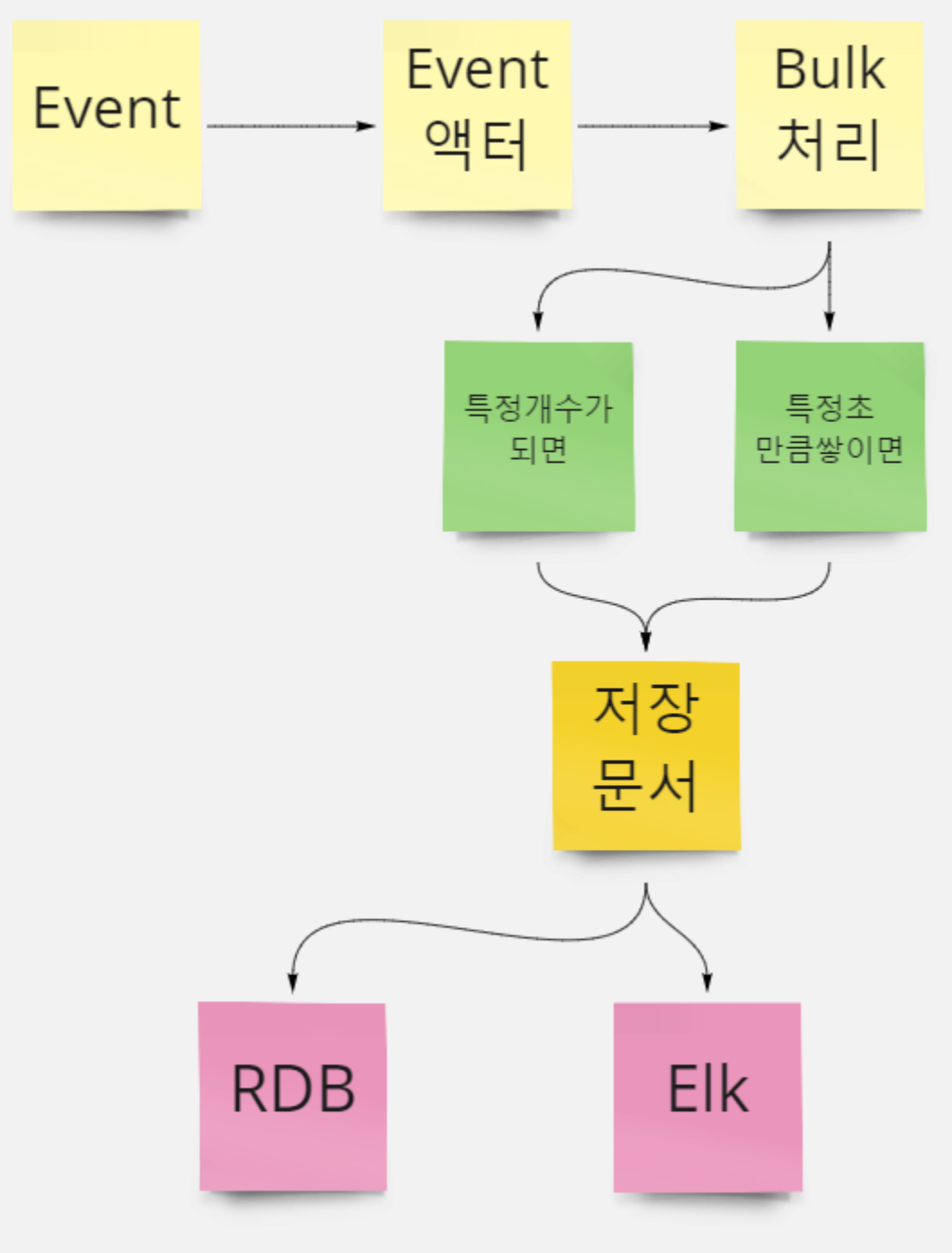
# NetCoreBulkInsertWithAKKA

NetCore(Akka)  InsertOrUpdate .

git : https://github.com/psmon/AkkaNetBulkBatch




,

RDB ElasticSearch . 1 Event 1 .

SQL NOSQL(ELK) .

( )

- mysql .
- elastic search .

