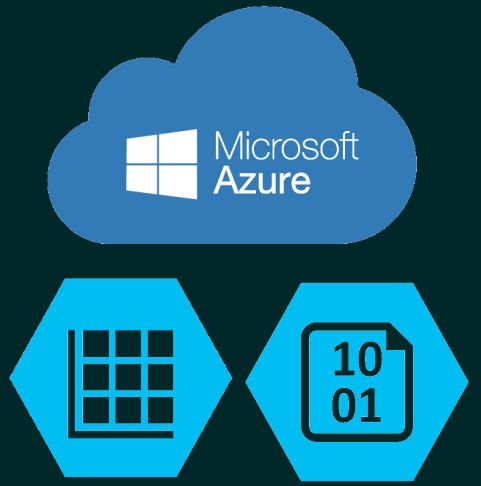


Вагиф Абилов

Журнал событий в ажуре (и без всякого космоса)

Консультант в Miles

Работаю с F# и C#

@ooobject
vagif.abilov@mail.com

О чем сегодня пойдет речь

- Журналы событий (event journals)
- События как источники состояния сущностей (event sourcing)
- Моделирование систем на основе источников событий
- DDD и Event Sourcing
- Способы хранения и извлечения событий
- Принципы построения журналов событий
- Журналы событий в облаке
- Примеры реализации журналов событий на основе Azure Table Storage и Blob Storage

Как родилась идея доклада

Различные презентации и статьи, описывающие избыточные методы реализации журналов событий

Что значит «избыточные»?

Избыточные по стоимости используемых облачных ресурсов

Что значит «избыточные»?

Избыточные по функциональным возможностям, не требующимся для реализации журналов событий

В этом месте должен был быть опрос аудитории

- Кто использовал в проектах или хорошо знаком с журналами событий (Event Journal)?
- Кто использовал или знаком с методом построения состояния сущностей из истории событий (Event Sourcing)?
- Кто знает разницу между ними?

Краткое введение в историю вопроса

Построение систем на основе источников событий (event sourcing)

Мартин Фаулер (2005)

Запись всех изменений состояния системы как последовательности событий

https://martinfowler.com/eaaDev/EventSourcing.html

Уди Дахан (2020)

Хранение полной истории предмета как последовательности событий вместо хранения лишь текущего состояния

https://www.slideshare.net/udidahan/udi-dahan-event-sourcing-keynote-ddd-eu

Значительная часть систем использует репозитории, реализованные по принципу хранения текущего состояния сущностей

Немалая часть из них хранит
ТОЛЬКО
текущие состояния

Customer

Customer Address

- AddressType
- Address
- City
- PostalCode
- Country
- LastChanged

Учет причины изменения адреса

- Переезд (CustomerRelocated)
- Коррекция (WrongAddressCorrected)

Учет причины изменения адреса

- Переезд (CustomerRelocated)
- Коррекция (WrongAddressCorrected)

Различные причины изменения адреса могут оказывать влияние на логику обработки такого события

Учет причины изменения адреса

- Переезд (CustomerRelocated)
- Коррекция (WrongAddressCorrected)

Переезд: Высланные на старый адрес товары не требуют дополнительных операций

Коррекция: Товары высланы на неверный адрес, требуется их возврат и новая рассылка

Как это отразить в модели данных?

- Добавляем поле: ChangeReason
- Храним предыдущие версии

Как это отразить в модели данных?

- Добавляем поле: ChangeReason
- Храним предыдущие версии

То же самое придется делать со многими полями различных таблиц

Как это отразить в модели данных?

