dotNEXT St.Petersburg

Dino Esposito
JetBRAINS

Common Scalability Practices
that just work

Scalability as teenage sex

Yes

Mmm

Yes

- Everyone talks about it
- Nobody really knows how to do it
- Everyone thinks everyone else is doing it
- So everyone claims they are doing it…

NEED

Domain and app matching

No product or technology can magically make a system scalable and usable.

- Not because you save data to the cloud, your app survives thousands of concurrent users.
- Not because of HTML5 users will enjoy the application.

Scalability

System's ability to handle a growing number of requests without incurring in significant performance loss and failures.

Scalability became a problem with the advent of web

Queuing theory

A queue forms when frequency at which requests for a service are placed exceeds the time it takes to fully serve a pending request.

- It's about the **performance** of a single task
- It's about **expanding** the system to perform more tasks at the same time

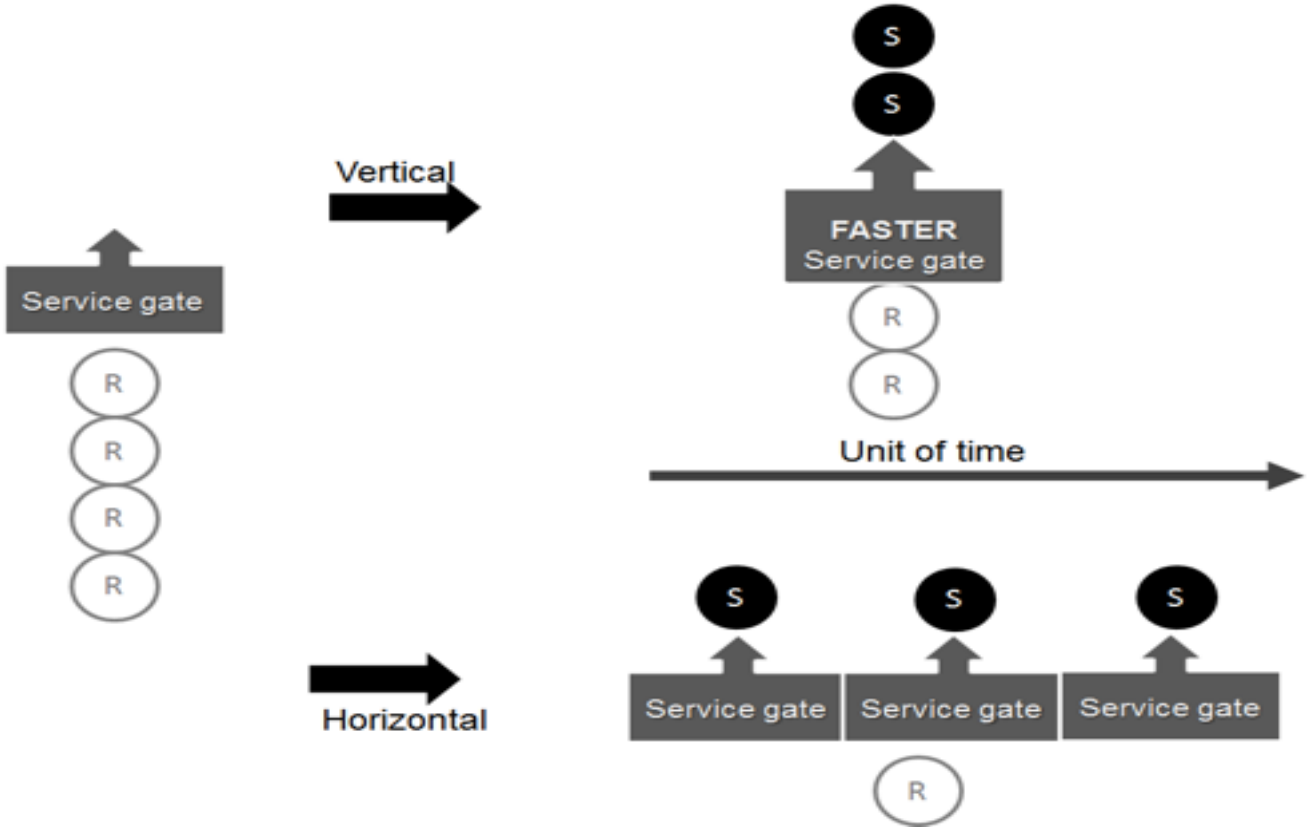
Ensure the system...

