Database DevOps спрямована на підвищення ефективності управління базами даних. Така практика допомагає оптимізувати процес розгортання і модифікації баз даних, даючи можливість автоматизувати багато аспектів життєвого циклу бази даних. Однак реалізація Database DevOps стикається з певними труднощами. Особливості бази даних як об'єкта інформаційної системи призводять до того, що їх повноцінна автоматизація тестування можлива тільки на даних, максимально наближених до реальних даних у виробничій базі даних. Однак використання реальних даних породжує обґрунтовані ризики порушення конфіденційності даних. У статті розглядається методика підготовки даних для тестування, що відповідає вимогам реалістичності і забезпечує конфіденційність реальних даних. Для створення означеної методики проведено аналіз особливостей інтеграції процесу розробки баз даних в DevOps, виявлено проблемні з точки зору конфіденційності операції в конвеєрі доставки DevOps. Такими визначені операції тестування БД, що виконуються на різних етапах DevOps. Для збереження адекватності даних предметної області з одного боку, і забезпечення їх конфіденційності з іншого, запропоновано послідовність перетворень інформації в базі даних, що відповідає зазначеним умовам.

Ключові слова: *бази даних, DevOps, маскування даних, тестування бази даних.*

АКТУАЛЬНІСТЬ І ПОСТАНОВКА ЗАДАЧІ

DevOps – це методологія, заснована на наступних концепціях:

- автоматизація процесів розробки, розгортання, документації, тестування і моніторингу між розробниками програмного забезпечення та інженерами з експлуатації;
- інтеграція процесів розробки і експлуатації для ефективної синхронізації, перевірки, управління і застосування змін бази даних.

DevOps впевнено завойовує позиції в світі розробки додатків, постійно вдосконалюючи процеси доставки і інструменти, необхідні для їх підтримки.

Однак здебільшого DevOps зосередився в першу чергу на самому додатку, а розробка баз даних здійснюється за традиційною технологією. Процеси і інструменти DevOps були розроблені для створення коду і розгортання додатків, а не для особливостей управління базами даних.

Однак, більшість змін коду програми, над якими працюють розробники програмного забезпечення, також впливає на код бази даних. Згідно з дослідженням State of Database Deploymentsin Application Delivery [1], 57% всіх змін додатків вимагають відповідної зміни бази даних. Розробники часто повинні чекати завершення змін в базі даних (зроблених адміністраторами баз даних), перш ніж вони зможуть продовжити свою роботу. Такі затримки негативно позначаються на загальній продуктивності команди.

Лише недавно бази даних були істотно включені в процеси DevOps, з'явилися інструментальні засоби і технології їх використання. Практика Database DevOps спрямована на підвищення ефективності управління базами даних. Така практика допомагає оптимізувати процес розгортання і модифікації баз даних, даючи можливість автоматизувати багато аспектів життєвого циклу бази даних. Метою інтеграції змін бази даних як частини процесу DevOps є підвищення швидкості доставки будь-яких змін бази даних. Особливості бази даних як об'єкта інформаційної системи призводять до того, що їх повноцінна автоматизація тестування можлива тільки на да-

них, максимально наближених до реальних даних у виробничій базі даних. Однак використання реальних даних породжує обґрунтовані ризики порушення конфіденційності даних.

Постановка задачі. Основним завданням є розробка методики забезпечення конфіденційності інформації в базах даних, що беруть участь процесі розробки DevOps. Рішення поставленого завдання включає в себе:

- аналіз особливостей інтеграції процесу розробки баз даних в DevOps;
- виявлення проблемних з точки зору конфіденційності етапів в конвеєрі доставки DevOps;
- пропозиція методики забезпечення конфіденційності даних в процесі тестування бази даних.

Проблеми інтеграції розробки бази даних в процес DevOps

Одне з істотних відмінностей між розгортанням додатків і баз даних полягає в тісній інтеграції з даними. З базами даних не можна просто перезаписувати схеми або копіювати зміни з одного середовища в інше, не враховуючи впливу на дані.

