Azure Data Explorer

как основная база данных

Артем Манченков

Дисклеймер

О чем поговорим

- Краткий обзор ADX
- Примеры использования
- Задача и решение
- Опыт разработки
- Результаты эксперимента

Что такое ADX?

Kusto

Azure Data Explorer

Полностью управляемая, высокопроизводительная платформа для анализа больших данных, которая позволяет легко анализировать большие объемы данных практически в реальном времени.

Kusto Query Language (KQL)

```
requests
| where timestamp > ago(1d)
| summarize
    percentiles(duration, 50, 90, 95), count()
    by bin(timestamp, 1h)
| order by timestamp asc
```

Kusto

Azure Data Explorer

Пример использования

- Логи
- Метрики
- Снимки данных

Интеграции Azure

Azure Monitor

Application Insights

Azure Sentinel

Azure Log Analytics

Event Hub

Окей, ADX нужен для логов, понял

Задача

Что мы пытались сделать?

Проект X

Нам нужно ежедневно мониторить много сервисов и проверять их готовность по набору правил

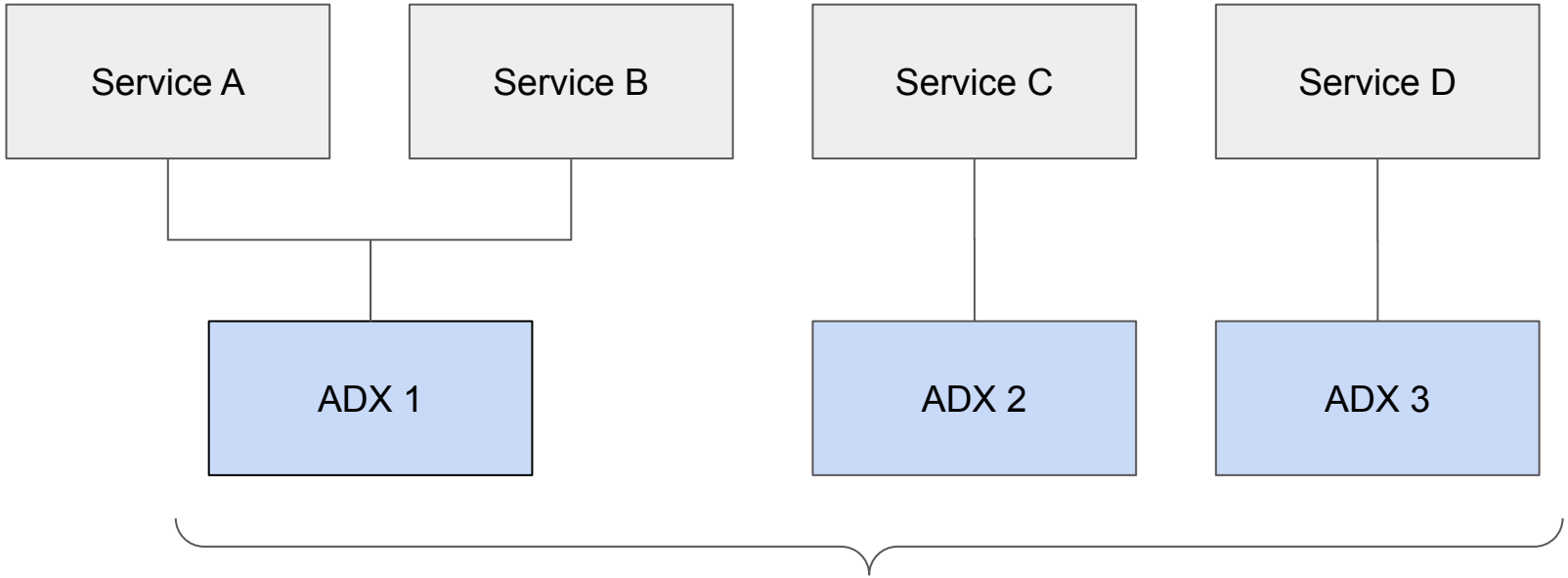
Пример набора правил

1. Безопасность
    a. Использует корректные сертификаты
    b. Настроено автопродление
    c. Аутентификация включена
    d. Авторизация включена
2. Платформа
    a. Используется .NET 8
    b. Безопасный deploy
    c. Настроен мониторинг

Проект X

Примеры логов для проверки в ADX

- Сервис-каталог
- Логи билдов в Azure DevOps
- Релизные метрики
- Runtime-логи с сервисов
- Информация из внутренних сервисов
- Проверки политик Azure
- Отчеты о Compliance
- *и прочее…*



Как данные идут в ADX

Начинаем проект

Требования

- Не дублировать source of truth
- Хранить всю историю изменений
- Сделать UI для клиента
- Подгружать информацию “быстро”

Начинаем проект

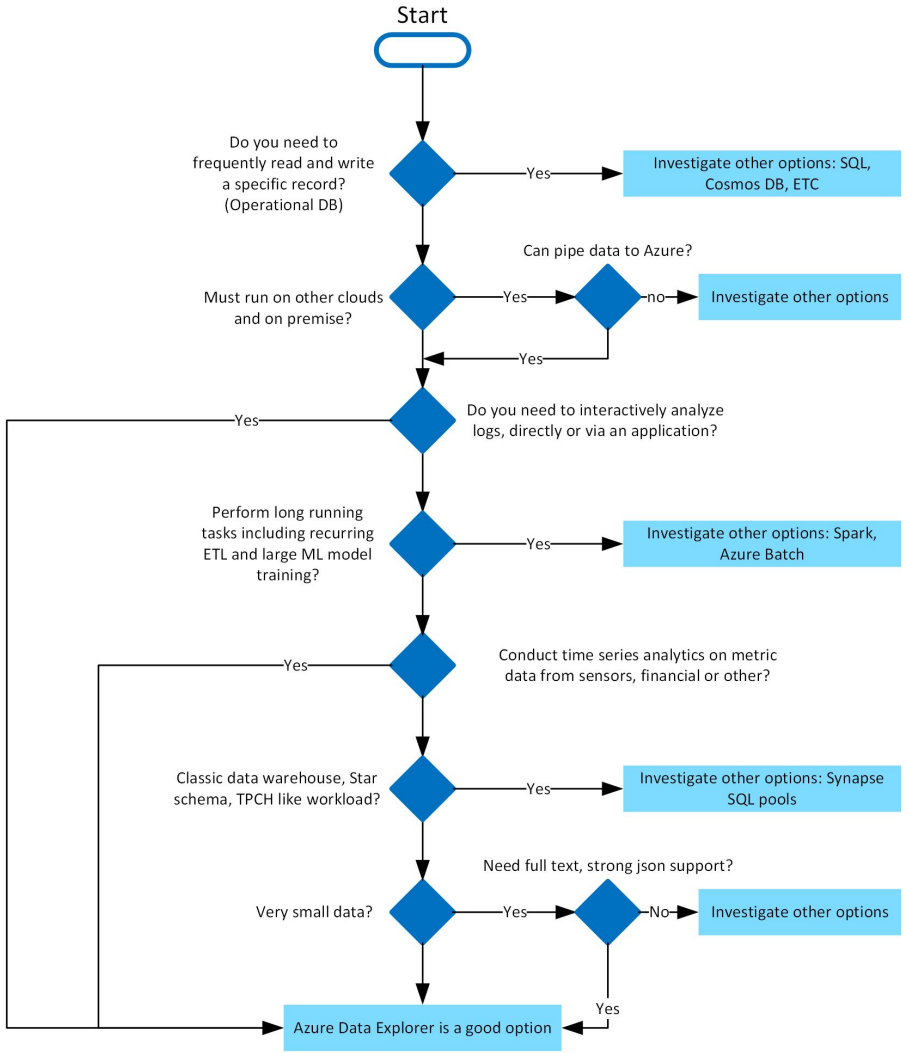
Проблемы

- Разработка множества API для чтения
- Огромный объем данных
- Схема данных пока неизвестна
- Обеспечить real-time запись от клиента

Начинаем проект

Идеи

- Избегать разработки API в пользу чтения ADX
- Держать данные и схему гибкими
- Агрегировать сложные данные в нашем кластере
- Ввод от пользователей через append



Подходит ли нам ADX?

Чтение из разных кластеров

```
cluster(Cluster1).database(Db).Table1
| join
    cluster(Cluster2).database(Db).Table2
    on $left.Id == $right.RefId
```

Почему все-таки ADX?

