From Legacy Monolith to Microservices via EventStorming
Thank You, CodeMash 2024 Sponsors!

They’re the reason why tickets are affordable! They help offset a lot of the costs!
Key Terms

- **Legacy** Monolith
- Microservices
- Modulith
- EventStorming
If you are doing simple CRUD apps or simple business systems, these solutions are overkill.
ALL T-SHIRTS ON SALE THIS WEEKEND

Previous

Showing 10 of 12 products - Page 1 - 2

Next

[ ADD TO BASKET ]
ROSLYN RED SHEET
$ 8.50

[ ADD TO BASKET ]
.NET FOUNDATION SWEATSHIRT
$ 12.00

[ ADD TO BASKET ]
PRISM WHITE T-SHIRT
$ 12.00

[ ADD TO BASKET ]
.NET CUP

[ ADD TO BASKET ]
.NET HOODIE BLACK

[ ADD TO BASKET ]
.NET HOODIE BLUE

[ ADD TO BASKET ]
.NET HOODIE GREY
Common Characteristics of a **Legacy** Monolith

- Monorepo
- Tightly coupled code
- Scalability is iffy
- Slow release cycles
- Difficult to update
- **Big Ball of Mud**
- **Spaghetti Code**
MICROSERVICES! MICROSERVICES!

MICROSERVICES!
Characteristics of Microservices

- suite of small services
- running in its own process and communicating with lightweight mechanisms
- built around business capabilities
- independently deployable
- bare minimum of centralized management
- may be written in different programming languages
- use different data storage technologies.

Characteristics of Microservices

• Small, independent, loosely coupled services
• Autonomous services
• Each service manages its own data
• Polyglot friendly
• Scalable
• Resilient – one failure doesn’t bring them all down
• DevOps friendly

Microservices from eShopOnContainers
Spring Petclinic - Microservices
Monolith || Microservices
Courage

Don't refactor complex systems without it.
This pattern has led many of my colleagues to argue that **you shouldn't start a new project with microservices**, even if you're sure your application will be big enough to make it worthwhile.

YAGNI

It may look like overkill, but I’m sure we’ll need it eventually.
PROJECT TYPE

Monolith  Modulith  Microservices
Characteristics of a Modular Monolith

- Single code base
- Module-based organization
- Separation of concerns
- Shared services
- Scalable
- Flexible
- Centralized datastore
- Interaction between modules happens via public APIs or messaging
- Intermediate step between Monolith and Microservices
<table>
<thead>
<tr>
<th>Synchronous</th>
<th>Asynchronous</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Communicate via APIs</td>
<td>• Communicate via messaging</td>
</tr>
<tr>
<td>• Pro: Easy to implement</td>
<td>• Temporally decoupled but…</td>
</tr>
<tr>
<td>• Pro: Reduced operational complexity</td>
<td>• <strong>Not guaranteed to be loosely coupled</strong></td>
</tr>
<tr>
<td>• Con: Strong coupling</td>
<td>• Con: More complex implementation</td>
</tr>
</tbody>
</table>
But wait!

Isn’t this still just a monolith?

Won’t we have the problems mentioned earlier?
When to Use Modular Monoliths vs Microservices

<table>
<thead>
<tr>
<th>Modular Monoliths</th>
<th>Microservices</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Greenfield, early stage</td>
<td>• Large, complex business systems</td>
</tr>
<tr>
<td>• Small to medium sized apps</td>
<td>• Scalability</td>
</tr>
<tr>
<td>• Low to medium scaling</td>
<td>• Polyglot support</td>
</tr>
<tr>
<td>• Low complexity business solutions</td>
<td>• Fault isolation</td>
</tr>
<tr>
<td>• Team expertise in a tech stack</td>
<td>• Multiple independent development teams</td>
</tr>
</tbody>
</table>
How do we grow and improve from this legacy monolith?
But it’s a mess now! How do we go from disaster to better?
EVENT STORMING!
What is EventStorming?

Collaborative workshop-based experience to gain a shared understanding of a complex business system
Key People in EventStorming

• Facilitator
• People with *questions*
• People with *answers*
  • Local experts, masters of their silo
EventStorming Grammar

- **Conversations** with sticky notes
- Eventually structured to follow a particular **grammar**
Merge the people, split the software.

- Alberto Brandolini
Types of EventStorming

- **Big Picture EventStorming**
  - Goal: Generate shared understanding
  - Used for discovery

- **EventStorming for Process Modeling**
  - Goal: Address the Hotspots
  - Used for exploring a process
  - Typically limited to a single end-to-end process

- **EventStorming for System Design**
  - Goal: Evaluate a system and propose a solution
  - Design a solution
  - Be aware of alternatives
  - Hide unnecessary complexity from the users

- **EventStorming for People Experience**
  - Goal: Understand customer/user/persona interactions and experiences

- **EventStorming for Refactoring**
  - Given an existing solution, use EventStorming to identify what is available and potential refactor points
Common Flow of EventStorming

• How do the EventStorming sessions flow into each other?
What if all you have is a Legacy Monolith?

