Transactional Cloud μServices
using Azure Service Bus

Chris Patterson

Principal Architect and Fellow
McKesson

Chris Patterson @**PhatBoyG**

- Principal Architect and Fellow McKesson
- Microsoft MVP (C#, .NET)
- Open Source Enthusiast

Agenda

- How did we get here?
- Why are we drawn to Microservices?
- What are the challenges of highly decoupled systems?
- How can we solve these problems?

Is this you?

- Curious about how to be cool, use Microservices, but still meet business requirements
- Learn how to decouple a finely-crafted monolith without giving up transactional consistency

Where did we start?

Database-Oriented Applications

- The era of the SQL DBA
- Everything connected to The Database
- Business logic in stored procedures
- Applications were “dumb” clients

Distributed Object Protocols

- The era of object-orient all the things
- Everything connected via CORBA or DCOM
- Location didn't matter, network transparency

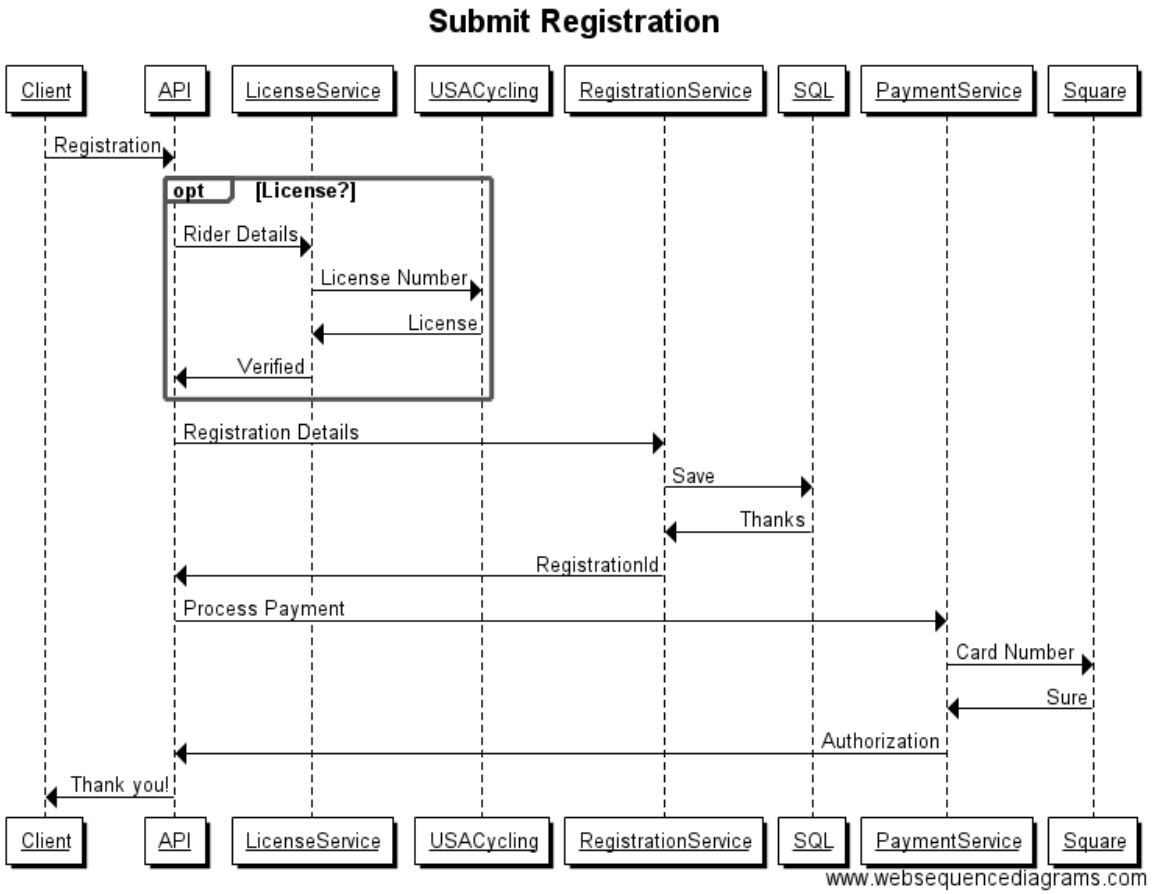
- It turns out, the network *does* matter

Service Oriented Architecture

- The first age of SOA
- Everything connected via SOAP
- Network was there, we just ignored it
- Rich client proxies, designer-driven architecture

- It turns out, the network still matters

The Nesting of RPC



Enter Microservices

Microservices

- Many small projects, many solutions
- Many services, many processes, many servers
- Connected via lightweight protocols, typically HTTP

- Requires extensive integration testing
- Requires more deployment skills and tools

Separation of Concerns

- Vertically partition services by business context
  - Micro- does not mean nano- (or pico-)
- Encapsulate dependencies, including storage
  - Database per context, or at a minimum schema per context
- Explicit public interface contracts

Increased Complexity

Once created, complexity cannot be destroyed – it can only be moved.

