TOP 5 ARCHITECTURE PATTERNS

Adding Tools to the Toolbox

CENTRAL OHIO PROGRAMMING COMPETITION

SEPTEMBER 7, 2019

Scan to Register & for Additional Details
(or visit link)
hmbnet.com/competition

Free Registration

Limited Seats Available
Register Now to Save Your Seat!

Prizes up for Grabs!

FEATURING JAVA AND C#

Beginner and Non-Beginner Skill Levels Welcome!

SPONSORED BY: HMB

Jim Everett

Developer, Architect, Software Enthusiast

Experience

HMBer

Married father of 2

Software Enthusiast

17 years of .NET Experience

Enterprise Architect

Developer

Scrum Master

Life Learner

https://twitter.com/CognitiveBurden

https://github.com/cognitiveburden

cognitiveburden@gmail.com

What is your favorite Architecture?

Introduction

Architecture Patterns

- Layered
- Microservices
- Event-Driven
- Space-Based
- Microkernel

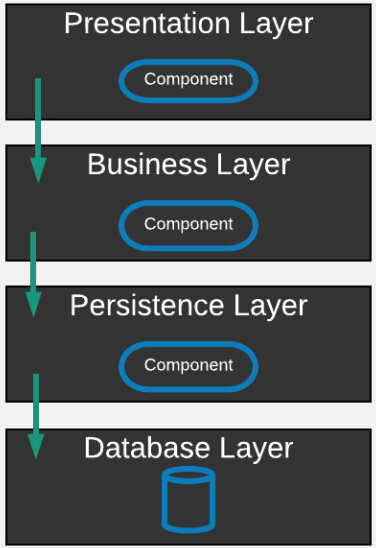
Discuss several well known architectural patterns with their pros and cons. The goal is for the listener to have an understanding of the patterns and be more informed on the types of architecture when faced with architectural decisions.

Layered Architecture

MMM...Tastes so good.

Typically:

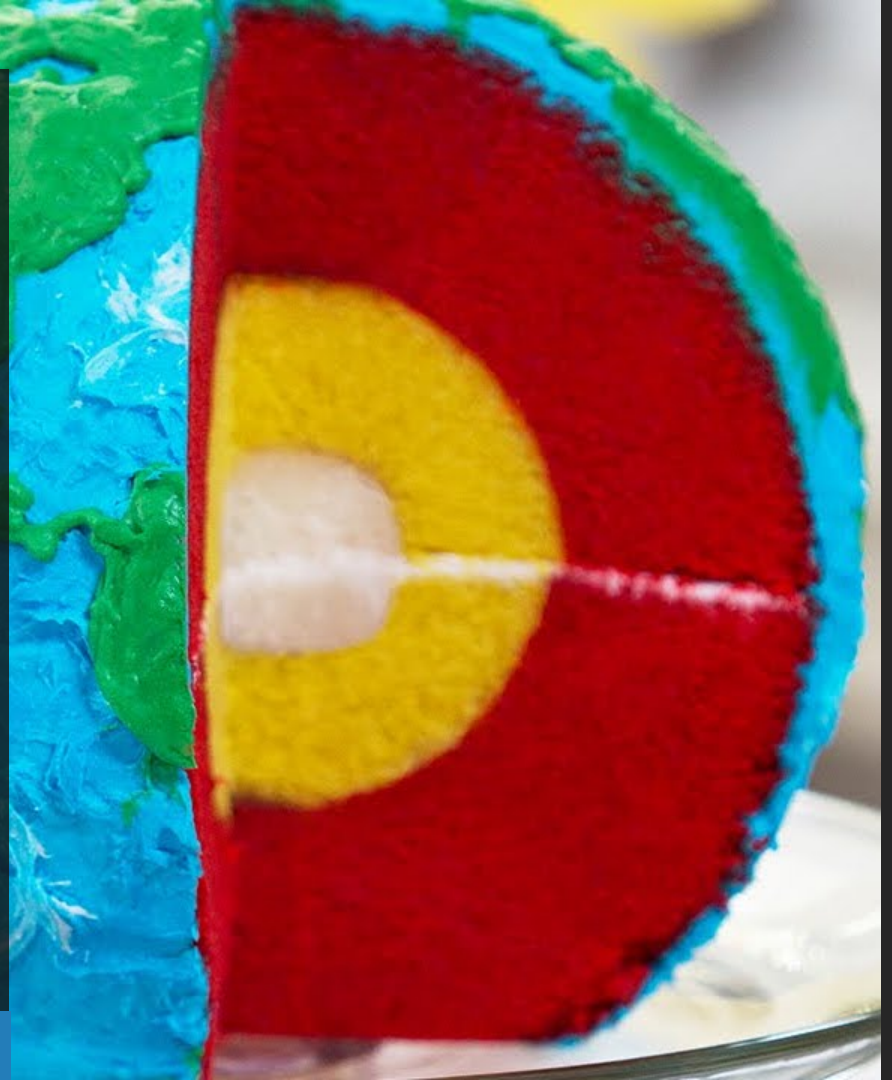
1. Presentation
2. Business
3. Persistence
4. Database



