UNISTACK

Unified Software Stack

About Us

- We are Aum Code LLC
- Group of practicing software engineers (3 key people)
- All of us have been in business for 20+ years
- Our background: medical, high-frequency trading, financial, real-time
- Our background: C/C++, Delphi, C#, Web, Erlang, Node.js
- Our background: Oracle, MsSQL, Mnesia, MongoDB, MySQL, Redis
- Our background: DOS, Windows, Linux, Android
- We love: Linux, Freedom, Big Memory, Distributed Systems, Startups
- Avoid: narrow mindset, closed-box thinking, corporate paralysis a'la MSFT

Goals

- We need to create software for users
- Users do not care how software is written, don’t care about purism
- We need to DELIVER quickly, we need to MAINTAIN/OPERATE
- We do not have time for serving the dev idols and catchy names/brands
- We are lazy, and not that smart. Our mind is limited - the less names/standards/ways we have to deal with - the better
- Names mean nothing. We have been in business for 20+ years. We know how catchy names have appeared and disappeared. A few names: REST, stateless, .COM bubble

What Would Happen...IF...

- If Bower, Grunt or Gulp disappear? - we would use npm and/or scripts
- Angular.js disappears? - we would use Backbone, React or anything else
- MS Entity Framework..? - we would use one of 100 others, Dapper anyone?
- MS MVC goes? - we would use Node, or ASP without webforms
- Log4X goes? - NLog + 20 others
- C# disappears? - we could use Java or Scala but that would SUCK!
- JavaScript disappears? - that's a **WAY bigger problem**
- **C/C++ disappears? - we are FU...ED completely!**

Bottom line: **there are important things, and the rest is pretty much noise...**

Noise Is Fragmented

But the important stuff is solid.

- C/C++ was, is, and will be for a long time. G++,VS C++,Intel C++
- C# was, is, will be for 10+ years. Either VS or Mono or DNX
- JavaScript was, is and probably will be around for 10 years
- MySQL/MariaDB, MongoDB, Riak, Mnesia, ORACLE - to stay for years
- Log4Net, NLog, Castle, Entrp Lib, Unity, Ninject, Autofac, Struts, Rails, Gulp, Grunt, Bower, JSON.NET, ServiceStack, CSLA, Telerik, Dojo, Kendo…….

Bower market share in 2020 - who knows? Node.JS in 2020 - definitely yes!

So, Lets Concentrate!

..on important stuff - **things that most server apps need**:

- Configuration, Dep Injection
- Logging with various sinks
- Security/Rights - Authenticate/Authorize
- Data Access - today no more RDBMS-only
- CRUD-like backend with automation
- Inter-process connection, queues, messaging
- Web server, some form of templatization, UI, MVC
- Data Cache
- Serialization JSON, Binary, Versioning, XML, interoperate
- UI with CRUD scaffolded yet flexible

UNISTACK

Your typical ASP.MVC 5 app:

NFX

UNISTACK

Used to be this:

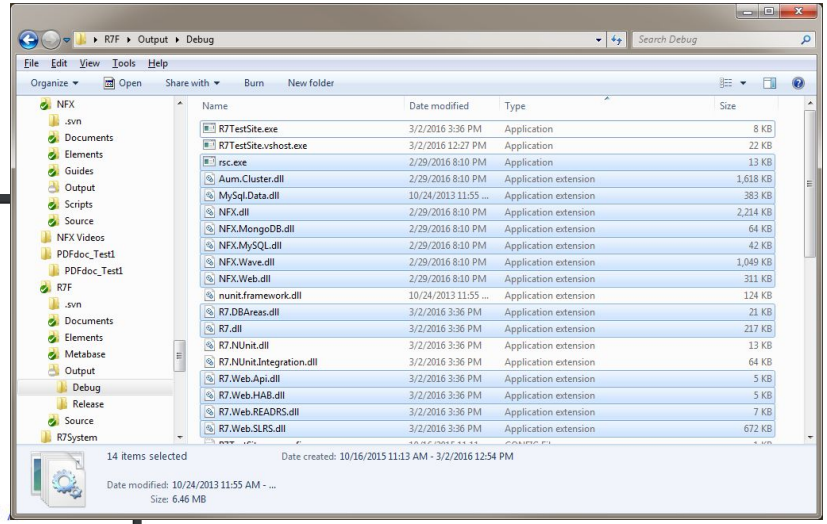
Было:

```
<?xml version="1.0" encoding="UTF-8"?>
<packages>
  <package targetFramework="net451" version="0.5.0.0" id="Abp"/>
  <package targetFramework="net451" version="0.5.0.0" id="Abp.AutoMapper"/>
  <package targetFramework="net451" version="0.5.0.0" id="Abp.Web"/>
  <package targetFramework="net451" version="0.5.0.0" id="Abp.Web.Api"/>
  <package targetFramework="net451" version="0.5.0.0" id="Abp.Web.Mvc"/>
  <package targetFramework="net451" version="0.5.0.0" id="Abp.Web.Resources"/>
  <package targetFramework="net451" version="0.5.0.0" id="Abp.Zero"/>
  <package targetFramework="net451" version="0.5.0.0" id="Abp.Zero.EntityFramework"/>
  <package targetFramework="net451" version="0.12.0" id="Angular.UI.Bootstrap"/>
  <package targetFramework="net451" version="0.2.13" id="Angular.UI.UI-Router"/>
  <package targetFramework="net451" version="0.1.1" id="Angular.UI.Utils"/>
  <package targetFramework="net451" version="3.3.0" id="AutoMapper"/>
  <package targetFramework="net451" version="3.3.3" id="Castle.Core"/>
  <package targetFramework="net451" version="3.3.3" id="Castle.Core-log4net"/>
  <package targetFramework="net451" version="3.3.0" id="Castle.LoggingFacility"/>
  <package targetFramework="net451" version="3.3.0" id="Castle.Windsor"/>
  <package targetFramework="net451" version="3.3.0" id="Castle.Windsor-log4net"/>
  <package targetFramework="net451" version="1.0" id="CommonServiceLocator"/>
  <package targetFramework="net451" version="6.1.2" id="EntityFramework"/>
  <package targetFramework="net451" version="1.0.2" id="json2"/>
  <package targetFramework="net451" version="1.2.10" id="log4net"/>
  <package targetFramework="net451" version="2.1.0" id="Microsoft.AspNet.Identity.Core"/>
  <package targetFramework="net451" version="2.1.0" id="Microsoft.AspNet.Identity.Owin"/>
  <package targetFramework="net451" version="5.2.2" id="Microsoft.AspNet.Mvc"/>
  <package targetFramework="net451" version="3.2.2" id="Microsoft.AspNet.Razor"/>
  <package targetFramework="net451" version="1.1.3" id="Microsoft.AspNet.Web.Optimization"/>
  <package targetFramework="net451" version="5.2.2" id="Microsoft.AspNet.WebApi"/>
  <package targetFramework="net451" version="5.2.2" id="Microsoft.AspNet.WebApi.Client"/>
  <package targetFramework="net451" version="5.2.2" id="Microsoft.AspNet.WebApi.Core"/>
  <package targetFramework="net451" version="5.2.2" id="Microsoft.AspNet.WebApi.WebHost"/>
  <package targetFramework="net451" version="3.2.2" id="Microsoft.AspNet.WebPages"/>
  <package targetFramework="net451" version="1.0.34" id="Microsoft.Bcl.Immutable"/>
  <package targetFramework="net451" version="3.0.0" id="Microsoft.Owin"/>
  <package targetFramework="net451" version="3.0.0" id="Microsoft.Owin.Host.SystemWeb"/>
  <package targetFramework="net451" version="3.0.0" id="Microsoft.Owin.Security"/>
  <package targetFramework="net451" version="3.0.0" id="Microsoft.Owin.Security.Cookies"/>
  <package targetFramework="net451" version="3.0.0" id="Microsoft.Owin.Security.OAuth"/>
  <package targetFramework="net451" version="1.0.0.0" id="Microsoft.Web.Infrastructure"/>
  <package targetFramework="net451" version="6.0.7" id="Newtonsoft.Json"/>
  <package targetFramework="net451" version="1.0" id="Owin"/>
  <package targetFramework="net451" version="2.1.0" id="toastr"/>
  <package targetFramework="net451" version="2.1.505.0" id="Unity"/>
  <package targetFramework="net451" version="1.6.0" id="WebGrease"/>
</packages>
```

