DOI: https://doi.org/10.36910/6775-2524-0560-2024-56-12
УДК 004.4`2
Бегліцов Сергій Валерійович, аспірант
http://orcid.org/0009-0000-9690-9881
Державний університет інформаційно-комунікаційних технологій, м. Київ, Україна

АВТОМАТИЗАЦІЯ МІГРАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ У ХМАРНУ АРХІТЕКТУРУ ІЗ ВИКОРИСТАННЯМ ІНСТРУМЕНТУ ІНФРАСТРУКТУРИ ЯК КОД TERRAFORM У СЕРЕДОВИЩІ AWS

Бегліцов С.В. Автоматизація міграції програмного забезпечення у хмарну архітектуру із використанням інструменту інфраструктури як код Terraform у середовищі AWS. У цій статті розглядається комплексний підхід до автоматизації надання інфраструктури за допомогою Terraform і AWS. Основною метою цього дослідження було оптимізувати розгортання та керування хмарними ресурсами за допомогою практик інфраструктури як коду (IaC). Стаття починається з пояснення фундаментальних принципів інфраструктури як коду, підкреслюючи її переваги з точки зору послідовності, повторюваності та співпраці. Центральне місце в цій методології займає Terraform, інструмент із відкритим кодом, розроблений для декларативного керування конфігурацією. Здатність Terraform визначати компоненти інфраструктури за допомогою простих, зрозумілих для людини файлів конфігурації забезпечує потужний механізм для надання та керування хмарними ресурсами в різних провайдерів. Ключовим моментом реалізації є використання AWS як хмарного провайдера. AWS пропонує широкий спектр послуг, від обчислювальних екземплярів (EC2) до керованих баз даних (RDS) і безсерверних обчислень (Lambda). У статті розглядаються конкретні випадки використання, коли Terraform використовується для автоматизації надання ресурсів AWS, таких як екземпляри EC2 для масштабованих веб-додатків, бакети S3 для зберігання об'єктів і конфігурації VPC для ізоляції мережі. Важливим аспектом, який обговорюється в статті, є керування станом Terraform, який відстежує поточний стан розгорнутих ресурсів і полегшує співпрацю між членами команди. Стратегії обробки файлів стану, включно з конфігураціями віддаленої серверної частини за допомогою Amazon S3 для блокування, вивчаються, щоб забезпечити узгодженість і уникнути конфліктів під час одночасного розгортання. Крім того, стаття стосується модулів Terraform, які інкапсулюють багаторазові конфігурації для різних компонентів інфраструктури, сприяючи багаторазовому використанню коду та зручності обслуговування. Завдяки модульній конфігурації команди можуть стандартизувати найкращі практики та застосовувати політики в різних середовищах, від розробки до виробництва. Інтеграція Terraform з конвеєрами CI/CD автоматизує тестування та розгортання змін інфраструктури, сприяючи гнучкості та зменшуючи ручне втручання. Ця автоматизація не тільки прискорює доставку оновлень інфраструктури, але й підвищує загальну надійність системи за допомогою автоматизованого тестування та механізмів відкату.
Ключові слова: Terraform, AWS, інфраструктура як код, керування хмарою, автоматизація, конвеєр

Behlitsov S. Software migration to a cloud architecture automation using an infrastructure as code tool Terraform AWS environment. This article explores a comprehensive approach to automating infrastructure provisioning using Terraform and AWS. The primary objective of this study was to streamline the deployment and management of cloud resources through Infrastructure as Code (IaC) practices. The article begins by elucidating the fundamental principles of Infrastructure as Code, highlighting its benefits in terms of consistency, repeatability, and collaboration. Central to this methodology is Terraform, an open-source tool designed for declarative configuration management. Terraform's ability to define infrastructure components using simple, human-readable configuration files provides a powerful mechanism for provisioning and managing cloud resources across different providers. Key to the implementation is the utilization of AWS (Amazon Web Services) as the cloud provider. AWS offers a vast array of services, from compute instances (EC2) to managed databases (RDS) and serverless computing (Lambda). The article delves into specific use cases where Terraform is leveraged to automate the provisioning of AWS resources, such as EC2 instances for scalable web applications, S3 buckets for object storage, and VPC configurations for network isolation. A critical aspect discussed in the article is the management of Terraform state, which tracks the current state of deployed resources and facilitates collaboration among team members. Strategies for handling state files, including remote backend configurations with Amazon S3 for locking, are explored to ensure consistency and avoid conflicts in concurrent deployments. Furthermore, the article addresses advanced topics such as Terraform modules, which encapsulate reusable configurations for different components of an infrastructure, promoting code reusability and maintainability. By modularizing configurations, teams can standardize best practices and enforce policies across various environments, from development to production. Integrating Terraform with CI/CD pipelines automates the testing and deployment of infrastructure changes, promoting agility and reducing manual intervention. This automation not only accelerates the delivery of infrastructure updates but also enhances overall system reliability through automated testing and rollback mechanisms.
Key words: Terraform, AWS, infrastructure as code, cloud management, automation, pipeline

