Tworzenie aplikacji serverless w Azure Functions z wykorzystaniem Pythona v2 i Terraform

mgr inż. Jakub Woźniak
Zakład Systemów Informatycznych
Instytut Informatyki Politechniki Poznańskiej

Abstract

Niniejszy dokument stanowi przewodnik laboratoryjny przez proces tworzenia kompleksowej aplikacji serverless w chmurze Microsoft Azure. Wykorzystujemy najnowszy model programistyczny Python v2, infrastrukturę jako kod (Terraform) oraz narzędzia developerskie Azure Functions Core Tools. Laboratorium obejmuje integrację z usługami Azure Storage, Cosmos DB, Storage Queue oraz Cognitive Services, z uwzględnieniem lokalnego testowania i praktycznych wzorców implementacyjnych.

1 Wprowadzenie do architektury serverless

Architektura serverless oferuje model wykonania gdzie dostawca chmury dynamicznie zarządza alokacją zasobów, rozliczając jedynie faktyczny czas wykonania kodu. Azure Functions to główna implementacja tego paradygmatu w ekosystemie Microsoft Azure.

1.1 Korzyści z podejścia serverless

- Zero administracji infrastrukturą
- Automatyczne skalowanie
- Model płatności za użycie
- Szybsze wdrożenia

Usługa	**Azure**	**AWS**	**Google Cloud**
Compute	Functions	Lambda	Cloud Functions
Storage	Blob Storage	S3	Cloud Storage
Baza danych	Cosmos DB	DynamoDB	Firestore
Kolejki	Storage Queue	SQS	Pub/Sub
AI	Cognitive Services	Rekognition	Vision AI

Table 1: Porównanie usług serverless między głównymi dostawcami chmury

2 Przygotowanie środowiska

2.1 Wymagania wstępne

- Azure CLI 2.45+
- Terraform 1.5+
- Azure Functions Core Tools 4.0+
- Python 3.10+

2.2 Konfiguracja narzędzi

```
# Instalacja Azure Functions Core Tools
npm install -g azure-functions-core-tools@4 --unsafe-perm true

# Logowanie do Azure CLI
az login

# Inicjalizacja Terraform
terraform init
```

3 Implementacja funkcji Hello World

3.1 Tworzenie projektu funkcji

```
func init lab-functions --python
cd lab-functions
func new --name hello-http --template "HTTP trigger" --authlevel anonymous
```

3.2 Implementacja logiki w Python v2

```
import azure.functions as func

app = func.FunctionApp()

@app.route(route="hello-http", auth_level=func.AuthLevel.ANONYMOUS)
def hello_http(req: func.HttpRequest) -> func.HttpResponse:
    name = req.params.get('name', 'Azure')
    return func.HttpResponse(f"Hello {name}!")
```

3.3 Lokalne testowanie

```
func start
curl http://localhost:7071/api/hello-http?name=Student
```

4 Integracja z Azure Storage

4.1 Definicja zasobów w Terraform

```
resource "azurerm_storage_account" "lab_storage" {
  name                     = "labstorage${var.environment}"
  resource_group_name      = azurerm_resource_group.lab.name
  location                 = "westeurope"
  account_tier             = "Standard"
  account_replication_type = "LRS"
}

resource "azurerm_storage_container" "uploads" {
  name                  = "uploads"
  storage_account_name  = azurerm_storage_account.lab_storage.name
  container_access_type = "blob"
}
```

4.2 Implementacja uploadu plików do Azure Blob Storage

```
import os
from azure.storage.blob import BlobServiceClient
from azure.identity import DefaultAzureCredential
import azure.functions as func

app = func.FunctionApp()

```

```python
@app.route(route="upload", auth_level=func.AuthLevel.FUNCTION)
def upload_file(req: func.HttpRequest) -> func.HttpResponse:
    # Autentykacja przez Managed Identity
    credential = DefaultAzureCredential()
    blob_service_client = BlobServiceClient(
        account_url=f"https://{os.environ['STORAGE_ACCOUNT']}.blob.core.windows.net",
        credential=credential
    )

    file = req.files.get('file')
    if not file:
        return func.HttpResponse("No file uploaded", status_code=400)

    blob_client = blob_service_client.get_blob_client(
        container="uploads",
        blob=file.filename
    )

    # Upload z automatycznym chunkingiem dla żduych plików
    blob_client.upload_blob(file.stream, overwrite=True)

    return func.HttpResponse(f"File {file.filename} uploaded successfully")
```