.... has become this:

Стало:

```
<?xml version="1.0" encoding="utf-8"?>

<packages>

  <package id="NFX" version="2.0.0.11" targetFramework="net451"

</packages>
```



Complex System < 7mb

Aum Cluster Package built using UNISTACK

With Unistack

- You get 1 library, where all sub-systems work in concert
- You get **freedom FROM choice -** as all system choices are made, you can change them, but why would you use, say nLog, if you already have logger as good if not better
- You get compression. Intellectual property compression. Less things to remember and keep in your head
- You get compression. Code compression - less code.
- You get independence - you don’t care about new versions of a small lib that just does this little xyz task.
- You get UNIFICATION of thought, pattern and practice

NFX UNISTACK

- **NFX is a UNISTACK philosophy incarnation, under Apache 2**
- Is written in C# from scratch
- Uses very BCL: arrays, List<>, Dictionary<>, Thread, Monitor, Task(only basic)
- Compiles on VS and Mono. Runs on Linux and Windows
- **Has 95% of what any distributed/web/cluster app needs in 1 lib**
- Does not depend on any 3rd parties but: MySQL client (if you need it)
- Does not use MS-Specific stuff like: IIS, ActiveDirectory, WCF, MVC, EF...
- Uses 2 language: C# and JavaScript for Web UI
- Does NOT concentrate on WEB-only, as there are many other cluster/cloud application types (i.e. high frequency trading)

1.1 Any App runs in a Container

```
//An example of a web server application serving APIs, pages, Ajax via MVC or handlers
class Program
{
  static void Main(string[] args)
  {
      try
      {
         using(new ServiceBaseApplication(args, null))//<--App Container
           using(var ws = new WaveServer())//Web server
           {
              ws.Configure(null);
              ws.Start();//All services in NFX descend from Service. Similar to Erlang lite process
              Console.WriteLine("Web server started. Press <ENTER> to terminate...");
              Console.ReadLine();
           }
      }
      catch(Exception error)
      {
        Console.WriteLine("Exception leaked");
        Console.WriteLine("----------------");
        Console.WriteLine(error.ToMessageWithType());
        System.Environment.ExitCode = -1;
      }
  }
}
```

1.2 Another App

```
//An example of a network service serving clients via NFX.Glue
class Program
{
    static void Main(string[] args)
    {
      try
      {
        using(new ServiceBaseApplication(args, null))//App container
        {
            ConsoleUtils.Info("Server is running. ");

            ConsoleUtils.Info("Glue servers:");//Print the server enpoints
            foreach(var s in ServiceBaseApplication.Glues.Servers)
              Console.WriteLine("  " + s);

            Console.WriteLine();
            ConsoleUtils.Info("Press <enter> to end server program");
            Console.ReadLine();
        }
      }
      catch(Exception error)
      {
        ConsoleUtils.Error(error.ToMessageWithType());
        Environment.ExitCode = -1;
      }
    }
}
```

1.3 App Container Feeds from Config

```
//An example of a network service config
 glue
 {
   server-log-level="Debug"
   client-log-level="Debug"

   bindings//determine the protocol, encoding and serialization
   {
     binding { name="sync"   type="NFX.Glue.Native.SyncBinding" }
     binding { name="inproc" type="NFX.Glue.Native.InProcBinding, NFX" }
     binding { name="mpx"    type="NFX.Glue.Native.MpxBinding, NFX" }
   }

   Servers //Server Endpoints
   {
     server {name="local" node="inproc://localhost" contract-servers="TestServer.Glue.JokeServer, TestServer;
TestServer.Glue.JokeCalculatorServer, TestServer"}
       server {name="sync"  node="sync://*:8000"   contract-servers="TestServer.Glue.JokeServer, TestServer;
TestServer.Glue.JokeCalculatorServer, TestServer"}
     server {name="mpx"   node="mpx://*:5701"  contract-servers="TestServer.Glue.JokeServer, TestServer;
TestServer.Glue.JokeCalculatorServer, TestServer"}
   }
 }//glue
```

