УДК 629.3 **doi: 10.26906/SUNZ.2024.1.104**

О. О. Копцев, В. О. Мартовицький, Н. М. Бологова, І. Б. Федак

Харківський національний університет радіоелектроніки, Харків, Україна

ОСОБЛИВОСТІ АВТОМАТИЧНОГО РОЗГОРТАННЯ ІНФРАСТРУКТУРИ ЯК КОДУ ДЛЯ ХМАРНИХ СЕРВІСІВ

Анотація. Хмарні сервіси надають сучасні обчислювальні ресурси, доступні на вимогу через Інтернет. Завдяки хмарним обчисленням команди стають більш ефективними та скорочують час виходу на ринок, оскільки вони можуть швидко набувати та масштабувати послуги без значних зусиль, які потребує управління традиційною інфраструктурою. Автоматизація дозволяє командам покращувати ключові показники. Команди відмовляються від тривалих процесів, пов'язаних із внесенням змін та запланованими розгортаннями. Вони також переходять від реактивного виявлення проблем до запобіжного моніторингу та забезпечення прозорості. Мета статті – дослідити популярні засоби для реалізації інфраструктури як коду, що включають Terraform, AWS CloudFormation, ARM Templates, Ansible, Puppet, Chef та інші. Ці інструменти допомагають створювати, керувати та відстежувати інфраструктурні ресурси через програмний код. Використання автоматизованих практик IaC дозволить зберегти час, зменшити ризики, підвищити сумісність та спростити процеси розгортання та управління інфраструктурою. Розглянувши популярні засоби для реалізації інфраструктури як коду, що допомагають створювати, керувати та відстежувати інфраструктурні ресурси через програмний код, ми дійшли висновку, що Bicep дозволяє більш ефективно та зрозуміло працювати з розгортанням інфраструктури в Azure, а також полегшує роботу з ARM Templates. Використання Bicep, у порівнянні з ARM шаблонами та іншими інструментами IaC, дає можливість створювати скрипти, які є значно компактнішими за розміром. Це досягається завдяки більш лаконічному та зрозумілому синтаксису Bicep, що дозволяє описувати однакові набори ресурсів меншою кількістю коду. Такий підхід не тільки спрощує розробку та підтримку інфраструктурного коду, але й знижує поріг входження для нових користувачів, які мають досвід роботи з програмуванням.

Ключові слова: хмарні сервіси, інфраструктура, масштабування, розгортання.

Вступ

Постановка проблеми. Хмарні сервіси (Cloud Services) - це послуги, які надаються через хмарну інфраструктуру, доступні через Інтернет [1]. Вони охоплюють широкий спектр обчислювальних та інших інформаційних послуг, які можуть бути використані організаціями та фізичними особами. Основною ідеєю хмарних сервісів є надання користувачам можливості використовувати обчислювальні ресурси, сховища даних та інші інфраструктурні ресурси за певну плату, без необхідності забезпечення та підтримки власного обладнання та програмного забезпечення.

Основні характеристики хмарних сервісів включають еластичність та масштабованість, що дозволяє користувачеві збільшити або зменшити обсяги використовуваних ресурсів залежно від потреб. Це забезпечує оптимальні обчислювальні потужності без зайвих витрат. Самообслуговування також є однією з основних характеристик хмарних сервісів. Користувачі можуть замовити ресурси та конфігурувати їх за допомогою веб-інтерфейсів або API, без потреби у взаємодії зі спеціалістами технічної підтримки. До характеристик хмарних сервісів також відносяться платіж за використання, доступність та відмовостійкість та широкий спектр послуг. Користувачі сплачують лише за той обсяг ресурсів, які вони зажадають. Це дозволяє зменшити витрати, втратити ви не платите за неактивний ресурс. Багато хмарних платформ забезпечують високу доступність за допомогою даних розташування та ресурсів на різних фізичних місцях. Це уникнути відмов у роботі у випадку збоїв обладнання. Хмарні сервіси можуть включати в себе обчислення, зберігання даних, аналітику, штучний інтелект, Інтернет-речі (IoT), розробки платформ та багато іншого.

