Kickstarting a charity with Serverless Technologies

James Randall
About me

https://github.com/JamesRandall

https://github.com/JamesRandall

https://www.azurefromthetrenches.com
Hello, we’re Bookmark. We want every child to read.

In an average class, 8 children leave primary school unable to read well.

These children often struggle in school, and beyond. Reading isn’t just about books. It’s about reading a road sign, a safety manual, a birthday card. It’s understanding a job application and the prescription that could help save your life.
The problem

- Connect schools, volunteers and Bookmark staff

- Three main areas to address:
  - Discovery
    - Schools finding volunteers
    - Volunteers finding schools
  - Scheduling
    - People live busy lives
    - Schools have constraints
  - Safety
Constraints

- A capped and small development budget
- A small operational budget
- A small development team – for the most part: me
- A small operational team – for the most part: me
- A none-technical internal and external audience
  - Bookmark “back office”
  - Schools
  - Volunteers
- A deadline – we needed to be available inside of 5 months to beta in the upcoming school year
Beginnings

• We started with the front end – what were we building
• Fairly typical wireframing and storyboard process
• These were worked through with potential early adopters
• This teased out the main domains

• It also highlighted there was a lot of work to do!
Modular Monoliths

• I wanted, needed, to ”have my cake and eat it”
  • Simple coding in a single codebase with low operational overhead and great support from tools
  • Strict separation of concerns and clear demarcation between bounded contexts
  • The capability to decompose later

• Subsystems broken down using Domain Driven Design
• The crossing of bounded contexts always takes place via a command
Either split API into school and volunteer or internally do the same via scopes. Will determine later as driven by API surface scope.

**REST API**
Component: ASP.NET Core
Configuration REST Controller

**Messaging**
Component: Azure Service Bus
Uses

**Mediator**
Component: AccordFramework/Relay

**Cache**
Component: Microsoft.Extensions.Caching

**School Application**
Component: Dr-Command Handler Application
School-specific settings for each application, such as any config, user permissions

**Relationship Application**
Component: Dr-Command Handler Application
Provides an interface between the school and volunteer network graph

**Pupil Application**
Component: Dr-Command Handler Application
Manages pupil details. The only place where there are actual relationships (e.g., teacher and pupil)

**Programme Application**
Component: Dr-Command Handler Application
Manages setup and sending of study programmes to the school

**Volunteer Application**
Component: Dr-Command Handler Application
Manages setup and sending of study programmes to the school

**Activity Feed Application**
Component: Dr-Command Handler Application
Provides access to the per volunteer feed

**Calendar Application**
Component: Dr-Command Handler Application
Provides access to the per volunteer feed

**Rewards Application**
Component: Dr-Command Handler Application
Manages available rewards and progress towards them

**Graph Database**
Component: Azure Cosmos DB
Anymarshes the relationships between volunteers, schools and students

**Document Database**
Component: Azure Cosmos DB
Stores basic details about public names, tags of each, year group.

**Relational Database**
Component: Azure SQL Database
Relational model of programmes, schedules, and volunteer and pupil assignments (via tags)

**Document Database**
Component: Azure Cosmos DB
Saves a volunteer profile

**Document Database**
Component: Azure Cosmos DB
Stores a volunteer profile

**Key/Attribute Store**
Component: Azure Key Vault
Utilities

**Key/Attribute Store**
Component: Azure Key Vault
Tableau for badge distribution and volunteer progress towards badges (correlation with badge objectives)

**Messaging**
Component: Azure Service Bus
Simple broker for cross-domain communications and relaying
Step in Serverless

• Pay for what you use
• Scale to billable zero
• Event based programming model
• Highly managed with low operational overhead
• We used a 100% serverless compute platform
No Silver Bullet

• We combined it with a constant focus on high value implementation patterns
Storage

- Azure Storage (blob and table)
- Cosmos DB (graph and document / SQL)
- Azure SQL Database
Other Services

- Application Insights
- Azure DevOps
- Azure Key Vault
- Auth0
Commands and Mediators

• A command is simple state associated with a C# type
• A command is dispatched for execution to a mediator
• The mediator invokes the configured executer:
  • In process via a command handler
  • Out of process via dispatch for remote execution (other Functions, Service Bus, APIs)
• Cross cutting concerns addressed consistently and once within the mediator
What does this look like?

