НАЗАД В БУДУЩЕЕ

Построение эффективных облачных сервисов

с помощью Orleans

Сергей Быков
Microsoft
@sergeybykov

Microsoft

Обо мне

В Microsoft с 2001 г.

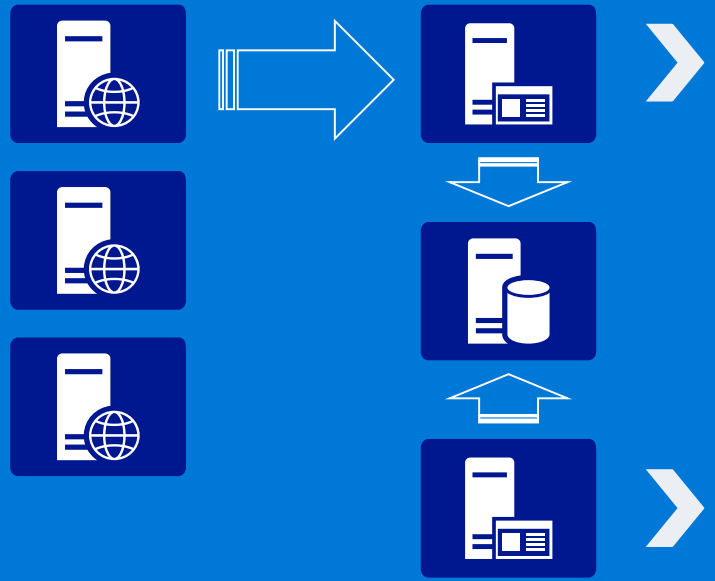
- Host Integration Server
- BizTalk Server
- Windows Embedded
- Bing
- Research
- Halo
- Xbox