Необхідно ретельно продумати, як зміни впливають на існуючі дані і як зберегти ці дані при внесенні змін. В іншому випадку виникають ризики втрати даних, порушення їх цілісності і конфіденційності.

Проблеми з масштабністю в базах даних також відрізняються від аналогічних проблем коду програми. Впровадження змін схеми в об'ємну базу даних у виробничому середовищі може виявитися серйозним заходом і призвести до проблем з продуктивністю і погіршення якості обслуговування. Зміни бази даних стають ще більш складними, якщо до бази даних звертається кілька додатків. Згідно зі звітом Accelerate: State of DevOps [2], зміни бази даних часто є основним джерелом ризику і затримок при розгортанні. При цьому в звіті говориться, що інтеграція баз даних в процес доставки додатків може справити позитивний вплив на безперервність доставки. Традиційне середовище розробки зміни в базі даних може затримувати доставку додатків, знижуючи продуктивність і витрати. Іншими словами, змінами в базі даних слід керувати так само, як і змінами додатків.

Інтеграція розгортання бази даних в процес DevOps

Підхід DevOps до процесу розробки баз даних робить цей процес більш гнучким і адаптованим. Команди реалізують більш дрібні і більш часті зміни в рамках скоординованих зусиль, отримуючи при цьому постійний зворотний зв'язок про процеси доставки і компоненти програми. Таким чином, розробка і розгортання бази даних більше не відключаються від загального робочого процесу доставки, а інтегруються в процес протягом усього життєвого циклу додатку. На рис. 1 представлена ілюстрація того, як може виглядати процес DevOps бази даних при інтеграції в конвеєр доставки додатків.

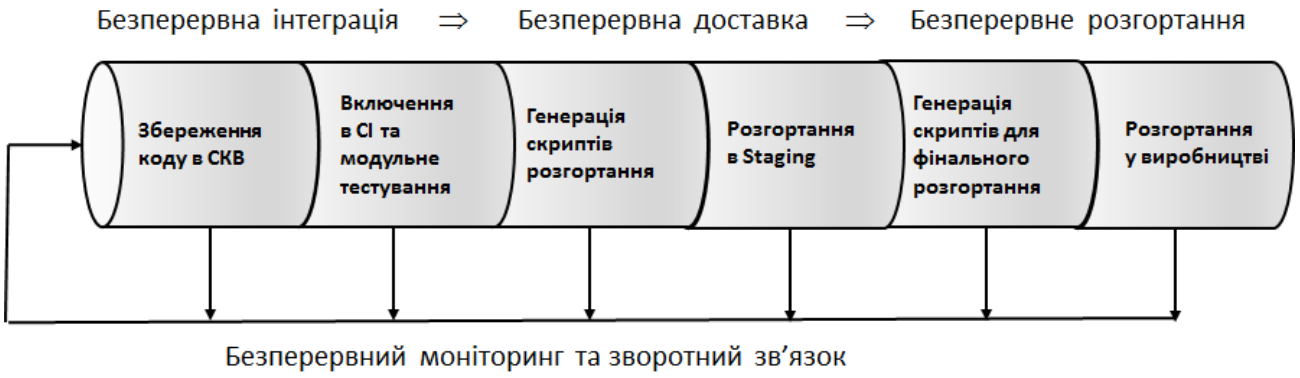

Рис. 1. Інтеграція розгортання бази даних в процес DevOps

Перший крок – розробники баз даних повинні зберігати весь код бази даних в системі контролю версій (СКВ). Сюди входять сценарії, які використовуються для створення баз даних і її

зміни. Наприклад, сценарії DDL, які створюють, змінюють або видаляють об'єкти бази даних, такі як таблиці, представлення, індекси, ролі, функції, тригери або збережені процедури. А також будь-які сценарії DML, які змінюють дані.

Прикладом управління вихідним кодом для SQL Server є SQL Server DataTools (SSDT)[3]. У цьому інструменті всі об'єкти бази даних виражаються в термінах сценаріїв CREATE. Це може бути зручно, якщо необхідно створити прототип нового об'єкта за допомогою SQL, а потім вставити остаточний варіант безпосередньо в проект бази даних в системі контролю версій. Інший приклад управління вихідним кодом – Entity Framework Migrations [4]. Замість сценаріїв SQL, база даних в основному представлена класами C# або VB.