2.1 Configuration

- Tree in memory
- Variables, env variables
- Scripting (Turing complete language)
- Structural overrides, inheritance, merges
- Includes/subtrees
- Includes from remote/injectable file systems
- Laconic format, XML, JSON
- <5000 LOC 2 interfaces and 3 classes to work with
- All UNISTACK components are configurable from the same source
- Ability to store config in cloud/cluster (is not necessarily a file)
- Plugable var/macros resolvers (i.e. string/time formatting)

3.1 NFX.Log

```
nfx
{

   disk-root=$"c:\nfx\"//This is a variable!
   log-root=$(/$disk-root)
   log-csv="NFX.Log.Destinations.CSVFileDestination, NFX"
   debug-default-action="Log,Throw"
   trace-disable=true

  log
  {
    name="Logger"
    destination
    {
       type=$(/$log-csv)
      name="IntegrationTestServer.Log"
      path=$(/$log-root)//which is referenced here and possibly in 10 other places
       name-time-format='yyyyMMdd'
       generate-failover-msg=false
    }
  }
}

Anywhere in code:

 App.Log.Write( new LogMessage{Type=MessageType.Info, Topic=GetType().Name, From=nameof(MyMethod), Text="Hello" });
```

3.2 NFX.Log Destinations (Sinks)

- Async or Sync
- Hierarchical
- Filter on dates, times, severity, injectable filter expressions
- Failover, SLA (if destination fails, route to another)
- Flood filter (gather many messages in 1 i.e. before emailing)
- CSV File, SMTP, Unix Syslog, MongoDB, MsSQL, MySQL etc…
- <5000 LOC 1 class to remember LogMessage. Component logger via mapper function
- 100% configurable from App container - may use vars etc.

4.1 NFX.Glue

- Glue instances between nodes/processes
- No data special contracts required
- **CLR object Teleportation via Slim**
- Design by contract
- Pluggable bindings
- Security: declarative or imperative
- Stateful or stateless server/client
- 150,000 ops/sec 2 way calls on a 4 core machine 3.6 GHz
- <10000 LOC <10 classes to remember
- 100% configurable from App container - may use vars etc.

4.2 NFX.Glue Example

```
    [Glued]
    [LifeCycle(ServerInstanceMode.Stateful, 20000)]
    [AuthenticationSupport]
    public interface IJokeCalculator
    {
       [Constructor]
       void Init(int value);

       [SultanPermission( 250 )] //notice declarative permissions
       int Add(int value);
       int Sub(int value);

       [Destructor]
       int Done();
   }

Anywhere in code:

 using(var cl = new JokeCalculator(new Node("sync://192.168.1.23:8934")))
 {
    cl.Init(234);
    cl.Add(3);
    Assert.AreEqual(237, cl.Done());
 }
```

5.1 Serialization - CLR Object Teleportation

- In UNISTACK everything uses the same runtime, so we can
- **Teleport object instances between cluster nodes**
- We don’t need to serialize in XML or JSON if we talk to internal nodes
- Internal nodes: DB Backend, middle tier, cache server etc.
- Teleporation = binary serialization that DOES NOT require special measures like DataContracts. Objects get “teleported” as-is, for example: Dictionary<string,List<Car>> may be teleported WITHOUT any extra work
- NFX.Serialization.SlimSerializer = Teleportation Device used by Glue
- Slim is 5-10 times faster than MS BinaryFormatter. See Serbench tool
- Slim uses sophisticated runtime expression tree generator with caching
- Slim supports polymorphism, readonly fields, ISerializable etc.
- Slim DOES NOT need extra DTO types!

5.2 Serialization - JSON

- In UNISTACK must support widespread JSON use. Everything is built-in
- NFX.Serialization.JSON
- Serialize: classes, structs, List, Dictionary, Rows
- Deserialize into MVC form models, Rows, JSONDataMap/JSONDataArray/dynamic

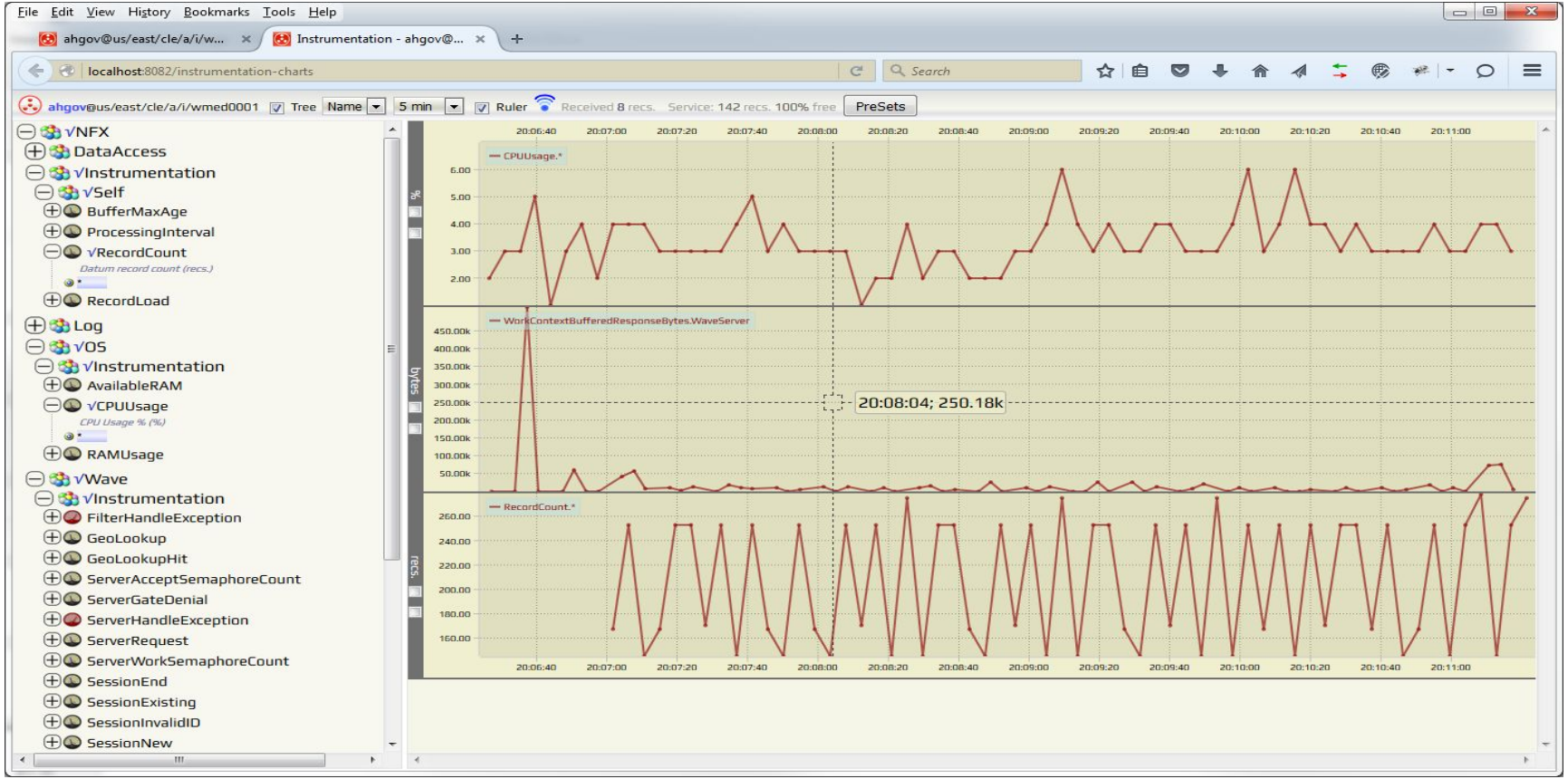
```
//NFX WAVE MVC example
[Action("match{method=GET}")]
public object GetTotal(MyComplexForm form)//notice how complex form gets injected from JSON
{
  var result = form.Balances.Sum();
  if (RequestedJSON)//if accept-type == app/json
   return {OK = True, original = form, total = result };
  else
   return new TotalView(result);//return the view
}
```