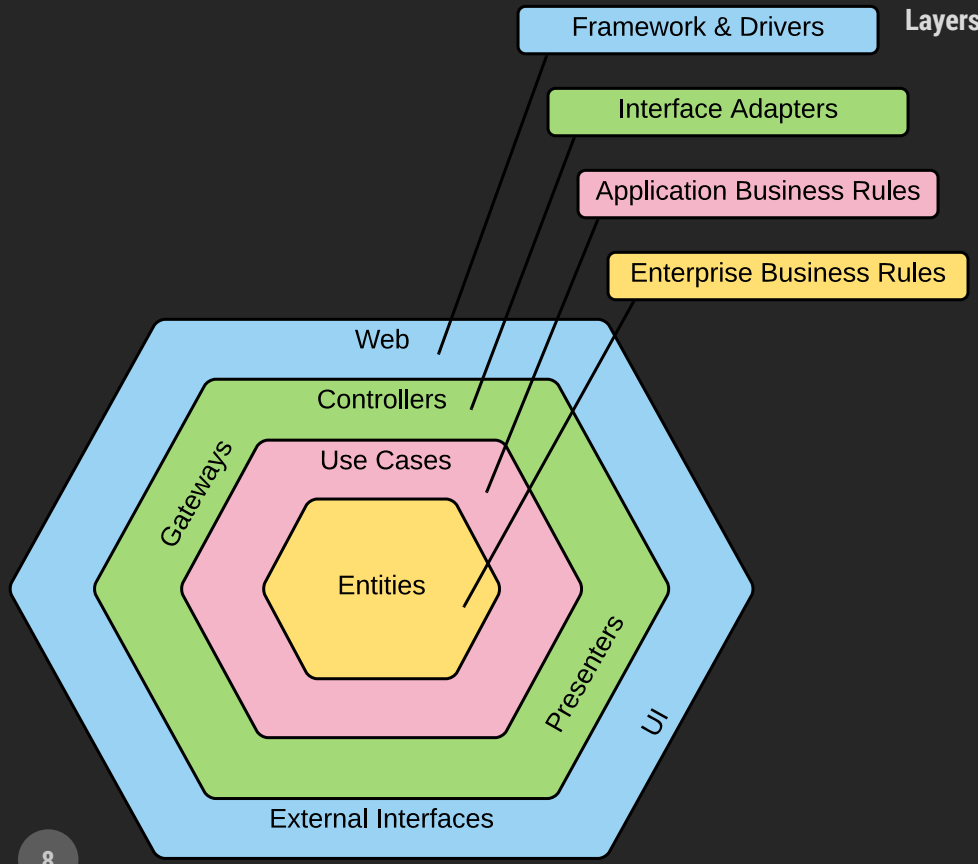
Clean, Onion, Hexagonal Architecture

Layered Architecture

- Independent of Frameworks
- Independent of UI
- Independent of Database
- Independent of External Integrations
- Testable



