Вагиф Абилов

Akka Streams
для простых смертных

Консультант в Miles

Работаю с F# и C#

@ooobject
vagif.abilov@mail.com

Демо презентации:
https://github.com/object/AkkaStreamsDemo

У вас уже может быть опыт работы с…

- Akka
  - Akka.NET
  - Akka для JVM/Scala
- Reactive Extensions
  - Rx.NET
  - RxJS
  - RxJava
  - RxScala
  - …
- Reactive Streams

…для понимания этого доклада он необязателен!

“Едва ли не худшее, что вы можете сделать, это заставить людей, не испытывающих боли, принимать ваш аспирин”

Макс Креминski.
“Закрытые двери, головная боль и интеллектуальные нужды”
http://tinyurl.com/AkkaStreamsNdc1

Если Akka Streams – это аспирин,
то что же должно быть болью?

Речь пойдет о потоках данных

… потоках данных в самом общем смысле

- Непрерывные измерения датчика температуры – поток данных
- Сообщения очередей RabbitMQ – поток данных
- Логи системы – поток данных
- Непрерывные измерения тысяч датчиков температуры, поступающие в систему через очереди RabbitMQ и сохраняемые в системных логах – поток данных
- В основе автоматизированного управления производством – поток данных

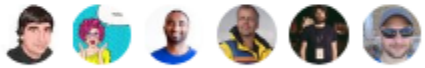
Aaron Stannard
@Aaronontheweb

Following

Looking through some .NET OSS projects for something simple and it's remarkable that none of these projects support any notion of streaming

5:14 PM - 4 Oct 2017

1 Retweet **5** Likes



5 1 5

Вам поможет Akka Streams, если…

- Вы знакомы с Akka, но хотите избавить себя от деталей, связанных с написанием кода акторов и их координацией
- Вы знакомы с Reactive Streams и хотели бы воспользоваться готовой релизацией их спецификации
- Для моделирования вашего процесса подходят блочные элементы стадий Akka Streams
- Вы хотите использовать преимущества обратного давления Akka Streams (backpressure) для управления и динамического уточнения пропускной способности стадий вашего рабочего процесса