- Doesn't crash at launch
- Gives signs of degrading performance timely

Make sure you know the most common tools and strategies to address scalability needs.

Strategies

Vertical vs. Horizontal



Vertical

Norm for 20 years – so long as DB was central point
Doesn't scale beyond a point

Front caching is a good way to do it

- Proxy servers with load balancing capabilities
- Working outside the core code of the application
- Squid, Varnish, Nginx

Horizontal

Mostly an architectural point

Critical parts can be expanded without

- Damaging other parts
- Introducing inconsistencies / incongruent data

Horizontal

Cloud apps are probably the easiest and most effective way to achieve forms of scalability today.

But, at the same time, you can have well responsive apps without re-architecting for the cloud.

Common Practices

Operational Practice #1

Remove bottlenecks

- Convoluted queries
- Long initialization steps
- Inefficient algorithms

HIGH throughput

MEDIUM cost

TIME consuming

DELICATE

Operational Practice #2

Move "some" requests to other servers

- CDN for static files
- Geographically distributed sites

LOW throughput

LOW cost

Quick

Improves the user's perception of the system

Operational Practice #3

Output Caching

- By param
- By locale
- By custom data

  for example, multi-tenant sites

MEDIUM throughput

LOW cost

Quick

MEDIUM risk

Operational Practice #4

Data Caching

- Problematic with farms
- Auto-updatable internal cache
- Use of distributed caches

  Redis, ScaleOut, NCache

HIGH throughput

MEDIUM cost

Relatively Quick

DELICATE

Operational Practice #5

Proxy caching for example **Varnish**

- Installed in front of any web site
- Fully configurable
- Cache and load balancer in one