✅

- Совместимость с источниками
- Cross-cluster запросы
- Динамическая схема данных
- Легкость аналитики
- Пользовательские функции
- Умеренная производительность

🚫

- Нет транзакций
- Нет обновлений строк
- Нет связей
- Нет консистентности
- **Нету Entity Framework**
- Умеренная производительность

Скорость запросов в ADX

```
Check
-- 10 records
-- 1 - 215ms
-- 2 - 114ms
-- 3 - 96ms
```

```
Check
| count
-- 10 records
-- 1 - 211ms
-- 2 - 217ms
-- 3 - 112ms
```

Скорость запросов в ADX

```
Group
| join kind=inner Check on $left.Id == $right.GroupId
-- 10 records
-- 1 - 228ms
-- 2 - 207ms
-- 3 - 118ms
```

Скорость запросов в ADX

```
ServiceCheck_Log
-- 40 100 records
-- 1 - 718ms
-- 2 - 459ms
-- 3 - 337ms
```

```
ServiceCheck_Log
| count
-- 40 100 records
-- 1 - 210ms
-- 2 - 124ms
-- 3 - 212ms
```

Скорость запросов в ADX

```
ServiceCheck_Log
-- 1 300 000 records
-- 1 - 2.718ms
-- 2 - 3.459ms
-- 3 - 2.337ms
```

```
ServiceCheck_Log
| count
-- 1 300 000 records
-- 1 - 210ms
-- 2 - 111ms
-- 3 - 107ms
```

Скорость запросов в ADX

```
Group
| join kind=leftouter Check on $left.Id == $right.GroupId
| join kind=leftouter ServiceCheck_Log on $left.Id1 == $right.CheckId
| summarize
    completed=countif(IsCompliant == true),
    incomplete=countif(IsCompliant == false),
    total=dcount(ServiceId)
    by GroupId, CheckId

-- 40 115 records
-- 1 - 215ms
-- 2 - 228ms
-- 3 - 127ms
```

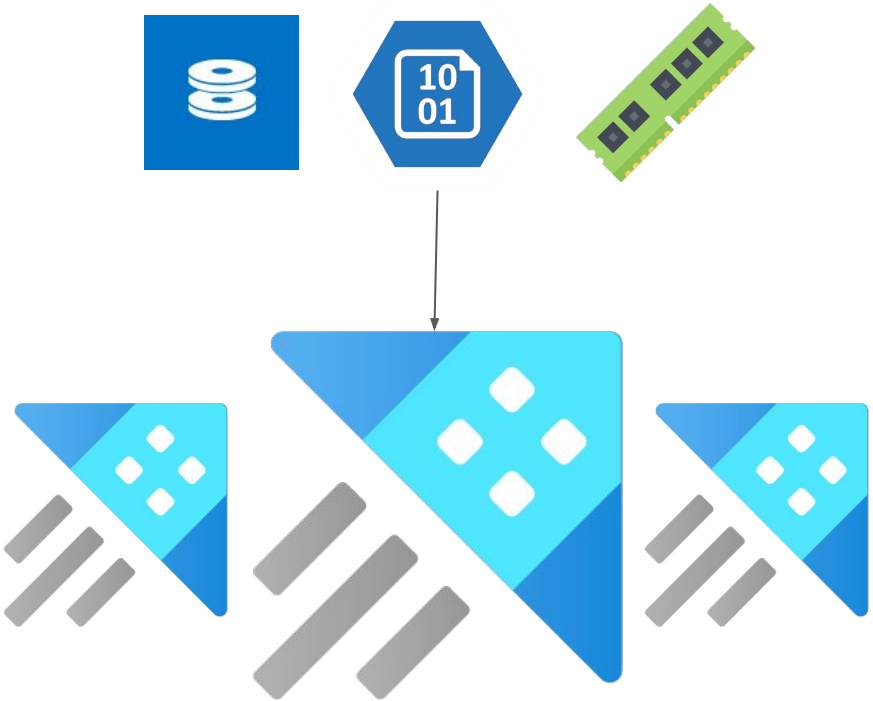
Скорость запросов в ADX

```
Group
| join kind=leftouter Check on $left.Id == $right.GroupId
| join kind=leftouter ServiceCheck_Log on $left.Id1 == $right.CheckId
| summarize
    completed=countif(IsCompliant == true),
    incomplete=countif(IsCompliant == false),
    total=dcount(ServiceId)
    by GroupId, CheckId

-- 1 300 000 records
-- 1 - 450ms
-- 2 - 324ms
-- 3 - 270ms
```

Как работает ADX внутри?

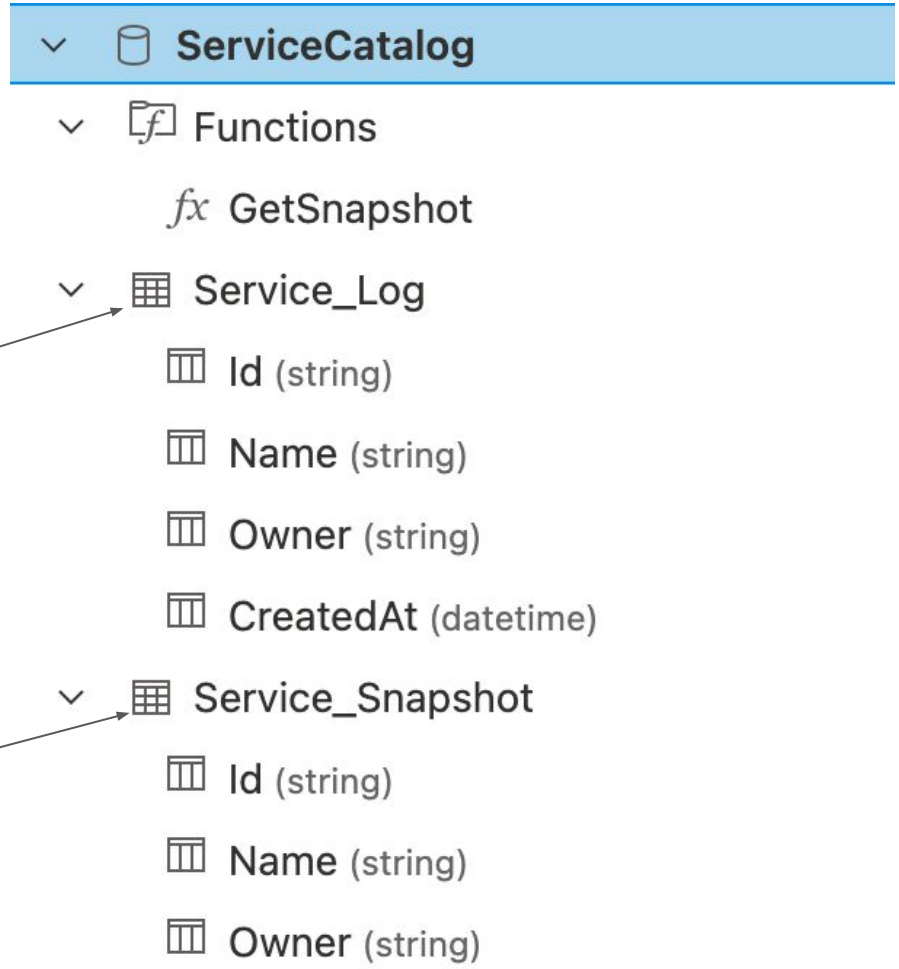
1. Хранение данных
2. Индексация текста
3. Сжатие колонок
4. Изоляция compute/storage
5. Кеширование
6. Распределенные запросы



Log & Snapshot 🔄

Хранит всю историю изменений

Хранит только актуальную версию



Snapshots 📚

```
.create-or-alter function GetSnapshot(data:(*), uid:string, key:string="CreatedAt")
{
    data
    | summarize
        key_agg = arg_max(column_ifexists(key, ''), *)
        by column_ifexists(uid, '')
    | project-away key_agg
}
```



```
GetSnapshot(Service_Log, uid='Id')

// OR

GetSnapshot(Service_Log, uid='Id', key='CreatedAt')
```