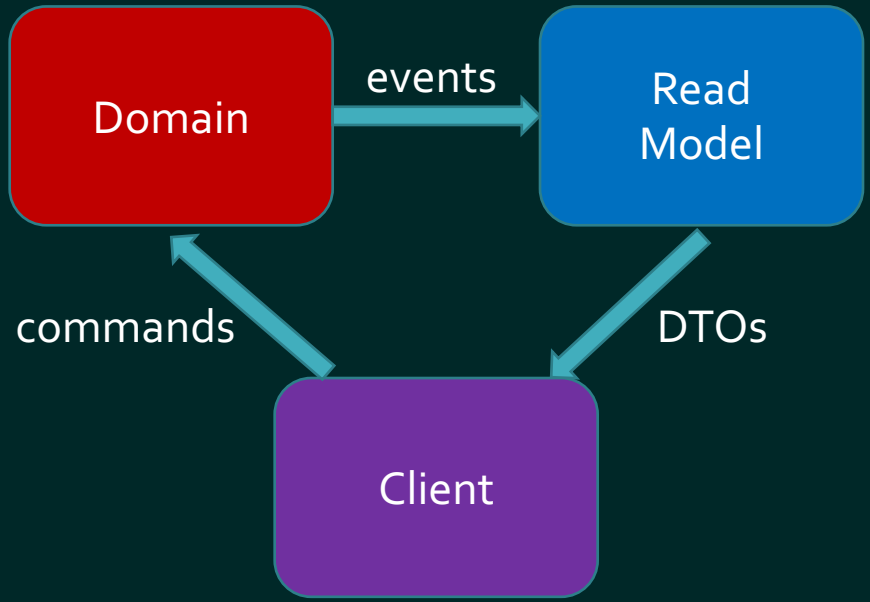
- Добавляем поле: ChangeReason
- Храним предыдущие версии

То же самое придется делать со многими полями различных таблиц
Схема становится громоздкой, совмещает текущие состояния и историю их изменения (которые редко используются)

На помощь приходят Event Sourcing и CQRS (Command/Query Responsibility Segregation)

Метод CQRS – не то же самое,
что event sourcing,
но часто применяется совместно

Грег Янг (2010)



https://cqrs.files.wordpress.com/2010/11/cqrs_documents.pdf

Хранилища данных для Event Sourcing / CQRS

- Хранилище для записи событий (Event Journal)
- Хранилище для чтения текущих состояний (классическая база SQL, GraphQL, NoSQL или их комбинация) - за пределами этого доклада

Применение Event Sourcing

Можно пойти дальше и отказаться от самостоятельного хранилища текущего состояния сущностей, собирая их на лету из истории событий

Применение Event Sourcing

Применение Event Sourcing требует
знания основ DDD
и правильного выбора корневых агрегатов

Примеры корневых агрегатов

- Финансовый сектор: банковский счет, пользователь банка
- Автодорожная инспекция: транспортное средство, владелец транспортного средства
- Телерадиовещательная компания: канал вещания, теле- или радиопрограмма, клип новостей

Применение Event Sourcing

- Event Sourcing использует журнал событий (Event Journal) для построения текущих состояний сущностей
- Допускается создание слепков (shapshot) состояний для оптимизации построения состояний сущностей с большой историей событий

Краткое введение в историю вопроса

Журналы событий (event journals)

Применение журналов событий

Реализация журналов событий принципиально не отличается для разных предметных областей

Применение журналов событий

У журналов событий есть схема данных, но она не зависит от конкретной области применения

Минимальная схема журнала событий

- EntityId - идентификатор сущности (корневого агрегата)
- SequenceNr/Timestamp - момент совершения события (порядковый номер, время или расширенное время)
- Payload - содержимое события

EntityId + SequenceNr составляют первичный ключ записи журнала событий

Почему SequenceNr, а не просто время?

- В одно и то же время может произойти несколько событий
- Для некоторых предметных областей достаточно порядкового номера
- Событие может описываться несколькими временами, некоторые предметные области требуют хранить их все

Домашнее упражнение для наиболее пытливых

- Разберитесь с битемпоральными (bitemporal) базами данных, обеспечивающими хранение времени совершения события и времени его регистрации

Домашнее упражнение для наиболее пытливых

- Разберитесь с битемпоральными (bitemporal) базами данных, обеспечивающими хранение времени совершения события и времени его регистрации
- Битемпоральные выглядят просто? Разберитесь с тритемпоральными, сохраняющими еще и время принятия решения