Вступ та постановка проблеми. Хмарна архітектура програмного забезпечення представляє ряд важливих переваг, одними з яких є низькі капіталовкладення та швидкі можливості входу. Тим не менш, розробка алгоритму міграції певної програми на архітектуру хмарного середовища з наземної інфраструктури, представляє певний виклик, як і будь-який звичайний проект розробки програмного забезпечення. Наприклад, вибір хмарного провайдера або адаптація архітектури та вихідного коду. Інша проблема полягає в тому, щоб адаптувати програму так, щоб вона також мала специфічні для хмари функції, такі як багатокористування або масштабованість.

Аналіз останніх досліджень і публікацій. У науково-дослідницькому просторі сьогодення з'являються роботи, присвячені розробці методів та засобів автоматизації процесів переключення сервісів наземної інфраструктури в хмарне середовище.

У роботі [1] представлено результати дослідження продуктивності та доступності двох інструментів «Інфраструктура як код» (IaC) – Google Cloud Deployment Manager і Terraform. У дослідженні оцінюється розгортання та керування хмарними ресурсами за допомогою цих інструментів, а також розглядаються переваги їх інтеграції, гнучкість і мультихмарні можливості. Вона підкреслює незмінно швидший час знищення ресурсів Terraform порівняно з Google Cloud Deployment Manager, про що свідчать тести з використанням команди «time» Linux, надаючи розуміння їхньої операційної ефективності та адаптивності в хмарних середовищах. Висновки свідчать про те, що рідний Google Cloud Deployment Manager краще інтегрується з його ресурсами. На противагу цьому Terraform є універсальним, забезпечує кращу гнучкість і швидше розгортається. Його незалежність більш корисна при роботі з різними хмарними середовищами – і не обмежується GCP. Крім того, щоб отримати розуміння, у цьому дослідженні обговорюються переваги та внутрішні проблеми, пов'язані з IaC. Підкреслюється важливість технічного досвіду для ефективного використання цих інструментів для управління хмарною інфраструктурою.

Дослідження дисертації [2], проведене AIT GmbH & Co. KG, зосереджено на перенесенні їхнього власного плагіна TFS ASAP із локального середовища на хмарну платформу Microsoft, зокрема Visual Studio Online (VSO) на Microsoft Azure. Керуючись такими перевагами хмарних обчислень, як скорочення капіталовкладень і масштабованість, AIT прагне використовувати можливості хмари та інтегрувати TFS якнайшвидше у VSO, забезпечуючи модулі обробки подій із можливістю використання кількох орендарів і масштабування. У дисертації досліджуються існуючі методології хмарної міграції та адаптуються вони відповідно до міграції TFS якнайшвидше. Висвітлюються такі проблеми, як вибір хмарних провайдерів, адаптація архітектури та вихідного коду, а також забезпечення реалізації специфічних для хмари функцій, таких як багатокористування та масштабованість. Стратегічне узгодження AIT з Microsoft диктує використання служб Microsoft Azure для цієї міграції. Перехід TFS вважався успішним, незважаючи на те, що він не повністю відповідав всім вимогам до функціональності та тестування, викладеним спочатку. TFS ASAP Online успішно розгорнуто на хмарній платформі Microsoft, що робить його доступним у всьому світі. Основні функціональні вимоги були реалізовані. Протягом усього процесу міграції AIT встановив контакти зі співробітниками Microsoft, у результаті чого TFS ASAP Online було офіційно внесено до списку як розширення для Visual Studio Online. Спочатку проект у рамках дипломної роботи TFS ASAP Online перетворився на повноцінний продукт у портфоліо AIT.

