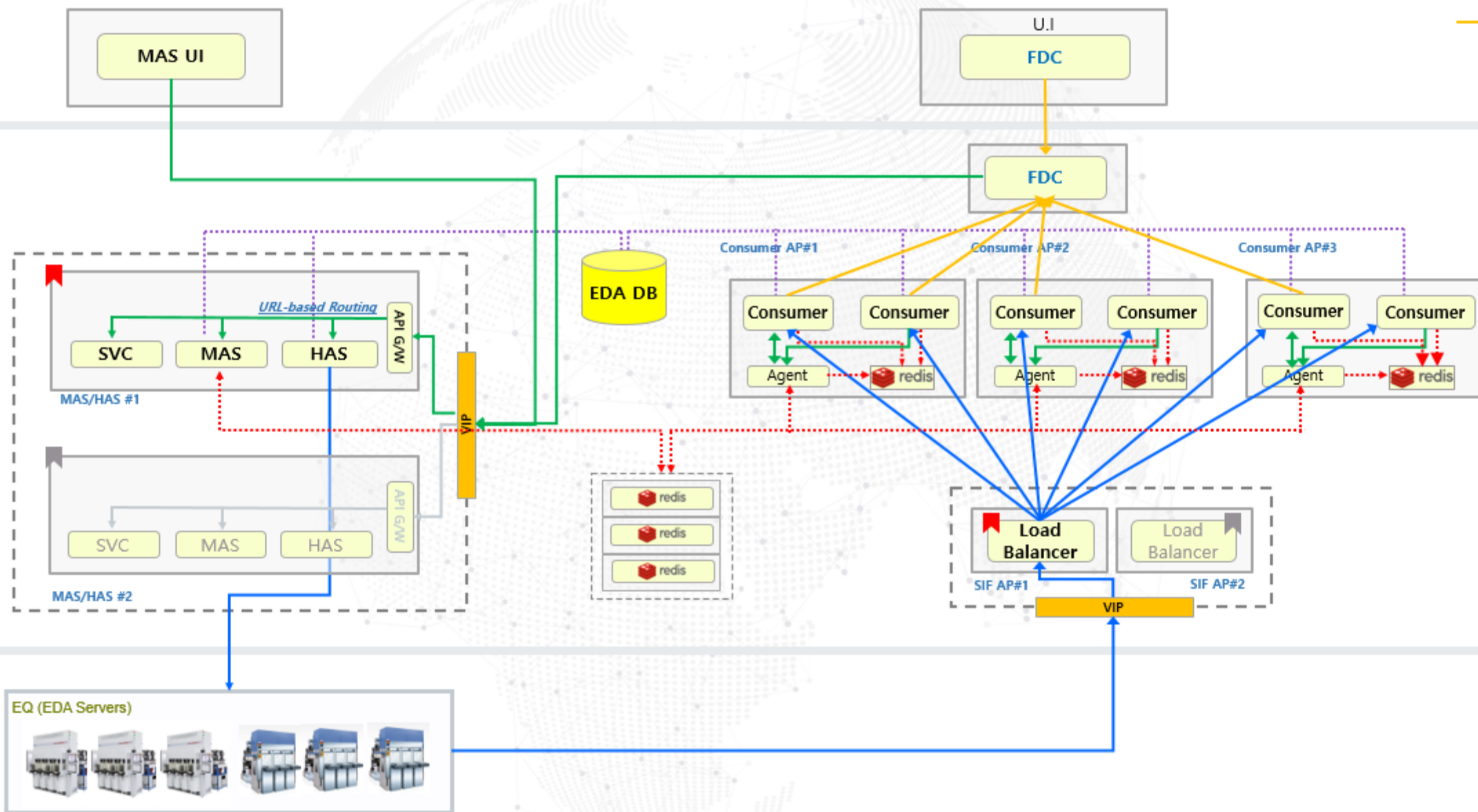
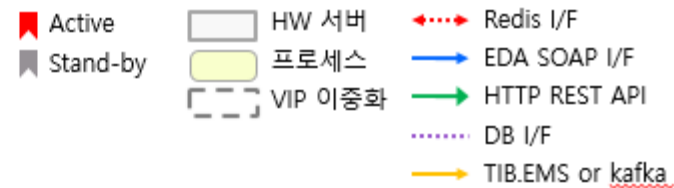