6.1 Instrumentation + Telemetry

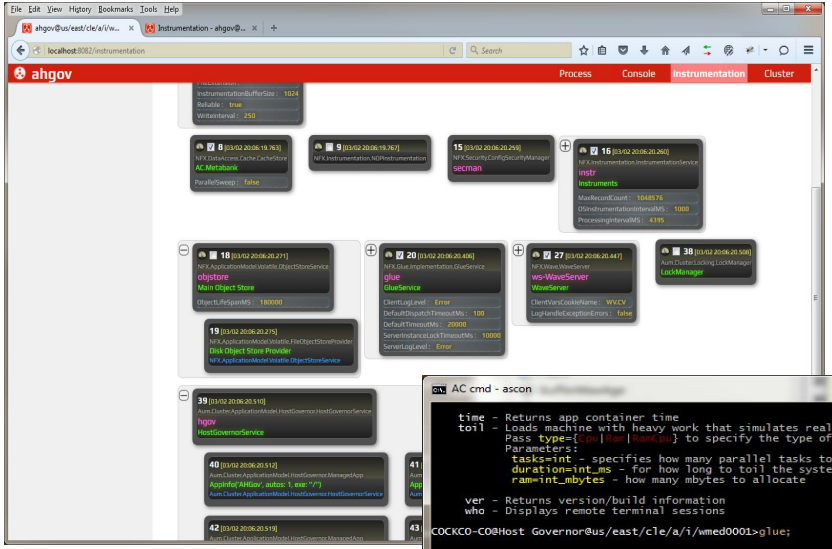
- Need to know what is going on in the process
- In cluster, need to know what is going in cluster zones, groups
- Fast and asynchronous
- Cluster does real-time host/zone/region telemetry MAP:REDUCE

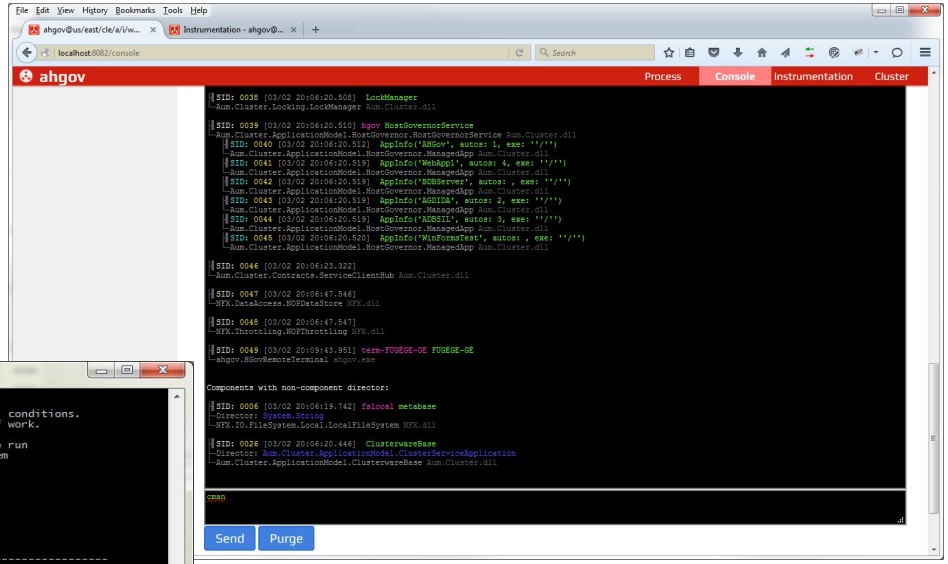
```
/// <summary>
/// How many response bytes were buffered
/// </summary>
[Serializable]
public class WorkContextBufferedResponseBytes : WaveLongGauge, INetInstrument, IMemoryInstrument
{
   internal WorkContextBufferedResponseBytes(string src, long value) : base(src, value) {}

   public override string Description { get{ return "How many response bytes were buffered";} }
   public override string ValueUnitName  { get { return NFX.CoreConsts.UNIT_NAME_BYTE; } }
   protected override Datum MakeAggregateInstance() { return new WorkContextBufferedResponseBytes(this.Source, 0); }
}

Use:
App.Instrumentation.Record(new WorkContextBufferedResponseBytes(httpListener.URL, totalSent));
```