Аналіз останніх досліджень і публікацій. Деякі провідні постачальники хмарних сервісів включають Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP), IBM Cloud тощо. Ці послуги призначені для розподілених центрів обробки даних для забезпечення доступності та швидкості обробки. Моделі обслуговування хмарних сервісів забезпечують різні рівні гнучкості, контролю та відповідальності, що дозволяє користувачам вибирати найбільш підходящий варіант у залежності від їх потреб та навичок. Найбільш популярні [2]:

1. Інфраструктура як сервіс (IaaS - Infrastructure as a Service). У цій моделі користувачам надаються віртуальні обчислювальні ресурси, такі як віртуальні машини, мережеві ресурси та сховища даних. Користувач самостійно керує операційною системою, забезпеченням безпеки та розгортанням програмного забезпечення. Приклади: Amazon EC2, віртуальні машини Microsoft Azure.

2. Платформа як сервіс (PaaS - Platform as a Service). У цій моделі користувачам надається платформа для розробки, тестування та розгортання власних програмних додатків. Вони можуть працювати з програмами без необхідності в управлінській інфраструктурі. Платформа надає інструменти для розробки та керування додатками. Приклади: Heroku, Google App Engine.

3. Програмне забезпечення як сервіс (SaaS - Software as a Service). У цій моделі користувачам надається готове програмне забезпечення, яке вони можуть використовувати через Інтернет. Вони не контролюють над інфраструктурою або платформою, але підтримують можливість використання програми з

будь-яким пристроєм з доступом до Інтернету. Приклади: Google Workspace, Microsoft Office 365.

4. Функції як сервіс (FaaS - Functions as a Service) - це модель, у якій користувачам надаються можливості для розгортання окремих функцій або скриптів без необхідності керування обчислювальною інфраструктурою. Користувачі платять за фактичний час виконання функцій. Приклади: AWS Lambda, функції Azure.

5. Інфраструктура як код (Infrastructure as Code, IaC) - це модель автоматизованого управління та розгортання інфраструктурних ресурсів за допомогою програмного коду. Замість традиційного ручного налаштування та управління інфраструктурою, при використанні коду IaC для опису ресурсів, які повинні бути створені, налаштовані та керовані. Розгортання інфраструктури - це процес створення та налаштування деяких обчислювальних ресурсів, мережевих компонентів, сховищ даних та інших елементів інфраструктури для підтримки роботи програмних додатків. Цей процес може бути виконаний вручну, але часто він автоматизується за допомогою практики інфраструктури як коду (IaC) та відповідних інструментів [3]. Основні концепції та переваги IaC:

Автоматизація. IaC дозволяє автоматизувати процеси створення, налаштування та управління інфраструктурою. Це втратити ризики помилок та спростити процеси впровадження.

Версіонування. Код інфраструктури може бути збережений у системі контролю версій, що дозволяє відстежувати зміни, порівнювати версії та відновлювати попередні стани.

Гнучкість та швидкість. За допомогою IaC можна легко та безпечно розгортати нові середовища та змінювати конфігурацію.

Документація. Код інфраструктури служить як документація для самої інфраструктури. Він описує всі налаштування та компоненти системи.

Впровадження кращих практик. IaC дозволяє впроваджувати стандартизовані підходи та кращі практики до конфігурації та управління інфраструктурою.

Повторюваність. Код інфраструктури може бути легко переносимим між світовими середовищами, що дозволяє забезпечити узгодженість середовищ у різних етапах розробки та тестування.

Мета статті – дослідити популярні засоби для реалізації інфраструктури як коду, що включають Terraform, AWS CloudFormation, ARM Templates, Ansible, Puppet, Chef та інші. Ці інструменти допомагають створювати, керувати та відстежувати інфраструктурні ресурси через програмний код.

Виклад основного матеріалу

До основних кроків розгортання інфраструктури належать:

1. Визначення вимог. Спочатку потрібно чітко виконати вимоги до інфраструктури. Це може включати обчислювальні ресурси, сховища даних, мережеві налаштування, безпеку тощо.

2. Вибір інструментів. Треба вибрати підходящі інструменти та засоби для реалізації розгортання. Якщо використовується практика Інфраструктури як коду, обирають відповідний інструмент, наприклад, Terraform, CloudFormation, Ansible тощо.

3. Створення коду інфраструктури. Треба написати код, який описує всі необхідні ресурси та налаштування для інфраструктури. Цей код може бути структурованим за допомогою файлів або скриптів, залежно від обраного інструменту.

4. Тестування. Перед розгортанням конкуренції необхідно провести тестування коду інфраструктури на відповідність вимогам та наявним стандартам.

