ALTENAR

Оптимизировать или обойти

Ускорение MongoDB сериализации

Алексей Троепольский

Алексей Троепольский

12+ years in .NET

Knows about high load

Разработчик в Altenar

YOUR IGAMING
GUIDE

BOOK A MEETING

SiGMA
EUROPE

YOUR LIFELONG
IGAMING
PARTNER

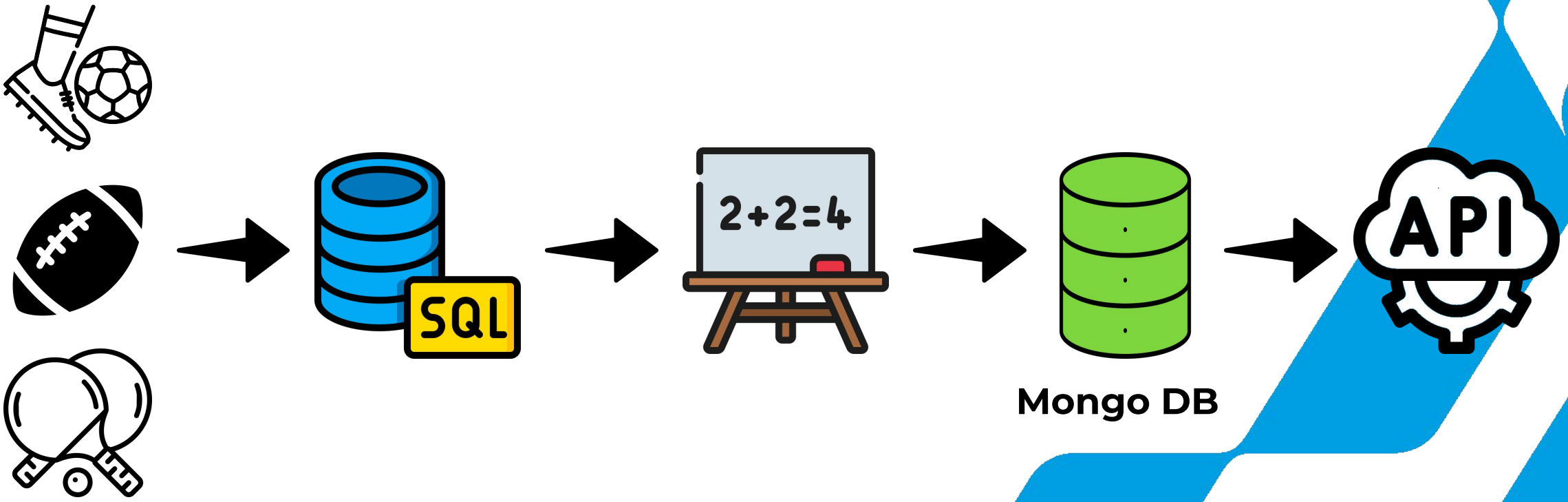
SiGMA
EUROPE