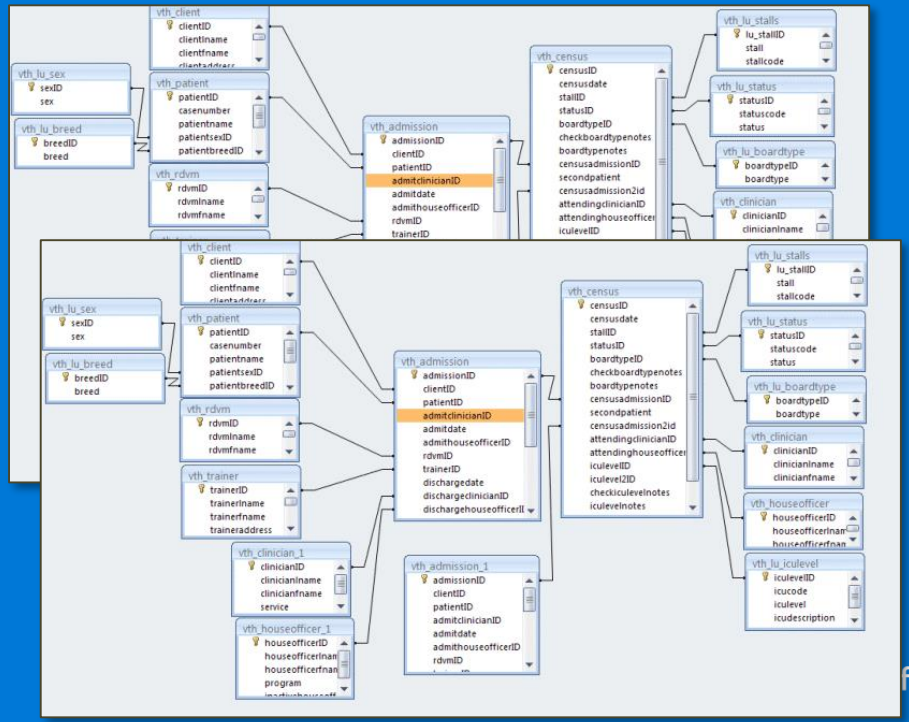
Microsoft

В старые добрые *stateful* времена

Клиенты

Сервер



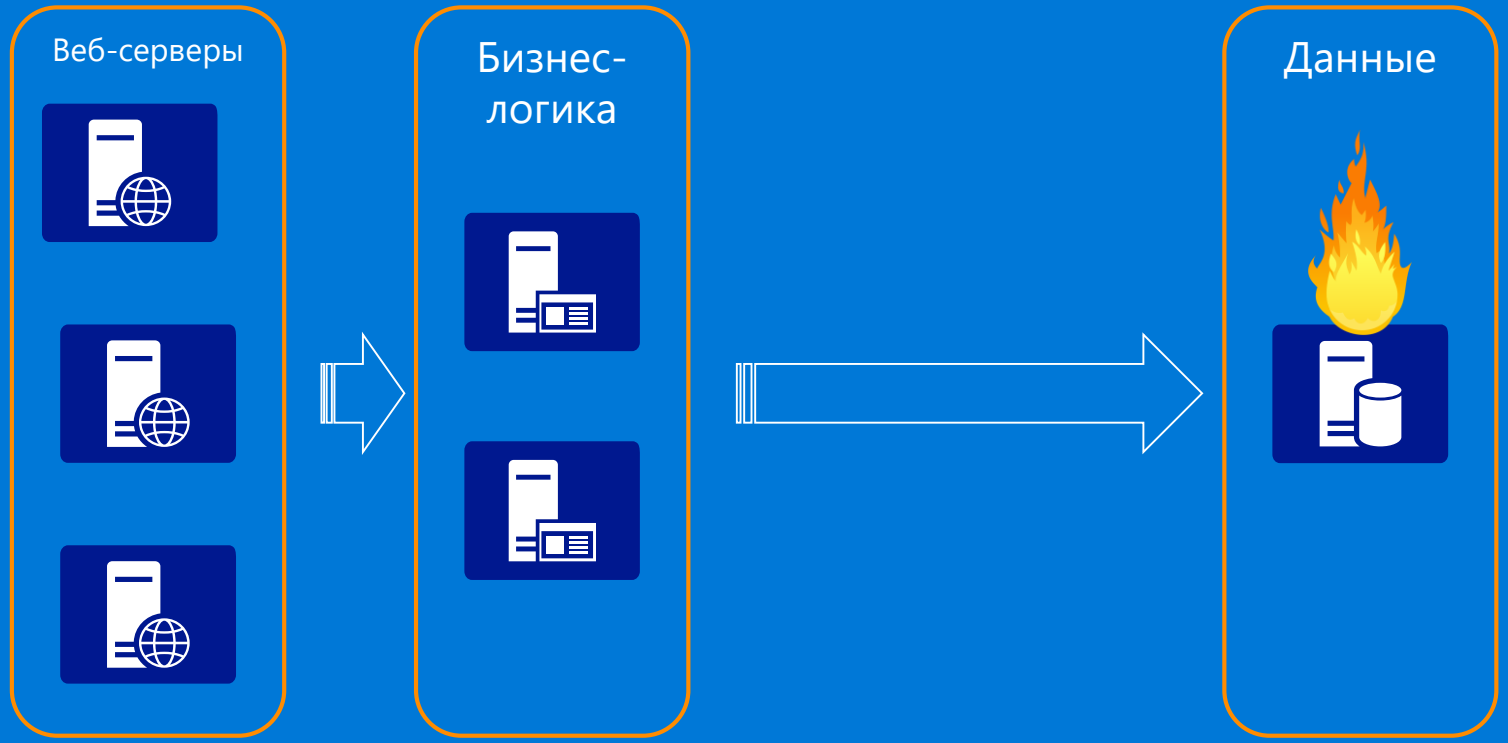


Плюсы и минусы *stateful* модели

+ Низкая задержка
+ Эффективный доступ к данным
+ Координация стандартными примитивами
+ Поддержка ООП

- Ограничена общей памятью
- Проблема с масштабируемостью
- Чувствительность с сбоям