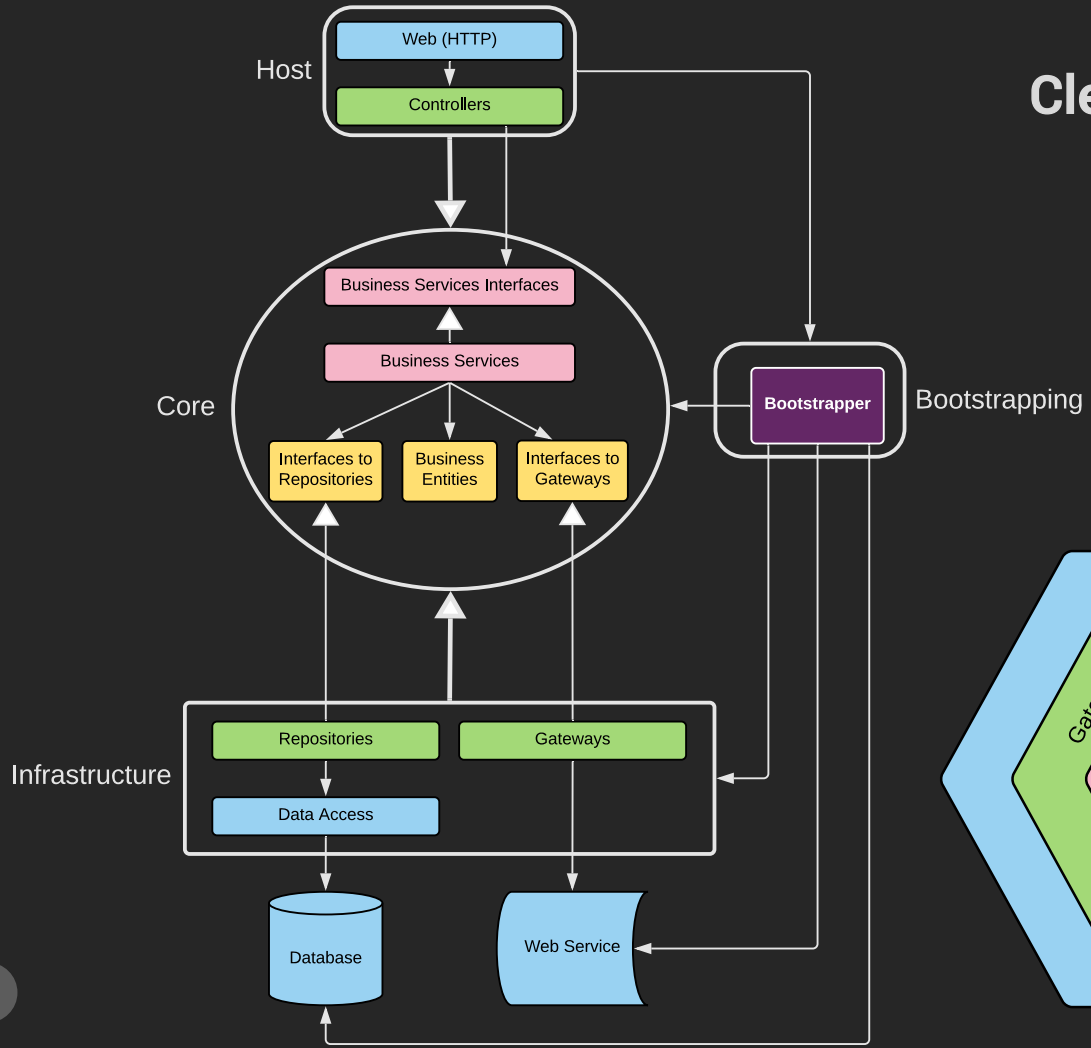
Clean Architecture

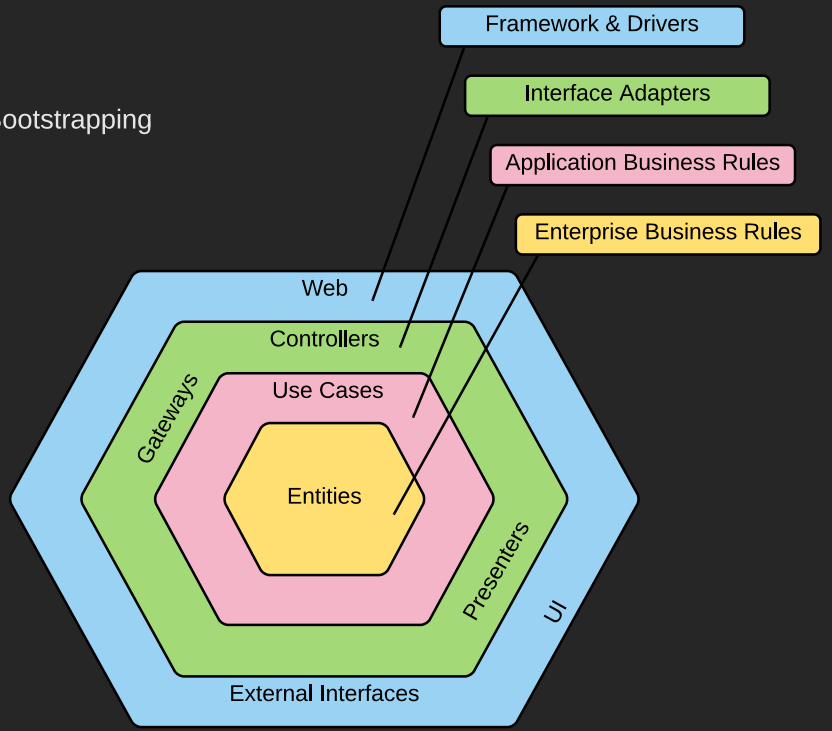


Layers

- Framework & Drivers
- Interface Adapters
- Application Business Rules
- Enterprise Business Rules

Clean Architecture





Pros & Cons

Layered Architecture

Pros

- General Purpose Pattern
- Starter Arch for new application

Cons

- Sinkhole anti-pattern
- Best for Monolithic applications

Pattern Analysis

Layered Architecture

Agility

Changes can be isolated to a layer, it is time-consuming to make changes due to Monolithic nature.

Ease of Deployment

One small change can result in redeploying the entire application.

Testability

Because components belong to specific layer, other layers can be faked, making it easy to test.

Performance

Pattern does not lend it self to performance because every request must go through multiple layers.

Scalability

Due to tightly coupled monolithic design is difficult to scale. Large granularity of layers makes expensive to scale.

Ease of Development

Pattern is well known and not overly complex. Matches well with companies communication patterns, Conway's Law.

Event-Driven Architecture

What's Going On

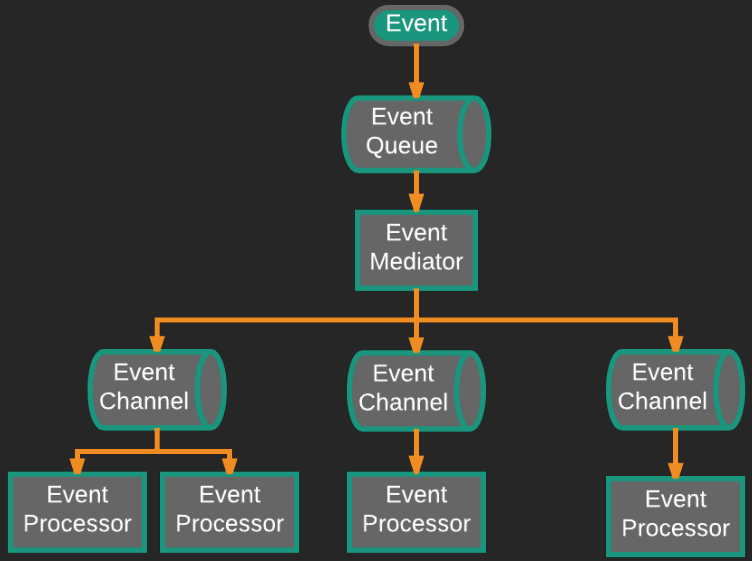
Distributed Async Pattern

Composed of highly decoupled, single purpose event processing components that asynchronously receive and process events.