Крім того, варто зазначити праці наступних науковців: Чинтапатла С. [3], Джонсон А. [4], Рахман А., Парто А., Моррісон П., Вільямс Л. [5], Сміт Л. [6], Далла Пальма С., Ді Нуччі Д., Паломба Ф., Тамбуррі Д.А. [7], Браун М. [8], Оздоган Е., Серан О., Устюндаг М. [9], Фейтоса Д., Пенча М.-Т., Берарді М., Боза Р.-Д. [10], Бату Дж. [11], Абулкішик А., Алван А., Гульзар Ю. [12], Елеракі М., Аніс Азіз Ст., Соліман Дж. [13], Мухопадьяй Н., Теварі Би. 14], Омофоєва Ю., Гребі А. [15] та інших.

Проте, беручи до уваги вище зазначену наукову документацію, питання, пов'язане з методологією по розробці методів та засобів автоматизації процесів переключення сервісів наземної інфраструктури в хмарне середовище, все ще залишається недостатньо дослідженим та потребує подальшого опрацювання.

Постановка завдання. Метою роботи є розробка методів та засобів автоматизації процесів переключення сервісів наземної інфраструктури в хмарне середовище із використанням інструменту інфраструктури як код Terraform у середовищі AWS.

Викладення основного матеріалу дослідження. Архітектура на основі хмарних обчислень зазвичай характеризується п'ятьма рисами:

– самообслуговування на вимогу, відповідно якому користувач може автоматично вимагати необхідні ресурси, коли це необхідно;

– широкий доступ до мережі, що означає здійснення доступу до ресурсів через Інтернет за допомогою стандартних механізмів;

– об'єднання ресурсів, завдяки якому ресурси управляються в пулі у віртуалізованій формі та можуть використовуватися кількома користувачами паралельно;

– швидка еластичність, яка передбачає можливість надання додаткових ресурсів користувачеві за потреби та звільнення, коли вони більше не потрібні;

– розмірене обслуговування, згідно якому споживання ресурсів контролюється та вимірюється з метою оптимізації.

Концепція хмарної інфраструктури реалізується завдяки трьом основним типам моделей, а саме програмне забезпечення як послуга (Software as a Service – SaaS), інфраструктура як послуга (Infrastructure as a Service – IaaS) і платформа як послуга (Platform as a Service – PaaS). Хмарна IaaS уможливлює впровадження платформи інфраструктури обладнання та програмного забезпечення в якості послуг. Згідно з концепцією хмарної PaaS користувачі мають можливість реалізовувати розроблюване програмне забезпечення використовувати існуючу платформу. В той же час використання концепції SaaS є можливість користуватися програмних забезпеченням лише в хмарній інфраструктурі. У своїй суті хмарні обчислення також можуть впроваджувати різні елементи програмного забезпечення в якості «коду», що здійснюється на підґрунті ідеї Інфраструктури як код (IaC – Infrastructure as Code), що є провідним об'єктом дослідження поточної роботи.

Автоматизація міграційних процесів у контексті переходу з наземної архітектури на хмарну, напряму залежить від характеру самої міграції. Найменш інвазивним типом хмарної міграції передбачає заміну окремих компонентів хмарними сервісами, в ході чого може знадобитися додаткова конфігурація та адаптація, щоб усунути потенційну несумісність компонентів, які переносяться. В ході часткової міграції здійснюється перенесення певних функцій програми. Міграція повного стеку передбачає міграцію всієї програми повністю, що є класичним випадком хмарної міграції. Зазвичай цей процес здійснюється шляхом інкапсуляції програми у віртуальну машину та запуску її в хмарному середовищі. Міграція за типом «Cloudify» передбачає повну трансформацію. У цьому відношенні, Міграція всього стека передбачає переміщення всієї наявної локальної інфраструктури в хмару з мінімальними змінами, часто використовуючи стратегію підйому та переміщення, в той час як, міграція за типом «Cloudify» означає перетворення додатків на хмарні, використовуючи специфічні функції та сервіси, що зазвичай передбачає перебудову архітектури або рефакторинг.