Stateless архитектура



Microsoft

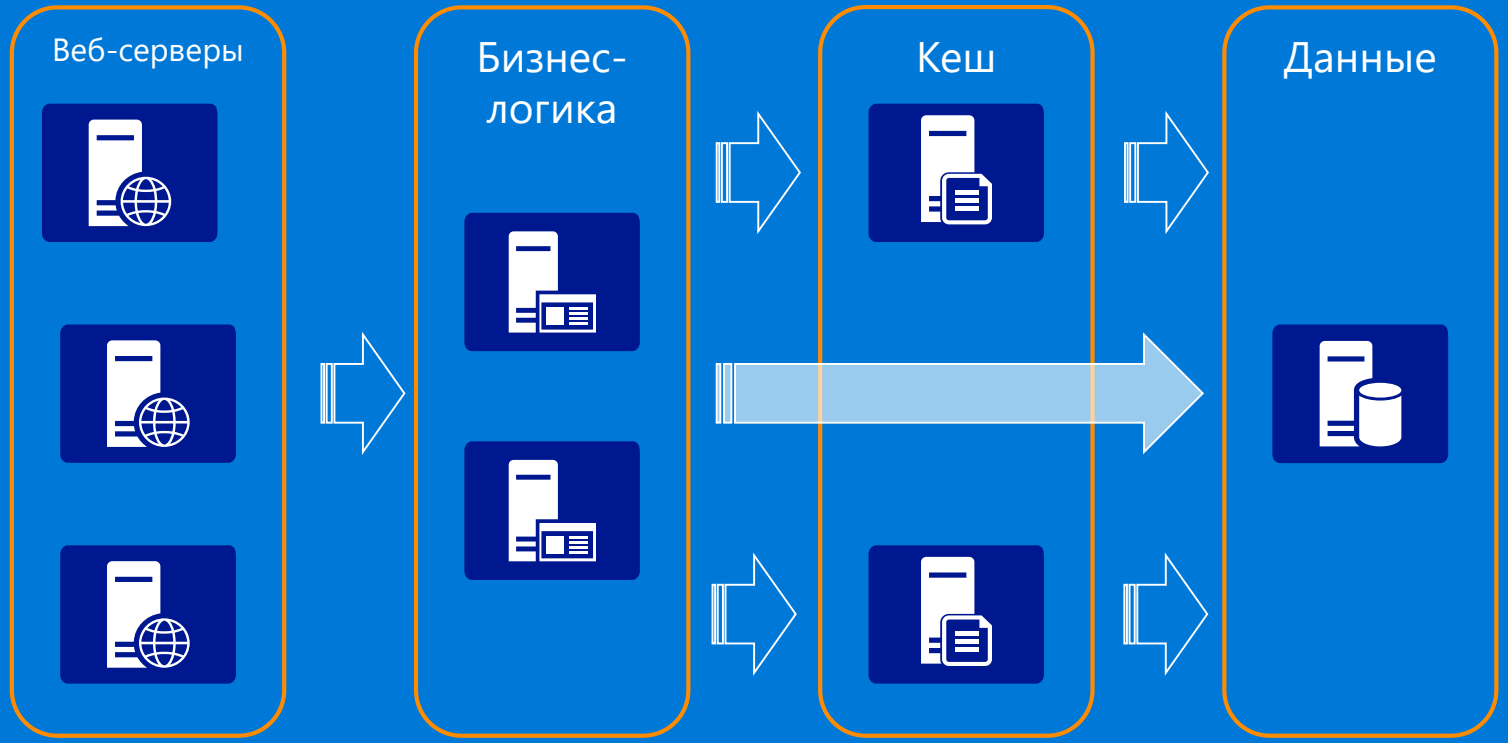
Плюсы и минусы *stateless* модели

+ Простота масштабирования
+ Простота восстановления после сбоя

- Б`ольшая задержка
- Неэффективный доступ к данным
- Отсутствие координации
- Перенос сложности и нагрузки в слой данных
- Забываем про ООП

Microsoft

Stateless архитектура с кешем



Microsoft

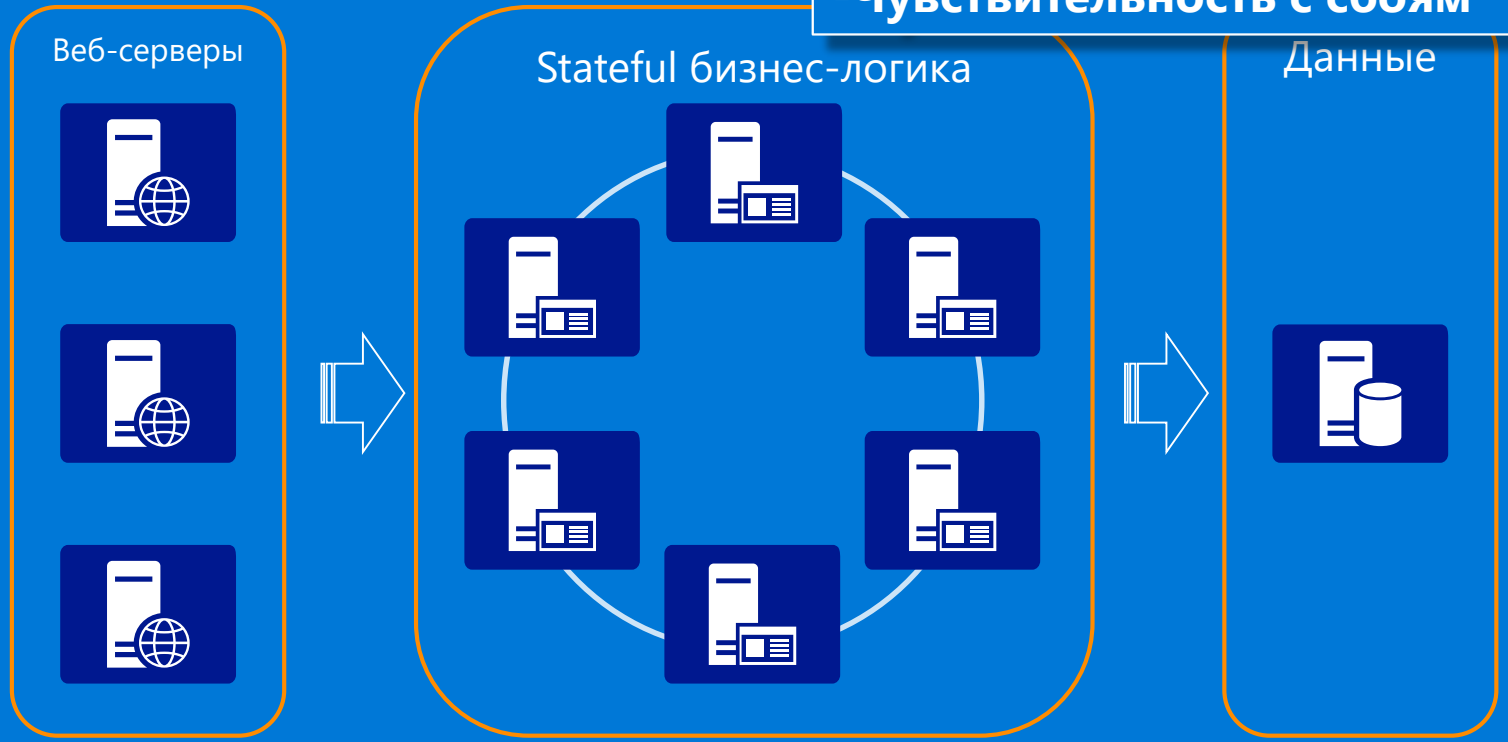
Плюсы и минусы *stateless* модели с кешем