От акторов
к Akka Streams

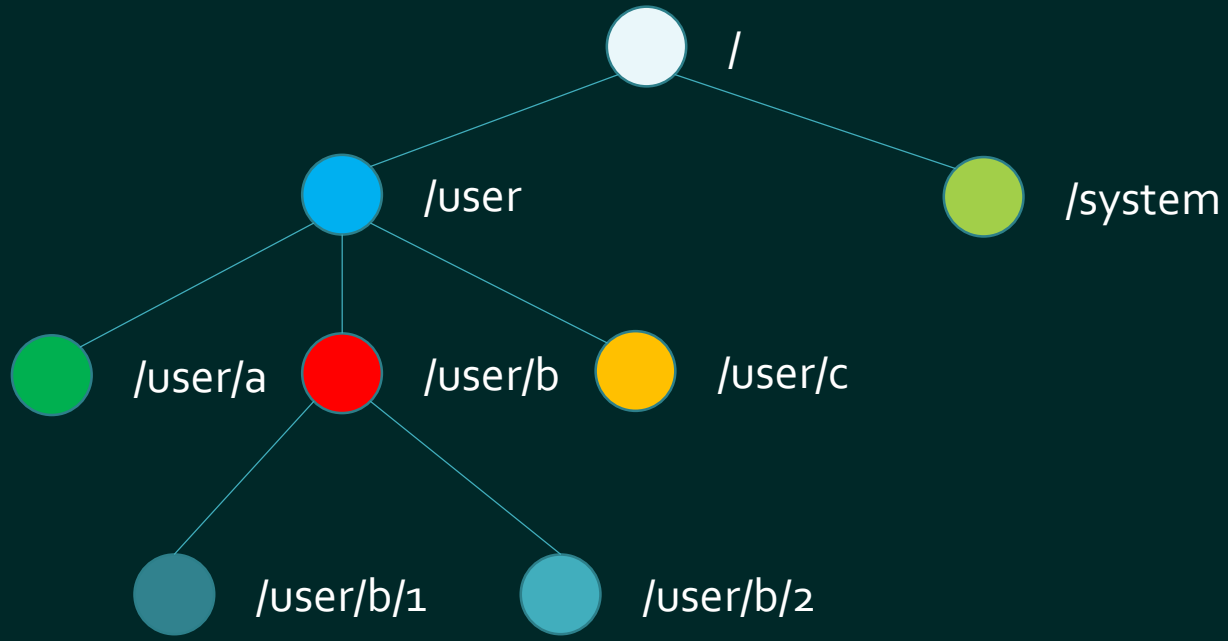

Thread Lock State

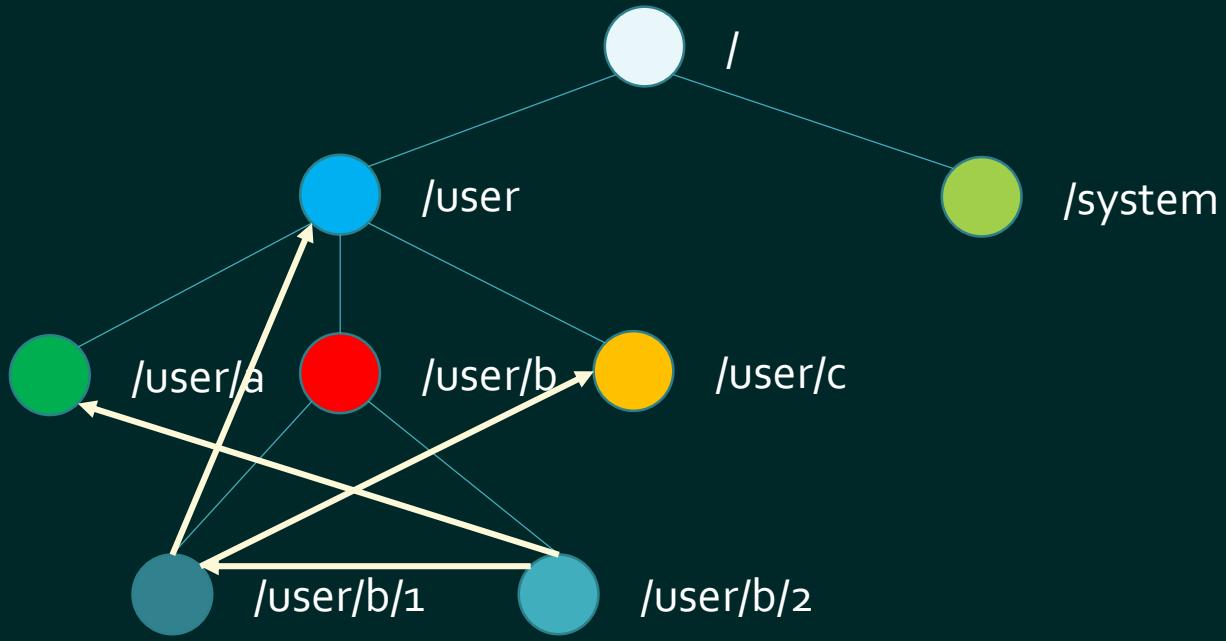
Утверждение: “акторы не компонуются”

“По умолчанию в код акторов вписаны получатели их сообщений. Если я создаю актор A, который посылает сообщение актору B, а вы хотите заменить получателя на актора C, в общем случае у вас это не выйдет.”

Ноэль Уэлш (Underscore.io)

http://tinyurl.com/AkkaStreamsNdc2


/
/user
/system
/user/a
/user/b
/user/c
/user/b/1
/user/b/2


/
/user
/system
/user/a
/user/b
/user/c
/user/b/1
/user/b/2

Утверждение: “акторы не компонуются”

“Если смотреть с лицевой стороны (т.е. глядя на API), акторы – это функции, которые не возвращают ничего: в Akka как tell, так и receive возвращают Unit. Это и является причиной утверждения, что они не компонуются, и смысл его в том, что их нельзя компоновать как обычные функции.”

Роланд Кун (Akka Tech Lead)

http://tinyurl.com/AkkaStreamsNdc3

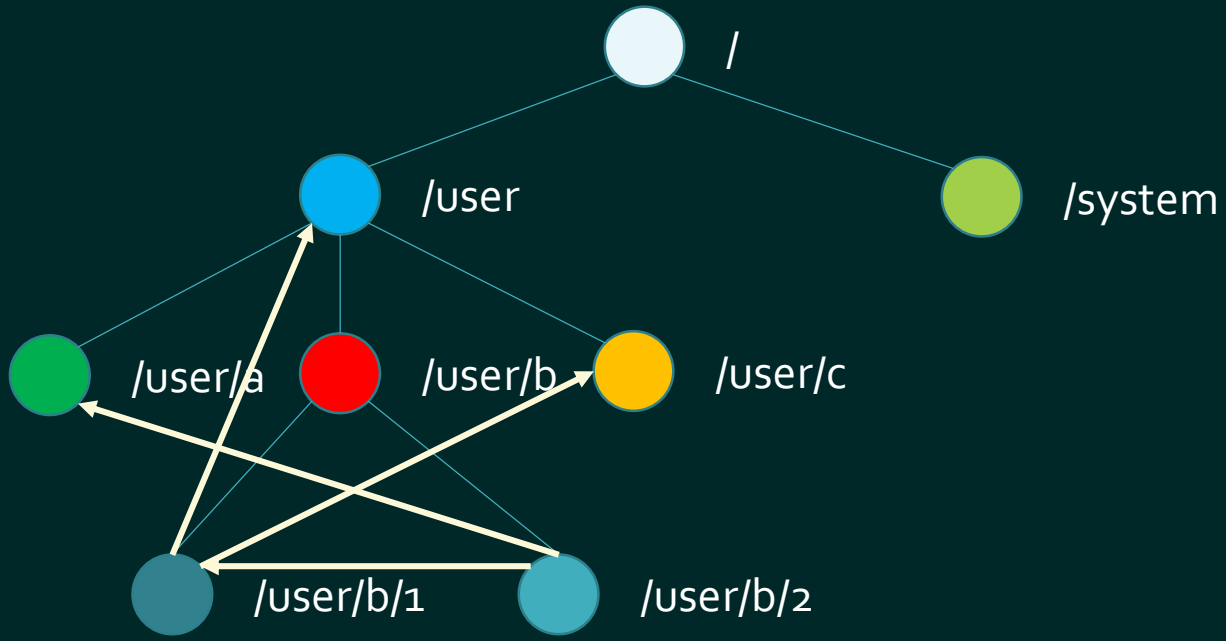
Утверждение: “акторы не компонуются”