Решение

Детали архитектуры и реализации

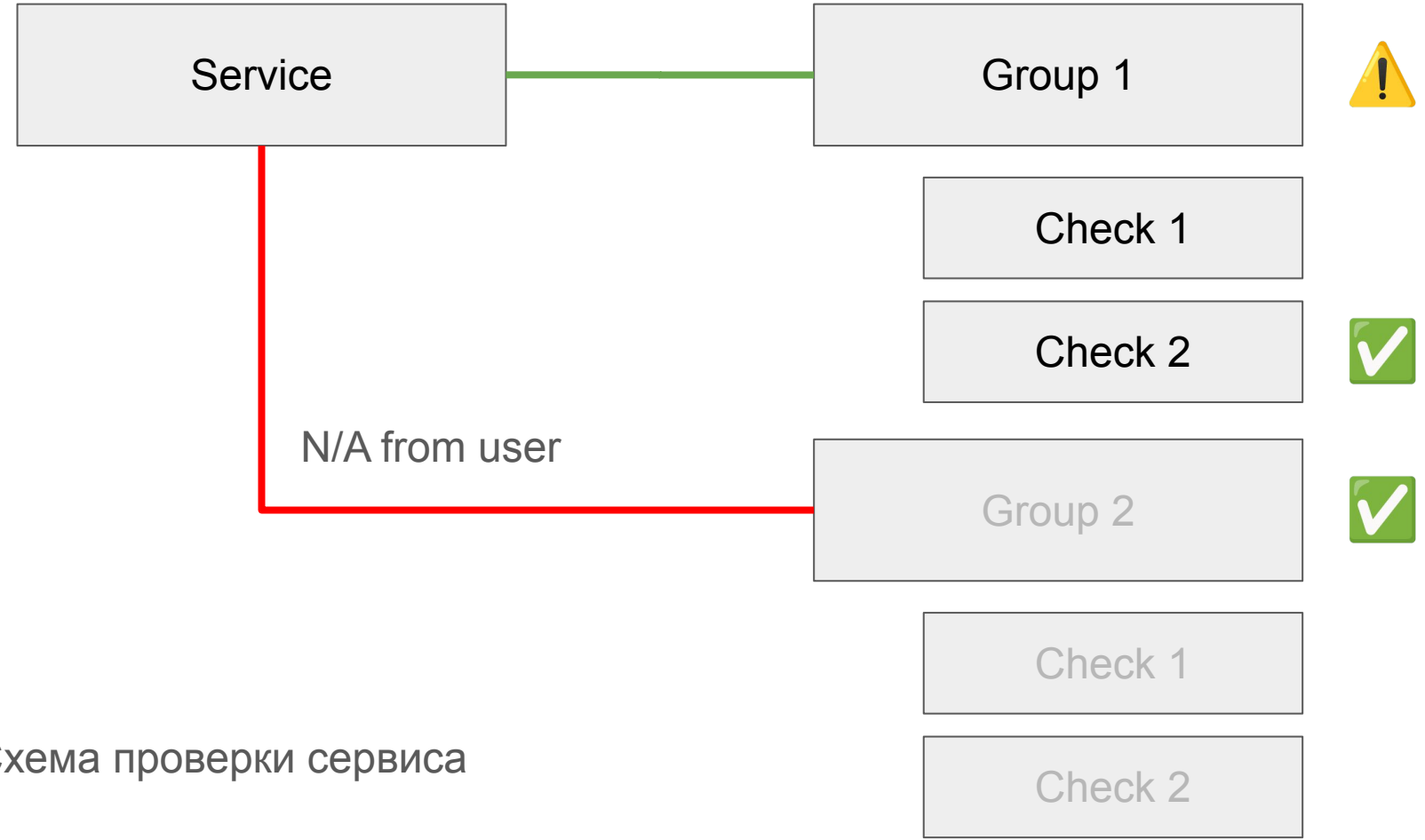
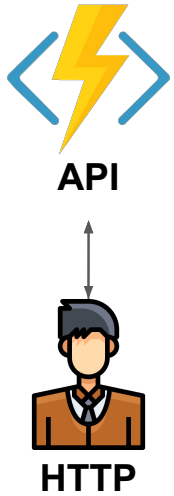
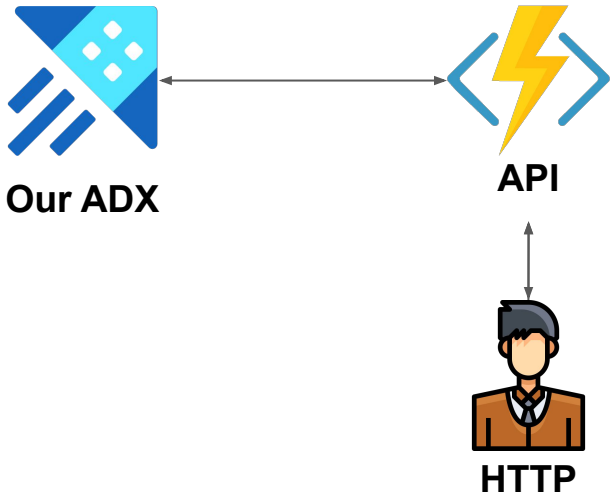