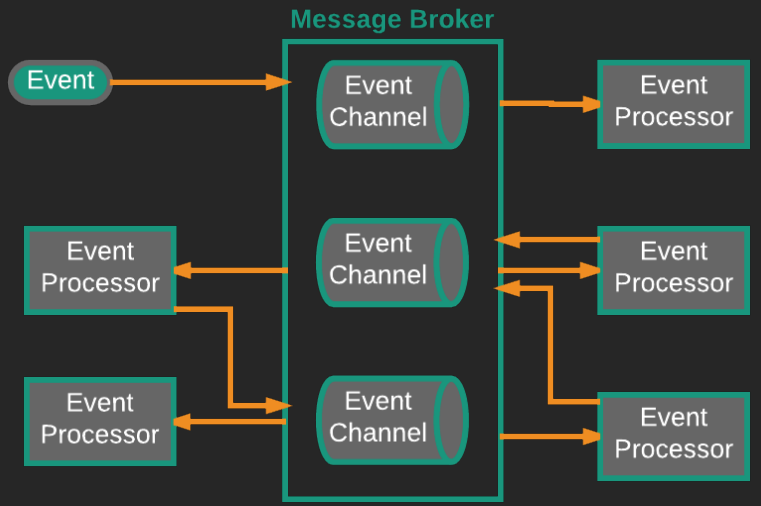
Mediator vs Broker

Orchestration vs Choreography

Mediator / Orchestrator



Broker / Choreography



Pattern Analysis

Event-Driven Architecture

Agility

Able to respond quickly to constantly changing environment. Since event-processor components are single purpose and decoupled, changes are isolated and quickly change without impacting others.

Ease of Deployment

Relatively easy to deploy because decoupled. Broker easier to deploy than mediator.

Testability

Unit testing not difficult, but testing is complicated by async nature of pattern.

Performance

Possible that will not perform well with all the messaging infrastructure, asynchronous capabilities, decoupled parallel execution outweighs cost.

Scalability

Naturally achieved through highly independent and decoupled event processors.

Ease of Development

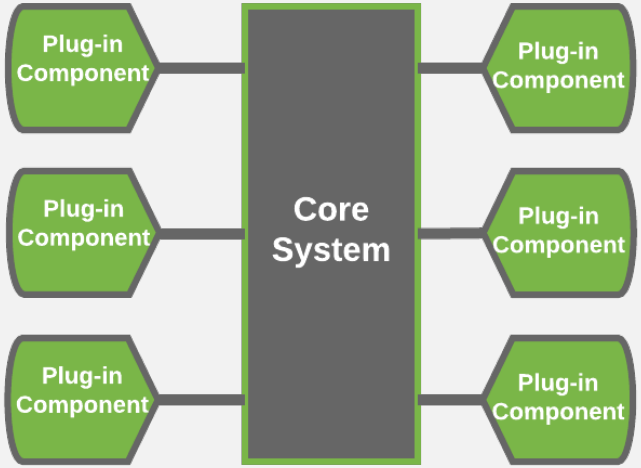
Development and can be complicated due to async nature of the pattern. Advanced error handling, unresponsive event processors, and failed brokers are complicated problems.

Microkernel Architecture

Plug it in, plug it in

Great for Products

- Add features as plug-ins
- Extensible
- Feature Separation
- Feature Isolation



Core vs Plugin

Microkernel

Core

Plug-in

- Minimal viable business logic to make application valuable
- Implements a plug-in registry

- Connected via OSGi, messaging, web services, point-to-point
- Remain independent of other plug-ins

Pattern Analysis

Microkernel Architecture

Agility

Able to respond quickly to constantly changing environment. Changes isolated and implemented quickly through loose coupled plug-in modules.

Ease of Deployment

Plug-ins can be added dynamically (hot-deploy).

Testability

Plug-ins can be tested in isolation and easily mocked by core system.

Performance

Not naturally high performance, often perform well because only include features you need.

Scalability

Core is often smaller in size and deployed as one unit, making it hard to scale.

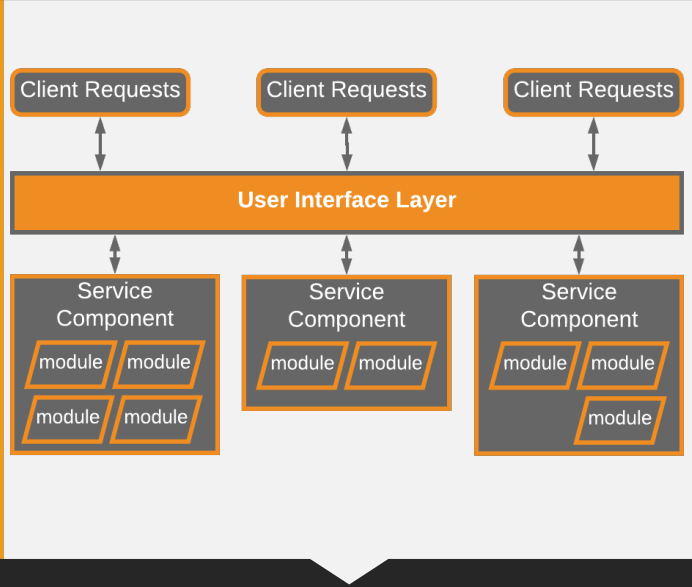
Ease of Development

Requires thoughtful design and contract governance. Contract versioning, plug-in registry, plug-in granularity, and wide array of connectivity choices make complex.