BOOK A
MEETING

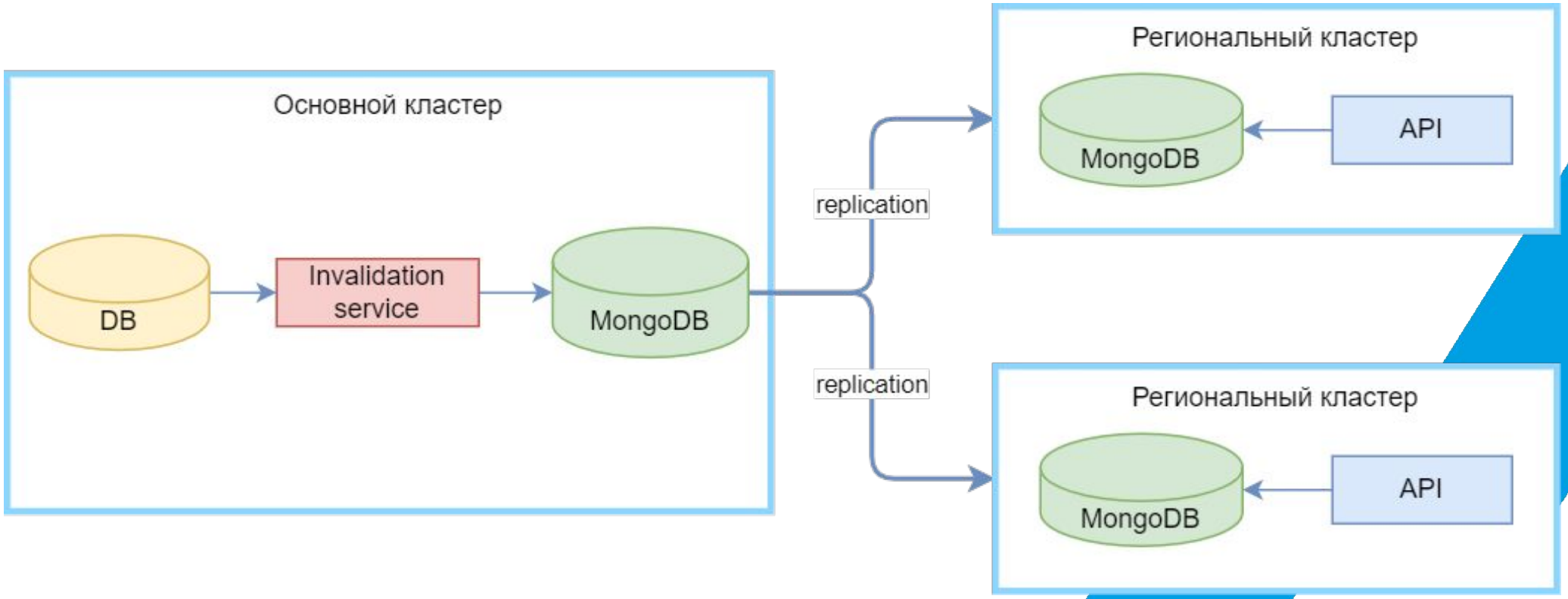
ALTENAR

YOUR
LIFELONG
IGAMING
PARTNER

От сырых данных до браузера



Как мы готовим данные



Как выглядит проблема

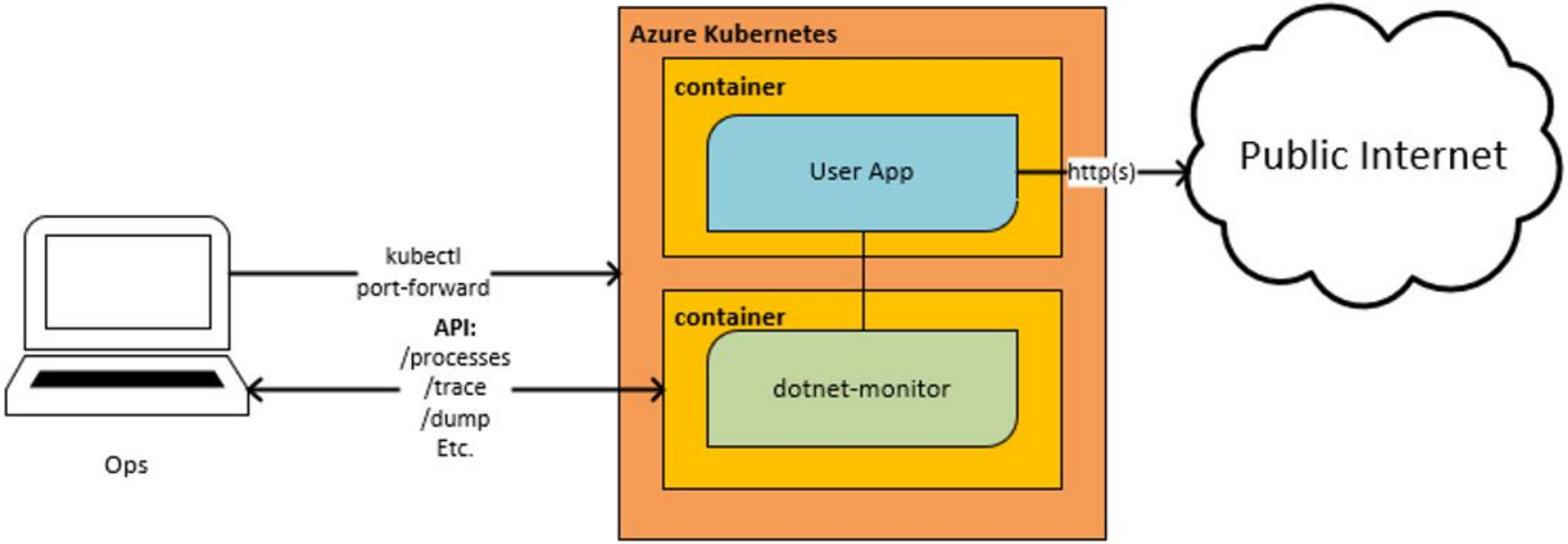
- 10 000 RPS
- 50 подов в каждом кластере
- 5 CPU, 10 GB RAM в каждом поде

Во всем виноват Снежок!