Схема проверки сервиса

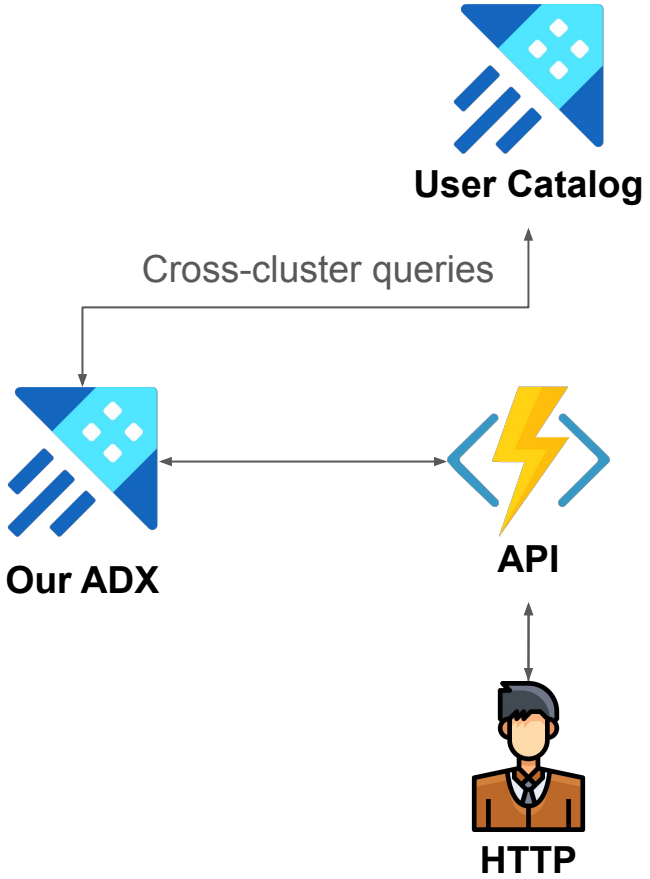
HTTP



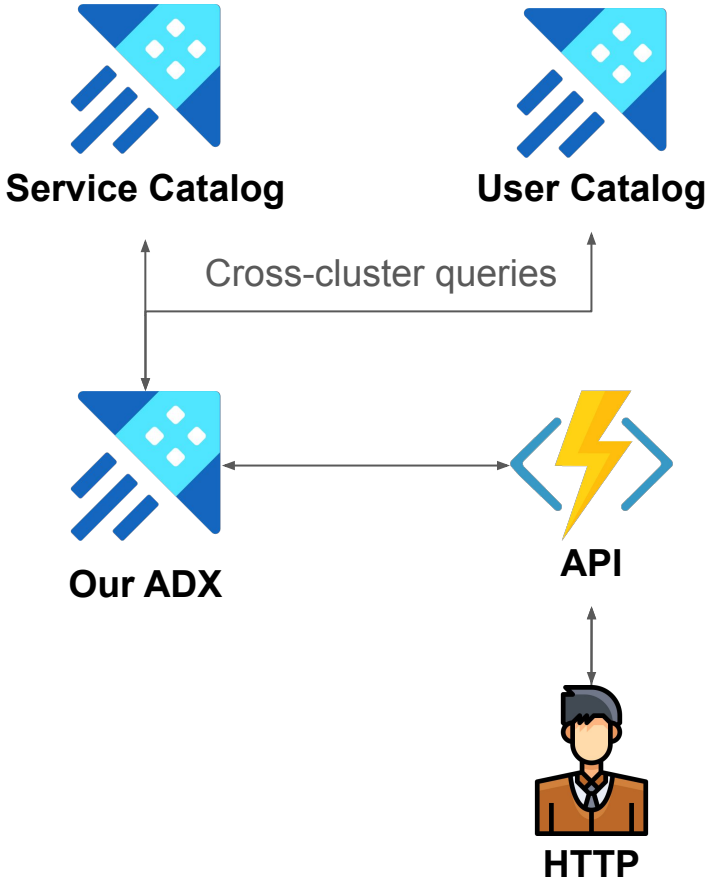
Как устроен Проект X



Как устроен Проект X



Как устроен Проект X



Как устроен Проект X

ADX 1

ADX 2

ADX 3

Service Catalog

User Catalog

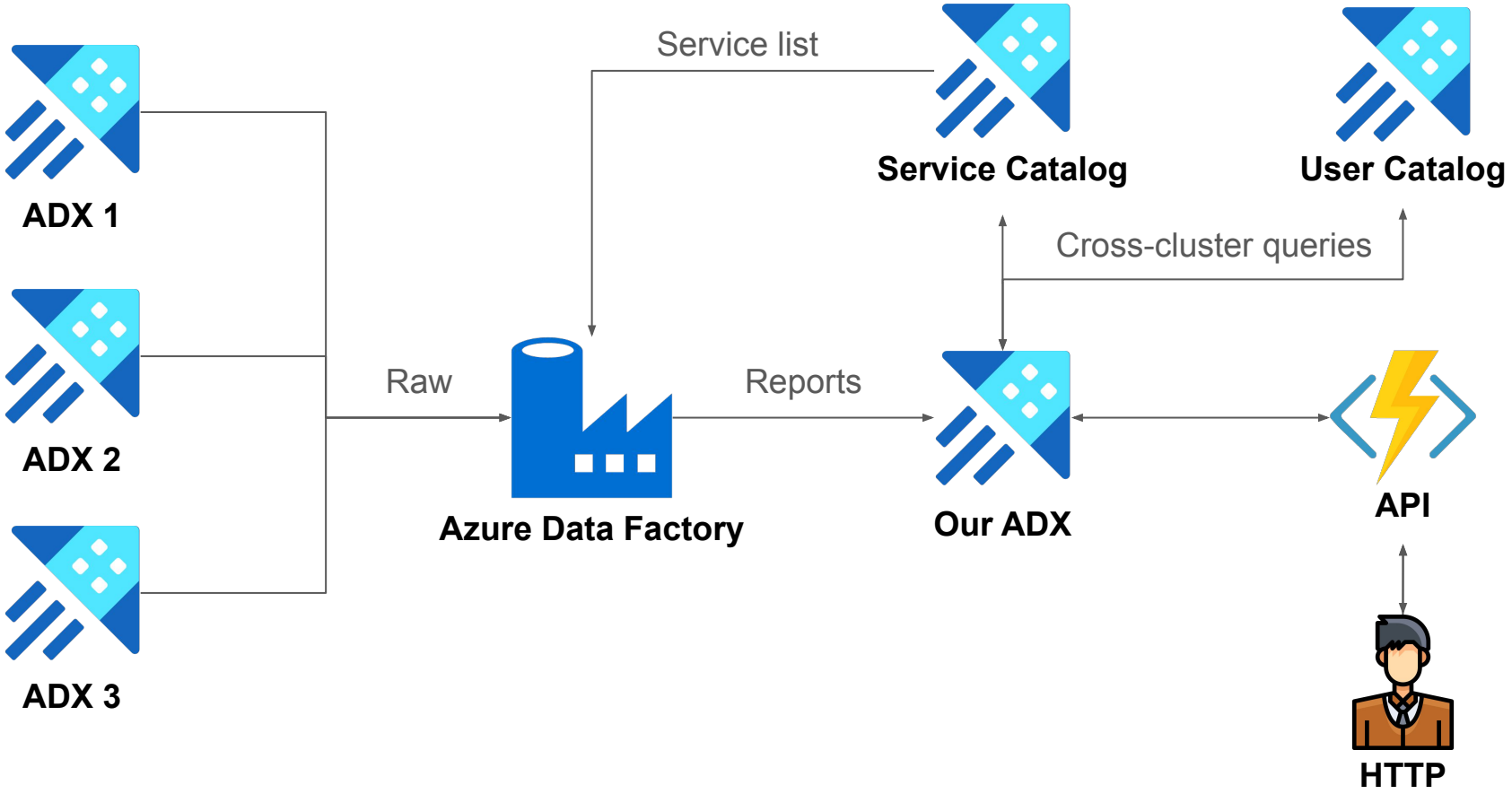
Cross-cluster queries

Our ADX

API

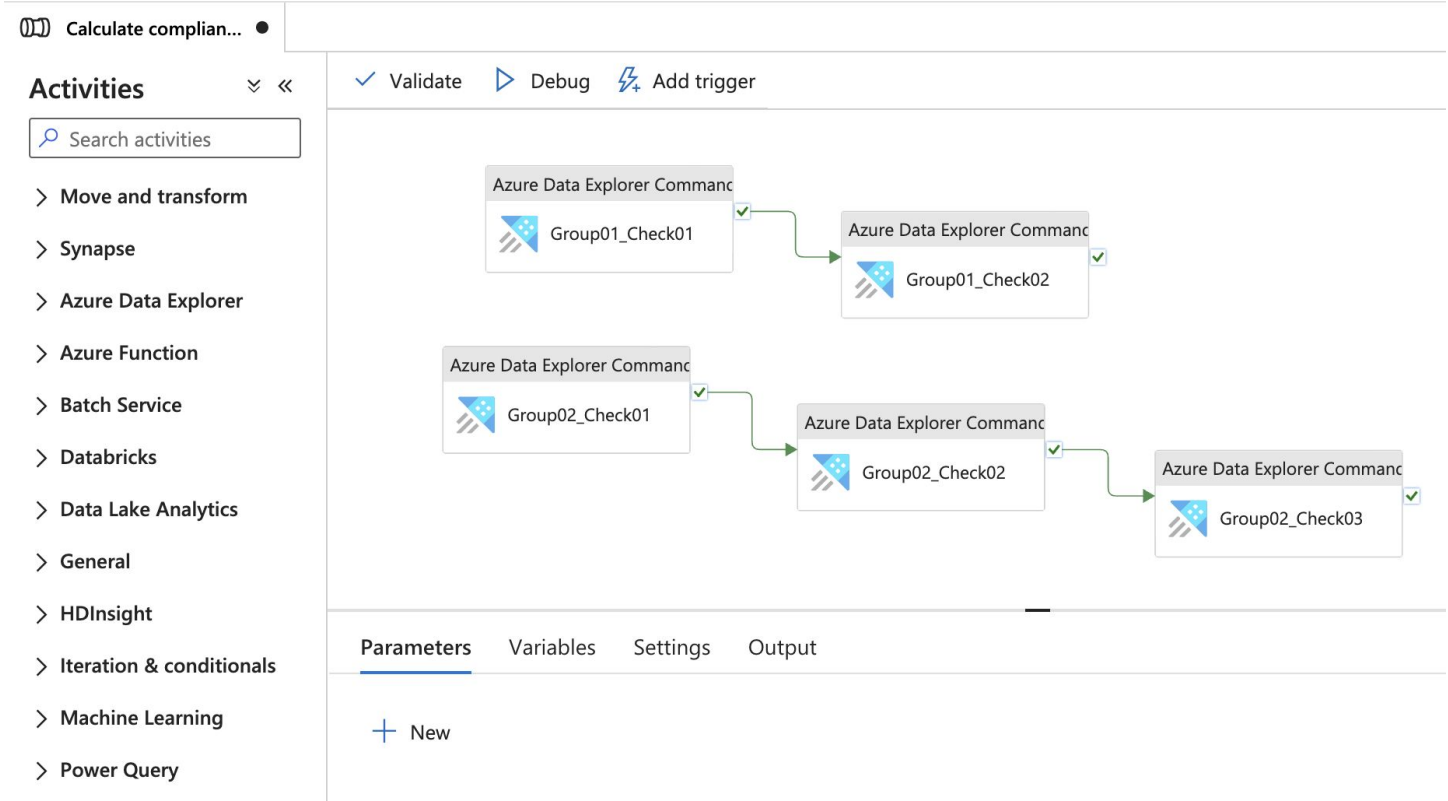
HTTP

Как устроен Проект X



Как устроен Проект X

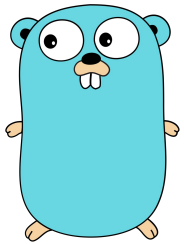
Azure Data Factory



Kusto SDK

Поддержка

- .NET
- Java
- Python
- Golang
- TypeScript
- PowerShell
- REST API



Kusto SDK

.NET C#
Azure Function

- Управление (DDL)
- Чтение данных (DML)
- Загрузка данных (DML)
- Языковые инструменты

Microsoft.Azure.Kusto.Data - Создание таблиц (DDL)

```
using (var kustoClient = KustoClientFactory.CreateCslAdminProvider(kcsb))
{
    var cmd = Kusto.Data.Common
        .CslCommandGenerator
        .GenerateTableCreateCommand(
            tableName,
            new[]
            {
                Tuple.Create("a", "System.String"),
                Tuple.Create("b", "System.Int32")
            }
        );

    await kustoClient.ExecuteControlCommandAsync(cmd);
}
```