\+ Простота масштабирования
\+ Простота восстановления после сбоя

- ~~Б`ольшая задержка~~
- ~~Неэффективный~~ доступ к данным
- Отсутствие координации
- Перенос сложности и ~~нагрузки~~ в слой данных
- Забываем про ООП
- *Проблема инвалидации кеша*

Назад в *Stateful* будущее

- **Ограничена общей памятью**
- **Проблема с масштабируемостью**
- **Чувствительность с сбоям**



Microsoft

Orleans: Фреймворк для облаков*

Модель программирования

Для всех

Простая, но мощная

3x –10x меньше кода

Лёгкость масштабирования

Нет узких мест

Нет единой точки отказа

Объединяет ряд проверенных методик распределённых систем

Microsoft

*Помимо облачных сервисов, Orleans подходит для создания широкого круга распределённых приложений

Суп из независимых объектов

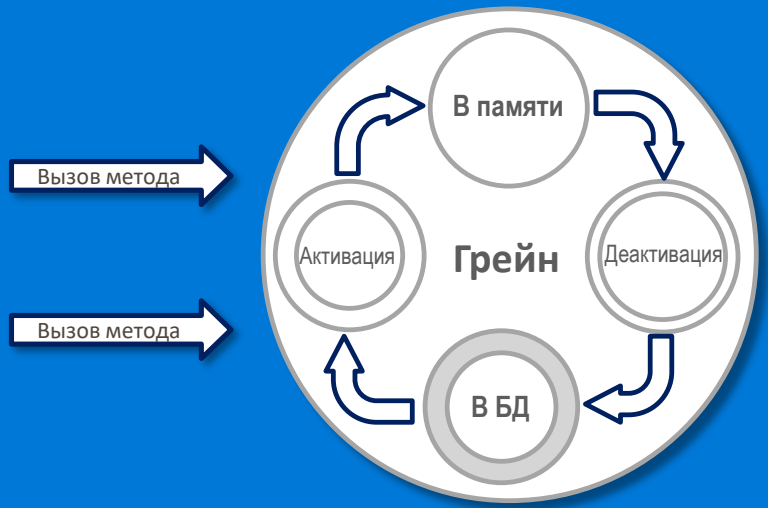
```
// grain interface
public interface IUser : IGrainWithGuidKey