Microservices

Collect them all

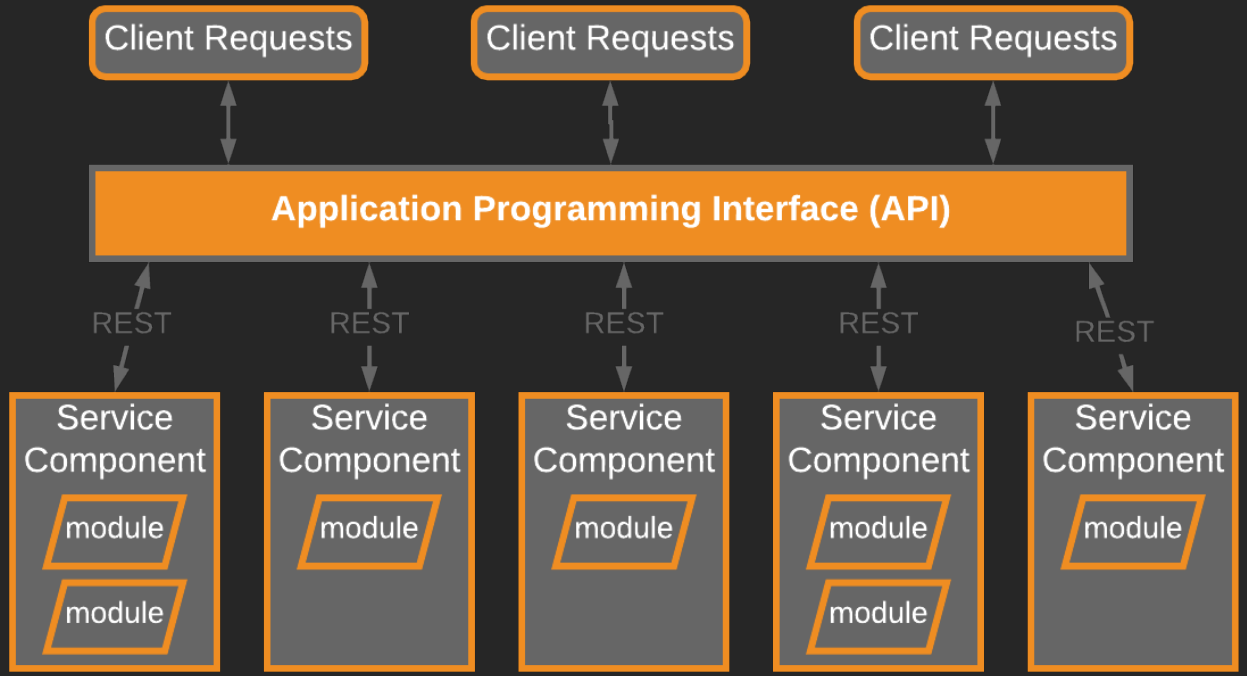
- Separately deployed units
- Distributed Architecture
- Easier deployment due to isolation
- Microservice as a service component
- Microservice has a single purpose
- Evolution of monolithic and SOA



API REST-base Topology

Microservices

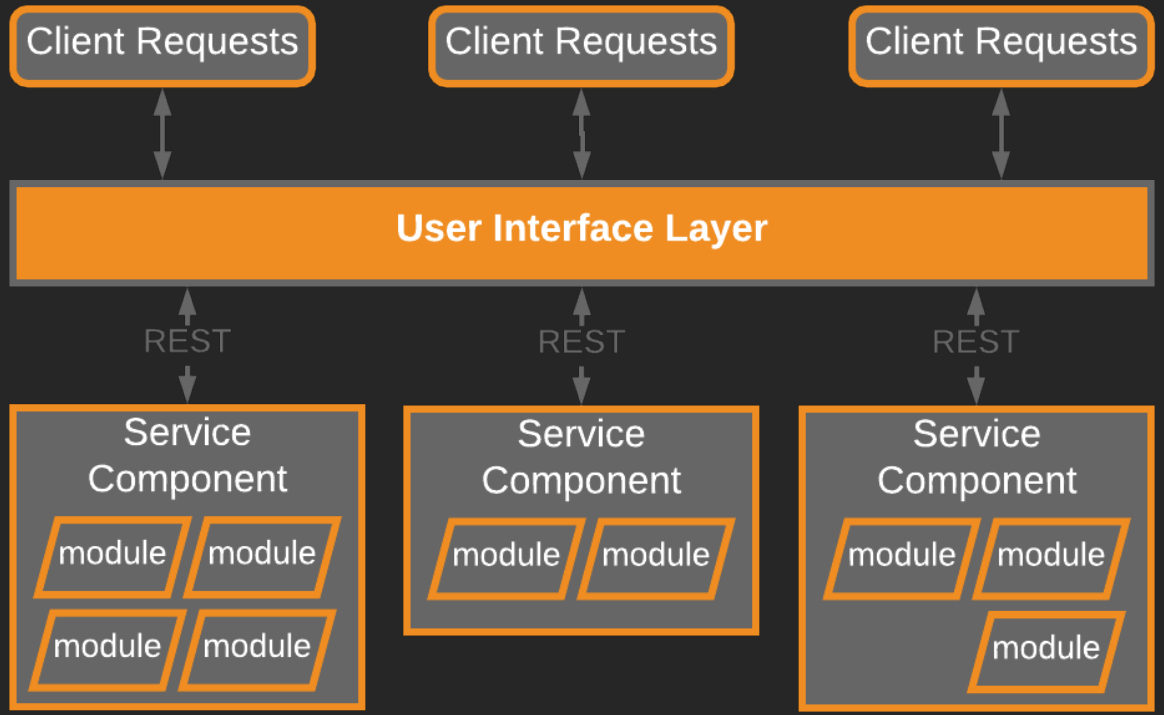
- Useful for websites
- Fine-grained services
- REST-based interface