Microsoft.Azure.Kusto.Data - Чтение (DML)

```
using (var kustoClient = KustoClientFactory.CreateCslQueryProvider(kcsb)) {
  string database = "UserCatalog";
  string query = "Users | where Email = 'john@doe.com'";

  using (var response = await kustoClient.ExecuteQueryAsync(database, query, null)) {
    // Handle IDataReader response
  }
}
```

Microsoft.Azure.Kusto.Data - Запись (DML)

```
using (var kustoClient = KustoClientFactory.CreateCslAdminProvider(kcsb)) {
  string database = "UserCatalog";
  string query = @"
    .set-or-append Users <|
    datatable(Id:string, Email:string)
    [
        'user-guid-1', 'john@doe.com',
        'user-guid-2', 'jane@doe.com',
    ]
  ";

  using (var response = await kustoClient.ExecuteControlCommandAsync(database, query, null)) {
    // Handle IDataReader response
  }
}
```

Microsoft.Azure.Kusto.Ingest - Массовая запись (DML)

```
using (var ingestClient = KustoIngestFactory.CreateQueuedIngestClient(ingestKcsb)) {
  string database = "UserCatalog";
  string table = "Users";

  string filePath = Path.Combine(
    Directory.GetCurrentDirectory(),
    "new_users.csv"
  );

  var ingestProps = new KustoIngestionProperties(database, table) {
    Format = DataSourceFormat.csv,
    AdditionalProperties = new Dictionary<string, string>()
    {
      { "ignoreFirstRecord", "True" },
    }
  };

  var result = await ingestClient.IngestFromStorageAsync(filePath, ingestProps);
}
```

Пара реальных примеров

Чтение

```
var cluster = "https://adxusers.eastus.kusto.windows.net";
var database = "ServiceCatalog";
var credential = new DefaultAzureCredential();
var kcsb = new KustoConnectionStringBuilder(cluster, database)
  .WithAadAzureTokenCredentialsAuthentication(credential);

using (var client = KustoClientFactory.CreateCslQueryProvider(kcsb))
{
    var serviceId = Guid.Parse("464C53CB-5FCE-4827-ABE0-E50EFA2127D6");
    var query = @"
        declare query_parameters(ServiceId: string);
        Service_Snapshot
        | where Id == ServiceId
    ";

    var parameters = new ClientRequestProperties()
    {
        ClientRequestId = $"GetServiceDetails_{serviceId}",
    };

    parameters.SetParameter("ServiceId", serviceId.ToString());

    var reader = await client.ExecuteQueryAsync(database, query, parameters);
    var dataTable = new DataTable();

    if (reader != null)
    {
        dataTable.Load(reader);
    }

    // Do something with the data
}
```

Подключение

```
var cluster = "https://adxusers.eastus.kusto.windows.net";
var database = "ServiceCatalog";
var credential = new DefaultAzureCredential();
var kcsb = new KustoConnectionStringBuilder(cluster, database)
  .WithAadAzureTokenCredentialsAuthentication(credential);

using (var client = KustoClientFactory.CreateCslQueryProvider(kcsb))
{
    var serviceId = Guid.Parse("464C53CB-5FCE-4827-ABE0-E50EFA2127D6");
    var query = @"
        declare query_parameters(ServiceId: string);
        Service_Snapshot
        | where Id == ServiceId
    ";

    var parameters = new ClientRequestProperties()
    {
        ClientRequestId = $"GetServiceDetails_{serviceId}",
    };

    parameters.SetParameter("ServiceId", serviceId.ToString());

    var reader = await client.ExecuteQueryAsync(database, query, parameters);
    var dataTable = new DataTable();

    if (reader != null)
    {
        dataTable.Load(reader);
    }

    // Do something with the data
}
```

Чтение

Подготовка запроса

```
var cluster = "https://adxusers.eastus.kusto.windows.net";
var database = "ServiceCatalog";
var credential = new DefaultAzureCredential();
var kcsb = new KustoConnectionStringBuilder(cluster, database)
  .WithAadAzureTokenCredentialsAuthentication(credential);

using (var client = KustoClientFactory.CreateCslQueryProvider(kcsb))
{
    var serviceId = Guid.Parse("464C53CB-5FCE-4827-ABE0-E50EFA2127D6");
    var query = @"
        declare query_parameters(ServiceId: string);
        Service_Snapshot
        | where Id == ServiceId
    ";

    var parameters = new ClientRequestProperties()
    {
        ClientRequestId = $"GetServiceDetails_{serviceId}",
    };

    parameters.SetParameter("ServiceId", serviceId.ToString());

    var reader = await client.ExecuteQueryAsync(database, query, parameters);
    var dataTable = new DataTable();

    if (reader != null)
    {
        dataTable.Load(reader);
    }

    // Do something with the data
}
```

Чтение

Привязка параметров

```
var cluster = "https://adxusers.eastus.kusto.windows.net";
var database = "ServiceCatalog";
var credential = new DefaultAzureCredential();
var kcsb = new KustoConnectionStringBuilder(cluster, database)
  .WithAadAzureTokenCredentialsAuthentication(credential);

using (var client = KustoClientFactory.CreateCslQueryProvider(kcsb))
{
    var serviceId = Guid.Parse("464C53CB-5FCE-4827-ABE0-E50EFA2127D6");
    var query = @"
        declare query_parameters(ServiceId: string);
        Service_Snapshot
        | where Id == ServiceId
    ";

    var parameters = new ClientRequestProperties()
    {
        ClientRequestId = $"GetServiceDetails_{serviceId}",
    };

    parameters.SetParameter("ServiceId", serviceId.ToString());

    var reader = await client.ExecuteQueryAsync(database, query, parameters);
    var dataTable = new DataTable();

    if (reader != null)
    {
        dataTable.Load(reader);
    }

    // Do something with the data
}
```

Чтение

Чтение

```
var cluster = "https://adxusers.eastus.kusto.windows.net";
var database = "ServiceCatalog";
var credential = new DefaultAzureCredential();
var kcsb = new KustoConnectionStringBuilder(cluster, database)
  .WithAadAzureTokenCredentialsAuthentication(credential);

using (var client = KustoClientFactory.CreateCslQueryProvider(kcsb))
{
    var serviceId = Guid.Parse("464C53CB-5FCE-4827-ABE0-E50EFA2127D6");
    var query = @"
        declare query_parameters(ServiceId: string);
        Service_Snapshot
        | where Id == ServiceId
    ";

    var parameters = new ClientRequestProperties()
    {
        ClientRequestId = $"GetServiceDetails_{serviceId}",
    };

    parameters.SetParameter("ServiceId", serviceId.ToString());

    var reader = await client.ExecuteQueryAsync(database, query, parameters);
    var dataTable = new DataTable();

    if (reader != null)
    {
        dataTable.Load(reader);
    }

    // Do something with the data
}
```

Выполнение запроса

Чтение

Обработка результата

```
var cluster = "https://adxusers.eastus.kusto.windows.net";
var database = "ServiceCatalog";
var credential = new DefaultAzureCredential();
var kcsb = new KustoConnectionStringBuilder(cluster, database)
  .WithAadAzureTokenCredentialsAuthentication(credential);

using (var client = KustoClientFactory.CreateCslQueryProvider(kcsb))
{
    var serviceId = Guid.Parse("464C53CB-5FCE-4827-ABE0-E50EFA2127D6");
    var query = @"
        declare query_parameters(ServiceId: string);
        Service_Snapshot
        | where Id == ServiceId
    ";

    var parameters = new ClientRequestProperties()
    {
        ClientRequestId = $"GetServiceDetails_{serviceId}",
    };

    parameters.SetParameter("ServiceId", serviceId.ToString());

    var reader = await client.ExecuteQueryAsync(database, query, parameters);
    var dataTable = new DataTable();

    if (reader != null)
    {
        dataTable.Load(reader);
    }

    // Do something with the data
}
```

Запись

```
var cluster = "https://adxusers.eastus.kusto.windows.net";
var database = "ServiceCatalog";
var credential = new DefaultAzureCredential();
var kcsb = new KustoConnectionStringBuilder(cluster, database)
    .WithAadAzureTokenCredentialsAuthentication(credential);

using (var client = KustoClientFactory.CreateCslAdminProvider(kcsb))
{
    var query = @"
        .set-or-append Service_Log <|
        datatable (Id: string, Name: string, Owner: string, CreatedAt: datetime)
        [
            '464C53CB-5FCE-4827-ABE0-E50EFA2127D6', 'Service_1',
            'AA32820D-0BFC-47AB-99C2-229616F2FEF5', datetime('2024-01-01'),
        ]
    ";

    var parameters = new ClientRequestProperties()
    {
        ClientRequestId = $"WriteServiceDetails_{serviceId}",
    };

    var result = await client.ExecuteControlCommandAsync(database, query, parameters);

    // Do something with the result
}
```

