Методи і засоби розробки подіє-керованих застосунків на serverless архітектурі

Виконав: Моренець Ігор
Керівник: Шабінський Антон

План

1. Екскурс в serverless
2. Azure Functions
3. Чотири типові сценарії використання
4. Висновки
5. Ваші запитання

Serverless

- Називають Function-as-a-Service та наносервісами
- Гнучке автоматичне масштабування
- Плата за час виконання
- Stateless
- Event-driven
- Асинхронна комунікація

https://adatum.no/azure/serverless-powershell-azure-functions

Azure Functions

FaaS від Microsoft

Типові сценарії використання

```
public class Post
{
    public string Id { get; set; }
    public string Title { get; set; }
    public string Author { get; set; }
    public string Body { get; set; }
    public int? Rating { get; set; }
}
```

HTTP наносервіси

Отримати пост по ідентифікатору

```
public static IActionResult GetPost(
    [HttpTrigger(methods: "get", Route = "posts/{author}/{id}")]
    HttpRequest req,
    [CosmosDB(
        databaseName: "MyBlog",
        collectionName: "Posts"
        PartitionKey = "{author}",
        Id = "{id}"
    )]
    Post? post,
    string id
)
{
    if (post == null)
    {
        return new NotFoundObjectResult(
            new {message = $"Couldn't find a post with id {id}"}
        );
    }

    return new OkObjectResult(post);
}
```

Створення поста

```
public static async Task<IActionResult> AddPost(
    [HttpTrigger(methods: "post",Route = "posts")]
    HttpRequest req,
    [CosmosDB(
        databaseName: "MyBlog",
        collectionName: "Posts"
    )]
    IAsyncCollector<Post> collector
)
{
    var data = JsonConvert.DeserializeObject<RequestBody?>(
        await new StreamReader(req.Body).ReadToEndAsync()
    );
    // ...валідація

    await collector.AddAsync(
        new Post {
            Author = data.Author!,
            Body = data.Body!,
            Title = data.Title!
         }
    );

    return new OkResult();
}
```

Видалення поста

```
public static async Task<IActionResult> DeletePost(
    [HttpTrigger(
        methods: "delete",
        Route = "posts/{author}/{id}"
    )]
    HttpRequest req,
    [CosmosDB(
        databaseName: "MyBlog",
        collectionName: "Posts"
    )]
    DocumentClient client,
    string author,
    string id
)
{
    var posts = client
                .CreateDocumentQuery<Document>()
                .Where(d => d.Id == id);

    foreach (var doc in posts)
    {
        await client.DeleteDocumentAsync(doc.SelfLink);
    }

    return new OkResult();
}
```

```
public class Author
{
    public string Id { get; set; }
    public string Name { get; set; }
    public string[] Subscribers { get; set; }
}
```

Зв'язування сервісів

Сповіщення про новий пост

```
public static async Task EmailerFun(
   [CosmosDBTrigger("MyBlog", "Posts")] IList<Document> posts,
   [CosmosDB("MyBlog", "Posts")] DocumentClient client,
   [SendGrid] IAsyncCollector<SendGridMessage> collector
)
{
   foreach (var doc in posts)
   {
       var author = doc.GetPropertyValue<string>("Author");

       var messages = client
                      .CreateDocumentQuery<Author>()
                      .Where(a => a.Name == author)
                      .Select(
                          a => ComposeMessage(
                              from: "i.morenets@ukma.edu.ua",
                              to: a.Subscribers,
                              author: author,
                              title:
doc.GetPropertyValue("Title")
                              )
                      );

       foreach (var message in messages)
       {
           await collector.AddAsync(message);
       }
   }
}
```