{
    Task<string> SayHello(string name);
}
```

```
// invoking a grain
IUser user = client.GetGrain<IUser>(userId);
string reply = await user.SayHello(name);
Console.WriteLine($"User {userId} said: {reply}");
```

```
// grain implementation class
public class UserGrain : Grain, IUser
{
    private int _counter;

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

Вызовы всегда однопоточны

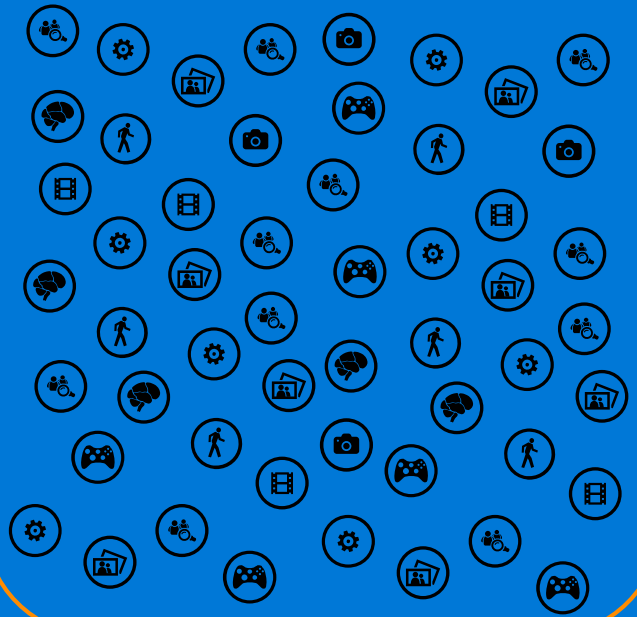


Microsoft

Скользящее подмножество всех возможных объектов: пользователи, устройства, сессии и т.д.

Веб-серверы

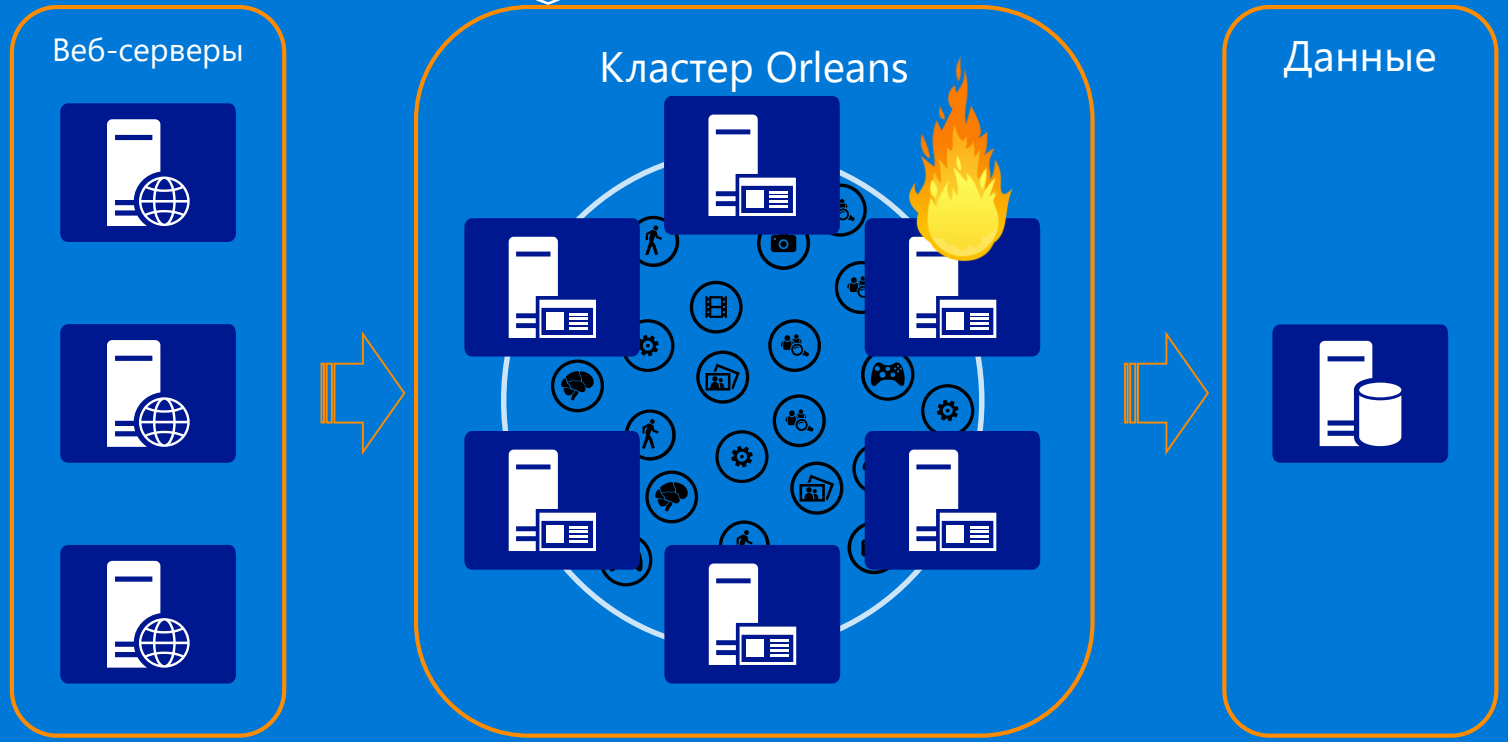
Бизнес-логика



Данные

Microsoft

Orleans прозрачно восстанавливается после сбоев серверов и переконфигурирует кластер



Microsoft

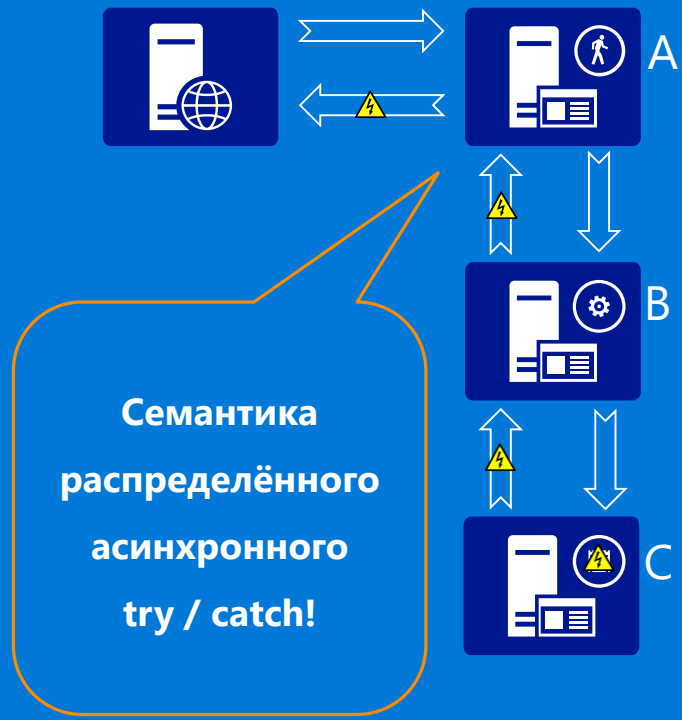
```
public interface IUser : IGrainWithGuidKey
{
    Task AddFriend(IUser friend);
}
```