Применение журналов событий

Многие реализации журналов событий
используют порядковые номера событий,
забота о времени перекладывается
на плечи разработчиков приложений
(которые включают его в содержимое событий)

Журнал событий Akka.NET (Microsoft SQL Server)

```
CREATE TABLE EventJournal (
  Ordering BIGINT IDENTITY(1,1) PRIMARY KEY NOT NULL,
  PersistenceID NVARCHAR(255) NOT NULL,
  SequenceNr BIGINT NOT NULL,
  Timestamp BIGINT NOT NULL,
  IsDeleted BIT NOT NULL,
  Manifest NVARCHAR(500) NOT NULL,
  Payload VARBINARY(MAX) NOT NULL,
  Tags NVARCHAR(100) NULL
  CONSTRAINT QU_EventJournal UNIQUE (PersistenceID, SequenceNr)
)
```

Где можно хранить журналы событий?

- Почти что где угодно (ну разве кроме blob storage, хотя и blob storage пойдет для хранения поля Payload)
- Для минимального журнала событий достаточно любого хранилища с возможностью упорядоченной просмотра записей по ключу идентификатора сущности

Журналы событий в облаке

Мы ограничимся облаками Microsoft Azure, но все популярные облачные сервисы имеют типы хранилищ, пригодные для хранения журналов событий

Ажурные облачные хранилища

- Blob storage - не подходит (но может использоваться в сочетании с другими)
- SQL Server - подходит
- Table storage - подходит для событий с размером содержимого до 64KB (или 1MB при делении содержимого между полями записи), в противном случае требуется совмещение хранилищ table/blob
- Cosmos DB (DocumentDB, MongoDB и т.п.) - подходит

Критерии выбора облачных сервисов

- Соответствие функциональным требованиям
- Стоимость
- Быстродействие
- Условия предоставления услуги (SLA)

Журналы событий и Cosmos DB

- Если стоимость не является приоритетным фактором принятия решения, то Cosmos DB предоставляет наиболее полный набор возможностей
- Индексация по умолчанию всех полей записей

Аргументы в пользу Cosmos DB

- Георепликация
- Гибридный API, поддерживающий популярные базы данных
- Надежность хранения данных (SLA)
- Индексация по умолчанию всех полей записей
- Дополнительный функционал, например Change Feed, позволяющий подписываться на изменения

Аргументы против Cosmos DB

- Высокая стоимость хранения
- Низкая пропускная способность при выборе наиболее экономичного плана
- Необходимость постоянного регулирования плана при неровной загрузке
- Для журнала событий: индексация по умолчанию всех полей записей

Прежде чем мы перейдем к техническим аспектам

- Azure Cosmos DB предлагает более надежное с точки зрения безопасности и восстановления данных соглашение об уровне обслуживания, чем Table/Blob storage

- “You will choose Cosmos DB when you need multiple region redundancy, the highest throughput, minimal latency, or control of failover scenarios” https://microsoft.github.io/AzureTipsAndTricks/blog/tip166.html

Соглашение об уровне обслуживания Azure Cosmos DB

Azure Cosmos DB — это многомодельная служба базы данных от Microsoft, распространяемая по всему миру. Она предлагает глобальное распределение под ключ для любого числа регионов Azure с прозрачным масштабированием и воспроизведением данных для пользователей в любых частях света. Служба предлагает полные 99,99 % SLA, которое покрывает гарантии на пропускную способность, соответствие, доступность и задержку для учетных записей базы данных Cosmos DB, ограниченных одним регионом Azure, с конфигурацией по одному из пяти уровней соответствия или учетные записи, охватывающие несколько регионов, с конфигурацией по одному из четырех нестрогих уровней соответствия. Azure Cosmos DB позволяет конфигурировать множество регионов Azure в качестве конечных точек для записи для учетной записи базы данных. В данной конфигурации база данных Cosmos DB обеспечивает доступность SLA 99,999 % для записи и чтения.