Подключение

```
var cluster = "https://adxusers.eastus.kusto.windows.net";
var database = "ServiceCatalog";
var credential = new DefaultAzureCredential();
var kcsb = new KustoConnectionStringBuilder(cluster, database)
    .WithAadAzureTokenCredentialsAuthentication(credential);

using (var client = KustoClientFactory.CreateCslAdminProvider(kcsb))
{
    var query = @"
        .set-or-append Service_Log <|
        datatable (Id: string, Name: string, Owner: string, CreatedAt: datetime)
        [
            '464C53CB-5FCE-4827-ABE0-E50EFA2127D6', 'Service_1',
            'AA32820D-0BFC-47AB-99C2-229616F2FEF5', datetime('2024-01-01'),
        ]
    ";

    var parameters = new ClientRequestProperties()
    {
        ClientRequestId = $"WriteServiceDetails_{serviceId}",
    };

    var result = await client.ExecuteControlCommandAsync(database, query, parameters);

    // Do something with the result
}
```

Запись

Подготовка запроса

```
var cluster = "https://adxusers.eastus.kusto.windows.net";
var database = "ServiceCatalog";
var credential = new DefaultAzureCredential();
var kcsb = new KustoConnectionStringBuilder(cluster, database)
    .WithAadAzureTokenCredentialsAuthentication(credential);

using (var client = KustoClientFactory.CreateCslAdminProvider(kcsb))
{
    var query = @"
        .set-or-append Service_Log <|
        datatable (Id: string, Name: string, Owner: string, CreatedAt: datetime)
        [
            '464C53CB-5FCE-4827-ABE0-E50EFA2127D6', 'Service_1',
            'AA32820D-0BFC-47AB-99C2-229616F2FEF5', datetime('2024-01-01'),
        ]
    ";

    var parameters = new ClientRequestProperties()
    {
        ClientRequestId = $"WriteServiceDetails_{serviceId}",
    };

    var result = await client.ExecuteControlCommandAsync(database, query, parameters);

    // Do something with the result
}
```

Запись

Привязка параметров

```
var cluster = "https://adxusers.eastus.kusto.windows.net";
var database = "ServiceCatalog";
var credential = new DefaultAzureCredential();
var kcsb = new KustoConnectionStringBuilder(cluster, database)
    .WithAadAzureTokenCredentialsAuthentication(credential);

using (var client = KustoClientFactory.CreateCslAdminProvider(kcsb))
{
    var query = @"
        .set-or-append Service_Log <|
        datatable (Id: string, Name: string, Owner: string, CreatedAt: datetime)
        [
            '464C53CB-5FCE-4827-ABE0-E50EFA2127D6', 'Service_1',
            'AA32820D-0BFC-47AB-99C2-229616F2FEF5', datetime('2024-01-01'),
        ]
    ";

    var parameters = new ClientRequestProperties()
    {
        ClientRequestId = $"WriteServiceDetails_{serviceId}",
    };

    var result = await client.ExecuteControlCommandAsync(database, query, parameters);

    // Do something with the result
}
```

Запись

```
1 var cluster = "https://adxusers.eastus.kusto.windows.net";
2 var database = "ServiceCatalog";
3 var credential = new DefaultAzureCredential();
4 var kcsb = new KustoConnectionStringBuilder(cluster, database)
5     .WithAadAzureTokenCredentialsAuthentication(credential);
6
7 using (var client = KustoClientFactory.CreateCslAdminProvider(kcsb))
8 {
9     var query = @"
10         .set-or-append Service_Log <|
11         datatable (Id: string, Name: string, Owner: string, CreatedAt: datetime)
12         [
13             '464C53CB-5FCE-4827-ABE0-E50EFA2127D6', 'Service_1',
14             'AA32820D-0BFC-47AB-99C2-229616F2FEF5', datetime('2024-01-01'),
15         ]
16     ";
17
18     var parameters = new ClientRequestProperties()
19     {
20         ClientRequestId = $"WriteServiceDetails_{serviceId}",
21     };
22
23     var result = await client.ExecuteControlCommandAsync(database, query, parameters);
24
25     // Do something with the result
26 }
```

Обработка результата

Запись

Первое впечатление

Плюсы

- Легко начать
- Раздельные read и write операции
- Немного зависимостей (~3)

Минусы

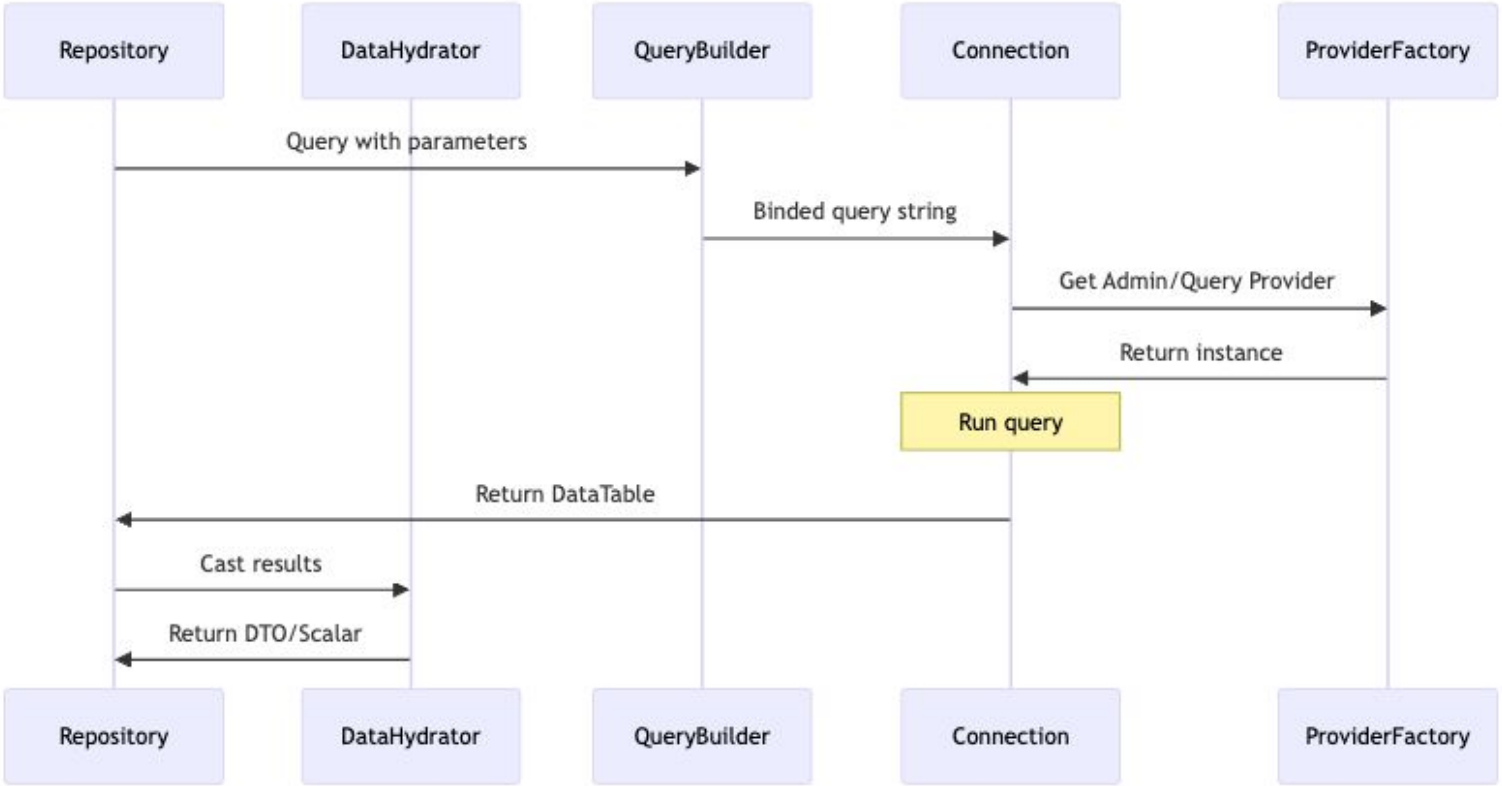
- Слишком много LOC/запрос
- Ручное время жизни QueryProvider
- Raw KQL
- Нюансы привязки параметров
- IDataReader как результат

Что делаем дальше?

1. Берем как есть 
2. Пишем расширение под Entity Framework 
3. Пишем свою библиотеку 

Что делаем дальше?