```
IUser me = client.GetGrain<IUser>(myId);
IUser friend = client.GetGrain<IUser>(friendId);

try
{
    await me.AddFriend(friend);
    Console.WriteLine($"Added friend {friendId}.");
}
catch (Exception e)
{
    Console.WriteLine(
        "Failed to add {friendId} as a friend: {e}.");
    throw;
}
```

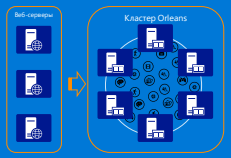
Ссылка на грейн как аргумент

Обработка исключений



Microsoft

Внутри



Orleans Silo

Scheduler

Activation Catalog

Dispatcher

Grain Activations

Streaming

Persistence

Grain Directory

Cluster Membership

Gateway

Messaging

Microsoft

Хостинг

Silo (Orleans сервер)

- Процесс или объект внутри процесса
- Два TCP порта: серверный и клиентский
- Конфигурация
- Код приложения (интерфейсы и классы грейнов)

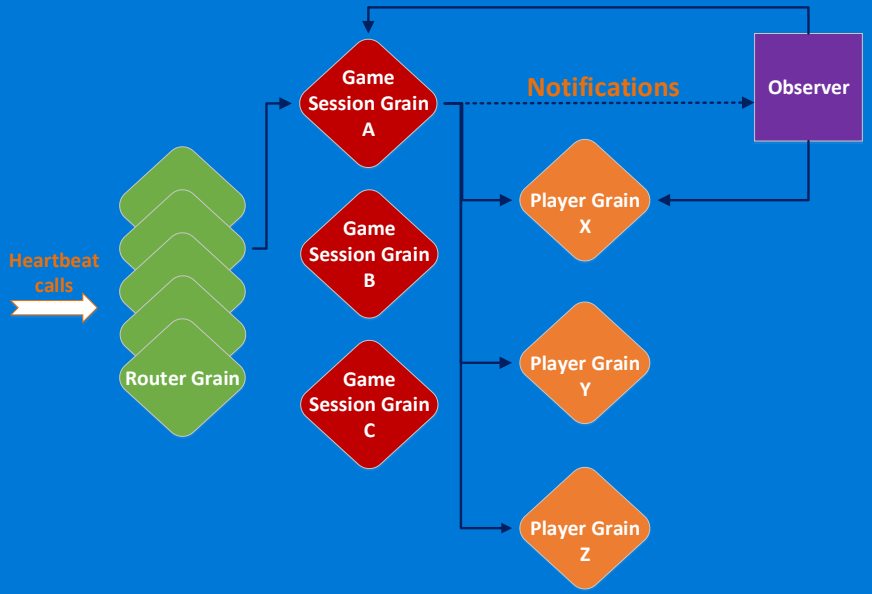
Встроенный протокол автоматического кластеринга

- Использует внешнее хранилище данных
- Взаимный мониторинг Silos и выявление отказов
- Автоматическое добавление в кластер новых Silos
- Клиенты (веб-сервера) подключаются ко всем Silos

Работает где угодно

- Azure, Amazon AWS, private cloud, произвольный набор серверов
- Плагины кластеринга: Azure Table, SF, SQL (MS, My, Postgre, Oracle), ZooKeeper, Consul

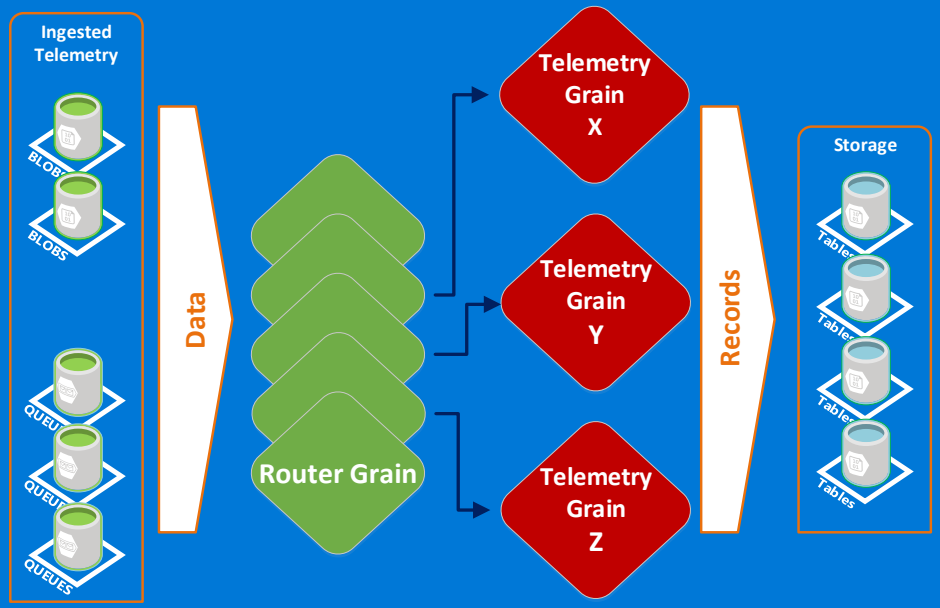