- По любому Linq!
- У нас его много
- Все знают что Linq медленный

Снимаем трейс

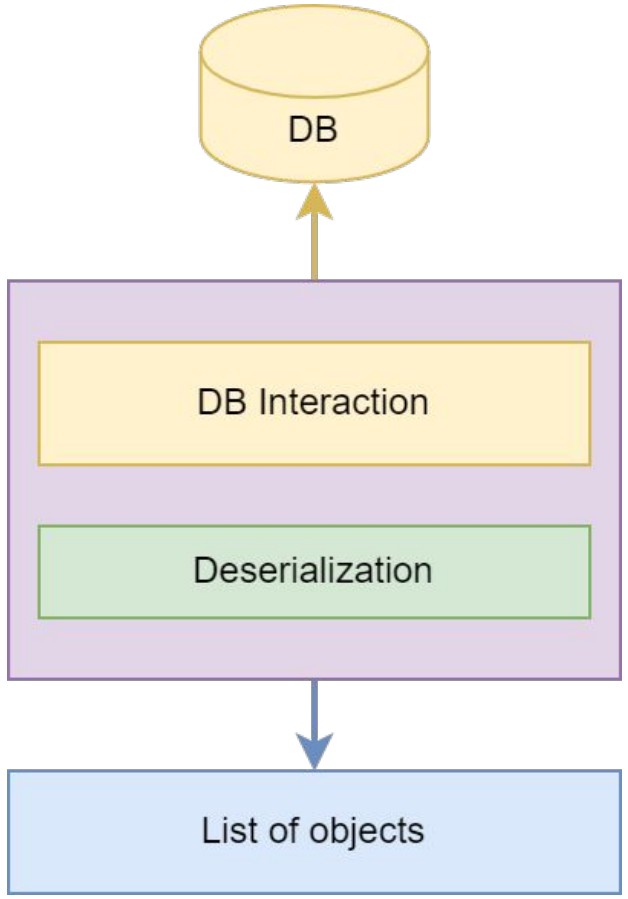
80% времени - десериализация



BSON

```
{
   "_id": 123456,
   "Status": 1,
   "EventId": 9154208,
   "Selections": [
      {
         "Id": 1653809987,
         "Name": "Mlakar",
         "Price": 2.0
      }]
}
```

Как менять драйвер




```
var col = GetCollection<Market>();
var markets  = await col.ToListAsync(filter);
```

Есть дефолтные сериализаторы

Возможность зарегистрировать собственные

Как делать бенчмарки

```
[MemoryDiagnoser]
public class ListDocumentsFromBson
 {
      static readonly byte[] _bsonArr = File.ReadAllBytes("Bson.txt");


      [Benchmark(Baseline = true)]
      public ArrWrapper Old()
      {
            var result= BsonSerializer.Deserialize<ArrWrapper>(_bsonArr);
            return result;
      }
}
```

Как делать бенчмарки

```
[GlobalSetup(Target = nameof(Optimized))]
public void InitOptimized()
{
    RegistrationHelper.Register(typeof(Market).Assembly.GetTypes()
        .Where(t => t.Namespace.StartsWith("MyNs")));

}
[Benchmark]
public ArrWrapper Optimized()
{
    var result = BsonSerializer.Deserialize<ArrWrapper>(_bsonArr);
    return result;
}
```

Убираем ненужное

```
public class BsonClassMapSerializer<TClass>
: SerializerBase<TClass>
```

Удалено **305 из 394** строки

Неотключаемые фичи:

Полиморфная десериализация

Контроль required полей

Десериализация без дефолтного конструктора

Убираем ненужное Benchmarks

Прирост ~12%

Method	Mean	Error	Ratio	Allocated	Alloc Ratio
Old	51.29 ms	1.002 ms	baseline	37.28 MB	
RemoveUnused	44.98 ms	0.876 ms	-12%	36.69 MB	-2%

Убираем рефлекшен

Рефлекшен- метаданные о типе

Позволяет:

- Получить список полей
- Получить список методов
- Установить значение поля

Работает медленно

Как используется рефлекшен

```
public class BsonMemberMap

private Action<object, object> GetPropertySetter()

(instance, value) => propertyInfo.SetValue(instance, value);
```

Деревья выражений

Сбилдить делегат. Скомпилировать. Закэшировать.

Expression.`Call` - вызвать метод
Expression.`Parameter` - объявить параметр
Expression.`Property`- работа с проперти
Expression.`Assign`- присвоить что-то чему-то
Expression.`Convert`- сконвертировать
Expression.`Lambda`- построить лямбду
Expression.`Compile`- скомпилировать дерево

Строим делегат

```
(obj, serializer, context) =>
{
    BsonDeserializationArgs args = new()
    {
        NominalType = serializer.ValueType
    };
    obj.Property = (TMember)serializer.Deserialize(context, args);
}
```

Две строчки кода - изи пизи!