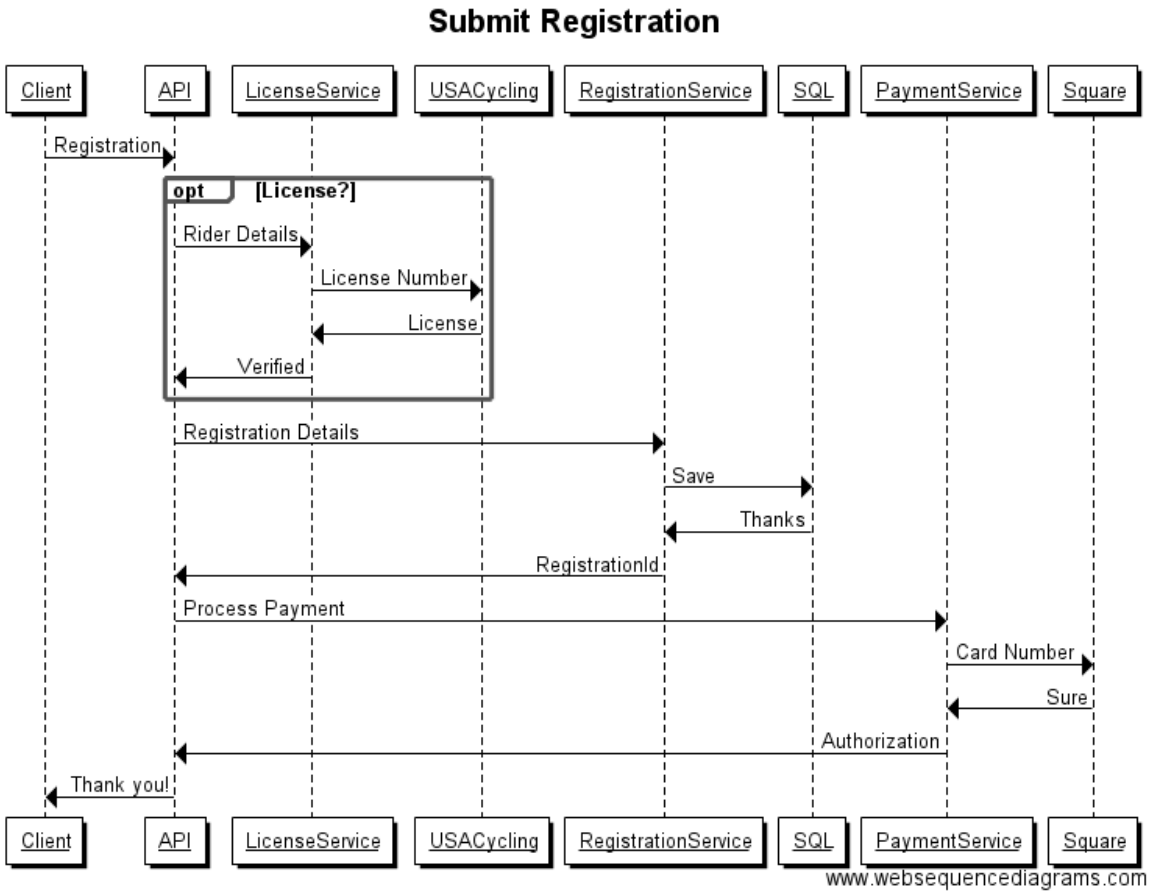
Microservices Complexity

- Service discovery
  - It's more than an endpoint address
    - URI format, methods, and access patterns
    - Content structure and organization
- Versioning service interfaces
- Transient fault handling
  - The fallacies of distributed computing
  - The network is not reliable
- Too many servers, containers, images, builds, repositories, etc.

Increased Latency

A network hop is two orders of magnitude slower than an in-process method reference.