5. Розгортання. Процес розгортання, під час якого інструмент автоматично створює та налаштовує необхідні ресурси відповідно до вашого коду.

6. Конфігурація та налаштування. Після розгортання можна налаштувати, а також додатково налаштувати ресурси, які вже створені.

7. Моніторинг та управління. Після розгортання слід встановити моніторинг для відстеження працездатності інфраструктури, а також забезпечити систему управління, щоб реагувати на зміни та події.

Важливо пам'ятати, що розгортання інфраструктури - це динамічний процес, який може змінюватися з часом під час зміни вимог до програмного забезпечення. Традиційне розгортання, означає налаштування та встановлення інфраструктурних ресурсів, таких як сервери, мережеві компоненти, бази даних тощо, вручну або за допомогою засобів, які не є автоматизованими. Однак цей підхід стає менш популярним при застосуванні автоматизованого розгортання за допомогою практичної реалізації інфраструктури як коду (IaC). Основні етапи традиційного розгортання інфраструктури полягають у наступному:

Перший етап полягає у плануванні. Це визначення вимог до інфраструктури, вибір обладнання, обчислювальних ресурсів, мережевої архітектури тощо. Другий етап — це придбання фізичного обладнання або оренда серверів у дата-центрі. Наступний - фізична інсталяція. Розміщення серверів, налаштування мережі, підключення до джерел електроживлення, охолодження тощо. Четвертий етап полягає у встановленні операційної системи на кожному сервері, налаштування мережі, безпеки та ін. Наступний етап - встановлення та конфігурація сховищ даних (наприклад, база даних), які використовуються додатками. Далі - налаштування заходів безпеки, включаючи файрволі, антивіруси, моніторинг та інші заходи. Сьомий етап полягає у розгортанні програмного забезпечення: Встановлення та конфігурація програмного забезпечення, яке буде працювати на інфраструктурі. Восьмий етап — це тестування, виконання тестів для перевірки працездатності та стабільності системи. І останній етап — це підтримка та управління. Додаткове управління та підтримка інфраструктури, включаючи регулярне оновлення, моніторинг та відповідь на проблеми.

Цей традиційний підхід до розгортання інфраструктури може бути працездатним, але він може бути гнучким, вимагати більше часу на ручне налаштування та зміну інфраструктури. У сучасному світі більше підприємств переходять до автоматизо-

ваних практик IaC для забезпечення більшої ефективності та узгодженості в розгортанні та керуванні інфраструктурою. Автоматизовані практики інфраструктури як коду (IaC) - це підхід до розгортання та управління інфраструктурою, коли всі дії, пов'язані зі створенням, налаштуваннями та управлінням ресурсами, забезпечуються за допомогою програмного коду. Основною метою є автоматизація процесів, забезпечення сумісності, зменшення ризиків помилок та забезпечення гнучкості в управлінській інфраструктурі. Ключові практики IaC включають:

1. Декларативний код. Код IaC описує бажану структуру та налаштування інфраструктури, а не послідовність дій для його створення. Це дозволяє системі самостійно розпізнавати, які зміни потрібно зробити, для досягнення бажаного стану.

2. Керованість версій. Код IaC може бути збережений та відстежуваний у системі керування версіями, що дозволяє відновлювати попередні стани та вести спільну роботу в команді.

3. Масштабованість. За допомогою IaC можна легко розгортати та масштабувати ресурси відповідно до потреб.

4. Повторюваність. Код IaC може бути застосований для розгортання інфраструктури на різних етапах розробки та в різних середовищах (наприклад, розробка, тестування, продакшн).

5. Автоматизована та неперервна доставка. IaC допоможе забезпечити автоматичне розгортання та оновлення інфраструктури, що підтримує практику неперервної доставки.

6. Тести та перевірки. Код IaC може бути підданим автоматизованим тестуванням, щоб виявити помилки до розгортання.

Використання автоматизованих практик IaC дозволить зберегти час, зменшити ризики, підвищити сумісність та спростити процеси розгортання та управління інфраструктурою.

