orleans

distributed systems

A distributed system is a system whose components are located on different networked computers, which communicate and coordinate their actions by passing messages to one another from any system. [1]

A good example of a distributed system is a group of containerised web API instances:

- placed behind a load balancer
- serving customer traffic

Where each API instance communicates with a group of data sources or supporting applications via message queues or direct messages, performs some decisions, reports on its health, etc.

actors

The Actor Model is a mathematical theory of computation that treats “Actors” as the universal primitives of concurrent digital computation [2]. The model has been used both as a framework for a theoretical understanding of concurrency, and as the theoretical basis for several practical implementations of concurrent systems. Recently, the Actor Model gained new popularity with the advent of cloud-based computing where:

- massive, distributed parallelism
- ability to quickly analyse and understand distributed systems

by humans are of critical importance.

In simplest terms, Actors are characterised by two features:

- they don't share state - each actor is an independent entity with its own memory
- they only communicate indirectly via messages - allowing for greater decoupling

Using the Actor Model in distributed systems allows for breaking down complex, resource-intensive problems into more manageable, focused fragments and sharing them between the components of a distributed system.

virtual actors

Orleans paper.

microservices and their challenges

Microservices are often characterised by:

- their lack of internal state
- application instances being expendable - relatively simple to spin up and to tear down

which makes it fair to categorise microservices as distributed stateless applications.

While microservices allow engineers to easily scale distributed systems (both horizontally and vertically), they also present a number of challenges which become especially apparent when mission-critical speed is essential, namely:

- scaling - the ability to correctly respond to sudden changes in traffic volumes (it is not uncommon to see 10x traffic spikes in our APIs). Application instances should be quick to wake up, warm up and start working with a load balancer to serve traffic.
- request processing speed - the API should ideally be able to perform its tasks in predictable, relatively constant time.
- performance bottlenecks - if the application has a performance bottleneck (e.g. one task requiring excesive amounts of resources), scaling the application horizontally will just replicate those performance bottlenecks across the fleet of application instances.
- [TODO] diagram - a fleet of web APIs - https://plantuml.com/component-diagram
- [TODO] coding focus
- [TODO] maintainability

actors in distributed stateless and stateful applications

references

1 - Distributed systems: principles and paradigms.

Tanenbaum, Andrew S.; Steen, Maarten van (2002). Distributed systems: principles and paradigms. Upper Saddle River, NJ: Pearson Prentice Hall. ISBN 0-13-088893-1.

2 - A Universal Modular Actor Formalism for Artificial Intelligence

Hewitt, Carl; Bishop, Peter; Steiger, Richard (1973). "A Universal Modular Actor Formalism for Artificial Intelligence". IJCAI.

resources

- https://github.com/dotnet/orleans
- https://dotnet.github.io/orleans/
- https://github.com/microsoft/coyote/