Коли код реєструється в репозиторії, включається служба Continuous Integration (CI) і запускає серію команд, які виконують такі завдання, як компіляція вихідного коду і запуск модульних тестів. Інструменти розгортання визначають, які зміни необхідні, порівнюючи поточний стан бази даних з версією в системі контролю версій. Це дозволяє швидко вносити зміни в базу даних і отримувати результати. Якщо база даних є частиною розгортання, служба CI тестує і оновлює базу даних або попереджає розробників про помилки. При виявленні проблем розробники можуть виправити їх і повторно відправити код, який перезапустить процес CI.

Результатом обробки коду SQL службою CI є так званий артефакт, який включає сценарій розгортання, необхідний для поновлення відповідних об'єктів бази даних і статичних даних. Артефакт також містить детальну інформацію про процес розгортання, в тому числі звіт про відмінності, який показує, що змінилося в базі даних. Артефакт являє собою конкретну підтверджену версію, яку можна використовувати в якості відправної точки для процесу випуску бази даних.

Артефакт розгортається в проміжній базі даних, яка в ідеалі є точною копією виробничої бази даних (в термінах DevOps цей процес називається staging).

Тут адміністратори баз даних можуть перевірити зміни і переконатися, що проміжна база даних готова до роботи. При необхідності вони також можуть зробити додаткові зміни. За результатами staging-тестування створюється остаточний сценарій розгортання, який потім може бути переданий по конвеєру для розгортання в виробничій базі даних.

Тестування бази даних

Одним з найбільш важливих аспектів конвеєра доставки додатків є постійне тестування. Тестування грає важливу роль в забезпеченні загальної безпеки бази даних. Зміни в базі даних повинні пройти ретельну перевірку, перш ніж випуск досягне проміжного середовища. Таким чином, розробники дізнаються про проблеми швидко, на ранніх етапах процесу розробки. При виявленні проблем вони відразу ж отримують повідомлення, щоб відразу перевірити виправлення в системі контролю версій і продовжити розробку.

Розробники та адміністратори баз даних повинні створювати тести, які передбачають якомога більше сценаріїв, щоб виявити якомога більше недоробок, перш ніж зміни бази даних потраплять в робоче середовище. Ці тести можуть бути засновані на фреймворках, специфічних для тестування баз даних. Наприклад, tSQLt – фреймворк з відкритим вихідним кодом для SQL Server, який дозволяє створювати модульні тести на T-SQL [5] для швидкої перевірки певних функцій або даних.

Крім модульних, на інших етапах конвеєра, використовуються і інші види тестування [6], наприклад, User Acceptance Testing(UAT).

Важливою особливістю тестування баз даних є реалістичність тестових даних. У випадках, коли додаток ніколи не тестується на реальних даних, поки він не випущений в виробництво, цілком ймовірно виникнення проблем, пов'язаних з даними. Код може не працює, тому що він зустрічає несподівані значення даних (поза очікуваного діапазону значень) або несподівані типи даних, або він працює повільно, тому що виконувані ним запити були протестовані на тому обсязі і

розподілі даних, які повністю відрізняються від того, що присутній у виробничому середовищі. Завдання використання даних близьких до реальних на ранніх етапах тестування є актуальним.

При цьому, однак, виникає проблема забезпечення конфіденційності реальних даних. Одним із шляхів вирішення цієї проблеми є використання в тестах маскованих реальних даних. Причому, операція маскування повинна виконуватися кожного разу, коли дані заносяться в базу.

Методика маскування даних

Для маскування баз даних можуть використовуватись сценарії ручного маскування даних з виробничого середовища перед тим, як зробити їх доступними для середовищ розробки і тестування.

Однак виробничі дані мають великий розмір, і створення сценаріїв для ретельної «деідентифікації» цих даних вимагає часу, і також необхідно підтримувати такі сценарії. Функція маскування представлена в сучасних СУБД.