Визначення елементів як коду дозволяє швидко та узгоджено створювати кілька екземплярів. Це забезпечує швидкий ремонт і реконструкцію, забезпечуючи однакову структуру елементів кожного разу, роблячи поведінку системи передбачуваною, а тестування більш надійним. Це полегшує постійне тестування та доставку. Крім того, код забезпечує чіткий план того, як був побудований кожен елемент, підвищуючи прозорість.

У контексті автоматизації міграції IaC дозволяє безперебійно та узгоджено розгортати компоненти інфраструктури в хмарному середовищі. Це мінімізує ризик людської помилки та гарантує, що всі компоненти розгортаються стандартизованим способом. Безперервне тестування та виконання за допомогою IaC сприяють продуктивній інтеграції та тестуванню протягом усього процесу міграції. Цей ітеративний підхід покращує загальну ефективність і надійність міграції.

У випадку даного дослідження увага зосереджується на використанні інструменту Інфраструктури як код Terraform у автоматизації міграції програмного забезпечення, що, насамперед мотивується наданням Terraform можливості визначати хмарну інфраструктуру та керувати нею за допомогою мови декларативної конфігурації. В основі Terraform лежить його декларативний синтаксис, який дозволяє користувачам вказувати бажаний стан файлів конфігурації інфраструктури. Ці конфігураційні файли, написані мовою конфігурації HashiCorp (HCL), коротко описують ресурси, залежності та зв'язки між різними компонентами інфраструктури. Потім Terraform автоматизує надання та керування цими ресурсами, щоб переконатися, що фактичний стан збігається з бажаним станом, указаним у файлах конфігурації. Terraform автоматизує надання хмарних ресурсів шляхом взаємодії з API, наданими хмарними провайдерами. Він перетворює бажаний стан, указаний у файлах конфігурації, у виклики API для надання та налаштування необхідних ресурсів. Постачальники ресурсів Terraform абстрагуються від складнощів взаємодії з хмарними API, дозволяючи користувачам зосередитися на визначенні бажаної конфігурації інфраструктури. Зокрема, даний інструмент пропонує підтримку багатьох хмарних провайдерів, включаючи AWS, Azure, Google Cloud Platform та інших.

Зокрема, змінні Terraform дозволяють користувачам параметризувати свої конфігурації та налаштовувати розгортання на основі динамічних вхідних даних. Змінні можуть бути визначені у файлах конфігурації Terraform або надані зовні за допомогою різних методів, таких як змінні середовища, прапорці командного рядка або файли змінних. Динамічне надання змінних забезпечує

гнучке та масштабоване розгортання інфраструктури, дозволяючи організаціям плавно адаптуватися до мінливих вимог і середовищ

Terraform Cloud є розміщеною службою, яка забезпечує централізоване керування станом, функції співпраці та інтеграцію з контролем версій і пайплайнами CI/CD. Він пропонує набір функцій, призначених для спрощення надання інфраструктури, співпраці та керування. Він забезпечує централізоване розташування для зберігання та керування файлами стану Terraform, забезпечуючи послідовність і одночасність між членами команди та середовищами. Робочі простори в Terraform Cloud дають змогу командам співпрацювати над налаштуваннями інфраструктури, ділитися змінними та керувати контролем доступу, сприяючи співпраці та повторному використанню коду. До того ж він легко інтегрується з системами контролю версій, забезпечуючи керування версіями, відстеження змін і можливості відкату для конфігурацій інфраструктури. Також, ця платформа підтримує політику у вигляді коду через Sentinel, що дозволяє організаціям запроваджувати політику управління, відповідності та безпеки в усіх інфраструктурних розгортаннях.

Поточне дослідження зосереджується на впровадженні функціональних можливостей Terraform та платформи Terraform Cloud у інтеграції з хмарними сервісами AWS (Amazon Web Services) задля автоматизації процесу міграції додатку електронної комерції. Додатковими пре реквізитами є S3 бакет для зберігання файлів стану Terraform та можливість створення сценаріїв Bash для автоматизації.