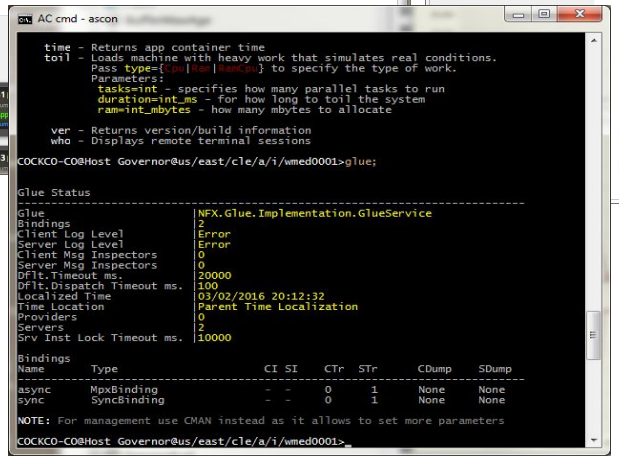
6.2 Visualize Telemetry



6.3 Instrument your App Container







7.1 Data Access

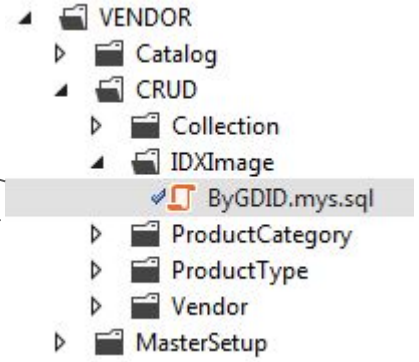
- Hybrid data stores: RDBMS, NoSQL, Services, Event Sourcing
- **Virtual Queries - code decoupled**
- CQRS - Command Query Responsibility Segregation (not mandatory)
- Model multi-targeting. Call fields different names depending on target
- Domain-Organized facades - Data Stores as interface groups
- CRUD auto gen for RDBMS, NoSQL, services (i.e. Erlang backend)
- Schema: DynamicRows, TypedRows, Rowsets, Amorphous data
- Validation per target: req, max/min/regexp, phone/email/date + metadata
- RSC Schema Compiler - write DB model as configuration
- GDID - global distributed ID 2^96 monotonically increasing 12 bytes
- Serialization Row > JSON > Row
- Domain model segregation: Data Models, Form Models, Filter Models

7.2 Data Access: TypedRow

```
[Table(targetName: SysConsts.R7_DS_MYSQL_TARGET, name: "tbl_idximage")]
[UniqueSequence(typeof(BLOB.BLOBRow))]
public class ImageRow :  R7RowWithGdidPK
{
  public ImageRow():base(){}
  [Field(required: true, description: "Vendor that this image is under", visible: false)]
  public GDID G_Vendor { get; set; }

  [Field(required: true, nonUI: true, kind: DataKind.DateTime)]
  public DateTime Modify_Date { get; set; }

  [Field(required: true,
         kind: DataKind.Text,
         maxLength: Domains.R7Mnemonic.MAX_LEN,
         description: "Catalog Name",
         metadata: @"Placeholder='Catalog Name' Hint='Catalogs are used for classification'")]
  public string Catalog { get; set; }

  [Field(required: true,
         kind: DataKind.Text,
         maxLength: Domains.R7Mnemonic.MAX_LEN,
         description: "Directory Name",
         metadata: @"Placeholder='Catalog Name' Hint='Directories are used for classification within catalogs'")]
  public string Directory { get; set; }

  [Field(kind: DataKind.Text,
         maxLength: Domains.R7Description.MAX_LEN,
         description: "Description",
         metadata: @"Placeholder='Description' Hint='Describe what the image is for, i.e. use SKU#'")]
  public string Description { get; set; }
}
```

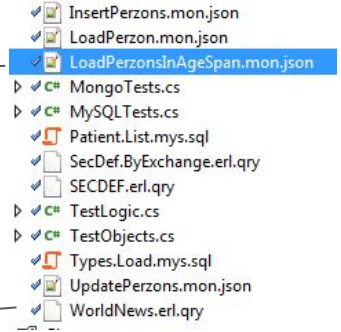
7.2 Data Access: Virtual Queries

```
public static Query<TRow> IDXImageByGDID<TRow>(GDID G_Vendor, GDID gdid) where TRow : Row
{
  return new Query<TRow>("Vendor.CRUD.IDXImage.ByGDID")
  {
    new Query.Param("pGDID", gdid),
    new Query.Param("pG_VENDOR", G_Vendor)
  };
}

var row = R7App.Data.Vendor.LoadRow(QVendor.IDXImageByGDID<ImageRow>(G_Vendor, imgGDID));
```



MySQL:

```
SELECT *
FROM -- can join tables, however only update some columns
 TBL_IDXIMAGE T1
WHERE
 (T1.`GDID` = ?pGDID) AND
 (T1.`G_VENDOR` = ?pG_VENDOR)
```



Mongo DB

```
#pragma
modify=MyPerzon

{ '$query': {'Age': {'$gt': '$$fromAge', '$lt': '$$toAge'}} , '$orderby': {'Age': -1} }
```

Erlang MFA (Module:Function:Arguments)

```
   nfx_test:world_news(Subscriber::pid(), Timestamp::long(), Count::int(), Period::int(*)
```

8.1 Big Memory

- Managed runtimes CAN NOT handle > 10MM objects without pauses
- Sometimes even 10 MM resident objects start to pause process
- GC Heap defragmentation kills your app by adding unpredictable stalls
- The more physical RAM you have, the more unpredictability. **Forget 64/128/256GB**
- All of the modern techniques (concurrent, background, parallel, server mode) tried
- CLR objects consume lots of ram. I.e. a string “ABC” holds around 30 bytes
- Many objects are needed for many apps: neural networks, caches, pre-computed data
- Java has the same issue, if their VMs have way more options to control GC
- The root of the problem: the GC “sees” references and has to traverse them to see what is still reachable and what is not
- Stateless design sucks. Need to have caches right in-process
- How fast could an app run if most of data were already in ram?
- Impossible to postpone GC for a long time
- Unmanaged object copies can not work 1:1 with CLR objects
- The way to go - “hide” data from GC
- Terracota did Big Memory for Java
- No such concept in CLR world.. Until recently