Відступ: Оркестрація

MapReduce

Word Count

Розподілені обчислення

MapReduce Word count

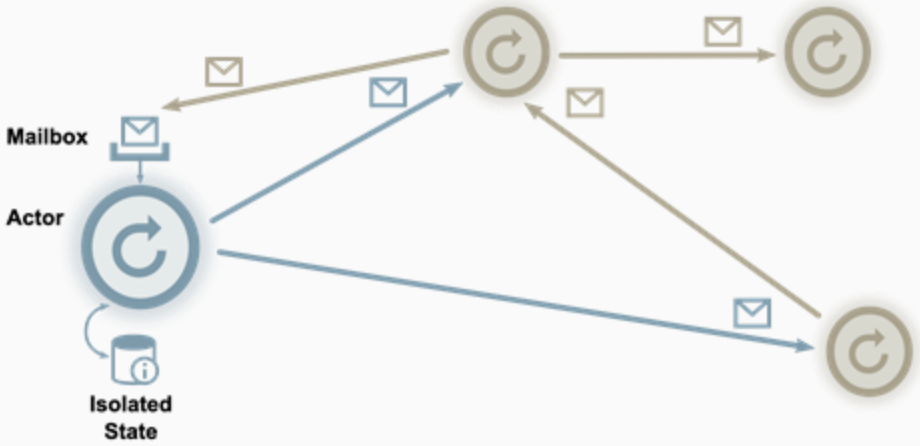
```
public static async Task WordCountOrchestrator(
    [OrchestrationTrigger] IDurableOrchestrationContext ctx
)
{
    var input = ctx.GetInput<WordCountInput>();
    var batches = Batching.ToBatches(ToLines(input.Content));

    var mapResults = await Task.WhenAll(
        batches.Select(
            batch => ctx.CallActivityAsync<IList<Result<string, int>>>(
                functionName: nameof(WordCountMap),
                input: batch
            )
        )
    );

    var groups = await ctx.CallActivityAsync<IList<Group<string, int>>>(
        functionName: nameof(WordCountGroup),
        input: mapResults
    );

    var reduceResults = await Task.WhenAll(
        groups.Select(
            group => ctx.CallActivityAsync<Result<string, int>>(
                functionName: nameof(WordCountReduce),
                input: group
            )
        )
    );

    await ctx.CallActivityAsync<string>(nameof(Output), reduceResults);
}
```



https://berb.github.io/diploma-thesis/original/054_actors.html

Модель акторів

Банківський рахунок

```
public class Account : IAccount
{
    public decimal Balance { get; set; } = decimal.Zero;

    public Task Replenish(decimal amount)
    {
        Balance += amount;
        return Task.CompletedTask;
    }

    public Task<bool> Withdraw(decimal amount)
    {
        var canWithdraw = amount <= Balance;

        if (canWithdraw)
        {
            Balance -= amount;
        }

        return Task.FromResult(canWithdraw);
    }

    public Task<decimal> GetBalance() => Task.FromResult(Balance);

    [FunctionName(nameof(Account))]
    public static Task Run(
        [EntityTrigger] IDurableEntityContext ctx
    ) => ctx.DispatchAsync<Account>();
}
```

Поповнення

```
public static async Task<IActionResult> ReplenishFun(
    [HttpTrigger(methods: "post", Route = "replenish")]
    HttpRequest req,
    [DurableClient] IDurableEntityClient client
)
{
    var data = JsonConvert.DeserializeObject<RequestBody?>(
        await new StreamReader(req.Body).ReadToEndAsync()
    );

    // ...валідація

    await client.SignalEntityAsync<IAccount>(
        entityKey: data.Account,
        operation: account => account.Replenish(data.Amount.Value)
    );

    return new OkObjectResult(new {
      message = "Successfully replenished"
    });
}
```

Зняття коштів

```
public static async Task<bool> WithdrawFun(
    [OrchestrationTrigger] IDurableOrchestrationContext ctx
)
{
    var input = ctx.GetInput<WithdrawArgs>();
    var account = ctx.CreateEntityProxy<IAccount>(input.Account);

    var wasSuccessful = await account.Withdraw(input.Amount);

    return wasSuccessful;
}


private class WithdrawArgs
{
    public string Account { get; set; }
    public decimal Amount { get; set; }
}
```

Переказ

```
public static async Task<bool> TransferFun(
    [OrchestrationTrigger] IDurableOrchestrationContext ctx
)
{
    var input = ctx.GetInput<TransferArgs>();

    var fromEntity = new EntityId(nameof(Account), input.From);
    var toEntity   = new EntityId(nameof(Account), input.To);

    using (await ctx.LockAsync(fromEntity, toEntity))
    {
        var from = ctx.CreateEntityProxy<IAccount>(fromEntity);
        var to   = ctx.CreateEntityProxy<IAccount>(toEntity);

        var hasEnoughFunds = await from.Withdraw(input.Amount);

        if (!hasEnoughFunds)
        {
            return false;
        }

        await to.Replenish(input.Amount);
    }

    return true;
}
```

Висновки

- Досліджено serverless як інструмент та архітектуру
- Досліджені основні постачальники
- Виділено та розроблено чотири сценарії використання
- Як розширення - реальний проект

Дякую за увагу