HIGH throughput

Relatively LOW cost

Relatively Quick

CQRS Architecture

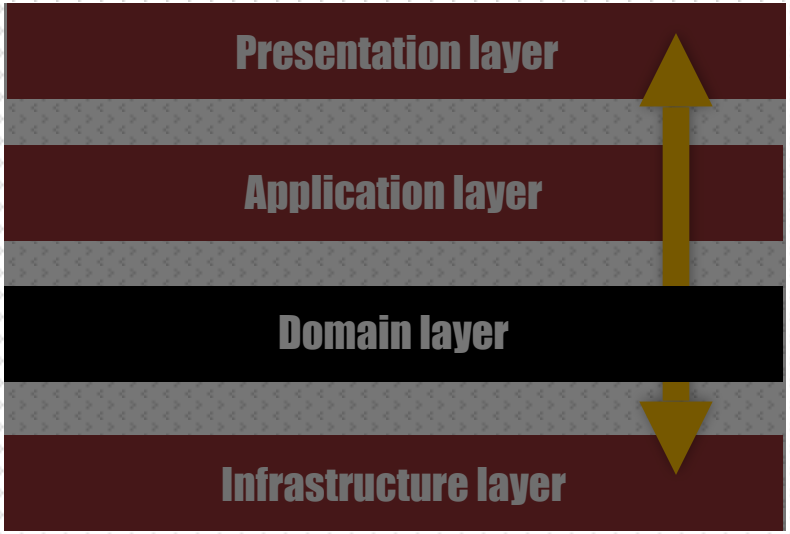
- Optimize stacks differently

HIGH throughput

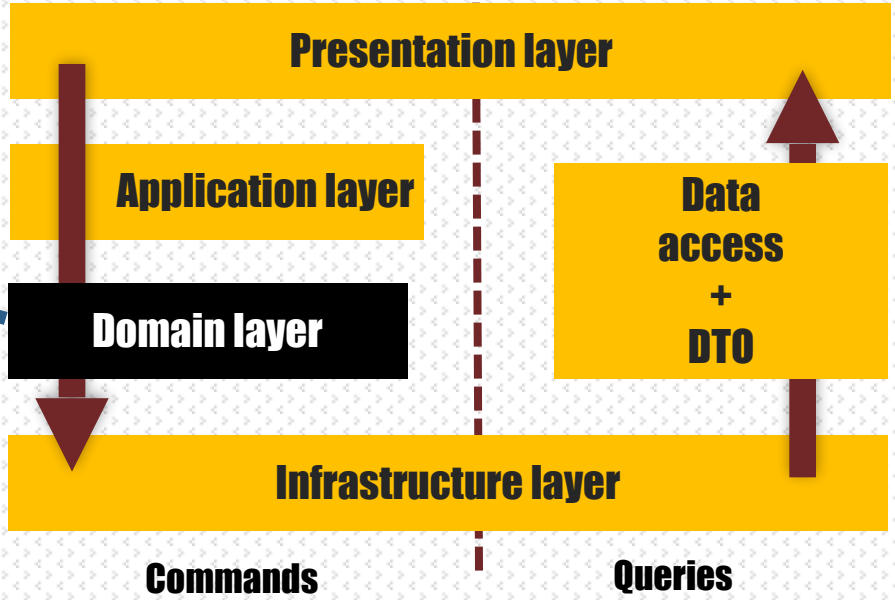
HIGH cost