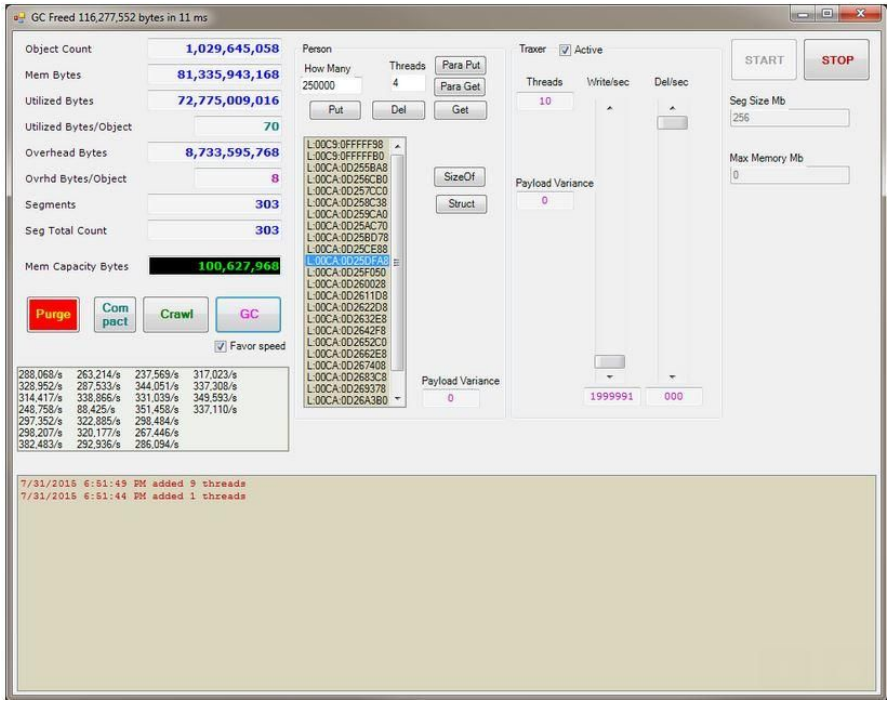
8.2 Big Memory Pile

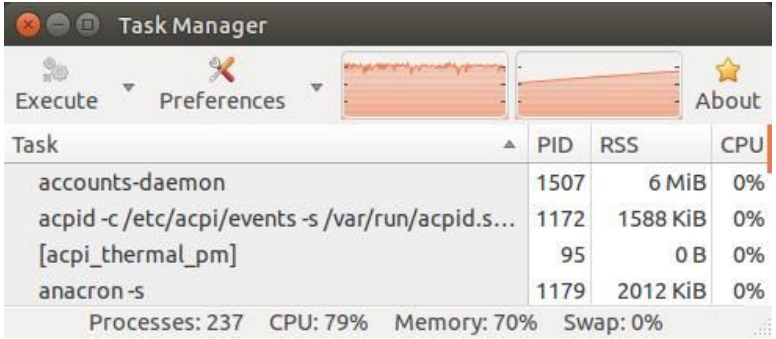
- Hold objects in huge byte[]. **GC does not see them**
- Uses teleportation device (Slim). **Compresses 25% over CLR**
- Turns CLR object into struct{int,int} and back
- Manages “unmanaged” managed memory for you!

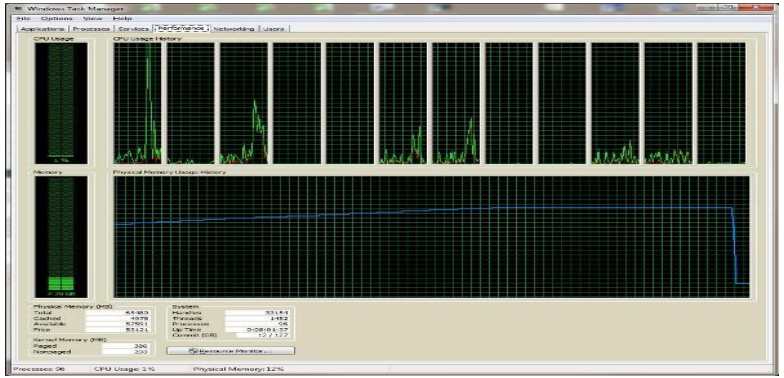
```
/// <summary>
/// Represents a pointer to the pile object (object stored in a pile).
/// The reference may be local or distributed in which case the NodeID is>=0.
/// Distributed pointers are very useful for organizing piles of objects distributed among many servers, for example
///  for "Big Memory" implementations or large neural networks where nodes may inter-connect between servers.
/// </summary>
public struct PilePointer : IEquatable<PilePointer>
{
.....
    /// <summary>
    /// Distributed Node ID. The local pile sets this to -1 rendering this pointer as !DistributedValid
    /// </summary>
    public readonly int NodeID;

    /// <summary>
    /// Segment # within pile
    /// </summary>
    public readonly int Segment;

    /// <summary>
    /// Address within the segment
    /// </summary>
    public readonly int Address;

}
```

8.3 Big Memory Pile Facts

- Simple data record (10 fields “Person”) 1 thread: put 400K/sec get: 550K/sec
- 100% thread safe. 6 core machine: put 1.3MM/sec get: 2+MM/sec. Full GC <15ms
- 64GB = 300,000,000 “Person” objects, still 10 Gb free. Full GC <20 ms
- 64GB machine, put 1,000,000,000 “Persons” swap into 85+GB. Full GC <30ms







8.4 Big Memory Pile Use

- We easily store 200,000,000 “hot” social connections with ability to traverse them in <1 ms by a single thread on a 64 Gb box
- Most of our apps turned from IO bound to CPU bound(**pile teleportation**)
- There is cache. We cache most of DB (except for financial rollups)
- Pile is great for Event Sourcing
- Because of pile we don’t need to make ANY socket calls to Redis/Memcached
- We don’t even need IPC/domain sockets
- The cache (NFX.ApplicationModel.Pile.Cache)has expiration, priority, low/high watermarks
- We tend to be stateful everywhere
- We tend to avoid talking to data layers at all, 95% requests are served from RAM
- IMPORTANT: we do not need to invent special DTOs, as pile/cache stores Rows (along with logic)
- **This is classical OOP! Data is here, it has methods, it has state!**
- Pile promotes stateful data, yet in an Actor-like sense **you get a copy**
- Pile pointer becomes an Actor token
- **In spite of teleportation involved, Pile yields FAR faster throughput and better latency had we used plain CLR objects that would have just stalled the process**