Недостатки индексация всех полей записи

- Уменьшает быстродействие записи в журнал
- Увеличивает размер хранилища (а значит и стоимость хранения)
- Не требуется для функционала журнала событий
- Может стать причиной принятия неверных архитектурных решений

Недостатки индексация всех полей записи

- Уменьшает быстродействие записи в журнал
- Увеличивает размер хранилища (а значит и стоимость хранения)
- Не требуется для функционала журнала событий
- Может стать причиной принятия неверных архитектурных решений

Последнее стоит пояснить

Утверждение

Все поля и методы классов C#/Java должны быть публичными, чтобы сделать их доступными для будущих сценариев

Утверждение

Все поля и методы классов C#/Java должны быть публичными, чтобы сделать их доступными для будущих сценариев

Кто согласен?

А зачем вообще индексировать внутреннюю информацию содержимого записи журнала событий?

В журнале событий перемешаны яблоки с апельсинами (а также шинами и мокасинами)

Бессмысленно пытаться создать общую схему разнородных событий системы,
а значит бессмысленно индексировать отдельные их поля

Выбирая Table/Blob storage, не забываем

- Отсутствие георепликации
- Фатальные последствия ошибочного удаления хранилища (AzCopy - ваш друг)
- Оптимизация быстродействия не может быть достигнута просто изменением сервисного плана (однако, Table/Blob и так обладают отменным быстродействием)

Ограничения размера записей Azure Table

- Максмимальный размер поля - 64 KB
- Максимум 16 полей в записи
- Максимальный размер записи - 1 MB

Упаковка содержимого событий позволяет хранить события с содержимым примерно в 1 мегабайт

Если вам этого не хватает, используйте Table вместе с Blob, а заодно подумайте, нужную ли информацию вы включаете в события

Ограничения на размер сообщений в очередях Azure

- Azure queue - 64 KB
- Azure service bus (standard) - 256 KB
- Azure service bus (premium) - 1 MB

Пример разницы в стоимости (наш проект)

- Tables: 0.568 NOK / Гб / месяц
- Blobs: standard page 0.3652 NOK / Гб / месяц
- Cosmos DB single region write: 2.0286 NOK / Гб / месяц
- 400 RU / коллекцию, 0.065 NOK / час / 100 RU

- Грубая оценка: ca. 100 000 NOK/год для Cosmos DB vs. 10 000 NOK/год для Table/Blob

Подходит ли структура Table для журнала событий?

- Достаточен ли для операционных нужд журнал событий единственным индексом EntityId + SequenceNr / Timestamp?
- Понадобится ли вам проигрывание всех событий во временном интервале?
- Такая потребность может возникнуть как при тестировании, так и при аварийном восстановлении системы после сбоя
- Azure Table storage содержит единственный кластерный индекс

Способы решения проблемы единственного индекса

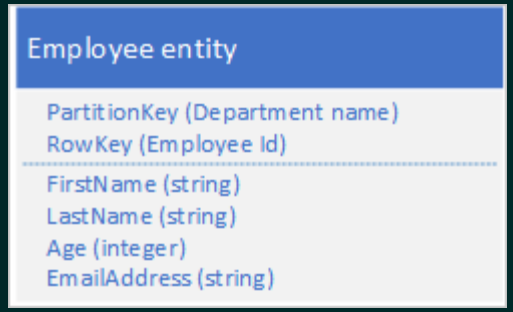
- Создание двух таблиц для каждого журнала
- Создание гибридного индекса в журнале с дупликацией записей (необязательно дублировать содержимое каждого события) – inter-partition secondary index pattern