Процес розпочинається визначенням необхідних ресурсів AWS за допомогою файлів конфігурації Terraform. У таблиці 1 міститься функція AWS Lambda та API Gateway.hcl. Нижче описано визначення функції AWS Lambda та шлюзу API для надання функції Lambda через кінцеву точку.

```
resource "aws_lambda_function" {
  function_name = "my-lambda-function"
  runtime       = "nodejs14.x"
  handler       = "index.handler"
  filename      = "${path.module}/lambda_function_payload.zip"
}

resource "aws_api_gateway_rest_api" {
  name        = "my-api-gateway"
  description = "My API Gateway"
}

resource "aws_api_gateway_resource" {
  rest_api_id = aws_api_gateway_rest_api.id
  parent_id   = aws_api_gateway_rest_api.root_resource_id
}

resource "aws_api_gateway_method" {
  rest_api_id   = aws_api_gateway_rest_api.id
  resource_id   = aws_api_gateway_resource.id
  http_method   = "GET"
  authorization = "NONE"
}

resource "aws_api_gateway_integration" {
  rest_api_id             = aws_api_gateway_rest_api.id
  resource_id             = aws_api_gateway_resource.id
  http_method             = aws_api_gateway_method.http_method
  integration_http_method = "POST"
  type                    = "AWS_PROXY"
  uri                     = aws_lambda_function.invoke_arn
}
```

```
resource "aws_api_gateway_deployment" " {
  depends_on  = [aws_api_gateway_integration]
  rest_api_id = aws_api_gateway_rest_api.id
  stage_name  = "dev"
```

У кінцевому випадку, функція Lambda оброблятиме запити HTTP GET, що направляються через шлюз API. Далі файли стану Terraform зберігаються у бакеті S3, щоб підтримувати узгодженість і уможливлювати співпрацю.

Наступним кроком є створення сценарію bash, щоб динамічно встановлювати конфігурацію серверної частини для Terraform, дозволяючи різні імена файлів стану на основі змінних середовища.

```
# backend.sh
cat <<EOF > main.tf
provider "aws" {
}

terraform {
  backend "s3" {
    bucket = "dev-area"
    key    = "Terraform/lambda/${state_name}.tfstate"
  }
  required_providers {
    aws = {
      version = ">= 4.67.0"
      source  = "hashicorp/aws"
    }
  }
}
EOF

cat <<EOL > lambda.tf
resource "aws_lambda_function" "${state_name}_function" {
  function_name = var.lambda_name
  filename      = "lambda.zip"
  handler       = "exports.handler"
  runtime       = "nodejs18.x"
  role          = "arn:aws:iam::${account_id}:role/LambdaRole"
}
EOL
```

Сценарій bash (backend.sh) динамічно генерує файл конфігурації серверної частини Terraform (main.tf) і файл конфігурації функції Lambda (lambda.tf) на основі змінних середовища. Це дозволяє створювати гнучкі та багаторазові конфігурації Terraform.

Потім команди сценарію Terraform і bash включаються у пайплайн CI/CD, щоб автоматизувати процес розгортання.

```
name: CI/CD Pipeline

on:
  push:
    branches:
      - main

jobs:
  deploy:
    runs-on: ubuntu-latest
```

```
    steps:
      - name: Checkout code
        uses: actions/checkout@v2

      - name: Set up Terraform
        uses: hashicorp/setup-terraform@v1
        with:
          terraform_version: 1.0.0

      - name: Set AWS credentials
        uses: aws-actions/configure-aws-credentials@v1
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}

      - name: Execute bash script
        run: |
          export state_name=lambda
          bash backend.sh

      - name: Initialize Terraform
        run: terraform init -reconfigure

      - name: Plan Terraform
        run: terraform plan -out out/lambda -var profile='aws1' -var
lambda_name='lambda_function_name'

      - name: Apply Terraform
        run: terraform apply "out/lambda "
```

GitHub Actions автоматизує перевіряє код, налаштовує Terraform, конфігурує облікові дані AWS, виконує сценарій bash для створення конфігурацій Terraform і запускає команди Terraform для ініціалізації, планування та застосування змін інфраструктури.

Terraform Cloud використовується для віддаленого керування станом, співпраці та розширених функцій, таких як контроль версій і журнали аудиту.

```
# terraform.tf
terraform {
  required_version = ">= 0.13.0"
  backend "remote" {
    organization = "organization_name"
    workspaces {
      name = "example-workspace"
    }
  }
}
```

Надана конфігурація вказує, що Terraform має використовувати віддалену серверну частину, надану Terraform Cloud.