MAS-Server

Introduction

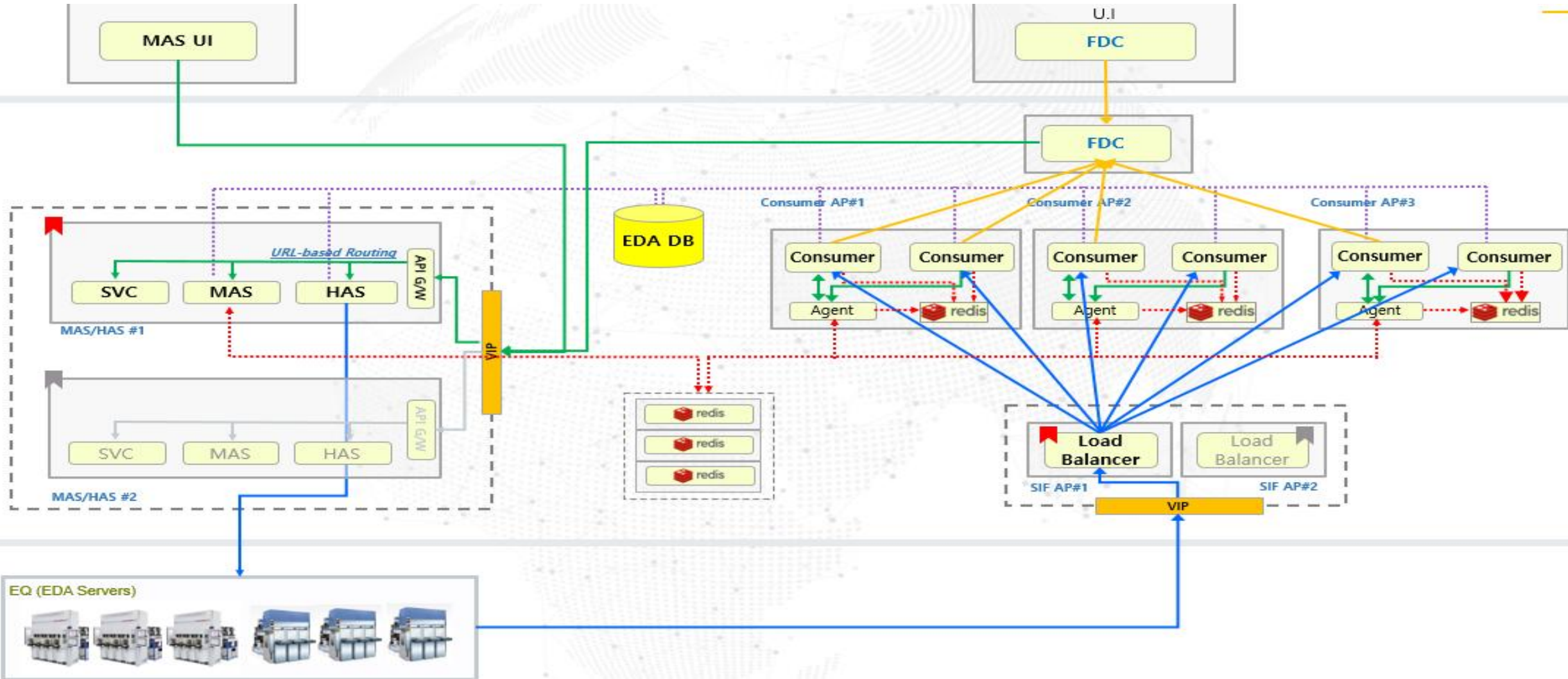
- EDA 전반적인 통합 UI 시스템 필요.
- Rest API 통신
- UI(Client) 와 EQ, Consumer, DB, Redis 를 연결 하는 Server 프로그램 필요.
- VIP 를 통해 Fail Over 기능 포함.

아키텍처 - Load balancer 기반 시스템 구성



Problem Setup

- API 통신, Redis Pub/Sub, PostgreSQL 필요.



Scope

- Controller Service Repository Pattern 적용
- Postgre DB Connection
- Redis Connection / Subscribe
- Redis Sentinel -> Master 를 따라가도록 적용.
- Proxy Master 상태 확인.