“Если смотреть с лицевой стороны (т.е. глядя на API), акторы – это функции, которые не возвращают ничего: в Akka как tell, так и receive возвращают Unit. Это и является причиной утверждения, что они не компонуются, и смысл его в том, что их нельзя компоновать как обычные функции.”

“Но акторы поддаются компоновке в таком же смысле, в каком поддается компоновке человеческое общество.”

Роланд Кун (Akka Tech Lead)

http://tinyurl.com/AkkaStreamsNdc3


/
/user
/system
/user/a
/user/b
/user/c
/user/b/1
/user/b/2

Релизация класса актора на C#

```
public class SampleActor : ReceiveActor
{
    public SampleActor()
    {
        Idle();
    }

    protected override void PreStart() { /* ... */ }

    private void Idle()
    {
        Receive<Job>(job => /* ... */);
    }

    private void Working()
    {
        Receive<Cancel>(job => /* ... */);
    }
}
```

Что если бы мы могли описывать процессы обработки данных на более высоком уровне?

Как в LINQ...

```
var results = db.Companies
                .Join(db.People,
                    c => c.CompanyID,
                    p => p.PersonID,
                    (c, p) => new { c, p })
                .Where(z => z.c.Created >= fromDate)
                .OrderByDescending(z => z.c.Created)
                .Select(z => z.p)
                .ToList();
```

… или серии функциональных трансформаций

```
HttpGet pageUrl
|> fun s -> Regex.Replace(s, "[^A-Za-z’]", " ")
|> fun s -> Regex.Split(s, " +")
|> Set.ofArray
|> Set.filter (fun word -> not (Spellcheck word))
|> Set.iter (fun word -> printfn "   %s" word)
```

Source: https://lorgonblog.wordpress.com/2008/03/30/pipelining-in-f/

Как насчет такого?

```
val in = Source(1 to 10)
val out = Sink.ignore
val bcast = builder.add(Broadcast[Int](2))
val merge = builder.add(Merge[Int](2))
val f1,f2,f3,f4 = Flow[Int].map(_ + 10)
```

```
source ~> f1 ~> bcast ~> f2 ~> merge ~> f3 ~> sink
              bcast ~> f4 ~> merge ~>
```

Источник: http://doc.akka.io/docs/akka/2.4/scala/stream/stream-graphs.html

Давайте разучимся
писать код определения
отдельных акторов

...и выучим вместо этого примитивы компоновки высокого уровня

...которые внутри себя
создадут и соединят
требуемые акторы

Пример на C#

```
var runnable =
    Source
        .From(Enumerable.Range(1, 1000))
        .Via(Flow.Create<int>().Select(x => x * 2)
        .To(Sink.ForEach<int>(x => Console.Write(x.ToString));
```

Графический DSL на C#

```
var graph = GraphDsl.Create(builder =>
{
    var bcast = builder.Add(new Broadcast<int>(2));
    var merge = builder.Add(new Merge<int, int>(2));
    var count = Flow.FromFunction(new Func<int, int>(x => 1));
    var sum = Flow.Create<int>().Sum((x, y) => x + y);

    builder.From(bcast.Out(0)).To(merge.In(0));
    builder.From(bcast.Out(1)).Via(count).Via(sum).To(merge.In(1));

    return new FlowShape<int, int>(bcast.In, merge.Out);
});
```

От Reactive Streams к Akka Streams

Что такое Reactive Streams?