Строим делегат

```
DeserializeMemberDeligate BuildDeserializeDelegate(){
var contextParameter = Expression.Parameter(typeof(BsonDeserializationContext), "context");
var serializerParameter = Expression.Parameter(typeof(IBsonSerializer), "serializer");
var documentParameter = Expression.Parameter(typeof(TClass).MakeByRefType(), "document");
var serializer = _memberMap.GetSerializer();
var argsVar = Expression.Variable(typeof(BsonDeserializationArgs), "args");
var nominalProperty = Expression.Property(argsVar, nameof(BsonDeserializationArgs.NominalType));
var setArgsNominalType = Expression.Assign(nominalProperty, Expression.Constant(serializer.ValueType));
var methos = typeof(IBsonSerializer).GetMethod("Deserialize", BindingFlags.Public | BindingFlags.Instance)!;
var deserializeMethod = Expression.Call(serializerParameter, methos, contextParameter, argsVar);
var convertedValue = Expression.Convert(deserializeMethod, _memberMap.MemberType);
var property = _memberMap.MemberInfo is FieldInfo fieldInfo
? Expression.Field(documentParameter, fieldInfo)
: Expression.Property(documentParameter, (PropertyInfo)_memberMap.MemberInfo);
var setProperty = Expression.Assign(property, convertedValue);
....
```

Строим делегат

```
(obj, serializer, context) =>
{
    BsonDeserializationArgs args = new()
    {
        NominalType = serializer.ValueType
    };
    obj.Property = (TMember)serializer.Deserialize(context, args);
}
```

Убираем рефлекшен. Benchmarks

Прирост +7% (Всего +19%)

Method	Mean	Error	Ratio	Allocated	Alloc Ratio
Old	51.29 ms	1.002 ms	baseline	37.28 MB	
RemoveUnused	44.98 ms	0.876 ms	-12%	36.69 MB	-2%
RemoveReflection	41.40 ms	0.794 ms	-19%	36.69 MB	-2%

Что можно улучшить

```
obj.Property = (TMember)serializer.Deserialize(context, args);
```

```
public interface IBsonSerializer
{
    object Deserialize(BsonDeserializationContext context, BsonDeserializationArgs args);
}
```

```
TValue Deserialize(BsonDeserializationContext context, BsonDeserializationArgs args)
```