Application REST-base Topology

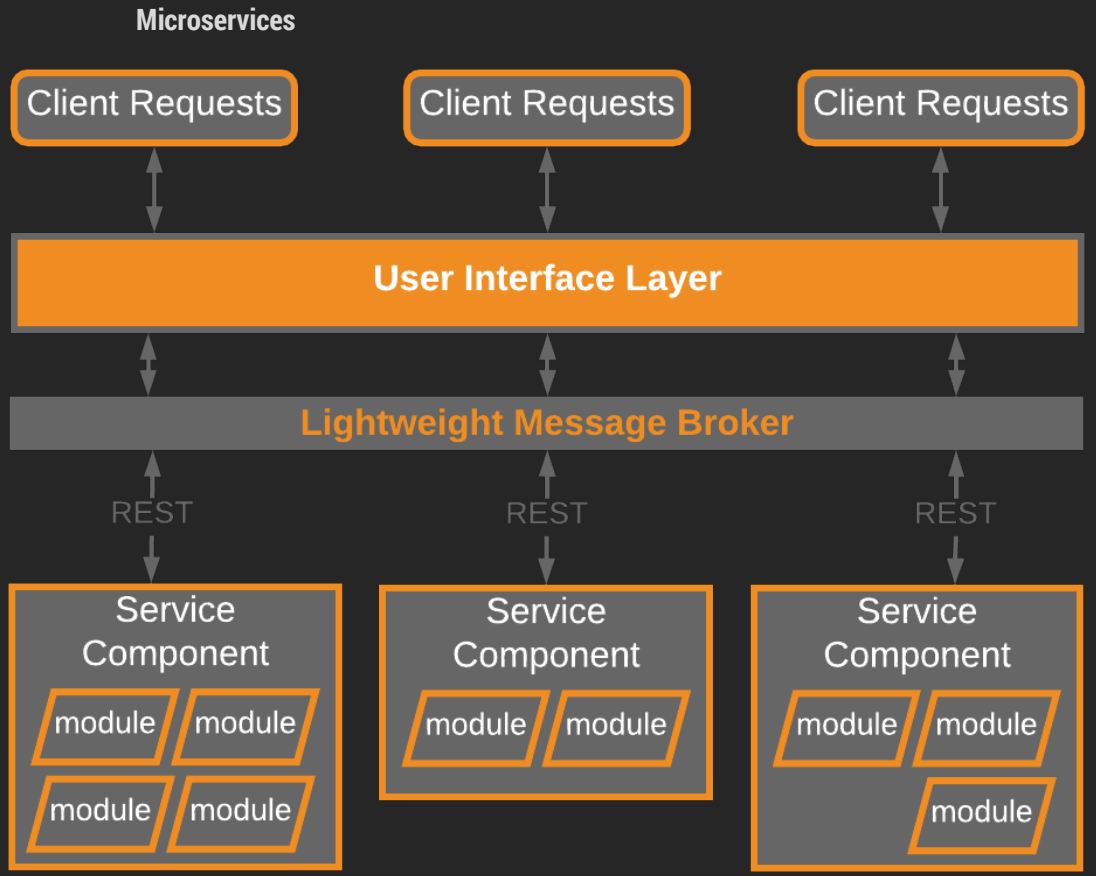
Microservices

- Useful for fat-client apps
- Through screen not API
- Coarse-grained components
- REST-based interface



Centralized Messaging Topology

- Remote Access via Message Broker
- NOT SOA-Lite
- Coarse-grained components
- REST-based interface



Granularity of Service Components

Avoid Dependencies and Orchestration

L

Too Coarse
May not realize benefits of deployment, scalability, testability, & loose coupling.

S

Too Fine
Leads to service orchestration, which turns lean microservices into heavy SOA, with complexity, confusion, expense, & cruft.

M

Just Right
Get all of the agility, deploy-ability, testability, scalability, & develop-ability

Pattern Analysis

Microservices Architecture

Agility

Able to respond quickly to constantly changing environment. Separate deployed units, changes limited to single service component.

Ease of Deployment

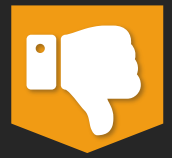
Easy to deploy due to the decoupled nature of event-processing components.

Testability

Isolation of business functionality into independent applications makes easy to test. Regression testing a single service is possible.

Performance

Possible to create performant solution, but pattern does not lend itself to performance, due to distributed nature of pattern.



Scalability

Because app is split into separate deployment unit, each service can be scaled.

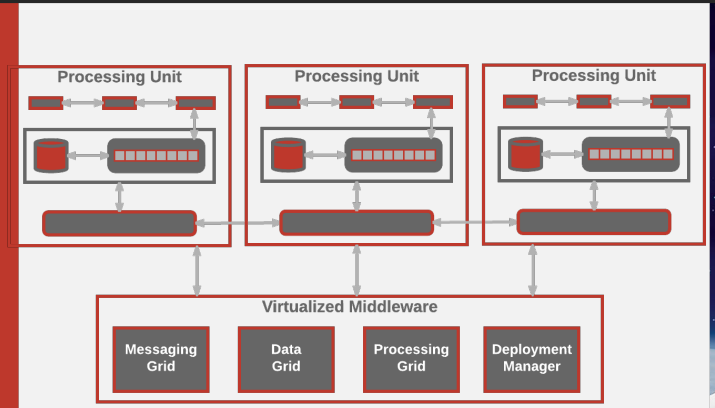
Ease of Development

Due to isolation of distinct service components , development is focused on smaller scope. Less chance a developer's changes will impact other service components.