The Return of the Nesting of RPC



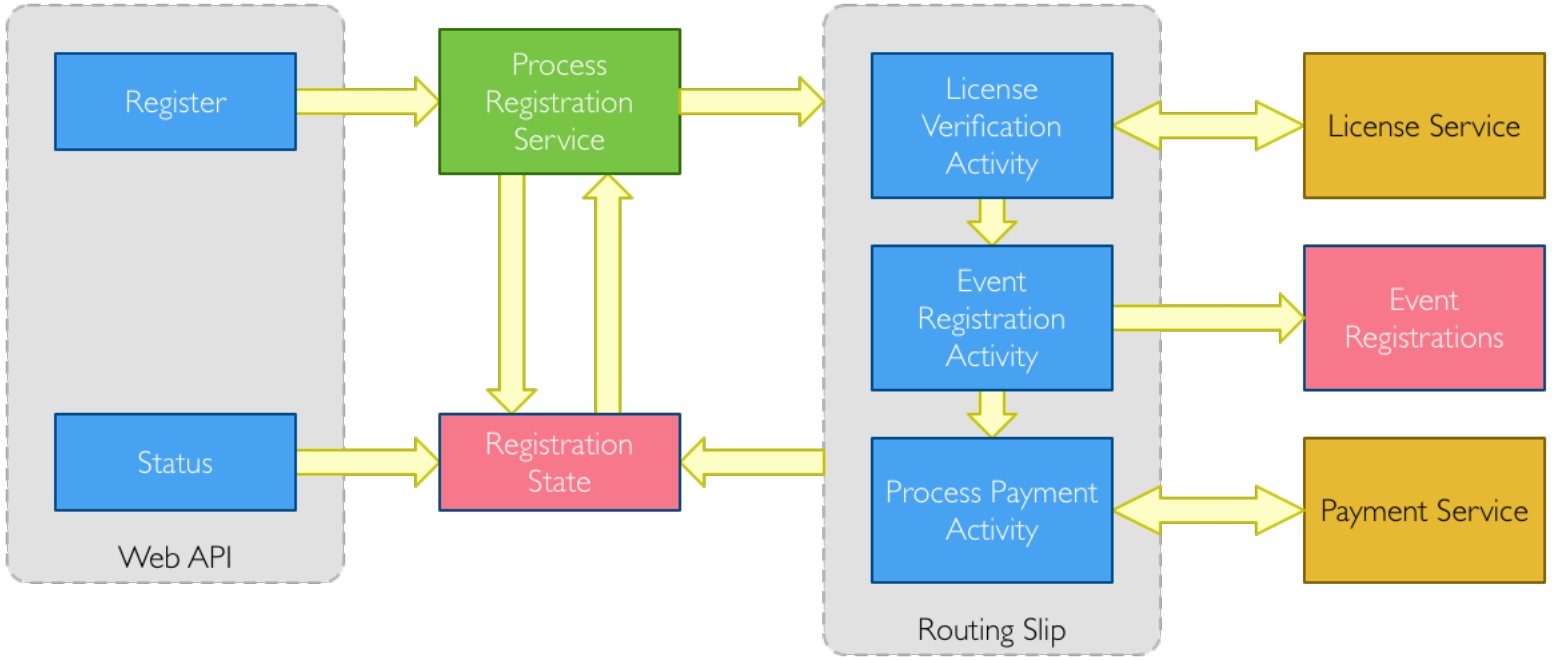
Deceased Consistency

The benefit of encapsulation provided by microservices comes at a very high cost – *consistency*.

Our Story

Race Registration

Event Registration Flow



Finely-crafted Monoliths

- Few projects, even fewer solutions
- Well factored code
  - Layered architecture
  - Separation of concerns
  - Clear interface segregation
  - Extensive unit tests
- High cohesion
- Easy to code, test, and deploy

Commands vs Queries

Separated behavior into two buckets

- Commands
  - Initiate action in the application
- Queries
  - Ask a question, no side effects
  - Two types of queries
    - Impatient, just curious
    - Important, willing to wait

Command Acceptance

- Redesigned command APIs to only *accept* requests
- Commands are written to a queue
- Intermediate response returned immediately
  - HTTP 202/Accepted
  - Resource identifier for status updates
  - Alternatively, command may include callback
- Endpoints remain stateless

Command Processing Service

- Standalone Windows service
- Handles commands from the service queue
- Publishes events during command processing
  - *Command Received*
  - *Command Completed*
  - *Command Faulted*
  - *Command Rejected*

Asynchronous Benefits

- Command handling decoupled from endpoint
- Service is active, versus reactive
- Service can be stopped, updated, restarted without affecting endpoint availability.
- Queues are easily load balanced, add workers to scale up, automatically handled by Azure.

Are we there yet?