Problem solution -Controller Service Repository Pattern 적용

- Controller/Action 으로 Post – Controller 에서 Repository 호출 후 처리.

```
[ApiController]
[Route("[controller]/[action]")]
참조 2개
public partial class ApplicationDetailController : Controller
```

```
[HttpPost]
참조 1개
public partial Task<ApplicationDetailModel []> GetAllAppServerDetailAsync();
```

```
public partial async Task<ApplicationDetailModel []> GetAllAppServerDetailAsync()
{
    try
    {
        Logger.LogInformation($"GetAllAppServerDetailAsync Start");
        var applicationInstance = await this.Repository.GetAllAppServerDetailAsync();
        return applicationInstance;
    }
    catch (System.Exception e)
    {
        Logger.LogError($"Message : {e.Message}");
        Logger.LogError($"StackTrace : {e.StackTrace}");
        throw;
    }
}
```

```
public interface IApplicationDetailRepository : IDisposable
{
    참조 3개
    Task<ApplicationDetailModel []> GetAllAppServerDetailAsync();
}
```

```
public async Task<ApplicationDetailModel []> GetAllAppServerDetailAsync()
{
    비즈니스 로직

    return listApplicationStatus.ToArray();
}
```

Problem solution - Postgres DB Connection

- Appsettings.json 에서 DBType 과 Connection String 관리.

```
"DBType": "POSTGRE", // ORACLE / POSTGRE / ...  
"ConnectionStrings": {  
  "DefaultConnection": "Server=192.168.0.218;Port=5432;Database= ;User Id= ;Password= ;CommandTimeout=20;"  
},
```

```
switch (this.Configuration["DBType"].ToUpper())  
{  
  case "ORACLE":  
    optionBuilder.UseOracle(connectionString);  
    break;  
  case "POSTGRE":  
    optionBuilder.UseNpgsql(connectionString);  
    break;  
  default:  
    break;  
}
```

DBType 에 따라 Builder switch
이후 Query.xml 에 정의 된 queryId 로 쿼리 실행.

```
public static async Task<int> ExecuteRawSqlQueryAsync(this DbContext dbContext, string queryId, string dbType, ILogger Log, (string parameterName, object value)[] parameters = null)  
{  
  int rowCnt = 0;  
  
  using (Serilog.Context.LogContext.PushProperty("MethodName", nameof(ExecuteRawSqlQueryAsync)))  
  {  
    return rowCnt;  
  }  
}
```

```
public static async Task SelectRawSqlQueryAsync(this DbContext dbContext,  
  ILogger Log, string queryId, string dbType, Func<System.Data.Common.DbDataReader, Task> action, (string parameterName, object value)[] parameters = null)  
{  
  var sbParams = new System.Text.StringBuilder();  
  
  using (Serilog.Context.LogContext.PushProperty("MethodName", nameof(SelectRawSqlQueryAsync)))  
  {  
    action(dbContext.GetRawSqlQuery(queryId, dbType, sbParams).ExecuteAsync());  
  }  
}
```

Problem solution - Redis Connection / Subscribe

- 최초 한번만 구독 하도록 실행 / Master 변경 시에도 구독 유지.

```
private readonly Lazy<ConnectionMultiplexer> _lazyConnection;  
참조 44개  
public ConnectionMultiplexer Connection => _lazyConnection.Value;  
  
참조 0개  
public RedisClientFactory(IConfiguration configuration)  
{  
    this.Configuration = configuration;  
  
    _lazyConnection = new (() =>  
    {  
        var redisHostString = Configuration["MultiRedisHost"];  
        var options = ConfigurationOptions.Parse(redisHostString, ignoreUnknown: true);  
        options.AbortOnConnectFail = false;  
        options.TieBreaker = "";  
        return ConnectionMultiplexer.Connect(options);  
    });  
}
```

```
public async Task InitRedisSubscriberAsync()  
{  
    await RedisSubscriber();  
    redisClientFactory.Connection.ConnectionFailed += (s, e) =>  
    {  
        if (e.ConnectionType == ConnectionType.Subscription)  
        {  
            this.logger.LogWarning($"Redis 연결 실패: {e.EndPoint}");  
        }  
    };  
  
    redisClientFactory.Connection.ConnectionRestored += (s, e) =>  
    {  
        if (e.ConnectionType == ConnectionType.Subscription)  
        {  
            this.logger.LogInformation($"Redis 연결 복구됨: {e.EndPoint}");  
        }  
    };  
}
```

```
public async Task RedisSubscriber()  
{  
    try  
    {  
        lock (subLock) ...  
        logger.LogInformation($"Redis 구독. Status : {subscriber.Multiplexer.GetStatus()}");  
  
        await subscriber.UnsubscribeAsync(RedisChannel.Pattern(RedisConstant.PUBSUB_CONSUMER_CONFIG_DEPLOY_RESULT));  
        var queue = await subscriber.SubscribeAsync(RedisChannel.Pattern(RedisConstant.PUBSUB_CONSUMER_CONFIG_DEPLOY_RESULT));  
        queue.OnMessage ...;  
    }  
}
```