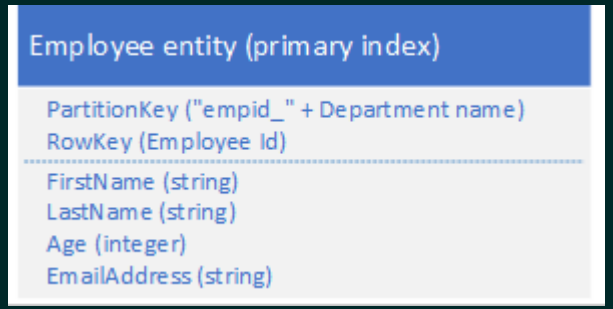
Inter-partition secondary index pattern

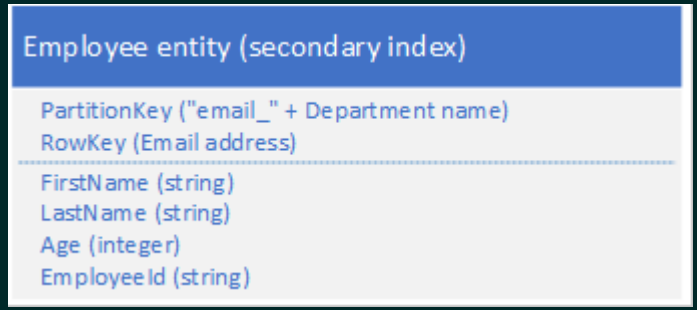
Создание множественных копий каждой сущности с различными значениями PartitionKey/RowKey той же самой таблицы, или в разных таблицах, для обеспечения эффективных запросов с различными ключами запроса и полями сортировки результатов

https://docs.microsoft.com/en-us/azure/storage/tables/table-storage-design-patterns

Создание вторичного индекса в хранилище Table







Создание вторичного индекса в хранилище Table

PARTITIONKEY	ROWKEY
date:20111124	2520801233544390000-1da8229e057d41a69ecc6bf6704b5ec5
date:20111124	2520801238446240000-df1f0fcddcb0405b90660e94c4ab9a47
date:20111124	2520801247020560000-700738d4b0f94045ba77138522212741
date:20111124	2520801247022590000-a7b143eabaf8429abc4424209efc95ac
date:20111124	2520801247025400000-459cfbf32e394ccf8339e0f2844e2c7d
date:20111124	2520801247027430000-39e324c70512421e9207a87a09f434e8
date:20111124	2520801247029310000-17ce22b5e29b485cb803e5e41a471c91
date:20111124	2520801247031030000-f3c1c5d634504d17b7b5ec762bdf775b
date:20111124	2520801247032900000-43173545208f4087af1324d73179124b
date:20111124	2520801247035710000-a4cfa8b0f4e7466d8f80a9205722b539
date:20111124	2520801247037590000-35584fb1c9e6494f9a14956988dbe83c
date:20111124	2520801247039930000-d5413e8c12a24a5fa42e5ef589d3a524
date:20111124	2520801247041650000-8b4e143083db4fb4a4c3ab330327a725
date:20111124	2520801247043370000-5fef07074a4045248585cf5c4a5e7860
date:20111124	2520801247045400000-009715d8b039492f935a3e33450dab7f
date:20111124	2520801247047430000-b90eb3c57c0f4bf29acad58dbb9c261c

PARTITIONKEY	ROWKEY
id:msus49001610	2518554144102920000-6bbb12e8ffd846a08c02690c67eef70a
id:msus49001610	2518557034207680000-e209b04ef7364960a317810695e65613
id:msus49001610	2518693634532870000-1048334667ae45b6a69efa8a5a648f10
id:msus49001610	2518694845960910000-acb4cfbcb7344aa784d502b48f3c172b
id:msus49001610	2518831018325740000-1e68eef7df2b4f6ab3fa18d352a2f3d4
id:msus49001610	2518832632869460000-95be691c2ac44875b940907b26557d80
id:msus49001610	2518832632902530000-e8c616007ea74de9867cbba500539467
id:msus49001610	2518833531547330000-d027522e9dfa43e689441bdc3f54588c
id:msus49001610	2518833531576500000-e9fbd4196129439ab9a444462e54bc55
id:msus49001610	2518833905484000000-e04ede67ed2447428580d1ad2a8ef145
id:msus49001610	2518833912547420000-0d9d482bc7e9414abda7b238b69631ba
id:msus49001610	2519079866139530000-22fd124745664bf889b5ed75904d9a14
id:msus49001610	2519079866816550000-4d44a2837c2144978d4a64d5f2659c0f
id:msus49001610	2519080672874680000-07d19dffb51640ac821ef7eb294b19a9
id:msus49001610	2519080676713870000-a8885bde77564ebba1c9b8d5aca224c2