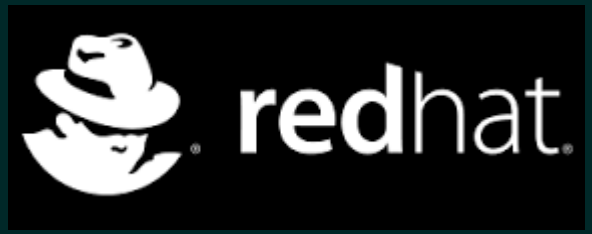
- Инициатива выработки стандарта асинхронной обработки потоков данных
- Определяет минимальный набор интерфейсов, методов и протоколов, описывающие необходимые операции и сущности для достижения цели – асинхроной обработки данных в реальном времени с неблокирующим обратным давлением (back pressure)
- Допускает различные реализации, использующие различные языки программирования

Reactive Streams позволяет

- Обрабатывать потенциально неограниченное количество элементов
- …в последовательности
- Асинхронно передавать элементы между компонентами
- …с неблокирующим обратным давлением

- http://www.reactive-streams.org/
- https://github.com/reactive-streams/

Инициаторы Reactive Streams



Компоненты Reactive Streams API

- Publisher
- Subscriber
- Subscription
- Processor

- Спецификация Reactive Streams не предназначена для использования как API разработчиков приложений

Интерфейсы Reactive Streams

```
public interface IPublisher<out T>
{
    void Subscribe(ISubscriber<T> subscriber);
}

public interface ISubscriber<in T>
{
    void OnSubscribe(ISubscripion subscription);
    void OnNext(T element);
    void OnError(Exception cause);
    void OnComplete();
}
```

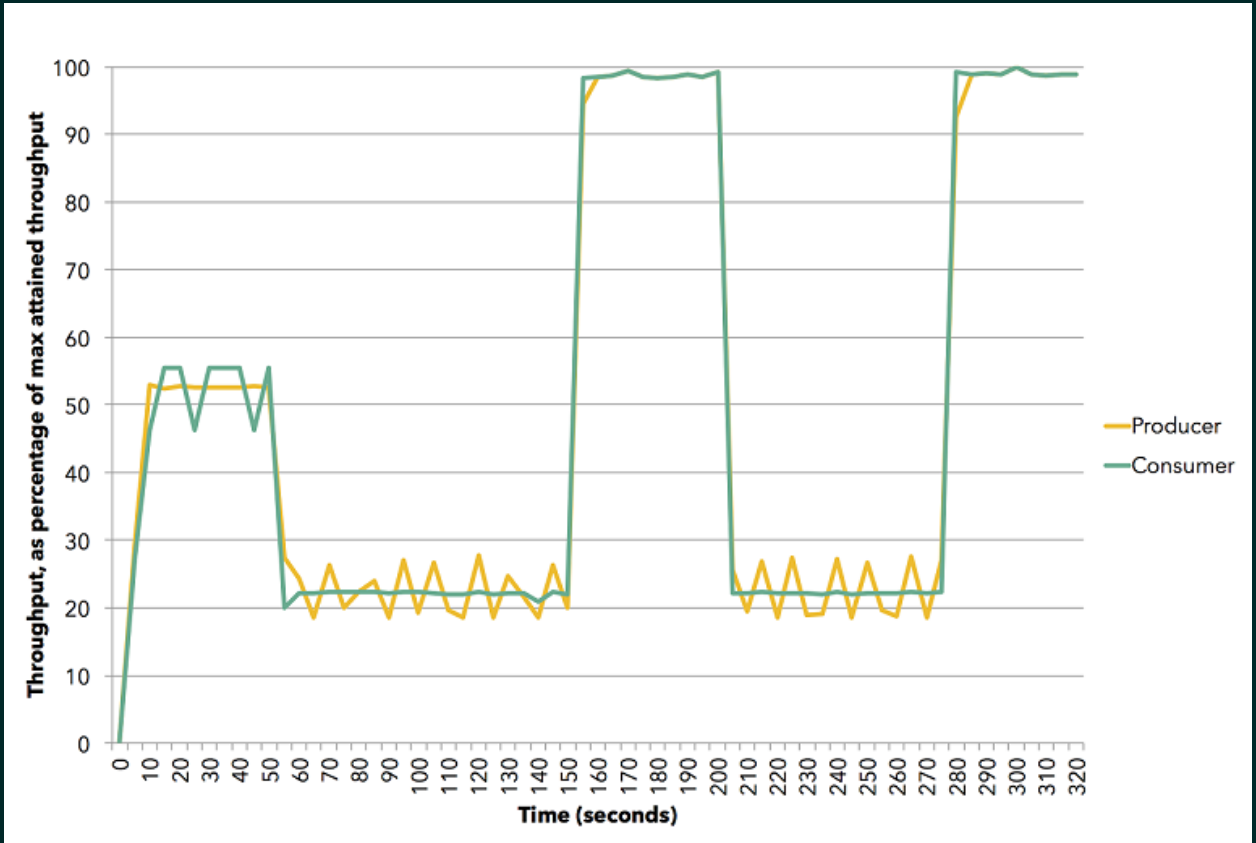
Интерфейсы Reactive Streams

```
public interface ISubscription
{
    void Request(long n);
    void Cancel();
}

public interface IProcessor<in T1, out T2>
               : ISubscriber<T1>, IPublisher<T2>
{
}
```