Time consuming

Canonical layered architecture



CQRS

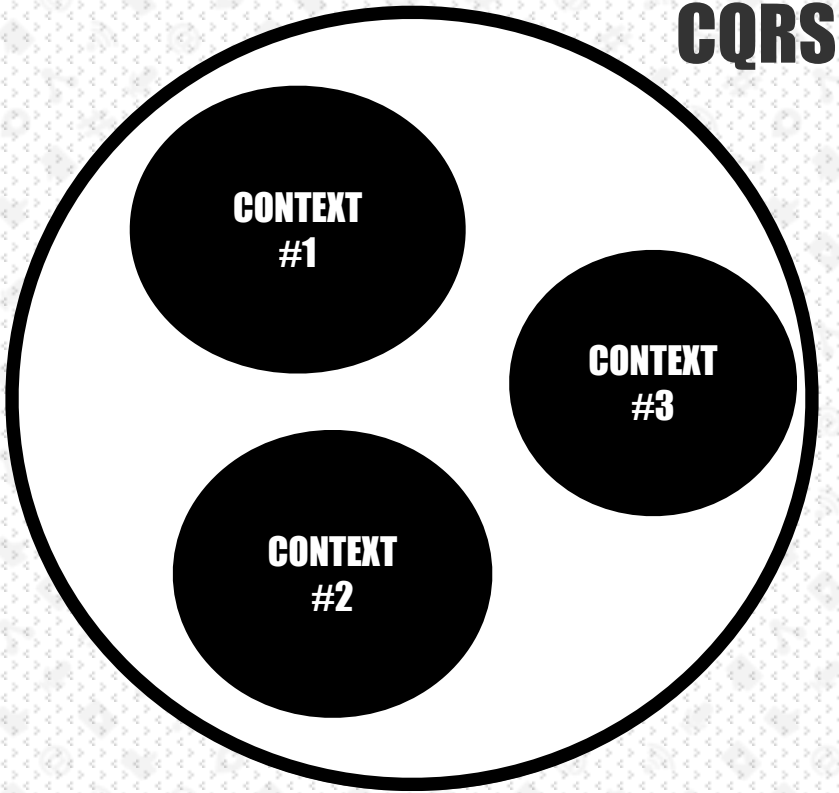


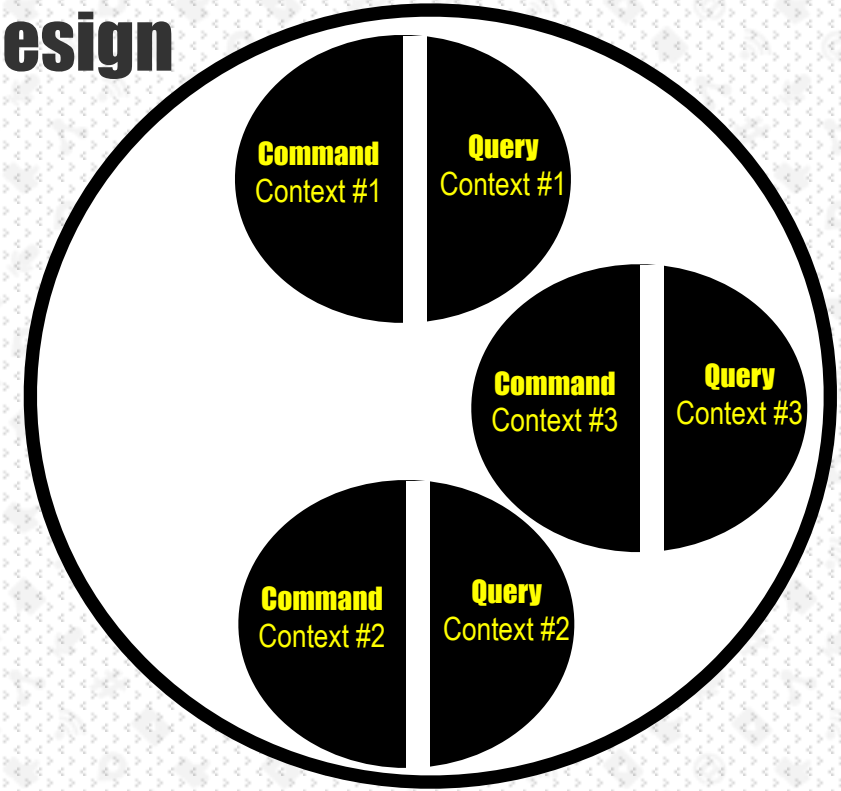
Message-based business logic implementation

Requirements

DDD Analysis