Space-Based Architecture

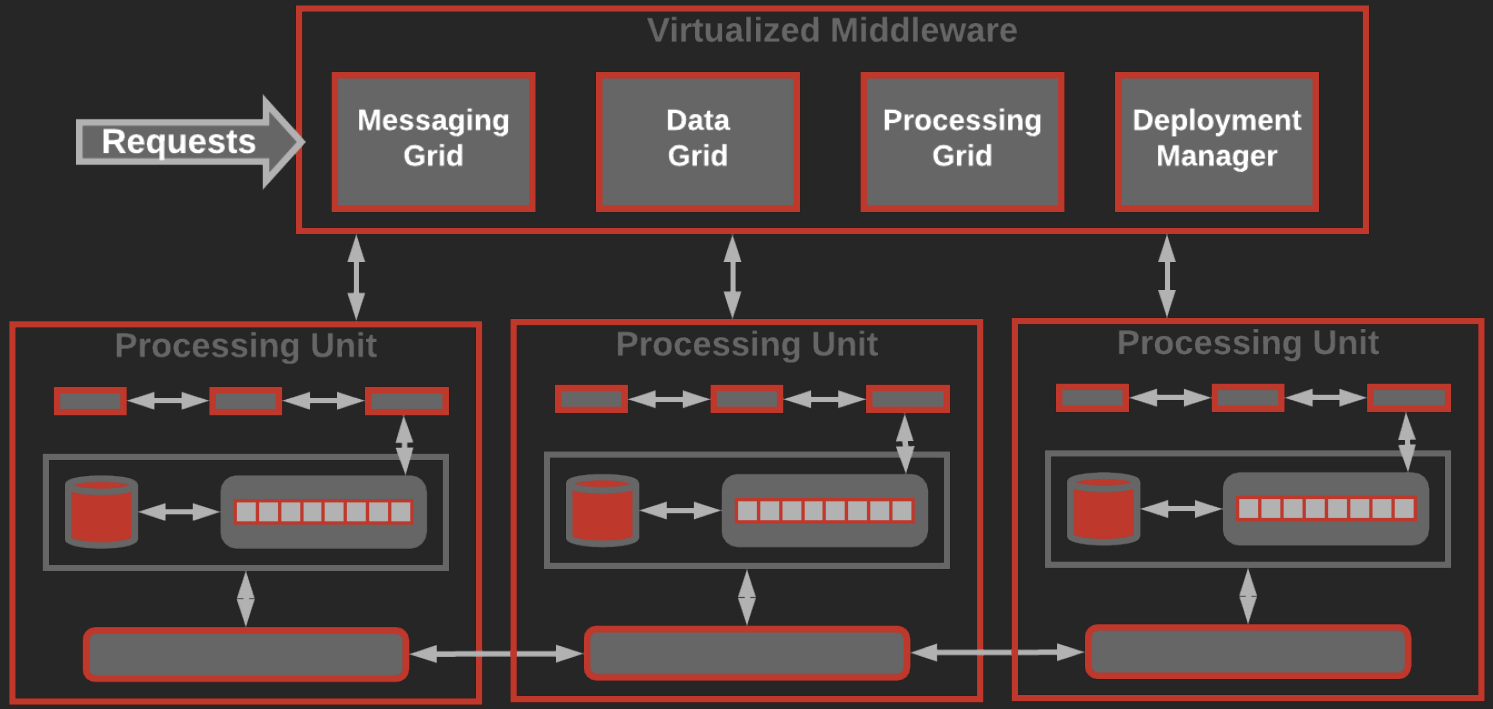
Build it

- Solve Scalability issues
- Address concurrency issues
- For high concurrent user volumes
- Aka the cloud architecture pattern
- Remove central DB
- Use replicated in memory data grids
- Complex & Expensive