- Services are still stateless
- Notifying the requester of the command disposition
  - Hint: use an event observer
- What about the status query?
  - Enter: state

Tracking Command State

- Behavior defined using a state machine
  - Introducing Automatonymous
- Observes command events
- Updates state based on event transitions

Querying Command State

- State is persistent
- Impatient
  - Read by endpoint directly, either from primary or secondary storage
- Important
  - Queried via request/response to state machine

Business Transactions

- A sequence of business activities that are performed to complete an operation.
- Some activities are harmless
  - Look up member address, verify membership
- Some activities have consequences
  - Charge payments, allocate seats, provision materials

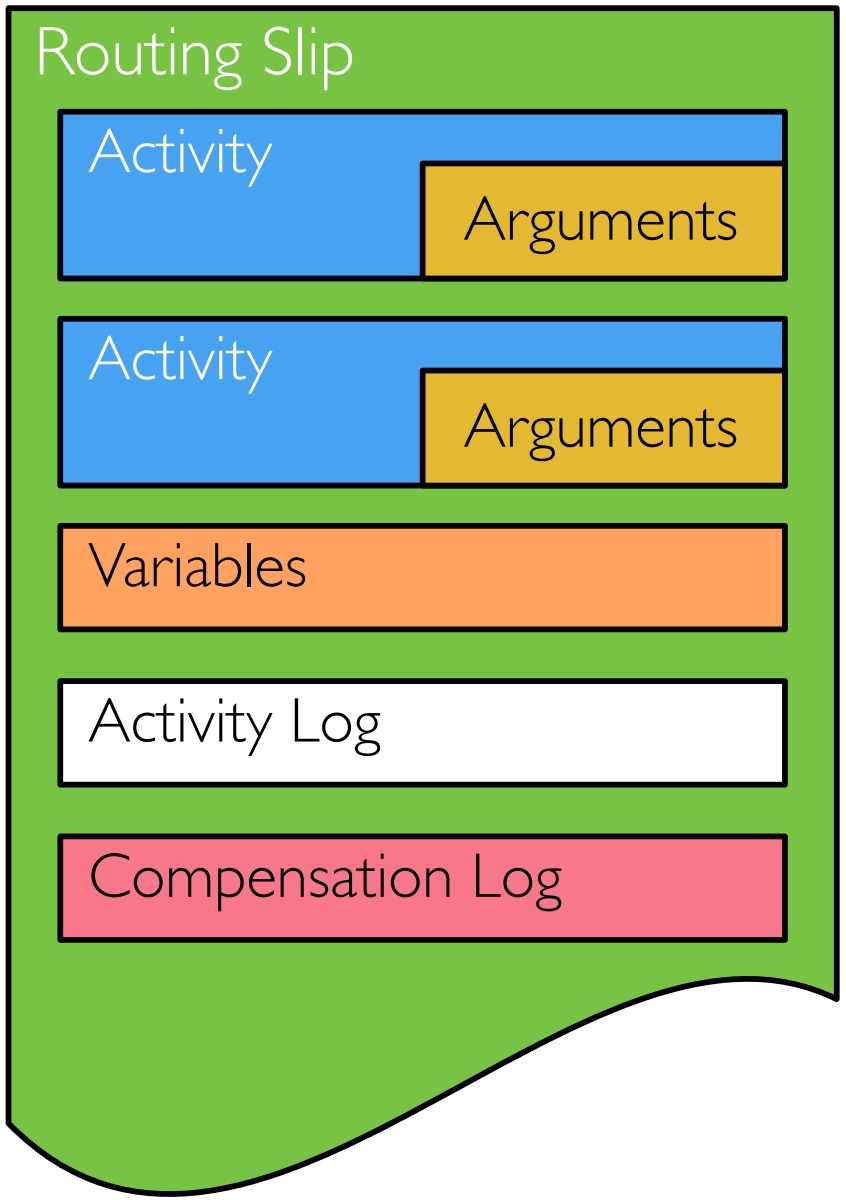
Reaching Consistency

- Immediate consistency in business is rare
- Records become consistent over time
  - Time may be seconds, days, even months
- Eventual consistency is the natural order of business
- Local consistency is key to user acceptance

Routing Slip Transactions

- Based on a real-world artifact, the routing slip
  - If messages are papers, this is the checklist
- An ordered sequence of activities (services)
- Activities transfer the routing slip directly
  - No centralized broker during execution
- Activities store compensation logs in the routing slip
- Activities may publish audit events

Routing Slip



Transaction Faults

- A faulted activity sends the routing slip to the last completed activity
- Each activity applies a compensating action, undoing what was previously done