Динамический push/pull в Reactive Streams

- Reactive Streams комбинируют модели push и pull для поддержки различных сценариев быстродействия
- Только “push” небезопасен для медленных потребителей данных
- Только “pull” неэффективен для медленных публикаторов данных
- Решение: динамическая адаптация



Источник: https://data-artisans.com/blog/how-flink-handles-backpressure (Apache Flink)

Источник (Source)

Стадия обработки с одним выходом

Сток (Sink)

Стадия обработки с одним входом

Пропускной пункт (Flow)

Стадия обработки с одним входом и одним выходом

Строительные блоки Akka Streams

Граф (Graph)

Стадия обработки, построенная из
источников, пропускных пунктов и стоков

Запускаемый граф (Runnable Graph)

Стадия обработки без входов и выходов
(закрытый контур, готовый к запуску)

Встроенные источники (названия согласно Akka.NET)

- FromEnumerator
- From
- Single
- Repeat
- Cycle
- Tick
- FromTask
- Unfold
- UnfoldAsync
- Empty
- Maybe
- Failed
- ActorPublisher
- ActorRef
- Combine
- UnfoldResource
- Queue
- AsSubscriber
- FromPublisher
- ZipN
- ZipWithN
- FromInputStream
- AsOutputStream
- FromFile

Встроенные стоки

- First
- FirstOrDefault
- Last
- LastOrDefault
- Ignore
- Cancelled
- Seq
- Foreach
- ForeachParallel
- OnComplete
- Aggregate
- Sum

- Combine
- ActorRef
- ActorRefWithAck
- ActorSubscriber
- AsPublisher
- FromSubscriber
- FromOutputStream
- AsInputStream
- ToFile

Встроенные пропускные пункты

- Select
- SelectMany
- StatefulSelectMany
- Where
- Collect
- Grouped
- Sliding
- Scan
- Aggregate
- Skip
- SkipWhile
- Take
- TakeWhile
- Recover
- RecoverWith
- Detach
- Throttle
- SelectAsync
- SelectAsyncUnordered
- TakeWithin
- SkipWithin
- GroupedWithin
- InitialDelay
- Delay
- Conflate
- ConflateWithSeed
- Batch
- BatchWeighted
- Expand
- Buffer
- PrefixAndTail
- GroupBy
- SplitWhen
- SplitAfter
- ConcatMany
- MergeMany
- InitialTimeout
- CompletionTimeout
- IdleTimeout
- BackpressureTimeout
- KeepAlive
- InitialDelay
- Merge
- MergeSorted
- Zip
- ZipWith
- ZipWithIndex
- Concat
- Prepend
- OrElse
- Interleave
- Unzip
- UnzipWith
- Broadcast
- Balance
- WatchTermination
- Monitor
- InitialTimeout

Материализация потоков

- Декларация потока является его исполнительным планом
- Исполнение начинается с материализации потока
- Материализовываться могут только запускаемые графы
- Несколько стадий потока могут быть при материализации исполняться одним актором (operator fusion) – Akka Streams 2.0
- Пропускные пункты, как правило, сохраняют очередность обработки элементов

Сравните с LINQ-выражениями

```
var results = db.Companies
                .Join(db.People,
                    c => c.CompanyID,
                    p => p.PersonID,
                    (c, p) => new { c, p })
                .Where(z => z.c.Created >= fromDate)
                .OrderByDescending(z => z.c.Created)
                .Select(z => z.p)
                .ToList();
```

Материализация потока

```
var runnable =
    Source
        .From(Enumerable.Range(1, 1000))
        .Via(Flow.Create<int>().Select(x => x * 2)
        .To(Sink.ForEach<int>(x => Console.Write(x.ToString));
```

Материализация потока

```
var runnable =
    Source
        .From(Enumerable.Range(1, 1000))
        .Via(Flow.Create<int>().Select(x => x * 2)
        .To(Sink.ForEach<int>(x => Console.Write(x.ToString));

var system = ActorSystem.Create("MyActorSystem");
using (var materializer = ActorMaterializer.Create(system))
{
    await runnable.Run(materializer);
}
```

Материализованные значения

```
var output = new List<int>();
var source1 = Source.From(Enumerable.Range(1, 1000));
var sink1 = Sink.ForEach<int>(output.Add);
IRunnableGraph<NotUsed> runnable1 = source1.To(sink1);
```

```
var source2 = Source.From(Enumerable.Range(1, 1000));
var sink2 = Sink.Sum<int>((x,y) => x + y);
IRunnableGraph<Task<int>> runnable2 =
            source2.ToMaterialized(sink2, Keep.Right);
```