4.3 Funkcja obsługująca triggery Blob Storage

```python
@app.blob_trigger(arg_name="blob", path="uploads/{name}",
                  connection="AzureWebJobsStorage")
def process_upload(blob: func.InputStream):
    logging.info(f"Przetwarzanie pliku: {blob.name}")
    # Logika przetwarzania pliku...
    file_content = blob.read()
    # łPrzykadowa logika przetwarzania...
    logging.info(f"Rozmiar pliku: {len(file_content)} bajtów")
```

5 Integracja z Cosmos DB

5.1 Konfiguracja Terraform

```
resource "azurerm_cosmosdb_account" "lab_db" {
  name                = "lab-cosmos-${var.environment}"
  location            = azurerm_resource_group.lab.location
  resource_group_name = azurerm_resource_group.lab.name
  offer_type          = "Standard"
  kind                = "GlobalDocumentDB"

  consistency_policy {
    consistency_level = "Session"
  }

  geo_location {
    location          = "westeurope"
    failover_priority = 0
  }
}

resource "azurerm_cosmosdb_sql_database" "lab_database" {
  name                = "lab-database"
  resource_group_name = azurerm_resource_group.lab.name
  account_name        = azurerm_cosmosdb_account.lab_db.name
}

resource "azurerm_cosmosdb_sql_container" "items" {
  name                = "items"
  resource_group_name = azurerm_resource_group.lab.name
  account_name        = azurerm_cosmosdb_account.lab_db.name
  database_name       = azurerm_cosmosdb_sql_database.lab_database.name
  partition_key_path  = "/id"
}
```

5.2 Implementacja operacji CRUD

```python
from azure.cosmos import CosmosClient, PartitionKey
import json

@app.route(route="items", auth_level=func.AuthLevel.FUNCTION, methods=["GET", "POST"])
def manage_items(req: func.HttpRequest) -> func.HttpResponse:
    # Konfiguracja łączenia z Cosmos DB
    endpoint = os.environ["COSMOS_ENDPOINT"]
    key = os.environ["COSMOS_KEY"]
    client = CosmosClient(endpoint, key)
    database = client.get_database_client("lab-database")
    container = database.get_container_client("items")

    # Obsługa GET - pobieranie wszystkich elementów
    if req.method == "GET":
        items = list(container.query_items(
            query="SELECT * FROM c",
            enable_cross_partition_query=True
        ))
        return func.HttpResponse(
            json.dumps(items),
            mimetype="application/json"
        )

    # Obsługa POST - dodawanie nowego elementu
    elif req.method == "POST":
        try:
            item = req.get_json()
            if not item.get('id'):
                item['id'] = str(uuid.uuid4())

            created_item = container.create_item(body=item)
            return func.HttpResponse(
                json.dumps(created_item),
                mimetype="application/json",
                status_code=201
            )
        except Exception as e:
            return func.HttpResponse(
                f"Error: {str(e)}",
                status_code=400
            )
```

5.3 Implementacja triggera Cosmos DB

```python
@app.cosmos_db_trigger(arg_name="documents",
                       database_name="lab-database",
                       collection_name="items",
                       connection="CosmosConnection",
                       create_lease_collection_if_not_exists=True)
def cosmos_trigger(documents: func.DocumentList):
    for doc in documents:
        doc_dict = json.loads(doc.to_json())
        logging.info(f"Przetworzono dokument: {doc_dict.get('id')}")
        # Tutaj logika przetwarzania zmienionych dokumentów
```

6 Obsługa kolejek Storage Queue

6.1 Definicja kolejki w Terraform

```
resource "azurerm_storage_queue" "processing_queue" {
  name                 = "processing-queue"
  storage_account_name = azurerm_storage_account.lab_storage.name
}
```

6.2 Wysyłanie wiadomości do kolejki

```python
from azure.storage.queue import QueueClient
from azure.identity import DefaultAzureCredential

@app.route(route="enqueue", auth_level=func.AuthLevel.FUNCTION)
def send_to_queue(req: func.HttpRequest) -> func.HttpResponse:
    message = req.params.get('message')
    if not message:
        try:
            req_body = req.get_json()
            message = req_body.get('message')
        except ValueError:
            message = None

    if not message:
        return func.HttpResponse(
            "Please provide a message to send to the queue",
            status_code=400
        )

    credential = DefaultAzureCredential()
    queue_client = QueueClient(
        account_url=f"https://{os.environ['STORAGE_ACCOUNT']}.queue.core.windows.net",
        queue_name="processing-queue",
        credential=credential
    )

    queue_client.send_message(message)

    return func.HttpResponse(f"Message sent to queue: {message}")
```