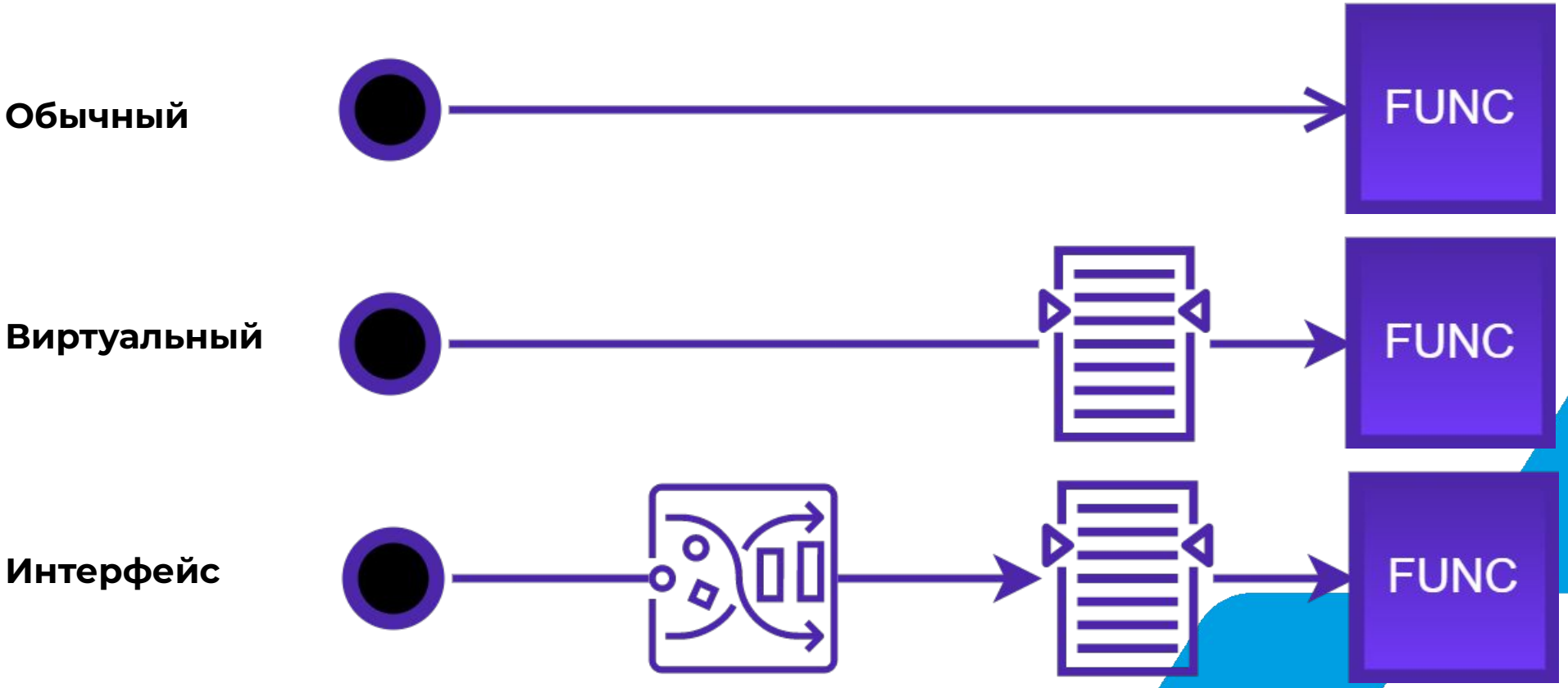
Что можно улучшить

Serializer известен и не меняется

```
obj.Property = (TValue)serializer.Deserialize(ctx, args);
```

```
obj.Property = ((SomeSerializer)serializer).Deserialize(ctx,args);
```

Вызовы методов



Девиртуализация

```
(TValue)serializer.Deserialize(ctx, args);
```

```
((SomeSerializer)serializer).Deserialize(ctx,args);
```

```
class SomeSerializer<TValue> : SerializerBase<TValue>
```

```
sealed class SomeSerializer<TValue> : SerializerBase<TValue>
```

Убираем боксинг. Benchmarks

Прирост скорости +7% (Всего +26%)

Аллокаций меньше на 10%

Method	Mean	Error	Ratio	Allocated	Alloc Ratio
Old	51.29 ms	1.002 ms	baseline	37.28 MB	
RemoveUnused	44.98 ms	0.876 ms	-12%	36.69 MB	-2%
RemoveReflection	41.40 ms	0.794 ms	-19%	36.69 MB	-2%
RemoveBoxing	38.17 ms	0.755 ms	-26%	33.71 MB	-10%

Десериализация коллекций

```
var accumulator = CreateAccumulator();

while (bsonReader.ReadBsonType() != BsonType.EndOfDocument)
{
    var item = _itemSerializer.Value.Deserialize(context);
    AddItem(accumulator, item);
}

return FinalizeResult(accumulator);
```

Десериализация коллекций

```
protected override TValue FinalizeResult(object accumulator){
    var accumulatorType = accumulator.GetType();
    foreach (var constructorInfo in typeof(TValue).GetTypeInfo().GetConstructors())
    {
        var parameterInfos = constructorInfo.GetParameters();
        if (parameterInfos.Length == 1 &&
        parameterInfos[0].ParameterType.GetTypeInfo()
        .IsAssignableFrom(accumulatorType))
        {
            return (TValue)constructorInfo.Invoke(new object[] { accumulator });
        }
    }
...
```

Переделаем FinalizeResult

```
Func<object, TValue>? _finalizeResult = null;

protected override TValue FinalizeResult(object accumulator)
{
    if (_finalizeResult == null)
        _finalizeResult = BuildFinalizeResultAction(accumulator);
    return _finalizeResult(accumulator);
}
```

Генерируем делегат

```
Func<object, TValue> BuildFinalizeResultAction(object accumulator)
{
    // if it the same type we can just convert and return
    var accumulatorType = accumulator.GetType();
    if (typeof(TValue).IsAssignableFrom(accumulatorType))
        return (accumulator) => (TValue)accumulator;

    .....
```

Десериализация коллекций

...ищем подходящий конструктор через рефлекшен...

```
var accumulator = Expression.Parameter(typeof(object), "accumulator");

var converted = Expression.ConvertChecked(accumulator, accumulatorType);


return Expression.Lambda<Func<object, TValue>>(
    body: Expression.New(constructor, converted)
    ,parameters: accumulator
).Compile();
```

Десериализация коллекций. Benchmarks

Прирост скорости +7% (Всего +33%)