Передача данных из потоков Akka Streams

- Встроенные стадии источников содержат широкий спектр реактивных потоков данных
  - Source.FromEnumerator и Source.From позволяют передавать данные из любого источника, реализующего IEnumerable
  - Unfold и UnfoldAsync формируют результаты вычислений функции при условии возврата ею ненулевых значений
  - FromInputStream преобразовывает Stream
  - FromFile обращает в реактивный поток содержимое файла
  - ActorPublisher преобразовывает сообщения актора
- Akka Persistence Query формирует поток содержимого Akka Event Journal
- Пользуйтесь коннекторами Alpakka (большое число коннекторов для Scala, несколько для.NET)

Коннекторы Alpakka (Scala/JVM)

- AMQP
- AWS (DynamoDB, Lambda, S3, SNS, SQS)
- Azure Storage Queue
- Cassandra
- FTP
- TCP
- Kafka
- Google Cloud Pub/Sub
- Hbase
- JMS
- Others

Коннекторы Alpakka (.NET)

- Azure
  - Storage Queue
  - EventHub
  - ServiceBus
- SignalR
- Text
  - CSV
  - XML
- AMQP (в процессе разработки)
- Kafka (в процессе разработки)

Альтернативы (RX, TPL DataFlow, Orleans Streams, …)

- Наиболее близким к Akka Streams по функционалу является Task Parallel DataFlow (TPL DataFlow или TDF)
- TPL DataFlow содержит аналогичные примитивы для компоновки потоков данных, так же как и средства буферизации и ограничения пропускной способности (BoundedCapacity, MaxMessagePerTask)
- Orleans Streams органично дополняют виртуальные акторы Microsoft Orleans функционалом потоков данных

Пример реализации
Поток журнала событий

Dashboard / Prod Oddjob Events

Add Save Share Options Last 15 minutes

*

Media Source

MDB PS

Storage Provider

Akamai Origin

Completed Events

23

Completed

Failed Events

7

Failed

Rejected Events

0

Rejected

Final Events

Completed Failed

Error Codes

5000

Prod Oddjob Persistence - All Events

1 2 3 4 »

Time	eventType	event.fileGroup	event.filePart	event.fileName	event.state	event.resultCode	event.mediaSource	event.storageProvider	event.sourcePath
June 10th 2017, 15:43:13.566	FileDistribution	a2b1be8397e54e87b1ba8397e58e879d		a2b1be83_205.mp4	Completed	0	MDB	Akamai	-
June 10th 2017, 15:43:13.238	FileDistribution	a2b1be8397e54e87b1ba8397e58e879d		a2b1be83_205.mp4	Initiated	0	MDB	Akamai	-
June 10th 2017, 15:43:13.238	FileDistribution	a2b1be8397e54e87b1ba8397e58e879d		a2b1be83_205.mp4	Requested	0	MDB	Akamai	\\manas01\Media_share\Potion\Archive\OnDemand\Prod\a2\a2b1be83-97e5-4e87-b1ba-8397e58e879d\20170610154157O4\london-politiet-tar-terrorist-636327060967780171_205.mp4
June 10th 2017, 15:43:13.160	FileGroup	-	-	-	-	-	-	-	-
June 10th 2017, 15:43:13.160	FileDistribution	a2b1be8397e54e87b1ba8397e58e879d		a2b1be83_205.mp4	Requested	0	MDB	Akamai	\\manas01\Media_share\Potion\Archive\OnDemand\Prod\a2\a2b1be83-97e5-4e87-b1ba-8397e58e879d\20170610154157O4\london-politiet-tar-terrorist-636327060967780171_205.mp4
June 10th 2017, 15:43:06.377	FileDistribution	a2b1be8397e54e87b1ba8397e58e879d		a2b1be83_2410.mp4	Completed	0	MDB	Akamai	-
June 10th 2017, 15:43:05.111	FileDistribution	a2b1be8397e54e87b1ba8397e58e879d		a2b1be83_2410.mp4	Initiated	0	MDB	Akamai	-
June 10th 2017, 15:43:04.721	FileDistribution	a2b1be8397e54e87b1ba8397e58e879d		a2b1be83_1394.mp4	Completed	0	MDB	Akamai	-
June 10th 2017, 15:43:04.299	FileDistribution	a2b1be8397e54e87b1ba8397e58e879d		a2b1be83_2410.mp4	Requested	0	MDB	Akamai	\\manas01\Media_share\Potion\Archive\OnDemand\Prod\a2\a2b1be83-97e5-4e87-b1ba-8397e58e879d\20170610154157O4\london-politiet-tar-terrorist-636327060967780171_2410.mp4
June 10th 2017, 15:43:04.143	FileGroup	-	-	-	-	-	-	-	-
June 10th 2017, 15:43:04.143	FileDistribution	a2b1be8397e54e87b1ba8397e58e879d		a2b1be83_2410.mp4	Requested	0	MDB	Akamai	\\manas01\Media_share\Potion\Archive\OnDemand\Prod\a2\a2b1be83-97e5-4e87-b1be-