- This was all supported with the Function Monkey library
  [https://functionmonkey.azurefromthetrenches.com](https://functionmonkey.azurefromthetrenches.com)
public class FunctionAppConfiguration : IFункциAppConfiguration
{
    public void Build(IFunctionHostBuilder builder)
    {
        builder
            .Setup((serviceCollection, commandRegistry) => { /* dependencies */ })
            .AddFluentValidation()
            .Authorization(authorization => authorization
                .TokenValidator<TokenValidator>()
                .AuthorizationDefault(AuthorizationType.TokenValidation)
                .Claims(claims => claims
                    .MapClaimToProperty(claimType: "userId", propertyName: "UserId")
                )
            )
            .Functions(functions => functions
                .HttpRoute("api/v1/todoItem", httpFunctionBuilder: route => route
                    .HttpFunction<AddToDoItemCommand>(HttpMethod.Post)
                    .HttpFunction<GetAllToDoItemsQuery>(HttpMethod.Get)
                    .HttpFunction<MarkItemCompleteCommand>(route: "/{itemId}/complete", HttpMethod.Put)
                )
                .ServiceBus(serviceBus => serviceBus
                    .QueueFunction<AddToDoItemCommand>(queueName: "newtodoitem")
                )
            );
    }
}
Sample Command

```csharp
public class AddToDoItemCommand : ICommand<ToDoItem>
{
    [SecurityProperty]

    public string UserId { get; set; }

    public string Title { get; set; }
}
```
internal class AddToDoItemCommandHandler : ICommandHandler<AddToDoItemCommand, ToDoItem>
{
    private readonly IToDoItemRepository _repository;

    public AddToDoItemCommandHandler(IToDoItemRepository repository)
    {
        _repository = repository;
    }

    public async Task<ToDoItem> ExecuteAsync(AddToDoItemCommand command, ToDoItem previousResult)
    {
        ToDoItem newItem = new ToDoItem
        {
            CreatedAtUtc = DateTime.UtcNow,
            CreatedByUserId = command.UserId,
            Id = Guid.NewGuid().ToString(),
            IsComplete = false,
            Title = command.Title
        };

        await _repository.Upsert(newItem);

        return newItem;
    }
}
• Our compute code is very lean
  • 95% + addressed business concerns
  • No boilerplate
  • Consistent
  • Its easy to move and repurpose

• Its easy to test – both acceptance and unit

• Its easy to change

• Operationally its been a breeze – nothing to do, everything is automated and was easy to automate
  • We had Azure DevOps Pipelines set up from the very start
  • Everything is a highly managed Azure service
• Operational costs are low – we have three always available environments running (dev, preview, live) and spend little

• Support has been straightforward
  • We really benefited from the consistent addressing of cross cutting concerns

• It’s easy to change – we went through two significant redesigns of scheduling in response to feedback and it was fairly straightforward

• The Service Bus acted as a low-UI operation manager
• Other than the function declarations our compute platform is completely decoupled from Functions themselves
• If required we can pick it up and drop it in a container
  • No intention to do so!
• Until “run from package” was released deployments were unreliable
  • Azure App Service locked file issues
• Requirement changes led to Cosmos capabilities going largely unused but we are paying its minimum 400RU footprint
Or rather: To do!

- Make use of API Management
  - Prior to its serverless variant this was disproportionately expensive
  - Would have cost more than the rest of the platform combined
- More data to integrate in the Warehouse from other external systems
- Move the React apps to Netlify
- Find a better meme!
Function Monkey -
https://functionmonkey.azurefromthetrenches.com

Function Monkey dev.to Tutorial -

Slides -
https://www.azurefromthetrenches.com/serverlesslondon/

Mediator -
https://commanding.azurefromthetrenches.com

Bookmark Reading -
https://www.bookmarkreading.org