Wrap Up

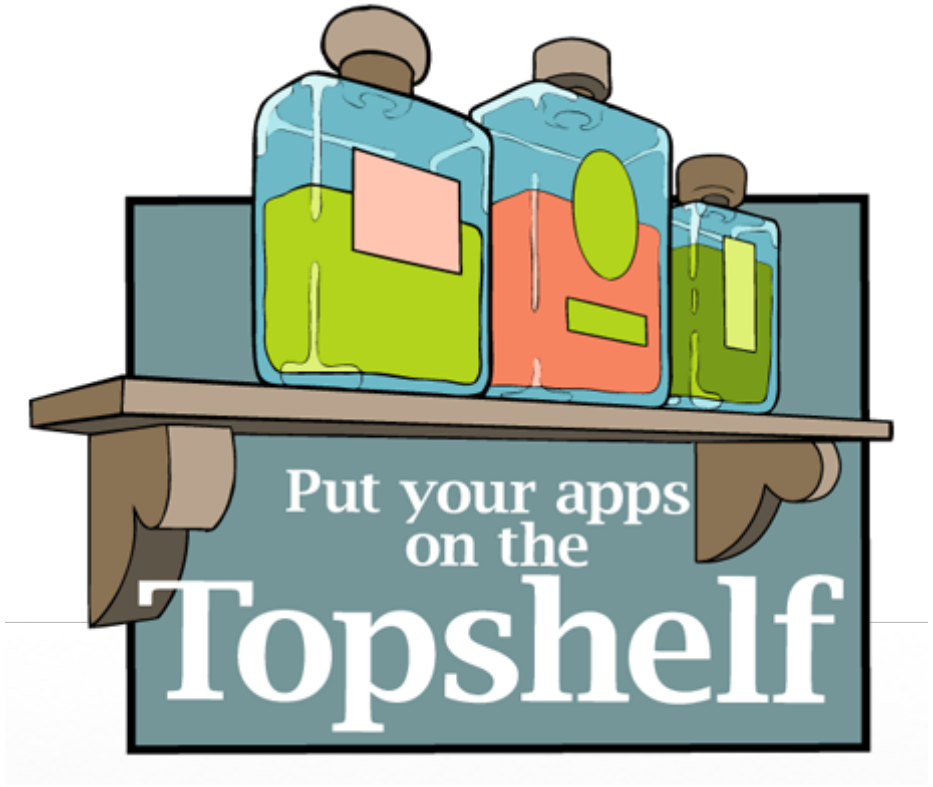
- Reviewed microservice benefits and challenges
- Showed how asynchronous decoupling increases availability
- Demonstrated how business transactions can be reliably modeled while maintaining service separation
- Learned how state machines can be used to track asynchronous transactions through events

The BITS

chris@phatboyg.com

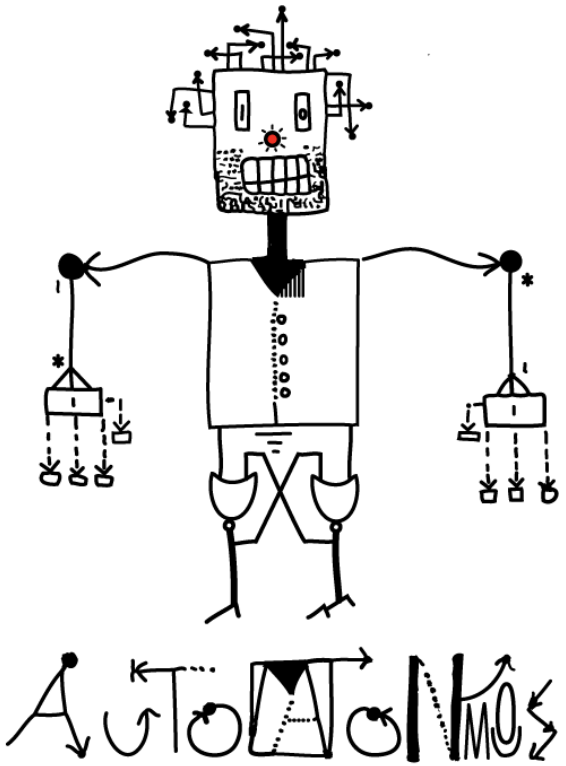
MassTransit

is a *free, open-source* distributed application framework for .NET. MassTransit makes it easy to create applications and services which leverage message-based, loosely-coupled asynchronous communication for higher availability, reliability, and scalability.



Topshelf

is an open source framework for creating and deploying .NET services.



Automatonymous

is a *free, open-source* state machine library for .NET developers. It provides an easy-to-use declarative syntax for defining states, events, and behavior. Automatonymous can be used standalone, and includes complete integration with MassTransit.

@PhatBoyG

http://masstransit-project.com

https://github.com/MassTransit