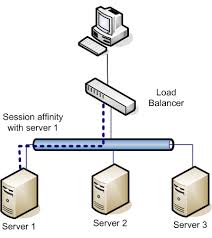
**Distributed applications** (**distributed** apps) are **applications** or software that runs on multiple computers within a network at the same time and can be stored on servers or with cloud computing.



**Orleans** is a framework that provides a straight-forward approach to building distributed high-scale computing applications, without the need to learn and apply complex concurrency or other scaling patterns. It was created by **Microsoft**  Research implementing the Virtual Actor Model and designed for **use** in the cloud.

<https://dotnet.github.io/orleans/>

silo : a [large](https://dictionary.cambridge.org/dictionary/english/large), round [tower](https://dictionary.cambridge.org/dictionary/english/tower) on a [farm](https://dictionary.cambridge.org/dictionary/english/farm) for [storing](https://dictionary.cambridge.org/dictionary/english/store) [grain](https://dictionary.cambridge.org/dictionary/english/grain) or [winter](https://dictionary.cambridge.org/dictionary/english/winter) [food](https://dictionary.cambridge.org/dictionary/english/food) for [cattle](https://dictionary.cambridge.org/dictionary/english/cattle):

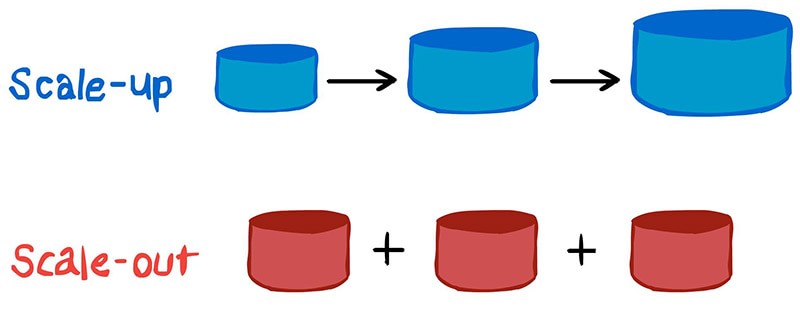
grain : seeds

scaffold : a [frame](https://dictionary.cambridge.org/dictionary/english/frame) that [hangs](https://dictionary.cambridge.org/dictionary/english/hang) on the [side](https://dictionary.cambridge.org/dictionary/english/side) of a [building](https://dictionary.cambridge.org/dictionary/english/building), [ship](https://dictionary.cambridge.org/dictionary/english/ship), etc. for [people](https://dictionary.cambridge.org/dictionary/english/people) to [work](https://dictionary.cambridge.org/dictionary/english/work) from

scalability : the [ability](https://dictionary.cambridge.org/dictionary/english/ability) of a [business](https://dictionary.cambridge.org/dictionary/english/business) or [system](https://dictionary.cambridge.org/dictionary/english/system) to [grow](https://dictionary.cambridge.org/dictionary/english/grow) [larger](https://dictionary.cambridge.org/dictionary/english/large):

intuitive : an [ability](https://dictionary.cambridge.org/dictionary/english/ability) to [understand](https://dictionary.cambridge.org/dictionary/english/understand) or [know](https://dictionary.cambridge.org/dictionary/english/know) something [immediately](https://dictionary.cambridge.org/dictionary/english/immediately) [based](https://dictionary.cambridge.org/dictionary/english/based) on [your](https://dictionary.cambridge.org/dictionary/english/your) [feelings](https://dictionary.cambridge.org/dictionary/english/feeling) [rather](https://dictionary.cambridge.org/dictionary/english/rather) than [facts](https://dictionary.cambridge.org/dictionary/english/fact):

**Scale out** is a type of capacity expansion concentrating on the addition of new hardware resources instead of increasing the capacity of already available hardware resources such as storage or processing silos.



**Orleans** is a framework that provides an approach to building distributed high-scale computing applications without the need to learn and apply complex concurrency or other scaling patterns.

#### Grains

Grains are the key primitives of the Orleans programming model. Grains are the building blocks of an Orleans application, they are atomic units of isolation, distribution, and persistence. Grains are objects that represent application entities. Just like in the classic Object Oriented Programming, a grain encapsulates state of an entity and encodes its behavior in the code logic. Grains can hold references to each other and interact by invoking each other’s methods exposed via interfaces.

One other interesting note about grains is that they are automatically managed by Orleans.  You don’t have to worry about instantiating or managing them.   Orleans terms them virtual actors as apposed to traditional actors.

#### Silos

Where do Grains live? Silos of course.