**DokcoerCompose**

```
version: '3.4'

services:
  bulk-mysql:
    image: mysql:5.7
    command: --default-authentication-plugin=mysql_native_password
    restart: always
    volumes:
      - ./firstsql.mysql:/docker-entrypoint-initdb.d/init.sql
    ports:
      - 13306:3306
    environment:
      MYSQL_ROOT_PASSWORD: root

  bulk-elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:7.4.0
    container_name: elasticsearch
    environment:
      - xpack.security.enabled=false
      - discovery.type=single-node
      - "ES_JAVA_OPTS=-Xms1500m -Xmx3000m"
    ulimits:
      memlock:
        soft: -1
        hard: -1
      nofile:
        soft: 65536
        hard: 65536
    cap_add:
      - IPC_LOCK
    volumes:
      - bulk-elasticsearch-data:/usr/share/elasticsearch/data
    ports:
      - 9200:9200
      - 9300:9300

  bulk-kibana:
    container_name: kibana
    image: docker.elastic.co/kibana/kibana:7.4.0
    restart: always
    environment:
      - ELASTICSEARCH_HOSTS=http://elasticsearch:9200
    ports:
      - 5601:5601
    depends_on:
      - bulk-elasticsearch

volumes:
  bulk-elasticsearch-data:
    driver: local
```

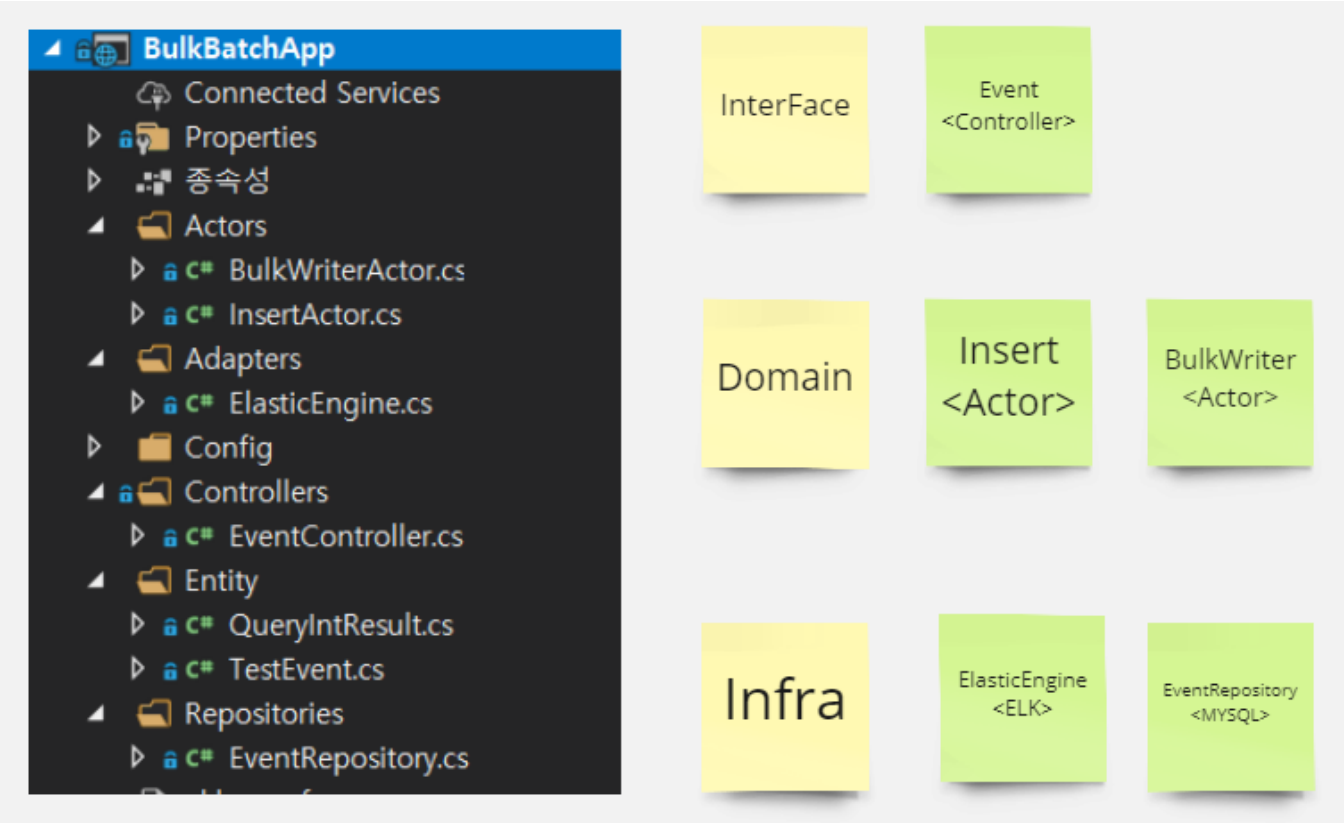
**firstsql.mysql**

```
/*
 * Action:   DB
 * Purpose:
 *  :
*/
CREATE DATABASE `webnori` default character set utf8 collate utf8_general_ci;
USE `webnori`;
SET NAMES 'utf8';

CREATE TABLE `tbl_test_bulk`  (
  `id` varchar(50) NOT NULL COMMENT 'ID',
  `action_type` int(0) NOT NULL DEFAULT 0 COMMENT 'Type',
  `action_name` varchar(50) NOT NULL COMMENT '',
  `upd_dt` datetime(0) NULL DEFAULT NULL ON UPDATE CURRENT_TIMESTAMP(0) COMMENT ' ',
  `reg_dt` datetime(0) NOT NULL DEFAULT CURRENT_TIMESTAMP() COMMENT ' ',
  PRIMARY KEY (`id`)
) CHARACTER SET = utf8 COLLATE = utf8_general_ci COMMENT = '';
```

```
    <PackageReference Include="AkkaDotModule.Webnori" Version="1.1.1" />
    <PackageReference Include="NEST" Version="7.5.1" />
    <PackageReference Include="Pomelo.EntityFrameworkCore.MySql" Version="3.1.1" />
```