6.3 Implementacja triggera kolejki

```python
@app.queue_trigger(arg_name="msg",
                   queue_name="processing-queue",
                   connection="AzureWebJobsStorage")
def process_message(msg: func.QueueMessage):
    message_text = msg.get_body().decode('utf-8')
    logging.info(f"Przetwarzanie świadomoci: {message_text}")

    # łPrzykadowa logika przetwarzania świadomoci
    # np. zapis do Cosmos DB, analiza śtreci, etc.
    try:
        # Symulacja przetwarzania
        logging.info("ęRozpoczcie przetwarzania...")
        time.sleep(1)  # Symulacja ążobcienia
        logging.info("ńZakoczono przetwarzanie")
    except Exception as e:
        logging.error(f"łąBd przetwarzania: {str(e)}")
        # W przypadku łębdów żmoemy ponownie śćumieci śćwiadomo w kolejce
        # lub ćzapisa ąj do kolejki łębdów
```

7 Integracja z Cognitive Services

7.1 Dodawanie usługi w Terraform

```hcl
resource "azurerm_cognitive_account" "vision" {
  name                = "lab-vision-${var.environment}"
  location            = azurerm_resource_group.lab.location
  resource_group_name = azurerm_resource_group.lab.name
  kind                = "ComputerVision"
  sku_name            = "S1"
}
```

7.2 Analiza obrazów w Python

```python
from azure.cognitiveservices.vision.computervision import ComputerVisionClient
from azure.cognitiveservices.vision.computervision.models import VisualFeatureTypes
from msrest.authentication import CognitiveServicesCredentials
import json

@app.route(route="analyze-image", auth_level=func.AuthLevel.FUNCTION)
def analyze_image(req: func.HttpRequest) -> func.HttpResponse:
    # Pobieranie obrazu z żądania
    image_file = req.files.get('image')
    if not image_file:
        return func.HttpResponse(
            "Please upload an image file",
            status_code=400
        )

    # Inicjalizacja klienta Computer Vision
    vision_client = ComputerVisionClient(
        endpoint=os.environ["VISION_ENDPOINT"],
        credentials=CognitiveServicesCredentials(os.environ["VISION_KEY"])
    )

    # Analiza obrazu
    features = [
        VisualFeatureTypes.description,
        VisualFeatureTypes.tags,
        VisualFeatureTypes.categories,
        VisualFeatureTypes.objects,
        VisualFeatureTypes.adult
    ]

    image_stream = image_file.stream.read()
    results = vision_client.analyze_image_in_stream(
        image=image_stream,
        visual_features=features
    )

    # Konwersja wyników do JSON
    analysis_result = {
        "description": results.description.captions[0].text if results.description.captions else "",
        "confidence": results.description.captions[0].confidence if results.description.captions else 0,
        "tags": [{'name': tag.name, 'confidence': tag.confidence} for tag in results.tags],
        "is_adult_content": results.adult.is_adult_content,
        "adult_score": results.adult.adult_score,
        "is_racy_content": results.adult.is_racy_content,
        "racy_score": results.adult.racy_score
    }

    # Zapisanie wyników do Cosmos DB lub innego magazynu
    # ... (kod ązapisujcy wyniki)

    return func.HttpResponse(
        json.dumps(analysis_result),
        mimetype="application/json"
    )
```