CQRS Design





Architectural Practice #2

Single-tier and stateless server

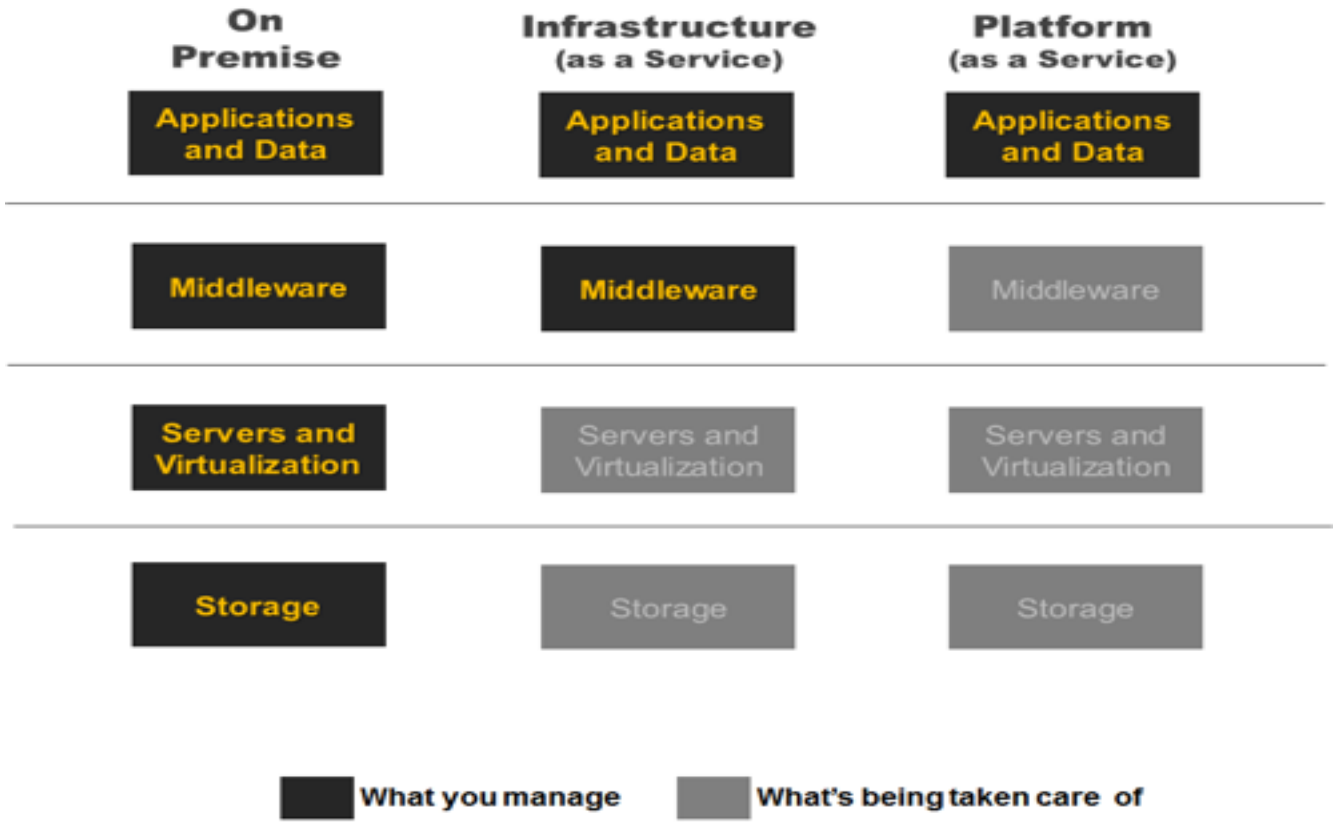
- One server
- No session state
- Quick and easy to duplicate
- Behind a load balancer