- AkkaDotModule.Webnori :  . ( AKKA )
- NEST :  API  .
- Pomelo.EntityFrameworkCore.MySql : Entity ORM + MYSQL  .  ORM  .

## Application LayOut




- EventController : Event RestAPI / .
- InsertActor : Event Queue .
- BulkWriterActor : .
- ElasticEngine / EventRepository : .

## Entity (+)

Event , InsertOrUpdate .

- POCO , ElasticSearch ,MySQL .

```
using System;

namespace BulkBatchApp.Entity
{
    public class TestEvent
    {
        public string id { get; set; }

        public int action_type { get; set; }

        public string action_name { get; set; }

        public DateTime upd_dt { get; set; }

        public DateTime reg_dt { get; set; }
    }

    public class InsertOrUpdateTestEvent : TestEvent
    {

    }
}
```

- MYSQL

```
using BulkBatchApp.Config;
using BulkBatchApp.Entity;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Logging;
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading.Tasks;

namespace BulkBatchApp.Repositories
{
    public class EventRepository : DbContext
    {
        private string database = "webnori";

        private readonly AppSettings _appSettings;

        public DbSet<QueryIntResult> Native { get; set; } //For Native Query

        private readonly ILogger<EventRepository> _logger;

        public EventRepository(AppSettings appSettings, ILogger<EventRepository> logger)
        {
            _appSettings = appSettings;
            _logger = logger;
        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder
                .Entity<QueryIntResult>(eb =>
                {
                    eb.HasNoKey();
                });
        }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            string dbOption = "Convert Zero Datetime=true;";
            string dbConnectionString = string.Empty;
            dbConnectionString = _appSettings.DBConnection + $"database={database};" + dbOption;
            optionsBuilder.UseMySql(dbConnectionString);
            base.OnConfiguring(optionsBuilder);
        }

        private string DateToDbString(DateTime dt)
        {
            return dt.ToString("yyyy-MM-dd hh:mm:ss");
        }

        public async Task BulkInsertOrUpdateToEventTable(List<TestEvent> testEvents)
        {
            StringBuilder queryStr = new StringBuilder("");
            queryStr.Append($"INSERT INTO tbl_test_bulk( id, action_type, action_name, reg_dt) VALUES ");

            int idx = 0;
            foreach (var testEvent in testEvents)
            {
                idx++;
                string appendQuery = $"('{testEvent.id}',{testEvent.action_type},'{testEvent.
action_name}','{DateToDbString(testEvent.reg_dt)}')";
                if (idx == testEvents.Count)
                {
                    queryStr.AppendLine(appendQuery);
                }
```

```
                else
                {
                    queryStr.AppendLine(appendQuery + ",");
                }
            }

            queryStr.AppendLine(@"
            ON DUPLICATE KEY UPDATE action_type= VALUES(action_type), action_name=VALUES(action_name),
reg_dt=VALUES(reg_dt);
            SELECT 1 as result;");

            _logger.LogDebug("======= BulkQuery  ========");
            _logger.LogDebug(queryStr.ToString());

            await Native.FromSqlRaw(queryStr.ToString()).ToListAsync().ConfigureAwait(false);
        }
    }
}
```