Аллокаций меньше на 12% (Всего на 22%)

Method	Mean	Error	Ratio	Allocated	Alloc Ratio
Old	51.29 ms	1.002 ms	baseline	37.28 MB	
RemoveUnused	44.98 ms	0.876 ms	-12%	36.69 MB	-2%
RemoveReflection	41.40 ms	0.794 ms	-19%	36.69 MB	-2%
RemoveBoxing	38.17 ms	0.755 ms	-26%	33.71 MB	-10%
Enumerable	34.36 ms	0.653 ms	-33%	29.1 MB	-22%

Проблема с ресайзами

```
var accumulator = CreateAccumulator(); -> new List<TValue>()
while (bsonReader.ReadBsonType() != BsonType.EndOfDocument)
{
    var item = _itemSerializer.Value.Deserialize(context);
    AddItem(accumulator, item); -> resize
}
return FinalizeResult(accumulator);
```

Используем пулы

```
DefaultObjectPool<List<TValue>> _listPool = new(200);

var list = _listPool.Get();
try
{
    ... Заполняем список из BSON ...
    return new List<TValue>(list);
}
finally
{
    list.Clear();
    _listPool.Return(list);
}
```

Побеждаем ресайзы. Benchmarks

Прирост скорости +4% (Всего +37%)

Аллокаций меньше на 8% (Всего на 30%)

Method	Mean	Error	Ratio	Allocated	Alloc Ratio
Old	51.29 ms	1.002 ms	baseline	37.28 MB	
RemoveUnused	44.98 ms	0.876 ms	-12%	36.69 MB	-2%
RemoveReflection	41.40 ms	0.794 ms	-19%	36.69 MB	-2%
RemoveBoxing	38.17 ms	0.755 ms	-26%	33.71 MB	-10%
Enumerable	34.36 ms	0.653 ms	-33%	29.1 MB	-22%
Final	32.20 ms	0.421 ms	-37%	26.14 MB	-30%

Можно ли лучше?

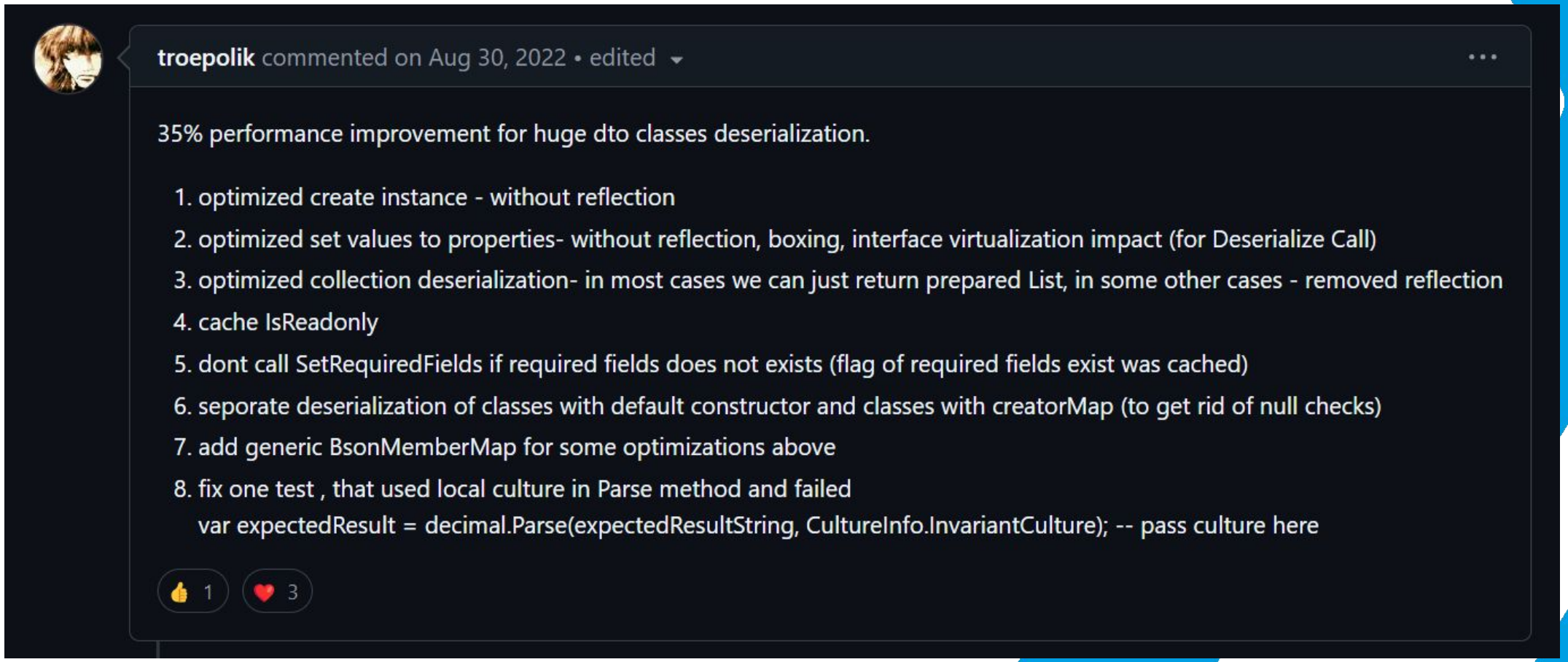
Можно, но придется форкать весь репозиторий.