HIGH throughput

Low cost

Quick

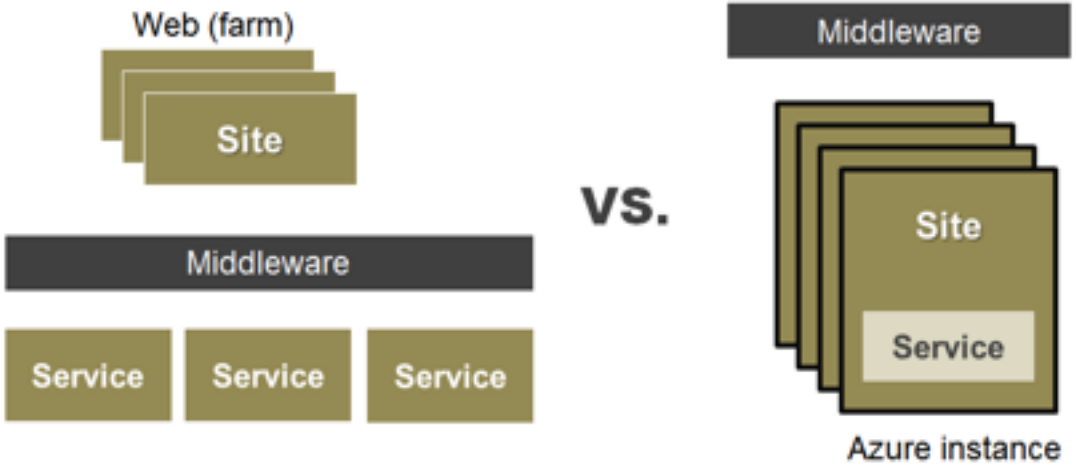
The point of the cloud



Architectural Practice #3

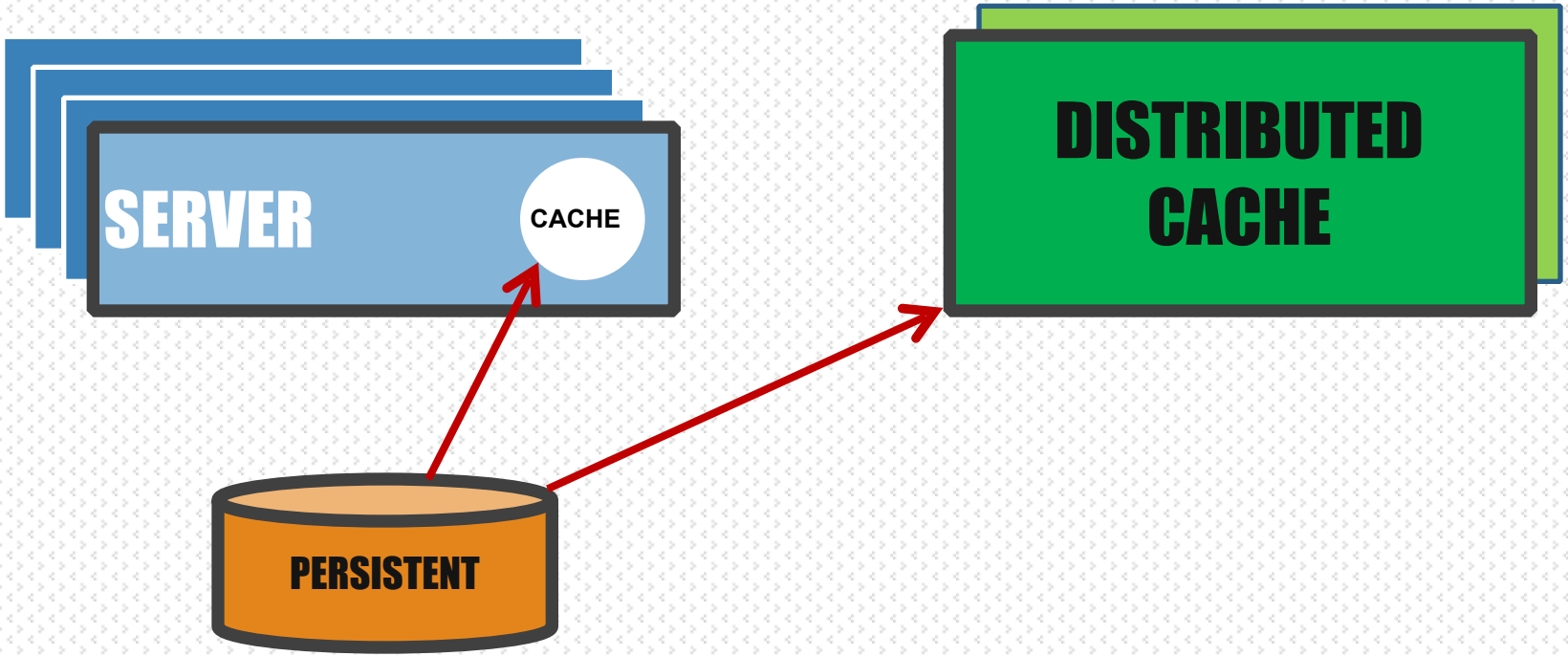
Cloud support

- On-demand servers
- Pay per use
- Configure easily
- No middleware costs
- Better failure policies



Auto-update internal cache

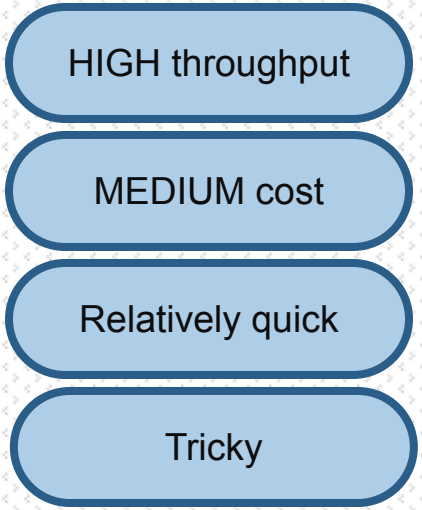
Global cache



Architectural Practice #4

Session state out of the server

- Client cookies if applicable
- Distributed cache
- Azure blob storage



PS: Best option for ASP.NET is probably using the **Redis**-based provider for out-of-process session state.

Architectural Practice #5

Be aware of NoSQL and polyglot persistence

- Relational is OK … until it works
- Sharding/growth of data

Azure SQL

+ Many small tables <500GB each
+ No extra license costs
+ Zero TCO
+ HA automatically on

SQL Server in a VM

+ Fewer large tables >500GB each
+ Reuse existing licenses
+ More machine resources
+ HA and management is your own

That's All Folks!

FOLLOW @despos

facebook.com/naa4e

dino.esposito@jetbrains.com

software2cents.wordpress.com