1. Берем как есть 🤔
2. Пишем расширение под Entity Framework 😄
3. **Пишем свою библиотеку** 🚲 ✅



Абстракция библиотеки

```
public class ServiceManager : IServiceManager
{
  private readonly IKustoRepository repository;

  public ServiceManager(IKustoRepository repository)
  {
    this.repository = repository;
  }

  // Client code goes here ...
}
```

Зависимость в клиентском коде

```
1  public async Task<ServiceDetailsDto> GetServiceDetails(Guid serviceId)
2  {
3      var query = @"
4          Service_Snapshot
5          | where Id == @serviceId
6      ";
7
8      var parameters = new Dictionary<string, object>
9      {
10         { "serviceId", serviceId }
11     };
12
13     return await repository.GetOne<ServiceDetailsDto>(query, parameters);
14 }
```

Чтение

```
1 public async Task StoreServiceCheck(ServiceCheckDto serviceCheck)
2 {
3     var items = new List<ServiceCheckDto> { serviceCheck };
4
5     return await repository.Append("ServiceCheck_Log", items);
6 }
```

BTS - CslCommandGenerator.GenerateTableSetOrAppendCommand()

Запись

DI Container

```
// Cluster metadata
var cluster = "https://clustername.westus2.kusto.windows.net/";
var database = "MyDatabase";

// Initial classes to work with Kusto
var connectionStringBuider = new KustoConnectionStringBuilder(cluster, database)
  .WithAadApplicationKeyAuthentication("AADAppId", "AADAppKey", "AADAuthority");

var kustoProviderFactory = new KustoProviderFactory(connectionStringBuider);
var connection = new KustoConnection(database, kustoProviderFactory);

// Query utilities
var stringFormatter = new StringFormatter();
var arrayFormatter = new ArrayFormatter();

var queryBuilder = new QueryBuilder(
    stringFormatter,
    new Dictionary<Type, IQueryFormatter>()
    {
        { typeof(string), stringFormatter },
        { typeof(List<object>), arrayFormatter },
    });

var dataHydrator = new DataHydrator();

// Base Kusto repository (use it as a dependency in custom repositories)
var repository = new KustoRepository(connection, queryBuilder, dataHydrator);
```

Адрес базы данных

```
// Cluster metadata
var cluster = "https://clustername.westus2.kusto.windows.net/";
var database = "MyDatabase";

// Initial classes to work with Kusto
var connectionStringBuider = new KustoConnectionStringBuilder(cluster, database)
  .WithAadApplicationKeyAuthentication("AADAppId", "AADAppKey", "AADAuthority");

var kustoProviderFactory = new KustoProviderFactory(connectionStringBuider);
var connection = new KustoConnection(database, kustoProviderFactory);

// Query utilities
var stringFormatter = new StringFormatter();
var arrayFormatter = new ArrayFormatter();

var queryBuilder = new QueryBuilder(
    stringFormatter,
    new Dictionary<Type, IQueryFormatter>()
    {
        { typeof(string), stringFormatter },
        { typeof(List<object>), arrayFormatter },
    });

var dataHydrator = new DataHydrator();

// Base Kusto repository (use it as a dependency in custom repositories)
var repository = new KustoRepository(connection, queryBuilder, dataHydrator);
```

DI Container

Строка подключения

DI Container

```
// Cluster metadata
var cluster = "https://clustername.westus2.kusto.windows.net/";
var database = "MyDatabase";

// Initial classes to work with Kusto
var connectionStringBuider = new KustoConnectionStringBuilder(cluster, database)
  .WithAadApplicationKeyAuthentication("AADAppId", "AADAppKey", "AADAuthority");

var kustoProviderFactory = new KustoProviderFactory(connectionStringBuider);
var connection = new KustoConnection(database, kustoProviderFactory);

// Query utilities
var stringFormatter = new StringFormatter();
var arrayFormatter = new ArrayFormatter();

var queryBuilder = new QueryBuilder(
    stringFormatter,
    new Dictionary<Type, IQueryFormatter>()
    {
        { typeof(string), stringFormatter },
        { typeof(List<object>), arrayFormatter },
    });

var dataHydrator = new DataHydrator();

// Base Kusto repository (use it as a dependency in custom repositories)
var repository = new KustoRepository(connection, queryBuilder, dataHydrator);
```

Соединение

```
// Cluster metadata
var cluster = "https://clustername.westus2.kusto.windows.net/";
var database = "MyDatabase";

// Initial classes to work with Kusto
var connectionStringBuider = new KustoConnectionStringBuilder(cluster, database)
  .WithAadApplicationKeyAuthentication("AADAppId", "AADAppKey", "AADAuthority");

var kustoProviderFactory = new KustoProviderFactory(connectionStringBuider);
var connection = new KustoConnection(database, kustoProviderFactory);

// Query utilities
var stringFormatter = new StringFormatter();
var arrayFormatter = new ArrayFormatter();

var queryBuilder = new QueryBuilder(
    stringFormatter,
    new Dictionary<Type, IQueryFormatter>()
    {
        { typeof(string), stringFormatter },
        { typeof(List<object>), arrayFormatter },
    });

var dataHydrator = new DataHydrator();

// Base Kusto repository (use it as a dependency in custom repositories)
var repository = new KustoRepository(connection, queryBuilder, dataHydrator);
```

DI Container

Утилиты для формирования / чтения запросов

```
// Cluster metadata
var cluster = "https://clustername.westus2.kusto.windows.net/";
var database = "MyDatabase";

// Initial classes to work with Kusto
var connectionStringBuider = new KustoConnectionStringBuilder(cluster, database)
  .WithAadApplicationKeyAuthentication("AADAppId", "AADAppKey", "AADAuthority");

var kustoProviderFactory = new KustoProviderFactory(connectionStringBuider);
var connection = new KustoConnection(database, kustoProviderFactory);

// Query utilities
var stringFormatter = new StringFormatter();
var arrayFormatter = new ArrayFormatter();

var queryBuilder = new QueryBuilder(
    stringFormatter,
    new Dictionary<Type, IQueryFormatter>()
    {
        { typeof(string), stringFormatter },
        { typeof(List<object>), arrayFormatter },
    });

var dataHydrator = new DataHydrator();

// Base Kusto repository (use it as a dependency in custom repositories)
var repository = new KustoRepository(connection, queryBuilder, dataHydrator);
```

DI Container

Репозиторий для использования

DI Container

```
// Cluster metadata
var cluster = "https://clustername.westus2.kusto.windows.net/";
var database = "MyDatabase";

// Initial classes to work with Kusto
var connectionStringBuider = new KustoConnectionStringBuilder(cluster, database)
  .WithAadApplicationKeyAuthentication("AADAppId", "AADAppKey", "AADAuthority");

var kustoProviderFactory = new KustoProviderFactory(connectionStringBuider);
var connection = new KustoConnection(database, kustoProviderFactory);

// Query utilities
var stringFormatter = new StringFormatter();
var arrayFormatter = new ArrayFormatter();

var queryBuilder = new QueryBuilder(
    stringFormatter,
    new Dictionary<Type, IQueryFormatter>()
    {
        { typeof(string), stringFormatter },
        { typeof(List<object>), arrayFormatter },
    });

var dataHydrator = new DataHydrator();

// Base Kusto repository (use it as a dependency in custom repositories)
var repository = new KustoRepository(connection, queryBuilder, dataHydrator);
```

Второе впечатление

Плюсы

- Меньше LOC/запрос
- Connection управляет QueryProvider
- Единственная зависимость
- Единая точка входа для Read/Write
- Автоматический маппинг DTO

Минусы

- Начать уже не так легко
- Непростая регистрация в DI
- Все еще не Entity Framework

Что если…?

У нас много идей

- Entities
- Migrations (*Delta*)
- Seeding
- Query Builder
- LINQ
- …

Давайте в open-source!

но это не точно…

Время релиза!