9.1 NFX.WAVE - Web Programming

- In UNISTACK **Web Programming is just the one of many things** that need to be done (unlike say PHP, Node that grew out of web)
- NFX.WAVE Server **does not use IIS**
- NFX.WAVE = Server + WAVE.js client (but you can use Angular etc.)
- NFX.WAVE is a **hybrid Http server**, filter/handler pipeline, templatization, MVC
- **Session state and security is built into NFX app container**, so a user from web may execute a Glue call to another machine under the same identity
- Server processing model is hybrid: **WorkContext may be served by a single thread** OR **many** WorkContext instances **may be served by the same thread**, i.e. chat application
- The programming model may be either: REST/Postback/MVC/Socket
- **We can be 100% stateless or 100% stateful**. The server side state (if you need it) is serviced by NFX App container that can survive process restarts
- Server knows how to scaffold metadata from Rows into reactive WAVE.js client
- The whole thing is < 10,000 LOC
- Everything is config-driven and can be set by code at runtime
- Attributes before/after. Session check, CSRF, NoCache, Security, UserIdentityConfirm…
- Flood Gate

9.2 A Simple Site - Start from Config

```
application
{
    wave
    {
        server
        {
            prefix { name="http://+:8080/" }

            dispatcher
            {
                handler
                {
                    type="NFX.Wave.Handlers.MVCHandler, NFX.Wave"
                    type-location { assembly="Wave.HelloWorld.exe" ns { name="Wave.HelloWorld.Controllers" } }
                    match { path="/{type=Hello}/{mvc-action=Index}" }
                }
            }
        }
    }
}
```

9.3 A Simple Site - Add Hosting Code

```
using System;
using NFX.ApplicationModel;
using NFX.Wave;

namespace NFXDemos.Wave.HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                using (var app = new ServiceBaseApplication(args, null))
                  using (var server = new WaveServer())
                  {
                      server.Configure(null);
                      server.Start();
                      Console.WriteLine("server started...");
                      Console.ReadLine();
                  }
            }
            catch (Exception ex)
            {
                Console.WriteLine("Critical error:");
                Console.WriteLine(ex);
                Environment.ExitCode = -1;
            }
        }
    }
}
```

9.4 A Simple Site - Add Controller

```
using System;
using NFX.Wave.MVC;
using Wave.HelloWorld.Pages;

namespace Wave.HelloWorld.Controllers
{
    public class Hello : Controller
    {
        [Action]
        public object Index()
        {
            return new Index();//This is view
        }
    }
}
```

Adding controller, and using MVC is not required, you could just serve dynamic pages directly, just change a line in config:

```
       handler
       {
            type="NFX.Wave.Handlers.TemplateHandler, NFX.Wave"
            type-location { assembly="Wave.HelloWorldNoMVC.exe" ns { name="Wave.HelloWorldNoMVC.Pages" } }
            match { path="/{type=Index}" }
       }
```

9.5 A Simple Site - Add a View/Page

```
#<laconf>
  compiler
  {
    using { ns="NFX" }
    using { ns="NFX.Wave" }
    using { ns="NFX.Wave.Templatization" }
    base-class-name="NFX.Wave.Templatization.WaveTemplate"
    namespace="Wave.HelloWorldNoMVC.Pages"
  }
#</laconf>
<!DOCTYPE html>
<html>
<head>
  <style>
    #root {
      width: 100%;
      margin-top: 100px;
      text-align: center;
      font-family: Verdana;
      font-size: 30px;
    }
  </style>
</head>
<body>
  <div id="root">Hello WAVE on ?[DateTime.Now]!</div>
</body>
</html>
```

9.6 A Simple Site - Lets add Data

```
using NFX.DataAccess.CRUD;
using System;

namespace Wave.DbApplication.Models
{

    public class Person : TypedRow
    {
        [Field(key: true, required: true)]
        public string ID { get; set; }

        [Field(required: true, maxLength: 32, description: "First Name")]
        public string FirstName { get; set; }

        [Field(required: false, maxLength: 32, description: "Middle Name")]
        public string MiddleName { get; set; }

        [Field(required: true, maxLength: 32, description: "Last Name")]
        public string LastName { get; set; }

        [Field(required: false, kind: DataKind.Date, description: "Date of Birth")]
        public DateTime DOB { get; set; }

        [Field(required: true, kind: DataKind.EMail, description: "Primary E-mail")]
        public string EMail { get; set; }
    }
}
```

9.7 A Simple Site - Lets add Data Controller

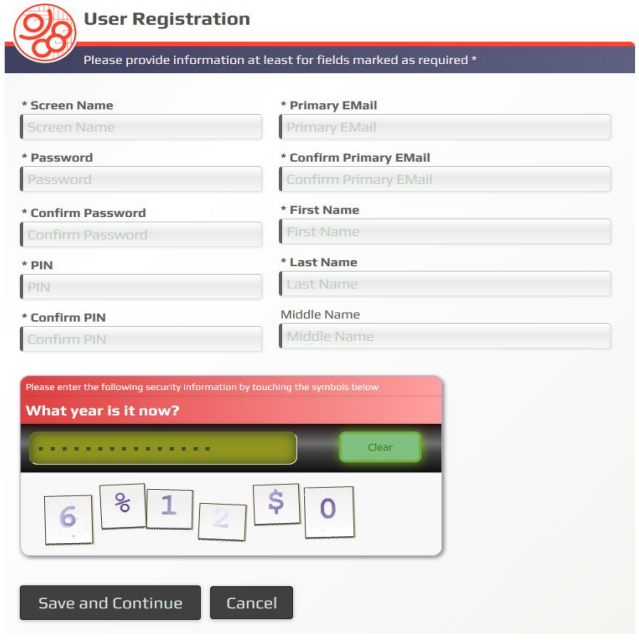
```
[Action(name: "edit", order: 0, matchScript: "match{methods=GET}")]
public object GetEdit(string personId)
{
   Person person = null;
   if (personId.IsNotNullOrWhiteSpace())
   {
      var query = new Query("Data.Scripts.GetPersonByID", typeof
(Person))
                  {
                     new Query.Param("id", personId)
                  };
      person = AppContext.DataStore.LoadOneRow(query) as Person;
   }

   if (person == null)
     person = new Person { ID = Guid.NewGuid().ToString(") };

   return new Edit { Person = person };
}
```