> *https://entityframework-extensions.net/bulk-update ORM*
>
> *BulkUpdate , EntityContext Repository(ORM, NativeSQL) .*

## - ELASTICSEARCH

```
using BulkBatchApp.Config;
using BulkBatchApp.Entity;
using Nest;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace BulkBatchApp.Adapters
{
    public interface IElasticEngine
    {
        IElasticClient GetClientForTestEvent();
    }

    public class ElasticEngine : IElasticEngine
    {
        private IElasticClient clientForTestEvent { get; set; }

        public ElasticEngine(AppSettings appSettings)
        {
            var defaultSettings = new ConnectionSettings(new Uri(appSettings.ElkConnection))
                .DefaultIndex(appSettings.ElkIndex)
                .DefaultMappingFor<TestEvent>(m => m
                    .IdProperty(p => p.id)
                );

            clientForTestEvent = new ElasticClient(defaultSettings);

        }

        public IElasticClient GetClientForTestEvent()
        {
            return clientForTestEvent;
        }

        public async Task BulkInsertOrUpdateToEventTable(List<TestEvent> testEvents)
        {
            var descriptorProductPerPage = new BulkDescriptor();

            foreach(var testEvent in testEvents)
            {
                descriptorProductPerPage.Index<TestEvent>(op => op
                .Document(testEvent));
            }

            var result = await GetClientForTestEvent().BulkAsync(descriptorProductPerPage);
            if (!result.ApiCall.Success)
            {
                throw new Exception($"Failed Elk-Bulk Insert : messge {result.ServerError}");
            }
        }
    }
}
```

*.DefaultMappingFor<TestEvent>(m => m*
*.IdProperty(p => p.id)*
*);*

*ElasticSearch id ,*

*id , Id . ( for InsetOrUpdate)*

*, BulkAsync .*

```
using Akka.Actor;
using AkkaDotModule.ActorUtils;
using AkkaDotModule.Models;
using BulkBatchApp.Entity;
using Microsoft.Extensions.DependencyInjection;

namespace BulkBatchApp.Actors
{
    public class InsertActor : ReceiveActor
    {
        private readonly IActorRef _bulkWriterActor;
        private readonly IActorRef _batchActor;

        //, bulkSec  , bulkCount
        int bulkSec = 3;
        int bulkCount = 1000;
        int eventCount = 0;

        public InsertActor(IServiceScopeFactory scopeFactory)
        {
            _bulkWriterActor = Context.ActorOf(Props.Create(() => new BulkWriterActor(scopeFactory)));
            _batchActor = Context.ActorOf(Props.Create(() => new BatchActor(bulkSec)));
            _batchActor.Tell(new SetTarget(_bulkWriterActor));  // (    )
            ReceiveAsync<InsertOrUpdateTestEvent>(async insertOrUpdateTestEvent =>
            {
                _batchActor.Tell(new Queue(insertOrUpdateTestEvent));
                eventCount++;
                if (eventCount > bulkCount)
                {
                    eventCount = 0;
                    //  ,
                    _batchActor.Tell(new Flush());
                }
            });
        }
    }
}
```

BatchActor  (AkkaDotModule.Webnori)

,  Writer .

, RDB/ELK  InsertOrUpdate .

```
using Akka.Actor;
using Akka.Event;
using AkkaDotModule.Models;
using BulkBatchApp.Adapters;
using BulkBatchApp.Entity;
using BulkBatchApp.Repositories;
using Microsoft.Extensions.DependencyInjection;
using System;
using System.Collections.Generic;

namespace BulkBatchApp.Actors
{
    public class BulkWriterActor : ReceiveActor
    {
        private readonly ILoggingAdapter logger = Context.GetLogger();
        private readonly IServiceScopeFactory _scopeFactory;

        public BulkWriterActor(IServiceScopeFactory scopeFactory)
        {
            _scopeFactory = scopeFactory;

            ReceiveAsync<object>(async message =>
            {
                //   DB  .
                if (message is Batch batchMessage)
                {
                    Batch batch = message as Batch;
                    //Type  (BulkWriteActor  Type )
                    if (batch.Obj[0] is InsertOrUpdateTestEvent)
                    {
                        using (var scope = _scopeFactory.CreateScope())
                        {
                            var eventRepository = scope.ServiceProvider.GetRequiredService<EventRepository>();
                            var elasitcEngine = scope.ServiceProvider.GetRequiredService<ElasticEngine>();

                            List<TestEvent> testEvents = new List<TestEvent>();
                            batch.Obj.ForEach(obj =>
                            {
                                var AddItem = obj as InsertOrUpdateTestEvent;
                                testEvents.Add(AddItem);
                            });

                            try
                            {
                                await elasitcEngine.BulkInsertOrUpdateToEventTable(testEvents);
                                await eventRepository.BulkInsertOrUpdateToEventTable(testEvents);
                            }
                            catch (Exception e)
                            {
                                logger.Error($"Failed BulkProductCateRelInsertOrUpdate: count:{testEvents.
Count} e:{e.Message}");
                            }
                        }
                    }
                }
            });
        }
    }
}
```