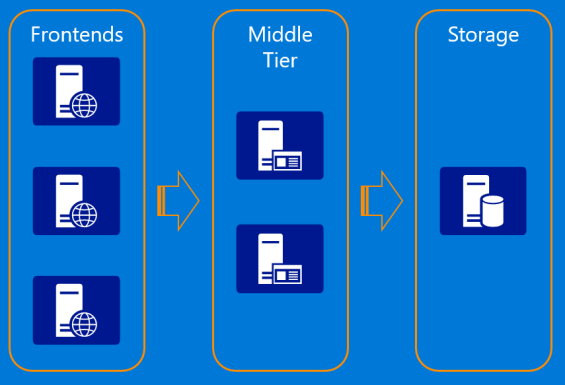
Silos are what host and execute Grains.  Again, Grains are your objects that expose behavior and encapsulate state.  Orleans creates your grains in the Silo and executes them here.  Your client code will reference only interfaces that your grains implement.  Don’t worry, sample code coming up.

Oh and of course, you can also have a cluster of silos.

**Background**

Cloud applications and services are inherently parallel and distributed. They are also interactive and dynamic; often requiring near real time direct interactions among cloud entities. Such applications are very difficult to build today. The development process demands expert level programmers and typically requires expensive iterations of the design and the architecture, as the workload grows.

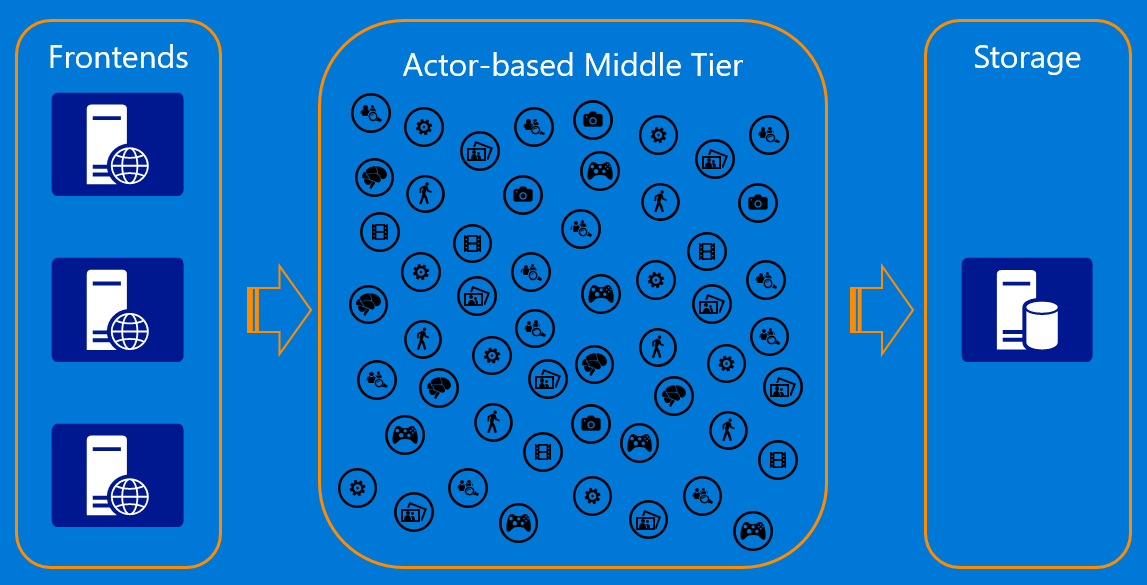
Most of today’s high scale properties are built as a composition of stateless n-tier services with most of the application logic residing in the middle tier.



While the model allows scaling out by adding more servers to the middle tier, it is constrained by the performance and scalability of the storage layer because most requests coming to the middle tier from the frontend web servers require one or more reads from storage. Updates are even more complicated and prone to concurrency issues and conflicts due to lack of coordination among the middle tier servers. It often requires caching in the stateless layer to get acceptable performance, adding complexity and introducing cache consistency issues. The other problem with the stateless n-tier model is that it doesn't support horizontal communications well among individual application entities exposed by the middle tier, which makes it hard to implement complex business logic with multiple entities performing individual operations as part of processing a request.

## Orleans as a Stateful Middle Tier

Orleans provides an intuitive way of building a stateful middle tier, where various business logic entities appear as sea of isolated globally addressable .NET objects (grains) of different application defined types distributed across a cluster of servers (silos).



A grain type is a simple .NET class that implements one or more application-defined grain interfaces. Individual grains are instances of application-defined grain classes that get automatically created by the Orleans runtime on servers on an as-needed basis to handle requests for those grains. Grains naturally map to most application entities, such as users, devices, sessions, inventories, and orders. This makes it very easy to build business logic that is object-oriented but scales transparently across a cluster of servers. Each grain has a stable logical identity (key) within its grain type chosen by the application logic, for example, user email or device ID or inventory SKU code. Orleans guarantees single-threaded execution of each individual grain, hence protecting the application logic from perils of concurrency and races. In the world of microservices, Orleans is used as a framework for implementing a microservice that can be deployed and managed by a microservices deployment/management solution of developer's choice.