Доменно-орієнтована мова (Domain-Specific Language, DSL) при розгортанні Інфраструктури як коду (IaC) - це спеціалізована мова програмування або конфігурації, яка розроблена для виразного опису інфраструктури та її розгортання. Основна ідея перетворюється в тому, щоб мову, спеціально налаштовану для виразу концепцій і понять, що відповідає конкретному домену (у цьому випадку - розгортанню інфраструктури), замість використання загальної мови програмування. Відмінність DSL від загальних мов програмування, таких як Python, JavaScript чи Ruby, виникає в тому, що DSL спрощує виразність, зрозумілість та специфічність для конкретної області, в цьому випадку - управління інфраструктурою. Це робить код більш читабельним і зрозумілим для тих, хто працює в даній області.

Є кілька прикладів DSL для розгортання Інфраструктури як коду. Декларативний інструмент Terraform для створення, налаштування та управління інфраструктурою різних провайдерів (AWS, Azure, Google Cloud тощо) - це приклад DSL. Файли конфігурації Terraform містять декларативний код, який описує ресурси, їх взаємозв'язки та налаштування. Аналогічно, шаблони CloudFormation для AWS також є формою DSL, спеціалізованою на описі ресурсів та послуг Amazon Web Services.

Pulumi - це інша платформа, яка дозволяє писати IaC за допомогою загальних мов програмування, але з концепціями, призначеними для розгортання та управління інфраструктурою. Навіть Ansible, хоча і використовує мову Python, має спеціалізовану конструкцію для конфігурації інфраструктури та автоматизованого розгортання.

У Microsoft Azure використання доменно-орієнтованої мови (DSL) дозволяє описати і керувати різними аспектами інфраструктури та ресурсів, які розгортаються в хмарному середовищі Azure. Основним інструментом для використання DSL в Azure є мова ARM (Azure Resource Manager).

Шаблони Azure Resource Manager (ARM Templates) - це JSON-подібна мова, яка дозволяє створювати ARM-шаблони - описи файлів, які використовують ресурси, їх взаємозв'язки та налаштування. Це DSL для розгортання та управління ресурсами Azure. У шаблонах ARM ви можете описати різні ресурси - від віртуальних машин та мережевих складових до баз даних та інших послуг (лістинг 1).

```
{"$schema":
"https://schema.management.azure.com/schemas/20
19-04-01/deploymentTemplate.json#",
  "resources": [
    {
      "type": "Microsoft.Resources/resourceGroups",
      "apiVersion": "2020-10-01",
      "location": "eastus",
      "name": "myResourceGroup"
    },
    {
      "type": "Microsoft.Compute/virtualMachines",
      "apiVersion": "2019-07-01",
      "name": "myVM",
      "location": "[resourceGroup().location]",
      "properties": {
        "hardwareProfile": {
          "vmSize": "Standard_DS1_v2"
        },
        "storageProfile": {
          "imageReference": {
            "publisher": "MicrosoftWindowsServer",
            "offer": "WindowsServer",
            "sku": "2016-Datacenter",
            "version": "latest"}
        }
      }
    },
    {
      "type": "Microsoft.Storage/storageAccounts",
      "apiVersion": "2019-06-01",
      "name": "mystorageaccount",
      "location": "[resourceGroup().location]"
```

Лістинг 1

Azure CLI - це командний рядок Azure, який також має підтримку для використання DSL. Можна використовувати команди CLI для створення, налаштування та керування ресурсами за допомогою команд спеціалізованої мови (лістинг 2). Terraform можна використовувати з підтримкою Azure. Він також надає DSL для опису ресурсів та їх взаємозв'язків, незалежно від платформи хмарних послуг (лістинг 3).

```
az group create --name myResourceGroup --location
      eastus
az vm create \
  --resource-group myResourceGroup \
  --name myVM \
  --image Win2016Datacenter \
  --admin-username azureuser \
  --admin-password myPassword12
az storage account create \
  --name mystorageaccount \
  --resource-group myResourceGroup \
  --location eastus \
```

Лістинг 2

```
resource "azurerm_resource_group" "my_rg" {
  name     = "myResourceGroup"
  location = "eastus"
}
resource "azurerm_virtual_machine" "my_vm" {
  name                = "myVM"
  location            =
      azurerm_resource_group.my_rg.location
  resource_group_name   =
      azurerm_resource_group.my_rg.name
  network_interface_ids =
      [azurerm_network_interface.my_nic.id]
  vm_size             = "Standard_DS1_v2"
  os_profile {
    computer_name  = "myvm"
    admin_username = "azureuser"
    admin_password = "myPassword12"
  }
}
resource "azurerm_storage_account" "my_sa" {
  name                     = "mystorageaccount"
  resource_group_name      =
      azurerm_resource_group.my_rg.name
  account_tier             = "Standard"
  account_replication_type = "LRS"
}
```