7.3 Integrowana całość: od uploadu do analizy i przechowywania

```python
@app.blob_trigger(arg_name="blob",
                  path="uploads/{name}",
                  connection="AzureWebJobsStorage")
def analyze_uploaded_image(blob: func.InputStream):
    logging.info(f"Analizowanie obrazu: {blob.name}")

    # Inicjalizacja klienta Computer Vision
    vision_client = ComputerVisionClient(
        endpoint=os.environ["VISION_ENDPOINT"],
        credentials=CognitiveServicesCredentials(os.environ["VISION_KEY"])
    )
```

```
12
13      # Analiza obrazu
14      features = [
15          VisualFeatureTypes.description,
16          VisualFeatureTypes.tags,
17          VisualFeatureTypes.adult
18      ]
19
20      image_data = blob.read()
21      results = vision_client.analyze_image_in_stream(
22          image=image_data,
23          visual_features=features
24      )
25
26      # Decyzja o przeniesieniu do odpowiedniego kontenera
27      is_inappropriate = results.adult.is_adult_content or results.adult.is_racy_content
28      target_container = "inappropriate" if is_inappropriate else "appropriate"
29
30      # Inicjalizacja klienta Blob Storage
31      credential = DefaultAzureCredential()
32      blob_service = BlobServiceClient(
33          account_url=f"https://{os.environ['STORAGE_ACCOUNT']}.blob.core.windows.net",
34          credential=credential
35      )
36
37      # Zapisanie obrazu w odpowiednim kontenerze
38      target_blob_client = blob_service.get_blob_client(
39          container=target_container,
40          blob=blob.name.split('/')[-1]
41      )
42
43      target_blob_client.upload_blob(image_data, overwrite=True)
44
45      # Zapisanie metadanych w Cosmos DB
46      cosmos_client = CosmosClient(
47          os.environ["COSMOS_ENDPOINT"],
48          os.environ["COSMOS_KEY"]
49      )
50      database = cosmos_client.get_database_client("lab-database")
51      container = database.get_container_client("items")
52
53      metadata = {
54          "id": str(uuid.uuid4()),
55          "fileName": blob.name.split('/')[-1],
56          "uploadTime": datetime.datetime.utcnow().isoformat(),
57          "container": target_container,
58          "description": results.description.captions[0].text if results.description.captions
    else "",
59          "tags": [tag.name for tag in results.tags],
60          "adultContent": results.adult.is_adult_content,
61          "racyContent": results.adult.is_racy_content
62      }
63
64      container.create_item(body=metadata)
65
66      # łWysanie powiadomienia do kolejki
67      queue_client = QueueClient(
68          account_url=f"https://{os.environ['STORAGE_ACCOUNT']}.queue.core.windows.net",
69          queue_name="processing-queue",
70          credential=credential
71      )
72
73      notification = {
74          "fileName": blob.name.split('/')[-1],
75          "processed": True,
76          "destination": target_container,
77          "timestamp": datetime.datetime.utcnow().isoformat()
78      }
79
80      queue_client.send_message(json.dumps(notification))
81
82      logging.info(f"Obraz {blob.name} przetworzony i przeniesiony do {target_container}")
```

8 Lokalne testowanie i debugowanie

8.1 Konfiguracja Azurite

```
# Instalacja emulatora
npm install -g azurite

# Uruchomienie łusug
azurite --silent --location azurite-data --debug azurite.log

# Konfiguracja connection string
export AZURE_STORAGE_CONNECTION_STRING="DefaultEndpointsProtocol=http;AccountName=
    devstoreaccount1;AccountKey=Eby8vdM02xNOcqFlqUwJPLlmEtlCDXJ1OUzFT50uSRZ6IFsuFq2UVErCz4I6tq
    /K1SZFPTOtr/KBHBeksoGMGw==;BlobEndpoint=http://127.0.0.1:10000/devstoreaccount1;"
```

8.2 Przykładowy plik local.settings.json

```
{
  "IsEncrypted": false,
  "Values": {
    "AzureWebJobsStorage": "UseDevelopmentStorage=true",
    "FUNCTIONS_WORKER_RUNTIME": "python",
    "VISION_ENDPOINT": "https://your-vision-service.cognitiveservices.azure.com/",
    "VISION_KEY": "your-vision-key",
    "COSMOS_ENDPOINT": "https://your-cosmos-account.documents.azure.com:443/",
    "COSMOS_KEY": "your-cosmos-key",
    "STORAGE_ACCOUNT": "yourstorageaccountname"
  }
}
```

8.3 Obsługa dużych plików

```
def upload_large_file(file_path: str):
    blob_client = BlobClient.from_connection_string(
        conn_str=os.getenv("AZURE_STORAGE_CONNECTION_STRING"),
        container_name="uploads",
        blob_name=os.path.basename(file_path)
    )

    with open(file_path, "rb") as data:
        blob_client.upload_blob(
            data,
            blob_type="BlockBlob",
            max_block_size=4*1024*1024,  # 4 MB chunks
            max_concurrency=8
        )
```