ДЕМО

Журнал событий в облаке на основе Azure Table/Blob storage

https://github.com/object/AzureEventJournal

Методы реализации облачного журнала событий

- Базовая версия: приложение вызывает напрямую метод библиотеки, записывающий событие в Table/Blob storage
- Использование облачных очередей: приложение пишет события в Azure Queue storage
- Использование облачных функций: приложение делает HTTP-запрос (обычно HTTP POST) для передачи события в Azure Function

```
[FunctionName(nameof(Enqueue))]
public static async Task<IActionResult> Enqueue(
    [HttpTrigger(
        AuthorizationLevel.Admin,
        "post",
        Route = "queue/{version}/{collectionName}")]
        HttpRequest req,
    Binder binder,
    string version,
    string collectionName,
    ILogger log)
```

```
var queueName = $"{QueueNameBase}-{version}";
log.LogInformation($"Posting '{collectionName}’
                   event to '{queueName}' queue.");

var queue = await binder.BindAsync<CloudQueue>(
    new Attribute[] { new QueueAttribute($"{queueName}") });
await queue.CreateIfNotExistsAsync();
```

```
var msgBody = await new StreamReader(req.Body).ReadToEndAsync();
var providerEvent =
    JsonConvert.DeserializeObject<ProviderEvent>(msgBody);
var compressedContent =
    Convert.ToBase64String(Utils.Zip(providerEvent.Content));
providerEvent.Content = compressedContent;

await SendToQueueAsync(queue, collectionName, providerEvent, log);
return new NoContentResult();
```

(проверка корректности данных и выброс исключительных состояний не показаны для краткости)

```
private static async Task SendToQueueAsync(CloudQueue queue,
    string collectionName, ProviderEvent providerEvent, ILogger log)
{
    var envelope = new Dictionary<string, object>
    {
        {EventCollectionElementName, collectionName},
        {EventPayloadElementName, providerEvent},
    };
    var messageText = JsonConvert.SerializeObject(envelope);
    var message = new CloudQueueMessage(messageText);
    await queue.AddMessageAsync(message);
}
```

```
[FunctionName(nameof(QueueTrigger))]
public static async Task QueueTrigger(
    [QueueTrigger("service-events-v1")] string messageText,
    Binder binder,
    ILogger log)
{
    var envelope = JObject.Parse(messageText);
    var providerEvent = new ProviderEvent(ParseEventEnvelope(
        envelope, out var collection));
    providerEvent.Content = Utils.Unzip(Convert.FromBase64String(
        providerEvent.Content));
    var storage = await Utils.GetStorageAsync(binder,
        collection.ToLower());
    await SaveToStorageAsync(storage, collection, providerEvent, log);
}
```

```
private static async Task SaveToStorageAsync(Storage storage,
    string collectionName, ProviderEvent providerEvent, ILogger log)
{
    var entityId = providerEvent.Id;

    var entity = new EventEntity(entityId, providerEvent.Created);
    entity.Id = providerEvent.Id;
    entity.Created = providerEvent.Created;
    entity.Content = Utils.Zip(providerEvent.Content);
    entity.Description = providerEvent.Description;

    await entity.SaveToStorageAsync(
    storage.Table, storage.BlobContainer);
}
```

```
public async Task SaveToStorageAsync(
    CloudTable table, CloudBlobContainer container)
{
    if (this.Content.Length >= MaxTableContentLength)
    {
        using (var blobStream = new MemoryStream(this.Content))
        {
            var blob = container.GetBlockBlobReference(
                        CreateContentId());
            await blob.UploadFromStreamAsync(blobStream);
        }
        this.Content = null;
    }

    (вторая часть метода на следующем слайде)
}
```

```
{
    (продолжение метода) SaveToStorageAsync)

    await table.ExecuteAsync(TableOperation.Insert(this));
    var entityCopy = new EventEntity()
    {
        PartitionKey = CreatePartitionKey(this.Created),
        RowKey = this.RowKey,
        Id = this.Id,
        Description = this.Description,
        Created = this.Created,
    };
    await table.ExecuteAsync(TableOperation.Insert(entityCopy));
}
```

```
public static string CreatePartitionKey(string id)
{
    return IdKeyPrefix + NormalizeId(id);
}

public static string CreatePartitionKey(DateTime date)
{
    var utcDate = date.ToUniversalTime();
    return DateKeyPrefix +
        $"{utcDate.Year:D4}{utcDate.Month:D2}{utcDate.Day:D2}";
}
```

```
public static string CreateRowKeyBase(DateTime created)
{
    return $"{(DateTime.MaxValue -
 created.ToUniversalTime()).Ticks + 1:D19}";
}

public static string CreateRowKey(DateTime created)
{
    var keyBase = CreateRowKeyBase(created);
    var rowGuid = Guid.NewGuid().ToString("N").ToLower();
    return $"{keyBase}-{rowGuid}";
}
```