У підсумку запускається пайплайн CI/CD для виконання команд Terraform, надання необхідних ресурсів AWS і розгортання програми електронної комерції.

```
export AWS_PROFILE=aws_profile
export state_name=lambda
```

```
echo $state_name
bash backend.sh

terraform init -reconfigure
terraform plan -out out/lambda-var profile='aws1' -var
lambda_name='lambda_function_name'
terraform apply "out/lambda"
terraform destroy -var profile='aws_profile' -var
lambda_name='lambda_function_name'
```

Ці команди виконують етапи міграції, включаючи експорт необхідних змінних середовища, запуск сценарію bash і виконання команд Terraform для ініціалізації, планування та застосування змін інфраструктури.

Висновки. Концепція IaC уможливлює швидке відтворення будь-якого елемента хмарної інфраструктури, оскільки все фіксується в одному сценарії. Ця можливість може усунути ризики та страхи, пов'язані зі змінами у процесі міграції програмного забезпечення. IaC вимагає, щоб системи проектувалися з припущенням, що хмарна інфраструктура буде змінюватися з часом і адаптуватися до мінливих потреб. Навіть якщо сервери видаляються або масштабуються, програми мають продовжувати працювати. Корисна функція інструментів IaC для легкого створення, видалення, заміни, масштабування та переміщення ресурсів хмарної інфраструктури полегшує вдосконалення та адаптацію поточної хмарної інфраструктури, що сприяє стійкості наявних сервісів.

Загалом, описана практика інтеграції Terraform і AWS, включаючи контроль версій коду інфраструктури, дотримання принципів безпеки та використання параметризації та змінних для динамічних конфігурацій гарантує, що розгортання інфраструктури є безпечним, доступним для перевірки та масштабованим, що відповідає вимогам сучасних хмарних програм. Завдяки такій реалізації, організації можуть досягти більшої гнучкості, зменшити операційні витрати та покращити співпрацю між командами розробки та операцій. Ідеї та методології, викладені в статті, забезпечують надійну основу для модернізації управління інфраструктурою в хмарних середовищах, що дає можливість організаціям впроваджувати інновації та масштабуватися в епоху хмарних обчислень.

Список бібліографічних джерел

1. Kishiyama B., Wang H., Lopez D., Yang J. An Overview of Infrastructure as Code (IaC) with Performance and Availability Assessment on Google Cloud Platform. 2024.
2. Sabacinski J. Der Weg in die Cloud: Entwicklung einer Migrations-Methodologie für Desktop- und Server-Anwendungen hin zu einer Software-as-a-Service Anwendung. *Diplomarbeit Nr. 3680, Institut für Architektur von Anwendungssystemen, Universität Stuttgart*. 2015.
3. Chinthapatla S. Unleashing the Power of AWS: Revolutionizing Cloud Management through Infrastructure as Code (IaC). 2024.
4. Johnson A., et al. Optimizing AWS Serverless Deployments with Terraform: ACaseStudy. *Journal of Cloud Computing.* 2021. № 10(2),
5. Rahman A., Partho A., Morrison P., Williams L. What questions do programmers ask about configuration as code? Proceedings of the 4th International Workshop on Rapid Continuous Software Engineering. 2018. pp 16–22, DOI:10.1145/3194760.3194769.
6. Smith L., et al. Automating Serverless Infrastructure on AWS with Terraform. *IEEE Transactions on Cloud Computing.* 2021. № 9(4), P. 789-802.
7. Dalla Palma S., Di Nucci D., Palomba F., Tamburri D. A. Within- project defect prediction of Infrastructure-as-Code using product and process metrics. IEEE Transactions on Software Engineering. 2020. № 14(8). P. 1-8.
8. Brown M., et al. Scalable Serverless Deployments on AWS using Terraform. *ACMTransactions on Cloud Computing.* 2021. № 7(3). P. 123-136.
9. Özdoğan E., Ceran O., Üstündağ M. Systematic Analysis of Infrastructure as Code Technologies. Gazi University Journal of Science Part A: Engineering and Innovation. 2023. № 10. DOI:10.54287/gujsa.1373305.
10. Feitosa D., Penca M.-T., Berardi M., Boza R.-D., Andrikopoulos V. Mining for Cost Awareness in the Infrastructure as Code Artifacts of Cloud-Based Applications: An Exploratory Study. 2023. DOI:10.2139/ssrn.4436874.
11. Batu J. Implementing Infrastructure-as-Code with Cloud Disaster Recovery Strategies. International Journal of Computer Trends and Technology. 2024. № 72. P. 41-45. DOI:10.14445/22312803/IJCTT-V72I2P108.
12. Abualkishik A., Alwan A., Gulzar Y. DisasterRecovery in Cloud Computing Systems: An Overview. *InternationalJournal of Advanced Computer Science and Applications.* 2020. № 11. DOI: 10.14569/IJACSA.2020.0110984.
13. Eleraky, Mohammed & Anis Aziz, Wagdy & Soliman, John. (2023). Using Cloud Infrastructure as a Code IaC to