Можно переписать на сорс генераторах.

Результаты на разных .NET

Method	Runtime	Mean	Ratio	Allocated
Final	.NET 6.0	51.80 ms	+72%	26.23 MB
Final	.NET 7.0	49.27 ms	+63%	26.14 MB
Final	.NET 8.0	30.16 ms	baseline	26.14 MB
Final	.NET 9.0	35.99 ms	+19%	26.03 MB

Где реквесты, Лебовски?



troepolik commented on Aug 30, 2022 • edited

35% performance improvement for huge dto classes deserialization.

1. optimized create instance - without reflection
2. optimized set values to properties- without reflection, boxing, interface virtualization impact (for Deserialize Call)
3. optimized collection deserialization- in most cases we can just return prepared List, in some other cases - removed reflection
4. cache IsReadonly
5. dont call SetRequiredFields if required fields does not exists (flag of required fields exist was cached)
6. seporate deserialization of classes with default constructor and classes with creatorMap (to get rid of null checks)
7. add generic BsonMemberMap for some optimizations above
8. fix one test , that used local culture in Parse method and failed
   var expectedResult = decimal.Parse(expectedResultString, CultureInfo.InvariantCulture); -- pass culture here

👍 1 ❤️ 3

Где реквесты, Лебовски?

DmitryLukyanov commented on Aug 30, 2022 — Contributor

Thank you for your PR. In order to consider accepting this PR, we require filing a CSHARP ticket in our JIRA project clearly explaining the problem along with a repro. Be sure to link to your PR.

BorisDog commented on Oct 17, 2022 — Contributor

Thank you for your research and the PR.

Closing this PR as we are going to consider those great ideas as part of a general serialization improvement effort CSHARP-3230.

Где реквесты, Лебовски?

Deanna Delapasse added a comment - Feb 07 2024 01:32:55 PM GMT+0000

This issue is severely impacting my team's project. We just paid $10k for a consultant to help us and the poor performance of the driver's C# deserialization was found to be the root cause of all of our "pain points". I see that many of the epic's tickets are already closed. Could you please release what you have in the hopes that it will offer some improvement?

We're reducing our projections when possible and minimizing our objects, but still giving us a very disappointing experience when compared to the pure query in the console (ie deserialization is 10x slower than the query itself).

Промежуточные итоги

Количество подов снизилось в 2 раза

Заслужил уважение коллег

Представление об авторах перевернулось

Развитие ставит новые вызовы

40% хорошо, но не кардинально.

Через год количество данных выросло.

В трейсе снова десериализация.

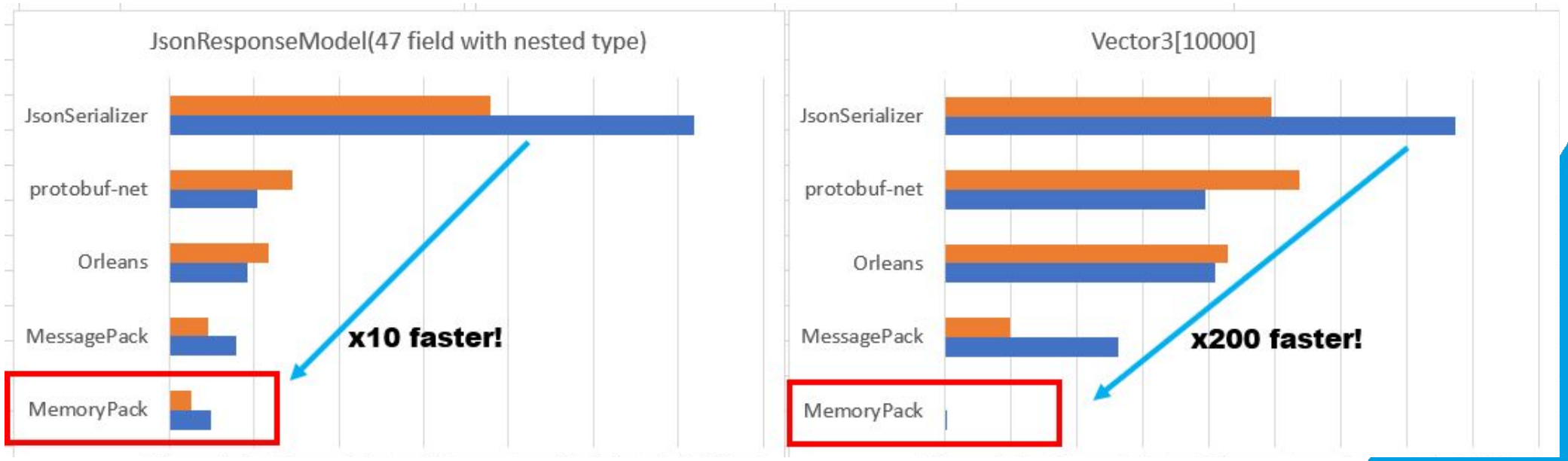
Может отказаться от BSON?