Панель текущих операций

- Панель Kibana отображает данные индекса Elasticsearch
- Индекс уничтожается и создается заново при пересмотре структуры данных
- Индекс пополняется содержимым журнала событий (event journal) Akka
- Журнал событий хранится базе данных Microsoft SQL Server
- И ранее сохраненные события, и события реального времени должны отбражаться на панели текущих операций

Схема журнала событий Akka для 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)
)
```

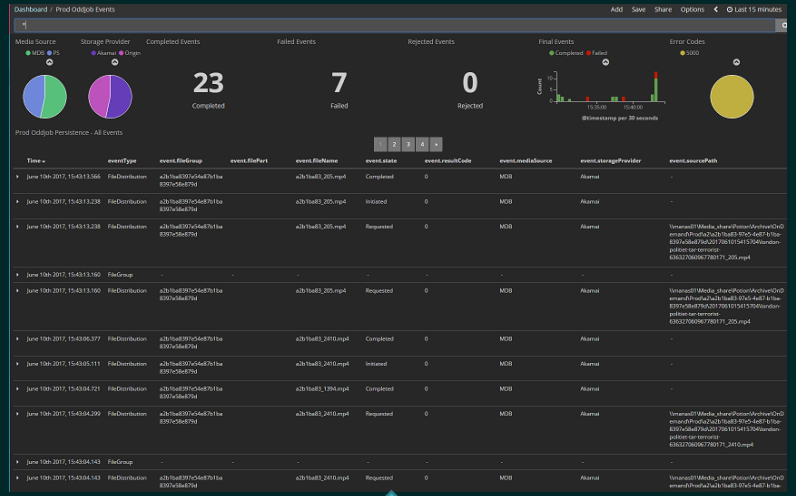
Actor system

Persistent actors

Event Journal

Indexer

Elastic Search



Actor system

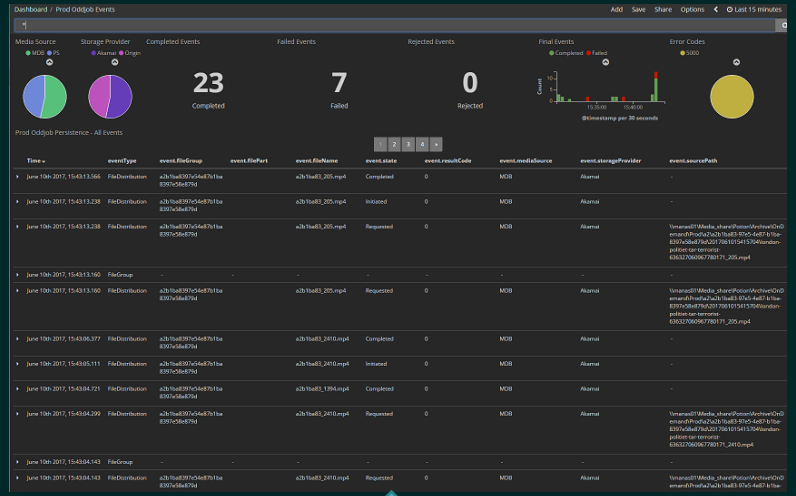
Persistent actors

Persistence query

Event Journal

Indexer

Elastic Search



Встроенные запросы (persistence queries)

- AllPersistenceIds
- CurrentPersistenceIds
- EventsByPersistenceId
- CurrentEventsByPersistenceId
- EventsByTag
- CurrentEventsByTag

Реализация пополнения индекса на F#

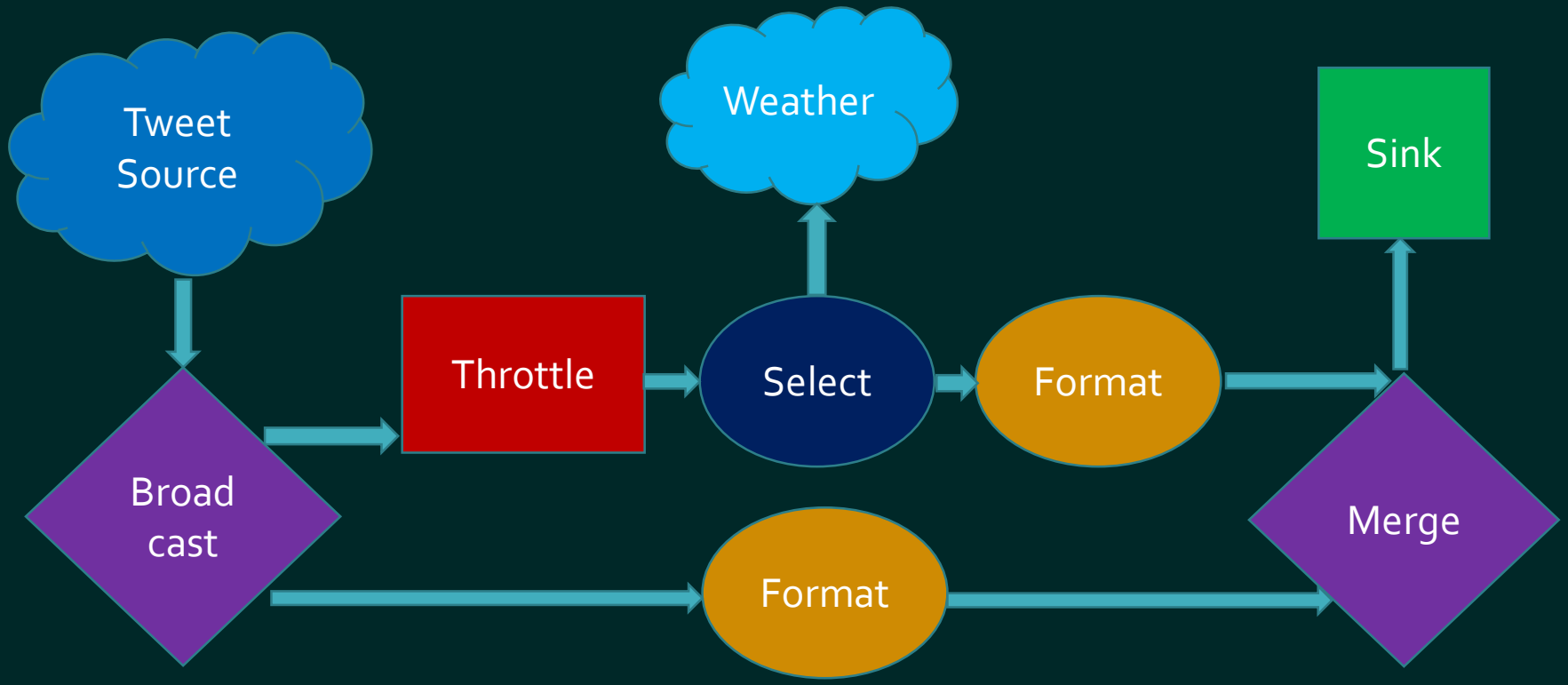
```
let system = mailbox.Context.System
let queries = PersistenceQuery.Get(system)
    .ReadJournalFor<SqlReadJournal>(SqlReadJournal.Identifier)
let mat = ActorMaterializer.Create(system)
let offset = getCurrentOffset client config
let ks = KillSwitches.Shared "persistence-elastic“