Set Up Web Applications. International Journal of Simulation: Systems, Science & Technology. 24.
14. Mukhopadhyay N., Tewari B. Efficient IaC-Based Resource Allocation for Virtualized Cloud Platforms. 2022. DOI: 10.1007/978-3-030-96040-7_16.
15. Omofoyewa Y., Grebe A., Leusmann P. IaC reusability for Hybrid Cloud Environment. 2021.

References

1. Kishiyama B., Wang H., Lopez D., & Yang J. (2024). An Overview of Infrastructure as Code (IaC) with Performance and Availability Assessment on Google Cloud Platform.
2. Sabacinski J. (2015). Der Weg in die Cloud: Entwicklung einer Migrations-Methodologie für Desktop- und Server-Anwendungen hin zu einer Software-as-a-Service Anwendung. Diplomarbeit Nr. 3680, Institut für Architektur von Anwendungssystemen, Universität Stuttgart.
3. Chinthapatla S. (2024). Unleashing the Power of AWS: Revolutionizing Cloud Management through Infrastructure as Code (IaC).
4. Johnson A., et al. (2021). Optimizing AWS Serverless Deployments with Terraform: A Case Study. Journal of Cloud Computing, 10(2).
5. Rahman A., Partho A., Morrison P., & Williams L. (2018). What questions do programmers ask about configuration as code? Proceedings of the 4th International Workshop on Rapid Continuous Software Engineering, 16–22. https://doi.org/10.1145/3194760.3194769
6. Smith L., et al. (2021). Automating Serverless Infrastructure on AWS with Terraform. IEEE Transactions on Cloud Computing, 9(4), 789-802.
7. Dalla Palma S., Di Nucci D., Palomba F., & Tamburri D. A. (2020). Within-project defect prediction of Infrastructure-as-Code using product and process metrics. IEEE Transactions on Software Engineering, 14(8), 1-8.
8. Brown M., et al. (2021). Scalable Serverless Deployments on AWS using Terraform. ACM Transactions on Cloud Computing, 7(3), 123-136.
9. Özdoğan E., Ceran O., & Üstündağ M. (2023). Systematic Analysis of Infrastructure as Code Technologies. Gazi University Journal of Science Part A: Engineering and Innovation, 10. https://doi.org/10.54287/gujsa.1373305
10. Feitosa D., Penca M.-T., Berardi M., Boza R.-D., & Andrikopoulos V. (2023). Mining for Cost Awareness in the Infrastructure as Code Artifacts of Cloud-Based Applications: An Exploratory Study. https://doi.org/10.2139/ssrn.4436874
11. Batu J. (2024). Implementing Infrastructure-as-Code with Cloud Disaster Recovery Strategies. International Journal of Computer Trends and Technology, 72, 41-45. https://doi.org/10.14445/22312803/IJCTT-V72I2P108
12. Abualkishik A., Alwan A., & Gulzar Y. (2020). Disaster Recovery in Cloud Computing Systems: An Overview. International Journal of Advanced Computer Science and Applications, 11. https://doi.org/10.14569/IJACSA.2020.0110984
13. Eleraky, M., Aziz, W. A., & Soliman, J. (2023). Using Cloud Infrastructure as a Code IaC to Set Up Web Applications. International Journal of Simulation: Systems, Science & Technology, 24.
14. Mukhopadhyay N., & Tewari B. (2022). Efficient IaC-Based Resource Allocation for Virtualized Cloud Platforms. https://doi.org/10.1007/978-3-030-96040-7_16
15. Omofoyewa Y., Grebe A., & Leusmann P. (2021). IaC reusability for Hybrid Cloud Environment.