ДЕСЕРТ

Поддержка Swagger

Aliencube.AzureFunctions.Extensions.OpenApi 1.5.4

This package helps render Open API document and Swagger UI through Azure Functions endpoints.

Package Manager | .NET CLI | **PackageReference** | Paket CLI

```
<PackageReference Include="Aliencube.AzureFunctions.Extensions.OpenApi" Version="1.5.4" />
```

For projects that support PackageReference, copy this XML node into the project file to reference the package.

Dependencies

Used By

Version History

Version	Downloads	Last updated
1.5.4	76,533	7/8/2019
1.5.3	4,810	6/12/2019
1.5.2	849	6/1/2019
1.5.1	8,139	4/12/2019
1.5.0	262	4/12/2019

Show more

Info

last updated 7/8/2019

Project Site

Source repository

License Info

Contact owners

Report

Download package (838.92 KB)

Statistics

106,198 total downloads

76,533 downloads of current version

212 downloads per day (avg)

View full stats

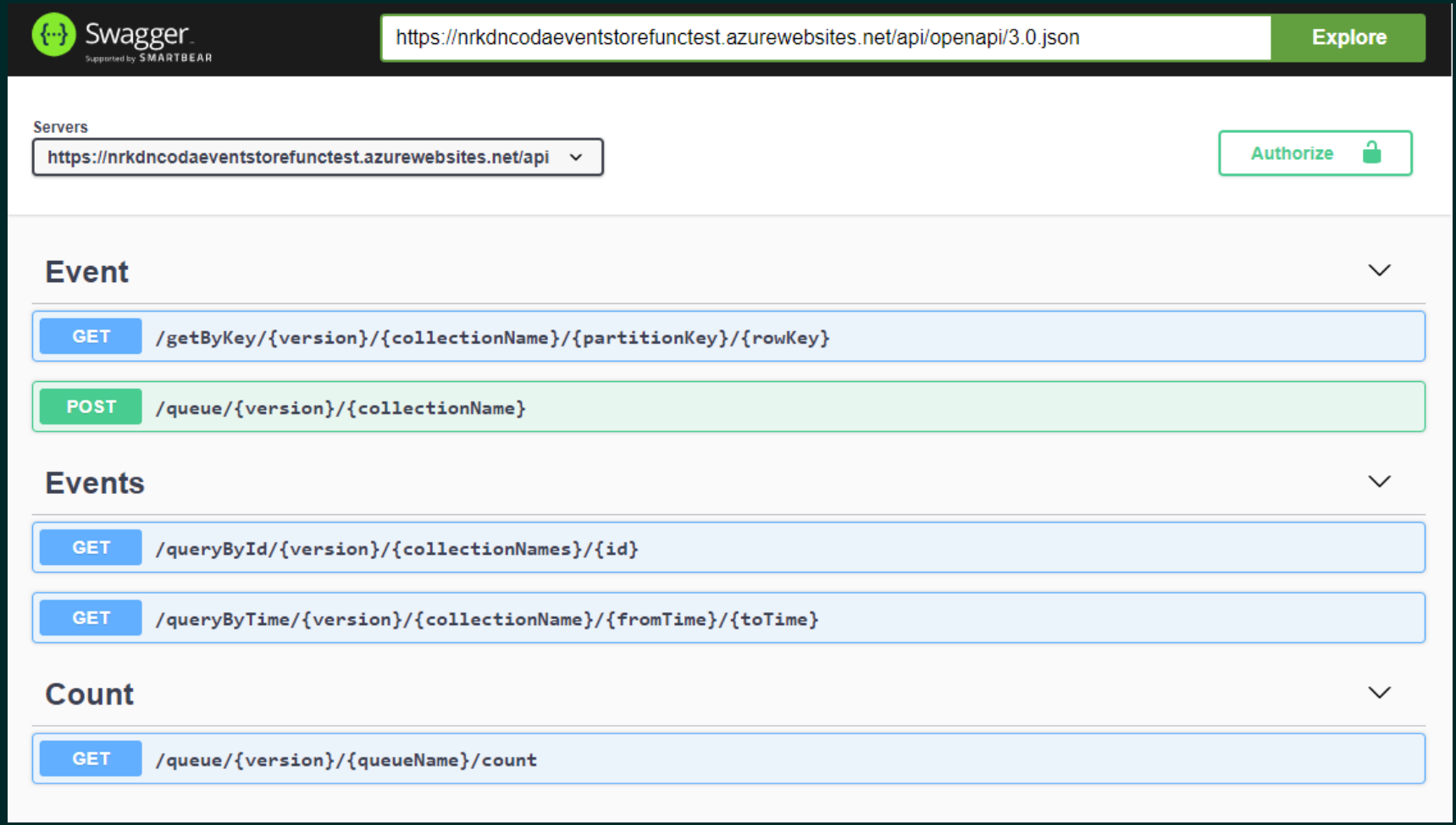
Owners

justinyoo

Authors

Justin Yoo

```
[FunctionName(nameof(Enqueue))]
[OpenApiOperation(nameof(Enqueue), "Event")]
[OpenApiParameter("version",
    In = ParameterLocation.Path, Required = true,
    Type = typeof(string), Description = "API version (e.g. v1)")]
[OpenApiParameter("collectionName",
    In = ParameterLocation.Path, Required = true,
    Type = typeof(string), Description = "Collection (e.g. omnibus)")]