9 Best practices i optymalizacja

- Użycie wiązań wejścia/wyjścia zamiast bezpośrednich wywołań SDK
- Implementacja wzorców idempotentności
- Monitorowanie z Application Insights
- Zarządzanie wersjami Pythona
- Obsługa współbieżności i synchronizacji
- Implementacja mechanizmów retry dla operacji zewnętrznych
- Optymalizacja cold startów

10 Bezpieczeństwo i kontrola dostępu

- Używanie Managed Identity dla funkcji
- Ograniczanie uprawnień przez zasady RBAC
- Szyfrowanie danych w spoczynku i transporcie
- Rotacja kluczy dostępu
- Ograniczenie dostępu do funkcji przez IP
- Zabezpieczenie komunikacji przez funkcje sieciowe

Usługa	**Zabezpieczenia**
Azure Functions	Managed Identity, Network Isolation
Cosmos DB	Role-Based Access Control, Firewall rules
Storage	SAS Tokens, Encryption at rest
Cognitive Services	Private Endpoints, Data encryption

Table 2: Mechanizmy bezpieczeństwa głównych usług

11 Rozwiązywanie typowych problemów

Problem 1 Funkcja nie pojawia się po deploymencie

Rozwiązanie Sprawdź model programistyczny (v1 vs v2), weryfikuj logi deploymentowe

Problem 2 Błędy importu w Python

Rozwiązanie Sprawdź requirements.txt, użyj wirtualnego środowiska

Problem 3 Limit czasu wykonania funkcji

Rozwiązanie Optymalizuj kod, rozważ Durable Functions

Problem 4 Problemy z połączeniem do usług zewnętrznych

Rozwiązanie Sprawdź konfigurację sieci, ustawienia firewall, poprawność kluczy dostępu

12 Wnioski i dalsze kroki

Niniejsze laboratorium demonstruje kompleksowy przepływ tworzenia aplikacji serverless w Azure z wykorzystaniem nowoczesnego stosu technologicznego. Studenci powinni eksperymentować z różnymi typami triggerów i integracją dodatkowych usług Azure. W praktyce, architektura serverless najlepiej sprawdza się w:

- Aplikacjach z nieregularnym obciążeniem
- Przetwarzaniu danych na żądanie
- Mikrousługach i API
- Przetwarzaniu strumieniowym danych
- Automatyzacji zadań

Dalszy rozwój umiejętności powinien obejmować poznanie Durable Functions, zaawansowanych wzorców integracji, oraz technik optymalizacji wydajności i kosztów.