Наприклад, така функція реалізована SQL Server 2014 і пізніших. Можна налаштовувати видимість даних для різних груп/ролей, тобто маскування є динамічним (dynamic datamasking, DDM). Однак, в такій реалізації існують деякі обмеження.

Наприклад, заміна порції конфіденційних даних однаковим набором символів («XXXXX»). Крім того, необхідно змінювати структуру бази даних. Головне, DDM не розроблений для захисту даних в таких середовищах, як середовище розробки, де розробники зазвичай працюють з підвищеними привілеями і можуть виконувати спеціальні запити.

По-перше, будь-який розробник з привілеями 'dbo' або 'sa' навіть не побачить маскування, а по-друге, в механізмі маскування є уразливості, які означають, що навіть без підвищених привілеїв спеціальні запити можуть розкривати конфіденційні дані [7]. Тестові дані в маскованому вигляді повинні відповідати реальним даним і в той же час не повинні давати можливість відновити вихідні дані. Дане обмеження можна представити у вигляді вимог:

- за тестовими даними не можна відновити вихідні дані;
- збереження типу даних. Результати маскування повинні відноситись до тих же базових типів даних, що і вихідні дані;
- масковані дані повинні зберігати приналежності до певного домену;
- збереження формату. Масковані дані повинні мати таку ж структуру, що і вихідні дані.

Ці вимоги важливі, особливо при маскуванні стовпців таблиць, що індексуються.

Авторами розроблена методика маскування, що відповідає таким вимогам [8]. Важливими для DevOps є властивості запропонованої методики:

- застосування алгоритму маскування до всього безлічі даних домену;
- маскування може бути автоматизованим, легко повторюваним процесом.

На додаток до перерахованих вище вимогам, запропонована методика дозволяє:

- збереження регістра символів;
- збереження довжини строкових змінних;
- знаходження голосних і приголосних букв на тих же позиціях, що і в вихідному рядку;
- заміна алфавітного символу на алфавітний, що належить до тієї ж кодової сторінці;
- заміна цифр цифрами;
- можливість залишати без змін деяких символів або підрядка початкового рядка.

В роботі [8] продемонстровані результати маскування стосовно до базових типів - рядок, число, дата. Як приклад, наведемо дані до маскування (рис. 2). І після маскування (рис. 3).

Як зазначалося, методика дозволяє залишити частину даних без змін, наприклад, залишити незміненим домен верхнього рівня в поштовій адресі або код оператора в телефонному номері.

	ФИО	ИНН	Почта	Адрес	Кредит карта	Баланс	Срок действия
1	Петренко П.І.	0123456789	petrenko@gmail.com	Одеса, вул. Буніна, 99	01234 1234 2345 3456	1000,00	2017-01-08 11:58:10.547
2	Петренко П.І.	0123456789	petrenko@gmail.com	Одеса, вул. Буніна, 99	01234 1234 2345 3456	1000,00	2017-01-08 11:58:10.547
3	Петренко П.І.	0123456789	petrenko@gmail.com	Одеса, вул. Буніна, 99	01234 1234 2345 3456	1000,00	2017-01-08 11:58:10.547
4	Петренко П.І.	0123456789	petrenko@gmail.com	Одеса, вул. Буніна, 99	01234 1234 2345 3456	1000,00	2017-01-08 11:58:10.547
5	Петренко П.І.	0123456789	petrenko@gmail.com	Одеса, вул. Буніна, 99	01234 1234 2345 3456	1000,00	2017-01-08 11:58:10.547
6	Петренко П.І.	0123456789	petrenko@gmail.com	Одеса, вул. Буніна, 99	01234 1234 2345 3456	1000,00	2017-01-08 11:58:10.547
7	Петренко П.І.	0123456789	petrenko@gmail.com	Одеса, вул. Буніна, 99	01234 1234 2345 3456	1000,00	2017-01-08 11:58:10.547
8	Петренко П.І.	0123456789	petrenko@gmail.com	Одеса, вул. Буніна, 99	01234 1234 2345 3456	1000,00	2017-01-08 11:58:10.547
9	Петренко П.І.	0123456789	petrenko@gmail.com	Одеса, вул. Буніна, 99	01234 1234 2345 3456	1000,00	2017-01-08 11:58:10.550
10	Петренко П.І.	0123456789	petrenko@gmail.com	Одеса, вул. Буніна, 99	01234 1234 2345 3456	1000,00	2017-01-08 11:58:10.550