BSON - часть протокола работы с MongoDB.

Если не BSON, то что?

Memory Pack

Zero encoding extreme performance binary serializer for C#



Немного об авторе

Yoshifumi Kawai

MessagePack

ZeroFormatter

Utf8Json

MemoryPack

Как это использовать

```
{
    " id": 123456,
    "Status": 1,
    "EventId": 9154208,
    "Selections": [
        {
            "Id": 1653809987,
            "Name": "Mlakar, Jan",
            "Result": 0
        }],
    "MemPacked": [1,1,0,0,0,1,1…]
}
```

Как это использовать

```
Builders<Market>.Projection.Include(x => x.MemPacked)


public class Mempacked
{
    public byte[]? MemPacked { get; set; }
}
```

Как это использовать

```
var mempackedMarkets = await GetCollection().ToListAsync(filter, projection);

List<Model> result = new(mempackedMarkets.Count);


foreach (var item in mempackedMarkets)
{
    var dto= MemoryPackSerializer.Deserialize<Model>(item.MemPacked);
    result.Add(dto);
}
return result;
```

MemoryPack. Benchmarks

В 10 раз быстрее дефолтного десериализатора

В 6 раз быстрее оптимизированного

В 2-3 раза меньше аллокаций

Method	Mean	Error	Median	Ratio	Allocated	Alloc Ratio
Old	55.437 ms	1.0943 ms	54.429 ms	baseline	37.28 MB	
Final	33.141 ms	0.6530 ms	32.987 ms	-41%	26.14 MB	-30%
MemPack	5.504 ms	0.1036 ms	5.534 ms	-90%	14.27 MB	-62%

Load Test


Requests received per Job
1.80K req/s
1.60K req/s
1.40K req/s
1.20K req/s
1K req/s
800 req/s
600 req/s
400 req/s
200 req/s
0 req/s
10:30
10:35
10:40
10:45
10:50
10:55
11:00
11:05
11:10
11:15
11:20
11:25
11:30

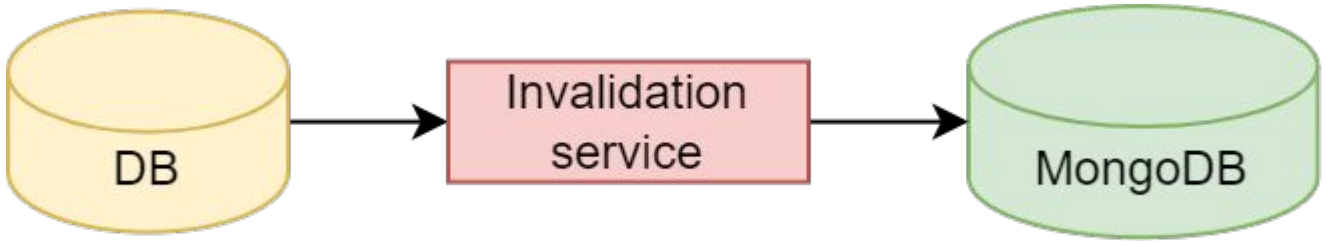
Ограничения

Привязка к языкам C#, TypeScript

Изменение структуры данных

Нельзя делать апдейты полей, только реплейс всего документа

Подход требует дополнительных усилий



Итоги

Количество подов снизилось в 2.5 раза

Появилось время на переработку архитектуры

Спасибо за внимание!

aleksei.troepolskii@altenar.com

troepolik

Алексей Троепольский