Building cloud-native services with Project Orleans
Cloud-Native Services

Term coined by Hoop Somuah

Services that are built for the cloud

- Reliable
- Scalable
- Elastic
- High throughput, low latency
- Fast to build and iterate
- DevOps friendly

No lift-and-shift
What is Project “Orleans”? 

Oversimplifying: “Distributed C#”
- Orleans runs your .NET objects on a cluster as if in a single process
- Define .NET interfaces and classes, deploy, send requests to them

Practically: “Toolset for building cloud-native services”
- Encapsulates best practices for building cloud-native services
- Framework for stateful near-real-time backends
- 3-5x less and simpler code to write, scalability by default

Academically: “Distributed virtual actor model”
- Adaptation of the Actor Model for challenges of the Cloud
- Actors that exist eternally and never fail
Patterns of a Big Game Launch

- Huge traffic spike on launch
- Downtime at launch is really bad
- Also spikes on weekends and holidays
- Load steadies out over time
Developer Experience
Distributed runtime

Built for .NET

- Actors (*Grains*) are .NET objects
- Messaging through .NET interfaces
- Asynchronous through async/await in C#
- Automatic error propagation

*Silo*: runtime execution container

- Implicit activation & lifecycle management
- Coordinated placement
- Multiplexed communication
- Failure recovery
public interface IHello : I Grain With Integer Key
{
    Task<string> SayHello (string name);
}
public class HelloGrain : Grain, IHello
{
    private int _counter;

    public async Task<string> SayHello (string name)
    {
        return string.Format(
            "Hello {0}. You are caller #{1}", name, _counter++);
    }
}

‘Hello World’ in Orleans – Implementation
‘Hello World’ in Orleans – Invocation

GrainClient.Initialize(); // client-only

IHello grainRef = GrainFactory.GetGrain<IHello>(0);
string reply = await grainRef.SayHello (name);
Console.WriteLine("HelloGrain said:" + reply);
public interface IUser : IGrain
{
    Task<string> GetName();
    TaskSetName(string name);

    Task<string> GetStatus();
    Task UpdateStatus(string status);

    Task<List<IUser>> GetFriends();
    Task AddFriend(IUser friend);
    Task<string> GetFriendsStatus();
    Task<List<string>> GetFriendsUpdates();
}
IUser me = GrainFactory.GetGrain<IUser>(myId);
IUser friend = GrainFactory.GetGrain<IUser>(friendId);

try
{
    await me.AddFriend(friend);
    Console.WriteLine("Added friend {0}.", friendId);
}
catch(Exception exc)
{
    Console.WriteLine("Failed to add {0} as friend: {1}", friendId, exc);
    throw;
}
Beyond ‘Hello World’ – Grains Class

```csharp
public class UserGrain : Grain, IUser
{
    private List<IUser> _friends;

    public async Task<string> GetFriendsStatus()
    {
        var tasks = new List<Task<string>>();
        foreach (var friend in _friends)
        {
            tasks.Add(friend.GetStatus());
        }

        await Task.WhenAll(tasks);

        var sb = new StringBuilder();
        foreach (var t in tasks)
        {
            sb.AppendLine(t.Result);
        }

        return sb.ToString();
    }
}
```

- Extend Grain
- Implement grain interface(s)
- Exclusive access to private fields
- No multi-threading
- Easy parallelism
- Handle returned TPL Task’s properly
- Just like in a desktop app
Lots More Features...

Automatic cluster membership, recovery from failures
Automatic resource management, elasticity
Flexible placement policies
Grain timers and reminders
Support for persistence with a provider model
Support for streaming event processing

...
Orleans Benefits

Very easy to program reliable distributed/cloud systems
Scalability by default
Uncompromised performance
Proven in many production services
Runs anywhere
Open source!
How You Can Benefit

A vibrant open source project to leverage

- Easy enough for undergrads
- Deep enough for PhD students
- Architected for the Cloud, great fit for IoT, social, gaming, even workflow

Build distributed scalable apps/services/systems in ‘user’ mode

Build system components/algorithms in ‘kernel’ mode (runtime)

Contribute to code used in production systems
Orleans Is Open Source

On GitHub under an MIT license

GitHub is the ‘master branch’
Active and growing community that
never sleeps

Easy to contribute
Pride of ownership – priceless

Join and enjoy the fun!

---

**June 7, 2015 – July 7, 2015**

**Overview**

<table>
<thead>
<tr>
<th>65 Active Pull Requests</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>60</strong> Merged Pull Requests</td>
</tr>
</tbody>
</table>

Excluding merges, **14 authors** have pushed **90 commits** to master and **144 commits** to all branches. On master, **465 files** have changed and there have been **19,964 additions** and **11,036 deletions**.
Orleans on GitHub:  
https://github.com/dotnet/orleans

Documentation:  
http://dotnet.github.io/orleans/

Ideas for Research and Course Projects:  
http://dotnet.github.io/orleans/Student-Projects
Backup
Distributed Runtime
Distributed Runtime

Messaging is multiplexed over a small number of TCP connections

Actor directory is a custom DHT

Single-threaded execution on a small number of threads, one per core

Performance benefits from cooperative multitasking

Actor activation management

- Automatic instantiation and placement (default is random)
- Garbage collection of idle activations

Custom cluster membership protocol, no Paxos
3-Tier Architecture

- Stateless frontends
- Stateless middle tier
- Storage is the bottleneck
  - Latency
  - Throughput
  - Scalability
- Horizontal calls are problematic
- Data shipping
Cache Tier for Performance & Scalability

- Much better performance
- Lost semantics of storage
- Lost concurrency control
- Horizontal calls are still problematic
- Still data shipping
Actors as Stateful Middle Tier

- Performance of cache
- Rich semantics
- Concurrency control
- Horizontal calls are natural
- OOP paradigm regained
- Function shipping
- But there are still problems...
Community That Never Sleeps

US
UK
Australia
Finland
Ukraine
Hungary, Netherlands
...