Лістинг 3

Pulumi - це інструмент для інфраструктури як коду, який дозволяє використовувати звичайні мови програмування (наприклад, Python, JavaScript, Go) для опису інфраструктури. Він також підтримує Azure, що дозволяє використовувати DSL з мовами програмування для роботи з ресурсами Azure. У прикладі ми використовуємо мову програмування Python (лістинг 4).

```
resource_group =
      core.ResourceGroup('myResourceGroup')
vm = compute.WindowsVirtualMachine('myVM',
    resource_group_name=resource_group.name,
    admin_username='azureuser',
    admin_password='myPassword12',
    size='Standard_DS1_v2'
    storageProfile: {
      imageReference: {
        publisher: 'MicrosoftWindowsServer'
        offer: 'WindowsServer'
        sku: '2016-Datacenter'
        version: 'latest'
      }
)
storage_account =
      storage.Account('mystorageaccount',
    resource_group_name=resource_group.name,
    account_tier='Standard',
    account_replication_type='LRS',
)
```

Лістинг 4

Вісер - це мова декларативного опису інфраструктури, розроблена спеціально для Microsoft Azure. У файлі з розширенням .bicep визначається інфраструктура, яку можна розгорнути в Azure, а потім використовується цей файл протягом усього життєвого циклу розробки для повторного розгортання інфраструктури.

Основна ідея Вісер полягає в тому, щоб зробити опис інфраструктури більш зрозумілим, менш складним і більш структурованим. Вісер дозволяє більш ефективно та зрозуміло працювати з розгортанням інфраструктури в Azure, а також полегшує роботу з ARM Templates (лістинг 5).

```
resource myResourceGroup
   'Microsoft.Resources/resourceGroups@2020-10-01'
   = {
  name: 'myResourceGroup'
  location: 'eastus'
}
resource myVM
   'Microsoft.Compute/virtualMachines@2020-06-01' = {
  name: 'myVM'
  location: myResourceGroup.location
  properties: {
    hardwareProfile: {
      vmSize: 'Standard_DS1_v2'
    }
  }
}
resource myStorageAccount
   'Microsoft.Storage/storageAccounts@2020-08-01-
   preview' = {
  name: 'mystorageaccount'
  location: myResourceGroup.location
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'StorageV2'
}
```

Лістинг 5

ARM (Azure Resource Manager) Templates і Вісер - це засоби для декларативного опису та розгортання інфраструктури в Microsoft Azure.

Порівняння двох цих засобів за критеріями: мова, зрозумілість, спостережуваність та модульність наведено у табл. 1.

Вісер дозволяє більш ефективно та зрозуміло працювати з розгортанням інфраструктури в Azure, а також полегшує роботу з ARM Templates. Однак, обираючи між ними, слід брати до уваги власні знання та потреби в конкретному проекті.

Висновки

Використання автоматизованих практик IaC дозволить зберегти час, зменшити ризики, підвищити сумісність та спростити процеси розгортання та управління інфраструктурою. Розглянувши популярні засоби для реалізації інфраструктури як коду, що допомагають створювати, керувати та відстежувати інфраструктурні ресурси через програмний код, ми дійшли висновку, що Вісер дозволяє більш ефективно та зрозуміло працювати з розгортанням інфраструктури в Azure, а також полегшує роботу з ARM Templates.

Таблиця 1 – **Порівняння засобів для декларативного опису та розгортання інфраструктури в Microsoft Azure**

Критерії порівняння до засобів	ARM Templates	Bicep
Мова	ARM Templates вибір JSON для опису ресурсів та їх налаштування. JSON є достатнім стандартним форматом, але великі шаблони можуть бути складними для читання та редагування.	Bicep використовує більш декларативний та структурований синтаксис, що надає властивості мови програмування. Це полегшує розуміння та редагування.
Зрозумілість	Доволі часто шаблони ARM можуть стати заплутаними через велику кількість JSON-коду та потребують вказати досить багато деталей.	Bicep робить код більш зрозумілим за допомогою свого синтаксису. Це особливо важливо при роботі з великими та складними шаблонами.
Спостережуваність	Якщо ARM Templates - це JSON, можна налагоджено відстежити зміни та розрізнити, які ресурси змінені чи додані.	Так як Bicep компілюється з шаблонами ARM, зміни стають більш видимими, щоб ви могли переглядати сирцевий Bicep-код, який згенерував певний шаблон ARM.
Модульність	Підтримка модульності та перевикористання коду не завжди є зручною, особливо для складних шаблонів.	Bicep має кращу підтримку модульності та перевикористання коду. Можна створювати власні модулі та використовувати їх для спрощення шаблонів.