Problem solution - Proxy Master 상태 확인.

- Proxy Socket 연결 후 Command 로 데이터 가져오도록 함.

```
public async Task<string[]> GetServerStateAsync(IPEndPoint iPEndPoint)
{
    using (var socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.IP))
    {
        // 서버에 연결
        await socket.ConnectAsync(iPEndPoint);

        // 명령어 전송
        byte[] requestData = Encoding.ASCII.GetBytes("show servers state\n");
        socket.Send(requestData);
    }
}
```

```
try
{
    var url = "http://" + $"{haProxyIP}" + "/haproxy.stats.csv";
    string Fulltext;
    List<string> sb = new List<string>();
    using var request = new HttpClient();
    using (var response = await request.GetAsync(url))
    {
        foreach (var item in sb)
        {
        }
    }
}
catch (Exception e)
{
    Logger.LogError($"Message : {e.Message}");
    Logger.LogError($"StackTrace : {e.StackTrace}");
    Logger.LogError($"http:// {haProxyIP} /haproxy.stats.csv");

    //return ProxyStatusList.ToArray();
    continue;
}
```

Proxy Url 로 http 통신으로 데이터 받아 옴.

```
await socket.ConnectAsync(iPEndPoint);
byte[] requestData;

if (isEnabled)
{
    requestData = Encoding.ASCII.GetBytes($"enable server {backendname}/{servicename}\n");
}
else
{
    requestData = Encoding.ASCII.GetBytes($"disable server {backendname}/{servicename}\n");
}
socket.Send(requestData);
```

Problem solution - Proxy Master 상태 확인.

- Proxy Socket 연결 후 Command 로 데이터 가져오도록 함.
- 해당 MAS 의 IP 주소가 Proxy Master 주소 를 가지고 있는지 확인.

```
public static bool IsCurrentNodeMasterState(ILogger logger, string? useClientSideFailOver, string? vip)
{
```

```
public async Task<string[]> GetServerStateAsync(IPEndPoint iPEndPoint)
{
    using (var socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.IP))
    {
        // 서버에 연결
        await socket.ConnectAsync(iPEndPoint);

        // 명령어 전송
        byte[] requestData = Encoding.ASCII.GetBytes("show servers state\n");
        socket.Send(requestData);
    }
}
```

```
try
{
    var url = "http://" + $"{haProxyIP}" + "/haproxy.stats.csv";
    string Fulltext;
    List<string> sb = new List<string>();
    using var request = new HttpClient();
    using (var response = await request.GetAsync(url))
    {
        foreach (var item in sb)
        {
        }
    }
}
catch (Exception e)
{
    Logger.LogError($"Message : {e.Message}");
    Logger.LogError($"StackTrace : {e.StackTrace}");
    Logger.LogError($"http:// {haProxyIP} /haproxy.stats.csv");

    //return ProxyStatusList.ToArray();
    continue;
}
```

Proxy Url 로 http 통신으로 데이터 받아 옴.

```
await socket.ConnectAsync(iPEndPoint);
byte[] requestData;

if (isEnabled)
{
    requestData = Encoding.ASCII.GetBytes($"enable server {backendname}/{servicename}\n");
}
else
{
    requestData = Encoding.ASCII.GetBytes($"disable server {backendname}/{servicename}\n");
}

socket.Send(requestData);
```

Result Notes

- 통합 UI 하나로 EDA Operation 수행 가능.
- 안정적인 Redis 통신 가능
- DBType 에 따른 Swich 용이.
- Data 끊김 없이 Proxy 관리 가능.