Пример 1: Halo Presence



- *Router Grain* распаковывает и расшифровывает пакеты
- *Session Grain* аккумулирует информацию одного матча и оповещает *Player Grains* игроков
- Мобильный клиент через *Player Grain* находит *Session Grain* текущего матча и подписывается на оповещения о его ходе
- Миллионы паралелных матчей могут обрабатываться независимо
- Пакеты от разных Xbox-ов участников матча доставляются одному *Session Grain*

Microsoft

Пример 2: Обработка телеметрии Skype



- Телеметрия с клиентов записывается в Azure BLOBs+Queues
- *Router Grains* распаковывает блоки телеметрии и направляют *Telemetry Grains*
- Идентификатором *Telemetry Grain* является первичный ключ индивидуальной записи в Azure Table
- Благодаря 1:1 отношению между грейном и записью, избегаются конфликты по записи (eTag violations)
- Пропускная способность выросла в 4x
- До 750K сообщения в секунду

Microsoft

Рецепты успеха

Множество независимых контекстов

- Пользователи, сессии, устройства, каталоги
- Объект (грейн) для каждого контекста

Малая задержка, высокая пропускная способность

- Взаимодействие в реальном времени, в большом масштабе
- Состояние держится в памяти
- Малые затраты на IO

Горизонтальное взаимодействие

