# MICROSOFT ORLEANS КАК РЕАЛИЗАЦИЯ МИКРОСЕРВИСНОЙ АРХИТЕКТУРЫ

*Мазуркевич Е.А.*

*Белорусский государственный университет информатики и радиоэлектроники*
*г. Минск, Республика Беларусь*

*Насуро Е.В. – к.т.н.*

Цель работы: исследовать структуру организации технологии microsoft orleans, рассмотреть применение данной технологии для реализации микросервисной архитектуры.

Microsoft Orleans - это инфраструктура, которая обеспечивает простой подход к созданию распределенных крупномасштабных вычислительных приложений без необходимости изучать и применять сложный параллелизм или другие шаблоны масштабирования [1]. Orleans предоставляет интуитивно понятный способ построения приложения, когда различные объекты бизнес-логики выглядят как множество изолированных глобально адресуемых объектов различных типов, именуемых Grain (зерно). Данные зерна распределены по кластеру серверов, которые в терминологии orleans называются Silo (хранилище). Данная структура отображена на рисунке 1.

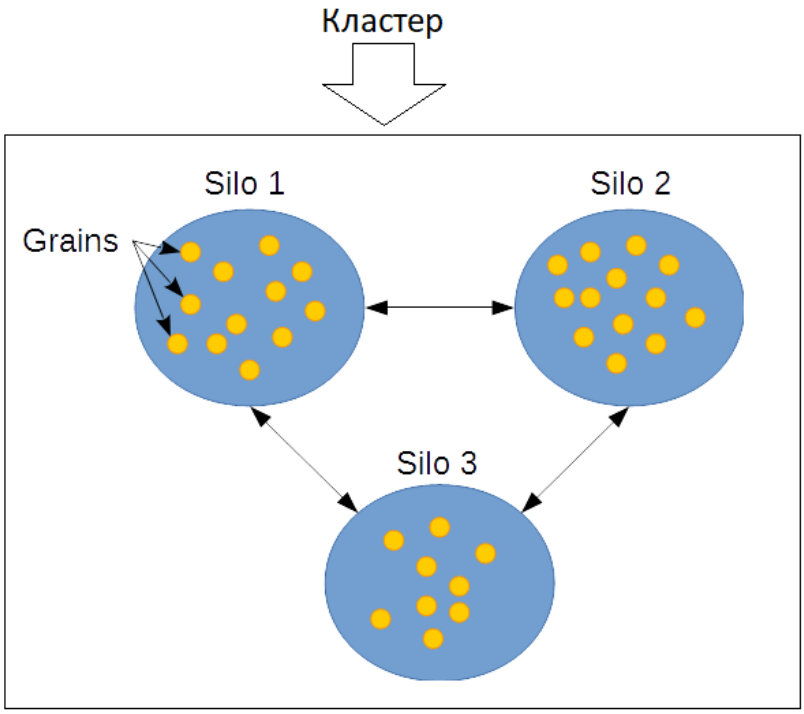


Рисунок 1 - Структура Microsoft Orleans

Реализация Orleans основана на модели актера, существующей с 1970-х годов. Однако, в отличие от актеров в более традиционных актерских системах, Orleans зерна - виртуальные актеры. Самым большим отличием является то, что физические экземпляры зерен полностью абстрагированы и автоматически управляются средой выполнения Orleans. Модель виртуального актора гораздо больше подходит для масштабных динамических рабочих нагрузок, таких как облачные сервисы, что является основным нововведением Orleans.

Преимущество виртуальных актеров в том, что в любой момент ваш код может получить у среды выполнения Orleans прокси для обращения к конкретному зерну по его интерфейсу и ключу [2]. При вызове методов прокси посылается сообщение кластеру из нескольких серверов, где оно будет доставлено одному зерну с заданным ключом. Среда выполнения гарантирует, что такое зерно будет создано в единственном экземпляре на одном из серверов и последующие вызовы будут доставлены ему. Среда выполнения также автоматически удаляет из памяти (деактивирует) зерна, которые не получали вызовов заданное время, таким образом постоянно собирая "мусор". Если сервер, на котором находились ранее зерна, стал недоступным — среда выполнения быстро поднимет экземпляры данных зерен на другом хранилище, что может сказаться только на небольшой задержке

на запуск грейнов. Если вызов должен прийти на тот же самый сервер — среда выполнения это оптимизирует и вызов будет локальным.