Załącznik: Pełna konfiguracja Terraform

```
# main.tf
terraform {
  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "~> 3.0"
    }
  }
}

provider "azurerm" {
  features {}
}

# Zmienne
variable "environment" {
  description = "Environment name (dev, test, prod)"
  default     = "dev"
}

variable "location" {
  description = "Azure region for resources"
  default     = "westeurope"
}

# Grupa zasobów
resource "azurerm_resource_group" "lab" {
  name     = "lab-serverless-${var.environment}"
  location = var.location
}

# Storage Account
resource "azurerm_storage_account" "lab_storage" {
  name                     = "labstorage${var.environment}"
  resource_group_name      = azurerm_resource_group.lab.name
  location                 = azurerm_resource_group.lab.location
  account_tier             = "Standard"
  account_replication_type = "LRS"
  allow_blob_public_access = false
}

# Kontenery Blob Storage
resource "azurerm_storage_container" "uploads" {
  name                  = "uploads"
  storage_account_name  = azurerm_storage_account.lab_storage.name
  container_access_type = "private"
}

resource "azurerm_storage_container" "appropriate" {
  name                  = "appropriate"
  storage_account_name  = azurerm_storage_account.lab_storage.name
  container_access_type = "private"
}

resource "azurerm_storage_container" "inappropriate" {
  name                  = "inappropriate"
  storage_account_name  = azurerm_storage_account.lab_storage.name
  container_access_type = "private"
}

# Kolejka Storage
resource "azurerm_storage_queue" "processing_queue" {
  name                 = "processing-queue"
  storage_account_name = azurerm_storage_account.lab_storage.name
}

# Cosmos DB
resource "azurerm_cosmosdb_account" "lab_db" {
  name                = "lab-cosmos-${var.environment}"
  location            = azurerm_resource_group.lab.location
  resource_group_name = azurerm_resource_group.lab.name
```

```
  offer_type          = "Standard"
  kind                = "GlobalDocumentDB"

  consistency_policy {
    consistency_level = "Session"
  }

  geo_location {
    location          = var.location
    failover_priority = 0
  }
}

resource "azurerm_cosmosdb_sql_database" "lab_database" {
  name                = "lab-database"
  resource_group_name = azurerm_resource_group.lab.name
  account_name        = azurerm_cosmosdb_account.lab_db.name
}

resource "azurerm_cosmosdb_sql_container" "items" {
  name                = "items"
  resource_group_name = azurerm_resource_group.lab.name
  account_name        = azurerm_cosmosdb_account.lab_db.name
  database_name       = azurerm_cosmosdb_sql_database.lab_database.name
  partition_key_path  = "/id"
}

# Cognitive Services
resource "azurerm_cognitive_account" "vision" {
  name                = "lab-vision-${var.environment}"
  location            = azurerm_resource_group.lab.location
  resource_group_name = azurerm_resource_group.lab.name
  kind                = "ComputerVision"
  sku_name            = "S1"
}

# Function App
resource "azurerm_service_plan" "lab_plan" {
  name                = "lab-plan-${var.environment}"
  resource_group_name = azurerm_resource_group.lab.name
  location            = azurerm_resource_group.lab.location
  os_type             = "Linux"
  sku_name            = "Y1" # Consumption plan
}

resource "azurerm_linux_function_app" "lab_function" {
  name                       = "lab-function-${var.environment}"
  resource_group_name        = azurerm_resource_group.lab.name
  location                   = azurerm_resource_group.lab.location
  service_plan_id            = azurerm_service_plan.lab_plan.id
  storage_account_name       = azurerm_storage_account.lab_storage.name
  storage_account_access_key = azurerm_storage_account.lab_storage.primary_access_key

  site_config {
    application_stack {
      python_version = "3.10"
    }
  }

  app_settings = {
    "FUNCTIONS_WORKER_RUNTIME"    = "python"
    "APPINSIGHTS_INSTRUMENTATIONKEY" = azurerm_application_insights.lab_insights.
    instrumentation_key
    "STORAGE_ACCOUNT"             = azurerm_storage_account.lab_storage.name
    "COSMOS_ENDPOINT"             = azurerm_cosmosdb_account.lab_db.endpoint
    "VISION_ENDPOINT"             = azurerm_cognitive_account.vision.endpoint
  }

  identity {
    type = "SystemAssigned"
  }
}

```

```
# Application Insights
resource "azurerm_application_insights" "lab_insights" {
  name                = "lab-insights-${var.environment}"
  location            = azurerm_resource_group.lab.location
  resource_group_name = azurerm_resource_group.lab.name
  application_type    = "web"
}

# Przypisanie ról
resource "azurerm_role_assignment" "storage_blob_contributor" {
  scope                = azurerm_storage_account.lab_storage.id
  role_definition_name = "Storage Blob Data Contributor"
  principal_id         = azurerm_linux_function_app.lab_function.identity[0].principal_id
}

resource "azurerm_role_assignment" "cosmos_contributor" {
  scope                = azurerm_cosmosdb_account.lab_db.id
  role_definition_name = "Cosmos DB Account Reader Role"
  principal_id         = azurerm_linux_function_app.lab_function.identity[0].principal_id
}

# Outputs
output "function_app_name" {
  value = azurerm_linux_function_app.lab_function.name
}

output "storage_account_name" {
  value = azurerm_storage_account.lab_storage.name
}

output "cosmos_db_endpoint" {
  value = azurerm_cosmosdb_account.lab_db.endpoint
}

output "vision_endpoint" {
  value = azurerm_cognitive_account.vision.endpoint
}
```

Podsumowanie

Przedstawiony materiał laboratoryjny kompleksowo obejmuje kluczowe aspekty tworzenia aplikacji serverless w Azure, uwzględniając zarówno praktyczne implementacje jak i teoretyczne podstawy. Wykorzystanie Terraform do zarządzania infrastrukturą oraz najnowszego modelu programistycznego Pythona v2 gwarantuje aktualność i zgodność z obecnymi najlepszymi praktykami w chmurze. Integracja z usługami Azure Storage, Cosmos DB, Queue Storage oraz Cognitive Services pozwala na budowanie zaawansowanych, skalowalnych aplikacji bezserwerowych przy minimalnym nakładzie administracyjnym.