- Единое адресное пространство в пределах кластера
- Грейны могут напрямую вызывать друг друга и организовываться в графы

Автоматическое управление ресурсами

- Виртуальность грейнов – ключевой момент
- Автоматическая «сборка мусора» неактивных грейнов

Возврат к ООП

- Лучше соответствует реальному миру, чем stateless сервисы

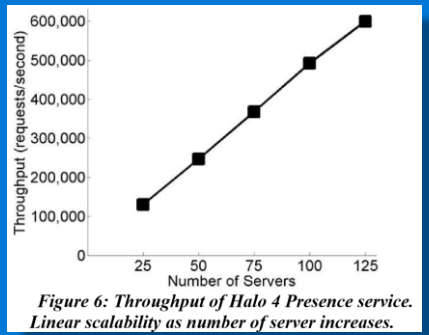


Figure 6: Throughput of Halo 4 Presence service. Linear scalability as number of server increases.

Microsoft

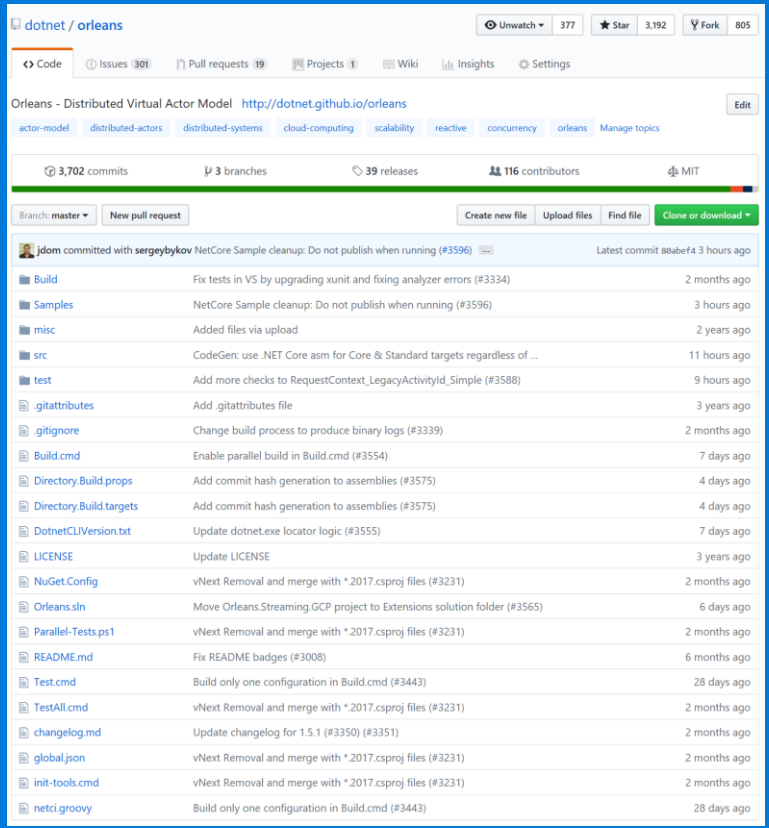
Исходный код на GitHub

Этапы большого пути

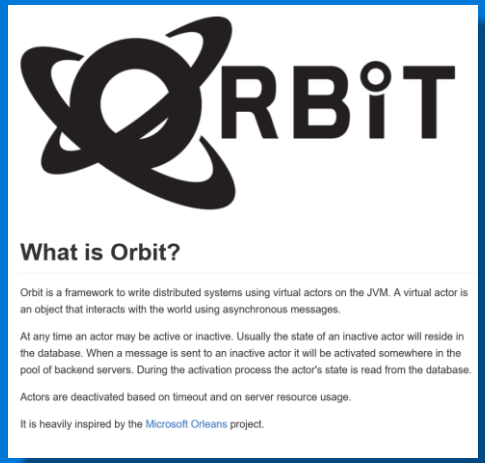
- Начат в Microsoft Research – 2009
- В production – конец 2011
- Выпуск Halo 4 – ноябрь 2012
- Public preview – апрель 2014
- GitHub (MIT) – январь 2015

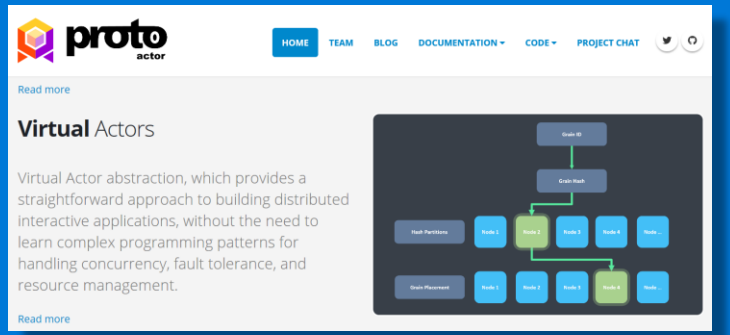
Открытая разработка на GitHub

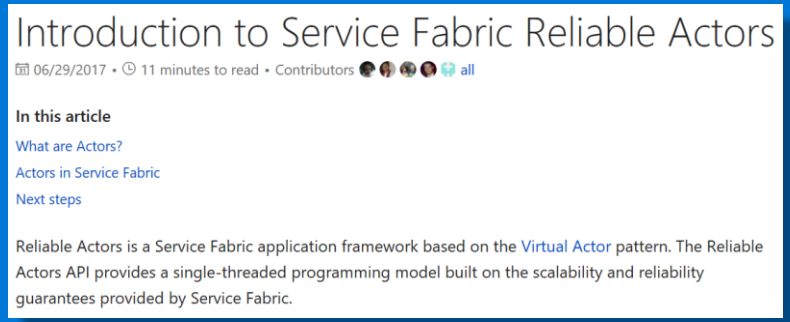
- Глобальное сообщество контрибьюторов
- Один из самых популярных OSS проектов Microsoft
- Кросс-платформенная v2.0.0 с CoreCLR
- Продолжается сотрудничество с Research

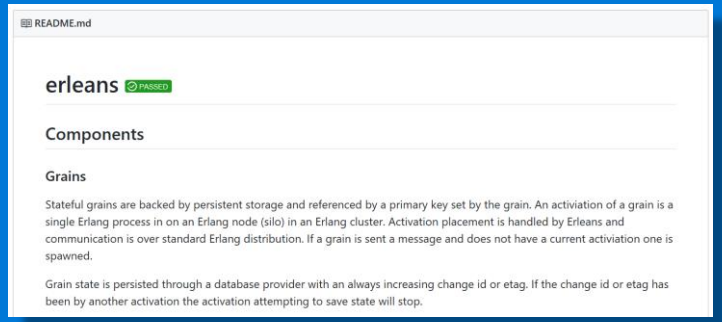


Имитация как высшая форма лести









Microsoft

Спасибо!

@SergeyBykov

@msftorleans

https://github.com/dotnet/orleans/

https://gitter.im/dotnet/orleans