[OpenApiRequestBody("application/json", typeof(ProviderEvent))]
public static async Task<IActionResult> Enqueue(
    [HttpTrigger(AuthorizationLevel.Admin, "post", Route = "queue/{vers
ion}/{collectionName}")] HttpRequest req,
    Binder binder,
    string version,
    string collectionName,
    ILogger log)
```


Swagger
Supported by SMARTBEAR
https://nrkdncodaeventstorefunctest.azurewebsites.net/api/openapi/3.0.json
Explore
Servers
https://nrkdncodaeventstorefunctest.azurewebsites.net/api
Authorize
Event
GET /getByKey/{version}/{collectionName}/{partitionKey}/{rowKey}
POST /queue/{version}/{collectionName}
Events
GET /queryById/{version}/{collectionNames}/{id}
GET /queryByTime/{version}/{collectionName}/{fromTime}/{toTime}
Count
GET /queue/{version}/{queueName}/count

Заключение

- Журналы событий позволяют собирать точную информацию о времени и причинах изменения состояний системы
- Современные облачные сервисы содержат все необходимое для вынесения журналов событий в облако
- Возможностей простых и недорогостоящих облачных хранилищ достаточно для ведения журналов событий
- Более дорогостоящие облачные сервисы могут понадобиться при повышенных требованиях к доступности и безопасности данных

Спасибо!

Контактная информация

- @ooobject
- vagif.abilov@mail.com