Рис. 2. Вихідна таблиця з не маскованими даними

	ФИО	ИНН	Почта	Адрес	Кредит карта	Баланс	Срок действия
1	Дуккєфгу П.О.	3127217273	zustubyi@rjeuw.xiw	Єнїъї, щір. ТирЇлё, 88	10964 4406 6696 9180	544,57	8971-10-06 00:00:
2	Вйбчендй Ш.И.	4197952415	qabwutni@lwiuq.den	Єнело, віф. Цїзєли, 93	13476 0501 3515 4461	2688,61	1753-11-20 00:00:
3	Нёщжікди Т.У.	2649869350	quprimwo@mvioc.ror	Йшіфе, кем. МанУза, 19	99029 5717 6114 9508	2807,64	2179-11-15 00:00:
4	Шіжвибрї Ж.І.	0035976257	wulqehla@jraof.qit	Єжупи, фец. ПёпЄси, 42	48470 0471 9788 9679	1501,81	4410-05-16 00:00:
5	Віхпімбї Ф.Ї.	0960690375	poyyupgo@pxuaw.nuw	Єдозё, шєщ. Ъогlшо, 78	22700 7771 0021 3805	8028,37	3951-02-25 00:00:
6	Гётъётчё М.І.	3547242765	felhacgo@zsaeg.nuk	Урежи, гиш. ЪипЙто, 14	63856 3457 1419 2159	9758,71	2299-02-14 00:00:
7	Феврецсї Ш.И.	5124997791	hanvahpo@qdoin.lom	Ётйзй, шуф. БёшЕно, 71	51712 5674 9906 7733	9618,15	5306-10-27 00:00:
8	Роцрёпцє Ш.І.	4531608181	qeckabja@dpeom.pis	Їлижи, ъиъ. БущИші, 97	97114 8204 5043 3562	6239,65	8088-08-23 00:00:
9	Хидгїгтї Ъ.Й.	6328386618	qotnuwyi@lbuoy.xem	Одёфа, ріф. РапЄщї, 01	52252 8378 4548 3009	212,10	3535-10-09 00:00:
10	Бенцоччй Г.И.	1535915908	gicmapfu@zpeul.kap	Ичепо, мёт. НёгЕва, 57	88936 5210 9604 1828	654,39	8826-01-01 00:00:

Рис. 3. Таблиця з маскованими даними

Методика в роботі [8] не розглядає важливий аспект цілісності бази даних - цілісність посилань. Якщо в таблицях використовуються природні (змістовні) ключі, що вимагають маскування, то в результаті статичного маскування може порушитися посилальна цілісність бази даних, що зробить базу даних непрацюючою. Рішення завдання забезпечення цілісності посилань розглянуто авторами в роботі [9]. Дана методика передбачає де-ідентифікацію первинних ключів, після чого відпадає необхідність їх маскування в основних і зв'язаних таблицях за методикою [8].

Тестові дані є узгодженими і захищеними, що забезпечують коректність тестів. Перетворені за методиками [8,9] конфіденційні дані не будуть доступні для перегляду в немаскованій формі в середовищі розробки.

Створивши копію маскованої бази даних на тестовому сервері, можна повторно використовувати файл з даними для генерації тих же даних, кожен раз, коли нам потрібно створювати нову копію цієї бази даних для цілей тестування. Для цього можна використовувати утиліту BCP для експорту даних у власному форматі з будь-яких таблиць з замаскованими даними в файл даних. Потім створити базу даних з системі контролю версій і за допомогою BCP імпортувати дані з файлу.