А что с тестами? 🤔

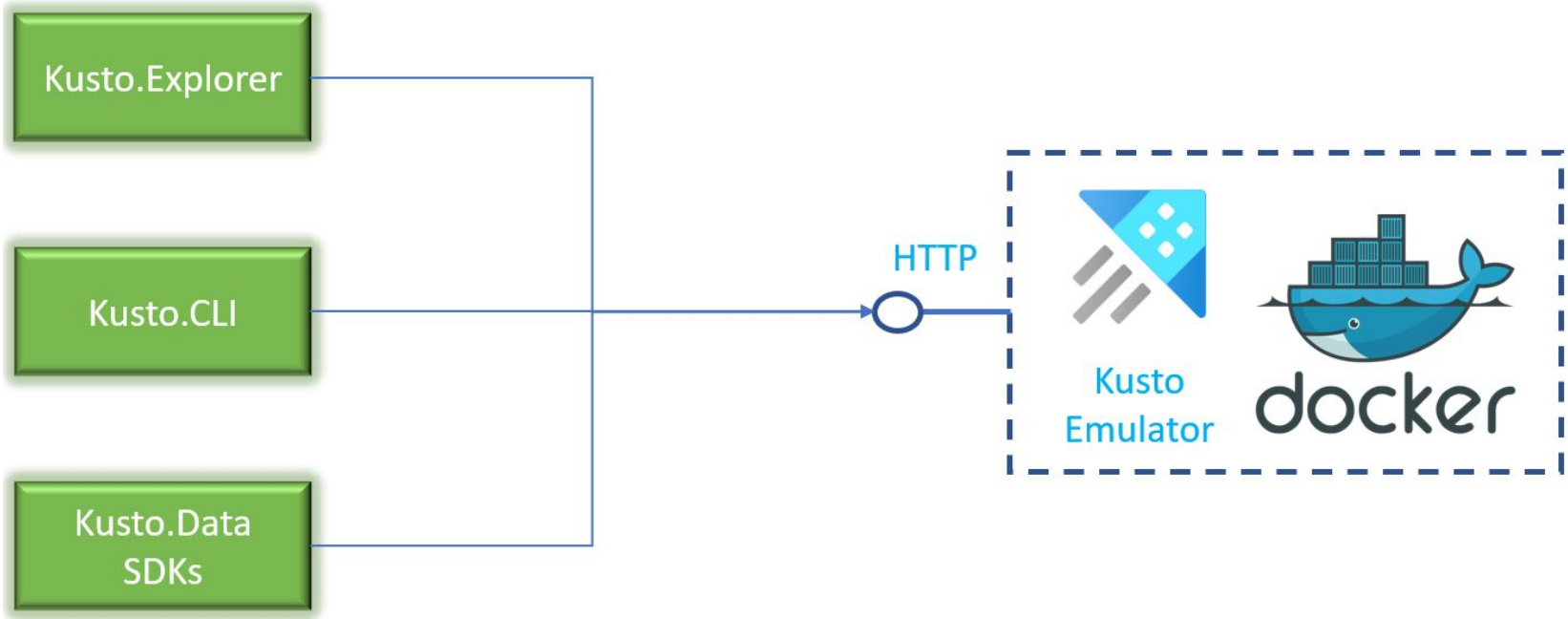
ADX + Тесты

… без Azure инфраструктуры

- Как тестить внешние источники?
- Как загрузить fake данные?
- Свой ADX на разработчика?
- Получится ли в CI?

Kustainer - Kusto in Container

Azure Data Explorer как Windows-based Docker образ.



Ограничения

Функция	**Kustainer**	**Бесплатный кластер**
Хранилище (без сжатия)	Место на хосте	100 GB
Базы и таблицы	10000 / 10000	10 / 100
Внешние таблицы	Локальные файлы	Не поддерживается
Безопасность	**Не поддерживается**	Azure аутентификация
Materialized views на базу	1000	5
Наличие интернета	**Необязательно**	Доступ к облаку Azure

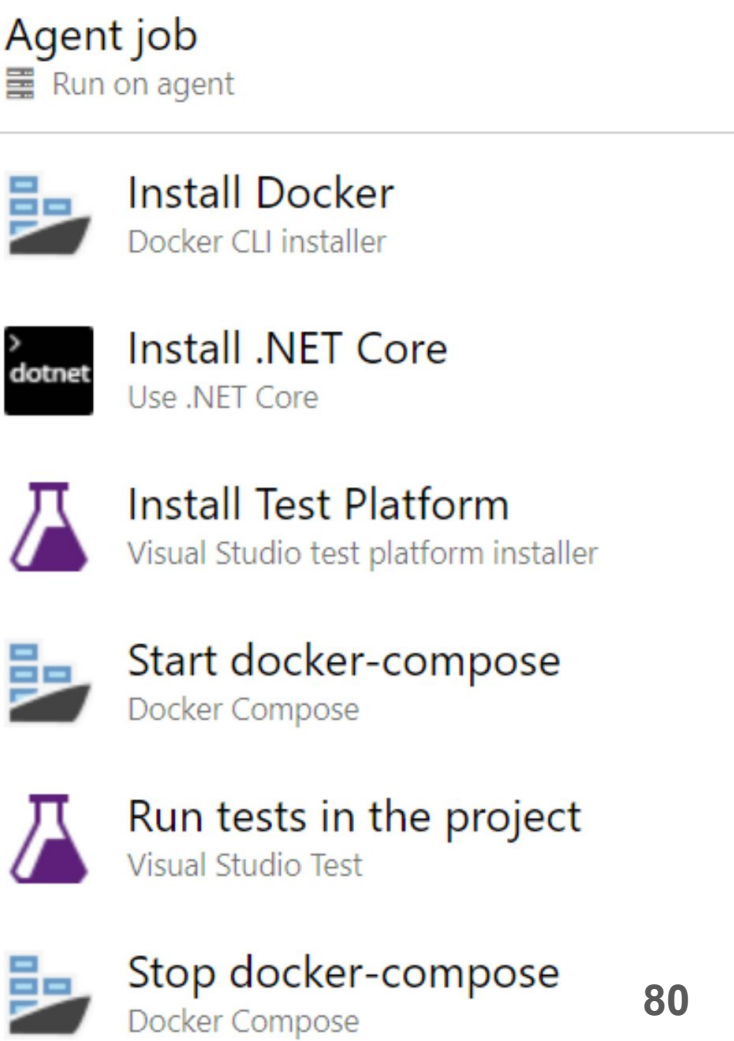
Загрузка тестовых данных

Исполнение ***.kql** файлов при старте контейнера

```
.drop tables (
  Group, Check, ServiceCheck_Log,
  ServiceCheck_Snapshot, ServiceException_Log,
  ServiceException_Snapshot
)

.create tables
    Group (Id: string, Name: string),
    Check (Id: string, Name: string, Description: string),
    ServiceCheck_Log (ServiceId: string, CheckId: string, IsCompliant: bool, CreatedAt: datetime),
    ServiceCheck_Snapshot (ServiceId: string, CheckId: string, IsCompliant: bool),
    ServiceException_Log (ServiceId: string, GroupId: string, IsApplicable: bool, CreatedAt: datetime),
    ServiceException_Snapshot (ServiceId: string, GroupId: string, IsApplicable: bool)
```

CI/CD - Pipeline



**windows-based images mode*

Результаты

Что мы получили

Эксперимент удался, но…

>4GB RAM на Docker

Random execution time

Timeout

Копались в исходниках

Мало документации

429 Too many requests

Много кода

Windows-based Docker image

Concurrent query limit per core

Cross-cluster Authentication issues

Стоит ли оно того?

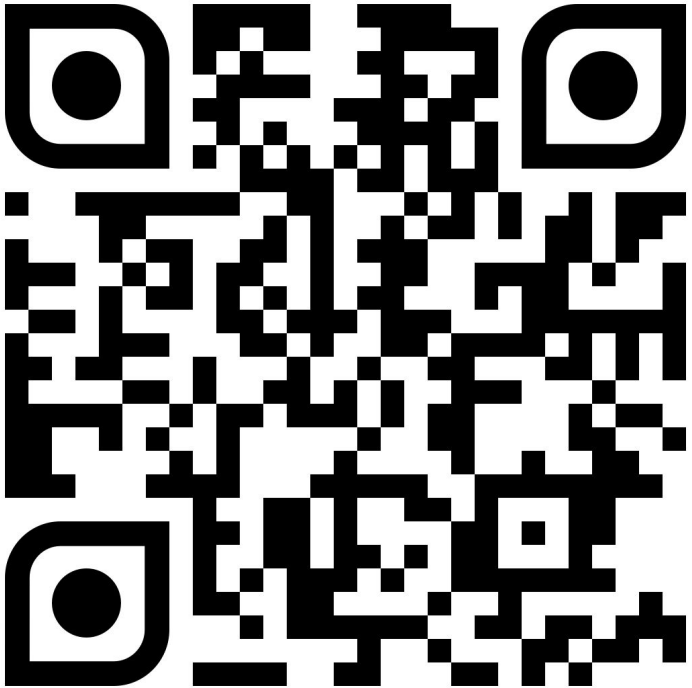
Зависит от ситуации

Когда стоит

- **Если вы в Azure**
- Нет строгого SLA
- Регулируемое кол-во пользователей
- Слабая структура данных
- Распределенное хранение
- Не нужны связи
- Пользовательский ввод

Ресурсы

- Azure Data Explorer - Docs
- Azure Data Explorer Technology 101
- ADX Policies and Limitations
- Kusto SDK
- Write First App
- Kustainer Docs
- Azure DevOps Orchestration
- Sync and Delta
- SDK Examples on Github



Scan!

Спасибо

github.com/manchenkoff