Використання Bicep, у порівнянні з ARM шаблонами та іншими інструментами IaC, дає можливість створювати скрипти, які є значно компактнішими за розміром. Це досягається завдяки більш лаконічному та зрозумілому синтаксису Bicep, що дозволяє описувати однакові набори ресурсів меншою кількістю коду. Такий підхід не тільки спрощує розробку та підтримку інфраструктурного коду, але й знижує поріг входження для нових користувачів, які мають досвід роботи з програмуванням. Коротший і чистіший код в Bicep сприяє кращому розумінню та легшому відстеженню змін у процесах розгортання та управління інфраструктурою, роблячи Bicep більш ефективним інструментом у сфері IaC.

СПИСОК ЛІТЕРАТУРИ

1. Red Hat. What are cloud services? URL: https://www.redhat.com/en/topics/cloud-computing/what-are-cloud-services
2. Вікіпедія. Моделі обслуговування та існуючі рішення URL: https://uk.wikipedia.org/
3. What is Infrastructure as Code? [Електронний ресурс] // Mike Jacobs, Ed Kaim. – 2021. – Режим доступу: https://docs.microsoft.com/enus/devops/deliver/what-is-infrastructure-as-code
4. Guerriero M., Michele G. Adoption, support, and challenges of infrastructure-as-code: Insights from industry. In: 2019 IEEE international conference on software maintenance and evolution (ICSME). IEEE, 2019. p. 580-589.
5. Rahman O., Akond J., Rezvan A. A systematic mapping study of infrastructure as code research. Information and Software Technology, 2019, 108: p. 65-77.
6. Microsoft. Comparing JSON and Bicep for templates. URL: https://learn.microsoft.com/en-us/azure/azure-resource-manager/bicep/compare-template-syntax
7. Rahman O., Akond J. Gang of eight: A defect taxonomy for infrastructure as code scripts. In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. 2020. p. 752-764.
8. Riti K. Pierluigi A. Infrastructure as Code. Beginning HCL Programming: Using Hashicorp Language for Automation and Configuration, 2021, p. 65-78.
9. Microsoft. What is Bicep? URL: https://learn.microsoft.com/en-us/azure/azure-resource-manager/bicep/overview?tabs=bicep
10. Bicep vs ARM Template URL: https://dev.to/evdbogaard/bicep-vs-arm-templates-bf9

Received (Надійшла) 22.11.2023
Accepted for publication (Прийнята до друку) 06.01.2024

Features of automatic deployment of infrastructure as code for cloud services

Oleg Koptsev, Vitalii Martovytskyi, Nataliia Bolohova, Ilko Fedak

Abstract. Cloud services provide modern computing resources available on demand over the Internet.Thanks to cloud computing, teams become more efficient and reduce time to market, as they can quickly acquire and scale services without significant efforts required for managing traditional infrastructure. Automation enables teams to improve key metrics. Teams get rid of lengthy processes associated with making changes and scheduled deployments. They are also moving from reactive problem detection to proactive monitoring and transparency. The goal of this article is to explore popular tools for implementing infrastructure as code (IaC), including Terraform, AWS CloudFormation, ARM Templates, Ansible, Puppet, Chef, and others. These tools help create, manage and monitor infrastructure resources through software code. Using automated IaC practices will save time, reduce risk, improve interoperability, and simplify infrastructure deployment and management processes. After looking at popular infrastructure-as-code tools that help create, manage, and monitor infrastructure resources through code, we came to the conclusion that Bicep allows you to work more efficiently and clearly with infrastructure deployment in Azure, and also makes it easier to work with ARM Templates. Using Bicep, compared to ARM templates and other IaC tools, makes it possible to create scripts that are much more compact in size. This is achieved thanks to the more concise and understandable syntax of Bicep, which allows describing the same sets of resources with less code. This approach not only simplifies the development and maintenance of infrastructure code, but also lowers the barrier to entry for new users with programming experience.

Keywords: Cloud services, infrastructure, scaling, deployment.