ВИСНОВОК

Запропонована методика захисту конфіденційних даних, коли вони знаходиться за межами виробничого середовища, дозволяє адекватно імітували те, що знаходиться у виробничому середовищі. В запропонованій методиці є кілька переваг:

- дані, доступні в невиробничому середовищі, контролюються і витік конфіденційних даних на етапі тестування виключається;
- процес створення тестових даних легко автоматизується. Можна отримати захищений набір даних в будь-який час, коли потрібно;
- можна зберегти невеликий набір даних, щоб скоротити час тестування і прискорити процес розробки. Використання системи контролю версій, безпечного тестування і автоматизації процесу доставки дозволяє отримати більш передбачуваний і простий в управлінні процес, який відповідає вимогам безпеки даних.

ЛІТЕРАТУРА

[1] Liquibase.URL:https://www.liquibase.com/resources/reports/database-deployments-2019.

[2] DORA-State of DevOps. URL: http: // cloud-platformonline.com / rs / 248-TPC-86 / images/ DORA-State of DevOps.pdf.

[3] SQL Server 2015 Release Notes. URL: https:// docs.microsoft.com/en-us/sql/ssdt/download-sql-server-data-tools-ssdt?redirectedfrom=MSDN &view=sql-server-ver15.

[4] Framework documentation. URL: https://docs.microsoft.com / en-us / aspnet / mvc / overview / gettingstarted / getting-started-with-ef-using-mvc / migrations-and-deployment-with-the-entity-fra mework-in-an-asp-net-mvc-application.

[5] TSQLT. Full user guide. URL: https: // tsqlt. Org / full-user-guide /.

[6] 4 best types of software testing. URL: https:// dzone.com/articles/4-best-types-of-software-testing.

[7] Unmasking the Dynamic Data Masking. URL: https://www.red-gate.com/simple-talk/ blogs/unmasking-the-dynamic-data-masking/.

[8] Коломыцев М.В., Носок С.А., Мазуренко А.Е. Маскирование таблиц базы данных с использованием технологии SQL CLR // Захист інформації – 2017. – Т.19, № 1. - С.16-22.

[9] Коломыцев М.В., Носок С.А., Мазуренко А.Е. Обеспечение целостности внешних ключей маскированной базы данных // Захист інформації – 2015. – Т.17, № 5. – С.306-311.

ENSURING THE CONFIDENTIALITY OF DEVOPS DATABASES

Data base development integration in to the DevOps delivery pipeline is a nessential challenge due to the growing popularity of this application development methodology. The goal of integrating data base changes as part of the DevOps process is to speed up the delivery of any data base changes. Data base DevOps practice aim storm prove the data base management efficiency. This practice helps optimize the process of data base deployment and modification, making it possible to auto mate many aspects of the data base life cycle. However,
the implementation of Data base DevOps faces certa in challenges. The peculiarities of the database as an object of the information system lead to the fact that their full automation of testing is possible only on data that are as close as possible to the real data in the production database. However, the use of real data creates legitimate data confidentiality risks. The article discusses the methodology of data preparation for testing, meet the requirements of realism and provides the confidentiality of real data. To create this methodology, the analysis of the database development process integration in DevOps features carried out and problematic identified in terms of operation confidentiality in the DevOps delivery pipeline. These are DB testing operations performed at different stages of DevOps.To maintain the adequacy of the subject area data on the one hand, and ensure its confidentiality on the other hand, a sequence of transformations of information in the database that meets these conditions is proposed.

Key words: data base, DevOps, data masking, database testing.

Коломиец Михайло Володимирович, кандидат технічних наук, доцент Фізико-технічного інституту КПІ ім. Ігоря Сікорського.

Kolomytsev Myhailo, candidate of technical sciences, associate professor of Institute of Physics and Technologies of the National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute".
Orcid ID: 0000-0001-8460-3041.
E-mail: box144.85@gmail.com.

Носок Світлана Олександрівна, кандидат технічних наук, доцент Фізико-технічного інституту КПІ ім. Ігоря Сікорського.

Nosok Svitlana, candidate of technical sciences, associate professor of Institute of Physics and Technologies of the National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute".
Orcid ID: 0000-0002-0016-9346.
E-mail: nos.sv.ol@gmail.com.