Overview

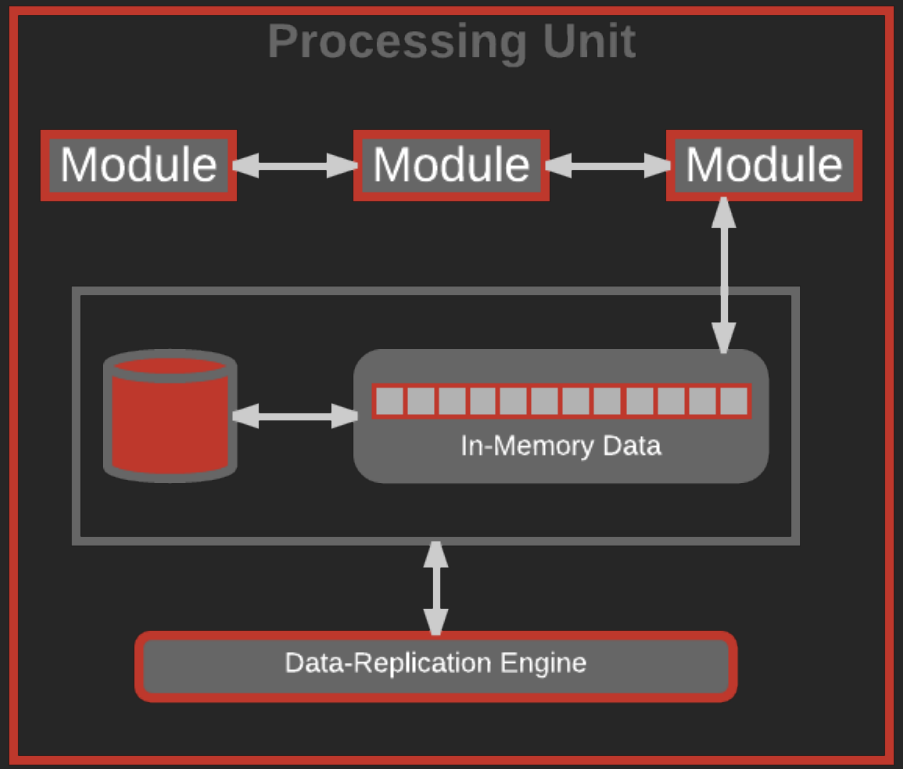
Space-Based Architecture



Processing Unit

Space-Based Architecture

- Application logic in modules
- In-memory data grid
- Optional async persistence store
- Data Replication Engine



Virtualized Middleware

Space-Based Architecture

Message Grid

- Manage input requests
- Manage Session
- Determine which active processing unit to use
- Scale of complexity

Data Grid

- Manages data replication
- Talks to each replicator
- Replication in parallel async

Processing Grid

- Optional component
- Manages distributed requests
- Coordinates processing units
- Orchestrates

Deployment Manager

- Manages startup/shutdown of processing units
- Monitors response times & user loads
- If load increases start new processing unit

Pattern Analysis

Space-Based Architecture

Agility

Because processing units can be brought up/down quickly.

Ease of Deployment

While not decoupled or distributed, sophisticated cloud-based tools allow for easy push button deploy.

Testability

Expensive and difficult to create user load to test architecture.

Performance

Achieved by in-memory data access and caching build into pattern.

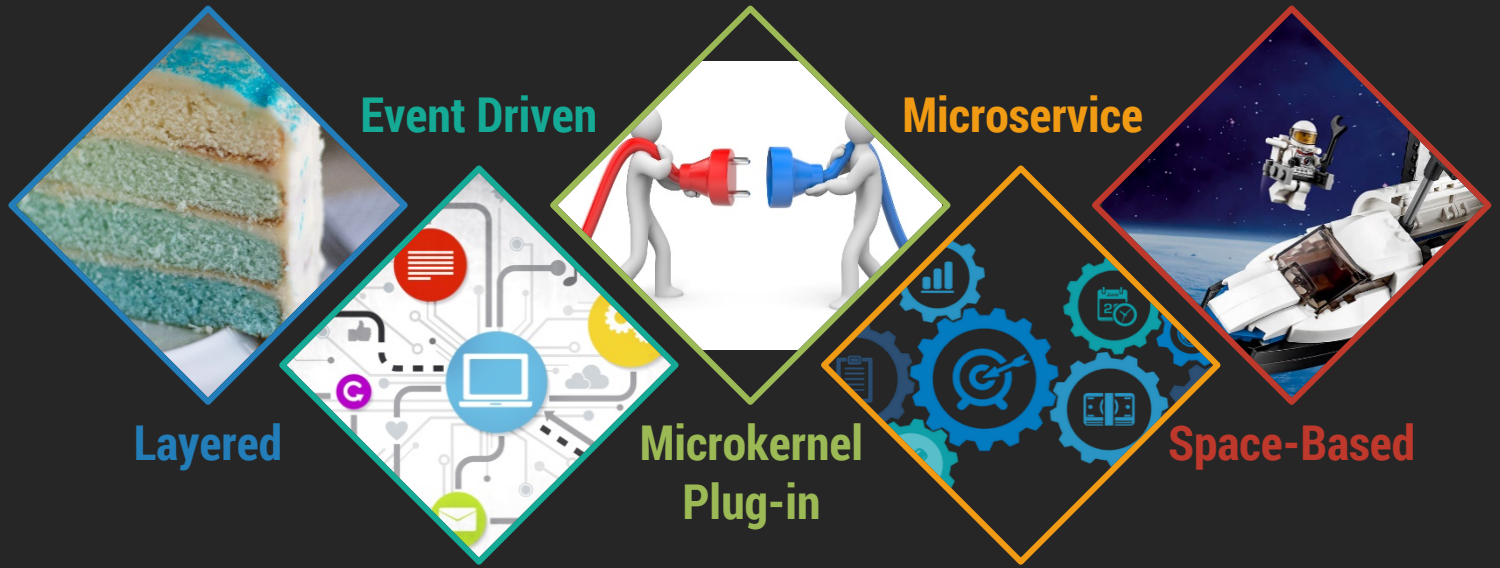
Scalability

Achieved by not depending on centralized database, which is often the limiting bottleneck.

Ease of Development

Complex caching and in-memory data grids make it difficult to develop. In addition, need to be diligent watchers of performance issues being introduced.

Review


Layered
Event Driven
Microkernel
Plug-in
Microservice
Space-Based

Contact Information

https://twitter.com/CognitiveBurden

https://github.com/cognitiveburden

cognitiveburden@gmail.com