• While you can build upon discussions in other EventStorming sessions, they are prerequisites for refactoring.

• No prior EventStorming session needed for refactoring discussions
EventStorming for Refactoring – Basic Sticky Notes

- What happened? (Domain Events)
- Who uses the system? (Users/Roles/Personas)
- What systems are involved? (Systems)
Identify Hotspots

Questions?
Pain Points?
EventStorming for Refactoring – Additional Sticky Notes

- What triggers the event?  
  Command

- Large Grouping of a Business Problem  
  Aggregate
Bounded Contexts

• Once **aggregates** are identified, **boundaries** are drawn
• Usually helps identify **microservices**
• Examples:
  • Product
  • Order
  • Customer
  • Delivery
  • Payment Processing
  • Tax Systems
Analyze EventStorming Results

• Once **events** are laid out, identify entities related to those events. Is there one that makes sense as an **aggregate**?

• Create **aggregates** with their functions.

• Draw boundaries.

• Use **bounded contexts** as guides for microservices... or modules.
Talk about splitting functionality as a team

• Do the splits make sense?
• Is there a ubiquitous language established? This would be a good point to get everyone on the same terms.
Modulith to Microservices?

Modulith as a stepping stone
Strangler Fig Pattern

• Create the adapter for the front-end to use
• Migrate one service at a time
  • Focus on the services where microservices will give the best ROI
  • Cost isn’t just $$$
  • Maintenance costs
  • Decoupling time
• Monolith shrinks over time
• If going wholly microservices, monolith will disappear

• But sometimes, a mix of the two may be where you are for awhile

Photo by David Clode on Unsplash
Low Hanging Fruit

• Break API layer out of monolith
• API calls to monolith and microservices
• Draw out service dependencies
  • Might determine order of migration
• New features follow the microservices pattern
• If specialized development teams, let them figure the priorities for their specialties.

Ease of Separation vs Benefit of Separation
Case Study: Amazon Prime Video Detector

• Moved from serverless, microservices to modular monolith for defects monitoring tool

• Problems:
  • Infrastructure at high scale was very expensive
  • Some components were hitting hard scaling limits
  • Orchestration with AWS Step Functions was costly – multiple state transitions every second of the stream, billed per state transition
  • Passing video around was expensive in making Tier-1 calls to S3
  • Cost of building the blocks of code was too high for large scale

Diagram of Amazon Prime Video Detector with Microservices — Source: Prime Video Tech
Amazon Prime Video Detector (continued)

• Moved to a monolith with vertical scaling for detectors
  • Detectors run within the same instance
• Benefits of moving to a monolith
  • Reduced costs by 90%
  • Streamlined for development, maintenance, and debugging
  • Minimized latency
• Microservices only when necessary
• Not an all-or-nothing approach – there is grey area!

Diagram of Amazon Prime Video Detector with Microservices – Source: Prime Video Tech
But wait... capacity limits?
Amazon Prime Video Detector (continued)
Recap
IS IT ALL-OR-NOTHING?
MONOLITH OR MICROSERVICES?
WHAT ARE THE KEY POINTS FOR USING MICROSERVICES?
When to Use Microservices

- High business complexity
- High need for fault isolation and resilience
- Specialized teams with polyglot backgrounds
- Independent scalability
- Enhanced maintainability
  - Agility for growth and improvement
WHAT ARE THE KEY POINTS FOR USING MODULITHS?
When to Use Moduliths

- Low-to-no business complexity
- Cost-effective
- Tight integration
- Reduced operational complexity
- Team expertise in a tech stack
Additional Resources

- **eShopOnContainers** - .NET microservices sample app
- **Telerik** – Creating Good Monoliths in ASP.NET Core
- **Fear of Oblivion** – Build the modular monolith first
- **Martin Fowler** – Monolith First
- **Nicholas Frankel (A Java geek)** – Chopping the Monolith
EventStorming Resources

- **EventStorming** by Alberto Brandolini
- **The EventStorming Handbook** by Paul Rayner
- **EventStorming.com**
- **50,000 Stickies Later** – Alberto Brandolini (Explore DDD 2017)
- **100,000 Stickies Later** – Alberto Brandolini (Øredev 2019)
Thank you!


/in/sadukie

sarah.dutkiewicz@nimblepros.com