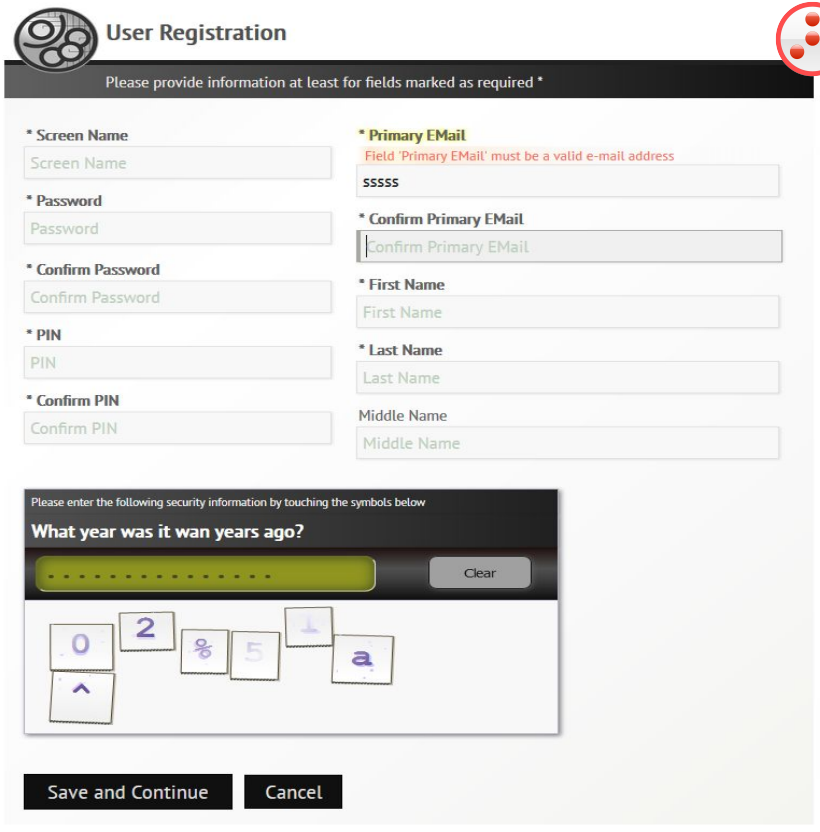
```
[Action(name: "edit", order: 0, matchScript: "match{methods=POST}")]
public object SaveEdit(Person person)
{
  if (person == null)
      person = new Person { ID = Guid.NewGuid().ToString("N") };

  var error = person.Validate();
  if (error == null)
  {
     AppContext.DataStore.Upsert(person);

     if (WorkContext.RequestedJSON)
        return new ClientRecord(person, null);
     else
        return new Redirect("/");
  }

  if (WorkContext.RequestedJSON)
    return new ClientRecord(person, error);
  else
    return new Edit { Person = person, ValidationError = error };
}
```

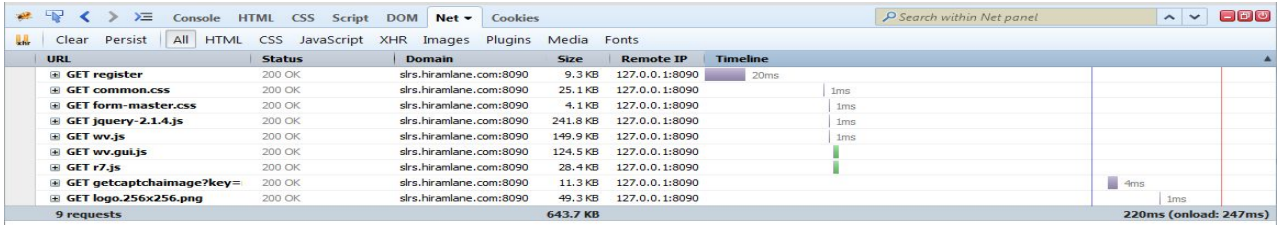
9.8 A Simple Site - Add Reactive View

```
#[override renderBody()]
<div id="form-container">
  <form id="frmContact" data-wv-rid="v1" action="/contacts/edit" method="POST" novalidate>
    <div data-wv-fname="FirstName"></div>
    <div data-wv-fname="MiddleName"></div>
    <div data-wv-fname="LastName"></div>
    <div data-wv-fname="DOB"></div>
    <div data-wv-fname="EMail"></div>
    <input type="submit" value="Submit" />
  </form>
</div>

<script>
  var REC = new WAVE.RecordModel.Record(?[:AppContext.FormJSON(Person, ValidationError)]);
  var RVIEW = new WAVE.RecordModel.RecordView("v1", REC);
  $('#frmContact').submit(function (e) {
    if (!REC.validate())
    {
      WAVE.GUI.toast('Please correct all validation errors marked in red','error');
      e.preventDefault();
      return false;
    }
    REC.resetModified();
    return true;
  })
  @[if (ValidationError != null){]
     WAVE.GUI.toast('Data validation error.<br/>Please, reenter data carefully.', 'error', 5000);
  @[}]
</script>
```

9.9 9 kb page...70K/sec







9.10 Real Code With Attributes

```
  [NoCache]
  [Filters.UserIdentityConfirmation]
  [R7.Security.Permissions.VendorUserPermission]
  public sealed class MasterSetup : VendorControllerBase
  {

… . . . . . .

    [Action("imagedetails", 1, "match { methods=GET,POST }")]
    public object ImageDetails(GDID? gImage, IDXImageForm form)
    {
     …
    }

    [Action("imagedetails", 2, "match { methods=DELETE accept-json=true }")]
    [SessionCSRFCheck]
    public object ImageDetails_DELETE(GDID gImage)
    {

  ……

     }
    protected override bool BeforeActionInvocation(WorkContext work,
                                                   string action,
                                                   MethodInfo method,
                                                   object[] args, ref object result)
    {
       return …...
    }
. . . . .
  }
```

Links and Resources

- Code https://github.com/aumcode/nfx
- Demos https://github.com/aumcode/nfx-demos
- Guides https://github.com/aumcode/nfx/tree/master/Guides
- Serbench Tool https://github.com/aumcode/serbench
- **Big Memory 1** http://www.infoq.com/articles/Big-Memory-Part-1
- **Big Memory 2** http://www.infoq.com/articles/Big-Memory-Part-2
- NFX Erlang https://youtu.be/o9utCAMLydA
- NFX WAVE (In Russian) https://www.youtube.com/channel/UCKv4mLAN-XjZF2ST0cT6pAQ
- NFX WAVE 50K/sec quad core https://youtu.be/F0MKPUD2bZ8
- Store 300,000,000 in CLR heap https://youtu.be/Dz_7hukyejQ
- http://blog.aumcode.com/ (does not get updated often)
- dmitriy@itadapter.com skype:**itadapter**