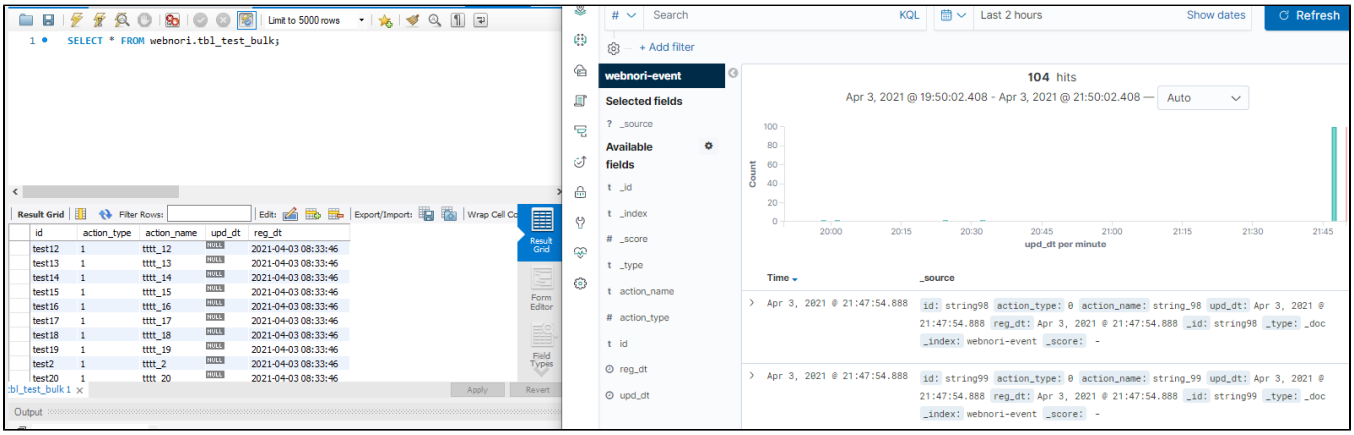
(3) .

.

| API |
| --- |
| |

```
using Akka.Actor;
using AkkaDotModule.Config;
using BulkBatchApp.Entity;
using Microsoft.AspNetCore.Mvc;
using System;
using System.Threading.Tasks;

namespace BulkBatchApp.Controllers
{
    [Route("api/[controller]")]
    public class EventController : Controller
    {
        private readonly IActorRef _insertActor;

        public EventController()
        {
            _insertActor = AkkaLoad.ActorSelect("InsertActor");
        }

        /// <summary>
        ///   .
        /// </summary>
        /// <response code="200"></response>
        /// <response code="412">
        /// ....
        /// </response>
        [HttpPost("EventRaise")]
        public async Task<string> EventRaise(
            [FromBody] InsertOrUpdateTestEvent userEvent)
        {
            _insertActor.Tell(userEvent);
            var result = "ok";
            return result;
        }

        /// <summary>
        ///    .
        /// </summary>
        /// <param name="repeat"></param>
        /// <response code="200"></response>
        /// <response code="412">
        /// ....
        /// </response>
        [HttpPost("EventBulkRaise")]
        public async Task<string> EventBulkRaise(
            int repeat,
            [FromBody] InsertOrUpdateTestEvent userEvent)
        {
            for (int i= 0; i<repeat; i++)
            {
                InsertOrUpdateTestEvent bulkEvent = new InsertOrUpdateTestEvent()
                {
                    id = userEvent.id + i,
                    action_type = userEvent.action_type,
                    action_name = userEvent.action_name + "_" + i,
                    reg_dt = DateTime.Now,
                    upd_dt = DateTime.Now
                };

                _insertActor.Tell(bulkEvent);
            }

            var result = "ok";
            return result;
        }

    }
}
```

RDB ELK .




- : https://github.com/psmon/AkkaNetBulkBatch
- FSM : 00.Finite State Machines
- RDB : https://entityframework-extensions.net/bulk-update - .