let task =
    queries.EventsByTag(PersistenceUtils.anyEventTag, offset)
        .Select(fun e -> ElasticTypes.EventEnvelope.FromAkka e)
        .GroupedWithin(config.BatchSize, config.BatchTimeout)
        .Via(ks.Flow())
        .RunForeach((fun batch -> processItems client batch), mat)
        .ContinueWith(handleStreamError mailbox,
            TaskContinuationOptions.OnlyOnFaulted)
    |> Async.AwaitTaskVoid
```

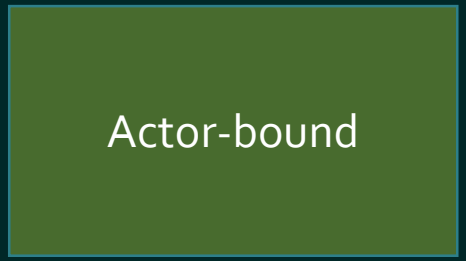
Пример реализации
Реактивные твиты

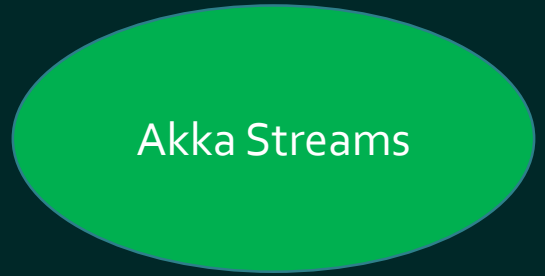
Реактивные твиты

- Как Akka (Scala/JVM), так и Akka.NET предлагают примеры реализации потока твитов Twitter
- Пример Akka.NET основан на Tweetinvi (https://github.com/linvi/tweetinvi)
- Tweetinvi определяет несколько потоков для выдачи данных Twitter, но не поддерживает Reactive Streams
  - IFilteredStream
  - ISampleStream
  - ITrackedStream
  - ITweetStream
  - IUserStream
- Reactive Tweets использует встроенный примитив Source.ActorRef для пересылки элементов из потока Tweetinvi в источник Akka Stream


Tweet Source
Weather
Sink
Throttle
Select
Format
Broad cast
Merge
Format

Заключительные соображения





Спасибо!

- Консультант в норвежской компании Miles
- Электронная почта: vagif.abilov@gmail.com
- Twitter: @ooobject
- GitHub: object
- Блог: http://vagifabilov.wordpress.com/
- Код демо: https://github.com/object/AkkaStreamsDemo