Пользой виртуальных актеров в том, что это структура очень легко масштабируется в облаках: хранилища проверяют доступность друг друга и определяют, когда кто-то недоступен, перераспределяют зерна недоступного хранилища между собой. Кластер будет доступен и выполнять свои функции, пока есть хотя бы одно активное хранилище, и при подключении новых участников кластера — они получат свой диапазон грейнов и начнут активно обрабатывать запросы.

Также среда выполнения дает гарантии что при выполнении метода в зерне, никакой другой вызов на тоже самое зерно не придет, то есть среда выполнения гарантирует однопоточное выполнение кода, в рамках одного зерна.

Микросервисная архитектура — вариант сервис-ориентированной архитектуры программного обеспечения, ориентированный на взаимодействие небольших, насколько это возможно, слабо связанных и легко изменяемых модулей — микросервисов. Философия микросервисов гласит, что каждый сервис должен «делать что-то одно, и делать это хорошо» и взаимодействовать с другими сервисами простыми средствами [3]. Основные характеристики микросервисной архитектуры определяются принципами слабой связи между сервисами и высокой связанностью внутри сервиса. Приложение с микросервисной архитектурой отличается четкой структурой сервисов, которые представляют часть функционала приложения.

Философия микросервисной архитектуры ложится на модель структуры Orleans, где зерна выступают в роли сервисов, реализующих узкую часть бизнес логики приложения, которая изолирована в рамках зерна. Коммуникация между зернами-сервисам реализована на уровне среды выполнения Orleans, посредством интерфейсов реализованных зернами.

Однако Orleans не позволяет в полной степени реализовать все возможности микросервисной архитектуры. В первую очередь, разработку приложения не получится легко разделить между несколькими командами разработчиков, поскольку все сервисы описываются в одном проекте.Также одним из плюсов микросервисной архитектуры, который невозможно реализовать с Orleans, является использование различных языков программирования и технологий, подходящих под конкретные задачи. Еще к минусам реализации микросервисной архитектуры с помощью Orleans можно отнести отсутствие возможности развертывания и обновления только части сервисов, поскольку необходимо разворачивать хранилищи целиком со всеми зернами.

Однако микросервисная архитектура с использованием Orleans будет очень полезна для начального этапа разработки микросервисного приложения, поскольку сразу позволяет разбивать логику приложения между зернами, которые в дальнейшем могут стать настоящими сервисами. Также нет необходимости в дополнительных трудозатратах на масштабирование приложения и настройки коммуникации между сервисами, поскольку среда выполнения реализует все эти функции самостоятельно. Данные трудозатраты являются основными причинами отказа от микросервисной архитектуры на начальном этапе разработки, однако Orleans позволяет уменьшить издержки.

***Список использованных источников:***

1. Microsoft Orleans | Microsoft Orleans Documentation [Электронный ресурс]. – Электронные данные. – Режим доступа https://dotnet.github.io/orleans

2. Getting Started with Microsoft Orleans [Электронный ресурс]. – Электронные данные. – Режим доступа https://medium.com/@kritner/getting-started-with-microsoft-orleans-882cdac4307f

3. Wagner, .NET Microservices: Architecture for Containerized .NET Applications / Bill Wagner, Cesar de la Torre, Mike Rousos – Redmond, Washington, 2018.

# СТРУКТУРЫ ДАННЫХ ДЛЯ РАБОТЫ СО СДЕЛКАМИ НА БИРЖЕ

*Митьковец А.А.*

*Белорусский государственный университет информатики и радиоэлектроники*
*г. Минск, Республика Беларусь*

*Поттосин Ю.В. – к.ф.-м.н.,доцент*

В статье обсуждается применение классических структур данных, позволяющих работать с потоком данных с биржи. Дается структура данных для ведения книги сделок на основе связного списка и хэш-таблицы. Обсуждается проведение сделок в контексте асинхронного программирования, даются рекомендации по использованию futures и promises.

Общение с биржей всегда строится по модели «клиент-сервер». Биржа представляет собой (готовый) сервер, который предоставляет API клиентам. В случае с конвенциальными биржами речь