VSH11 Exploring Microservices in a Microsoft Landscape

Marcel de Vries
CTO, Xpirit
Level: Intermediate

About me: Marcel de Vries
Also regional director
mdevries@xpirit.com
@marcelv
mdevries@xpirit.com
Xpirit
http://fluentbytes.com
Definitions of microservices

- Small and focused on doing one thing well
- Autonomous

“Loosely coupled service oriented architecture with bounded contexts”
Adrian Cockcroft (Netflix)

“SOA done right”
Anonymous

“...services are independently deployable and scalable, each service also provides a firm module boundary, even allowing for different services to be written in different programming languages.”
Martin Fowler (Thoughtworks)
Service Oriented Architecture

Services as a network of independent parts
Microservices architecture

Every service is independent

- Can have its own technology stack
- Maintains its own state
- Is delivered independent of other parts of the system
- Everything is highly automated
- Each Micro Service has its own team
  - Both delivery and operations are their responsibility (devops)
VSH11 - Exploring Microservices in a Microsoft Landscape - Marcel de Vries
Composition is key!

**Chosen strategy**

Obtain simplicity by adding complexity

<table>
<thead>
<tr>
<th>Monolith</th>
<th>Microservice</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simplicity</td>
<td>Partial deployment</td>
</tr>
<tr>
<td>Consistency</td>
<td>Availability</td>
</tr>
<tr>
<td>Inter-module refactoring</td>
<td>Preserve modularity</td>
</tr>
<tr>
<td>Single platform</td>
<td>Multiple platforms</td>
</tr>
</tbody>
</table>

VSH11 - Exploring Microservices in a Microsoft Landscape - Marcel de Vries
Benefits

- Selective scaling
- Resilience against failure
- Deployment of smaller pieces with lower risk
- Alignment with organization
- Ownership with teams
- Heterogeneous technology landscape
- Replaceability
- Composability

From horizontal to vertical
Change your approach to vertical partitions

- Organized by logical cohesion
- Teams own layers spanning domains
- Unified technology choice
- Modeled after organization’s domains
- Owned by team
- Top to bottom
- Isolated from each other as much as possible
Freedom of choice

Frontend integration

Social
Graph
Report
Blob
Product
Document

Bounded contexts

Sales Context
Opportunity
Pipeline
Territory
Sales Person
Customer
Product
Support Context
Ticket
Defect
Product Version
Example: otto.de

Example Microservices architecture - otto.de
Example Microservices architecture - otto.de

- Each with own UI (on website)
- Well defined interfaces
- No shared code or state
- REST interfaces for each vertical
  - Pull based data replication
  - Front end integration (JavaScript/Edge includes)

How to implement inter-system dependencies?
Event Driven Architecture

- Systems keep their own state.
- Events are used to notify changes.
- Systems use events to replicate data.
- Each service will have its own database.
- Downsides:
  - Latency in data changes
  - No central control of business logic
  - Challenge to reconcile in case of disaster

Characteristics of EDA

- Broadcast Model
  - Anyone can receive the events

- Completely Asynchronous
  - Publishers do not wait on any subscribers

- Fine Grained Events
  - Individual events

- Ontology
  - Hierarchy of events
  - Publishers can subscribe to these selected events
Why are event driven architectures relevant?

- Increased use of asynchronous notifications
  - Mobile push based notifications, twitter, whatsapp, etc.

- Increased application of asynchronous languages and frameworks
  - Javascript, node.js, AJAX, C# Async & Await, Signal R

- Increased need for fast, robust and scalable systems
  - Reactive Manifesto

- Increased need for systems agility
  - Faster, more frequent changes

Cross-service coupling

- Smart endpoints, dumb pipes
- No intelligent middleware
- Technologies to look at:
  - Queues (Azure, MSMQ, WebSphereMQ, ...)
  - Service Bus (NServiceBus, Azure ServiceBus, ...)
  - Language neutral data serialization
    (Google Protocol buffers, Apache Thrift)
Designed for failure

- Services can and will fail: prepare!
- Do not rely on another service’s availability
  – Introduce features in your app to deal with it
- Test under stress and with failing services

Microservices Architecture characteristics

- Organized around business capabilities
- Products not Projects
- Decentralized data management
- Decentralized governance
- Componentization via Services
- Design for Failure
- Infrastructure automation
- Evolutionary Design
SOME IMPORTANT PATTERNS

Circuit Breaker Pattern

- Handle faults that may take a variable amount of time to rectify when connecting to a
  - remote service or resource.
  - This pattern can improve the stability and resiliency of an application.
Circuit Breaker Pattern

- Works together with Retry Pattern

Retry Pattern

- Enable an application to handle temporary failures when connecting to a service or network resource by transparently retrying the operation in the expectation that the failure is transient. This pattern can improve the stability of the application.
Health Endpoint Monitoring Pattern

- Implement functional checks within an application that external tools can access through exposed endpoints at regular intervals.
  - This pattern can help to verify that applications and services are performing correctly
  - E.g. App insights

Adopting DevOps practices

- Allow individual services to be built and deployed independently
- Continuous integration (CI)
- Continuous delivery (CD)
Possible MSA with classic MSFT technologies

Problems and solutions

- Rolling upgrades
- Availability guarantees
- Scale out architecture
- Resource governance
- Density
- Packaging and deployment
- Policy enforcement
- Granular versioning
- Stateful workloads
- Leader election

Mesos
Kubernetes
Zookeeper
Redis
Raven
MongoDB
Yarn
Fleet
Hadoop
Containers
Microsoft Azure Service Fabric
A platform for reliable, hyperscale, microservice-based applications

Microservices

Service Fabric
- High Availability
- Simple programming models
- Hybrid Operations
- High Density
- Hyper-Scale

Data Partitioning
- Rolling Upgrades
- Automated Rollback

Low Latency
- Stateful services
- Placement
- Constraints
- Health Monitoring
- Fast startup & shutdown
- Container Orchestration & lifecycle management
- Load balancing
- Replication & Failover
- Self-healing

Windows Server
- Linux

Azure

Windows Server
- Linux

Private Clouds

Windows Server
- Linux

Hosted Clouds

What can you build with Service Fabric?

• Stateless applications
  – A service that has state where the state is persisted to external storage, such as Azure databases or Azure storage
    • e.g. Existing web (ASP.NET) and worker role applications

• Stateful applications
  – Reliability of state through replication and local persistence
  – Reduces latency
  – Reduces complexity and number of components in traditional three-tier architecture

• Existing apps written with other frameworks
  – node.js, Java VMs, any EXE
Service Fabric Applications

Applications composed of micro services

Reliable Actors API
- High Availability
- Simple programming models
- Hybrid Operations
- High Density
- Hyper-Scale
- Data Partitioning
- Rolling Upgrades
- Automated Rollback
- Service Fabric
- Low Latency
- Stateful services
- Placement
- Constraints
- Health
- Monitoring
- Container Orchestration & lifecycle management
- Self-healing
- Replication & Failover

Reliable Services API
- High Availability
- Simple programming models
- Hybrid Operations
- High Density
- Hyper-Scale
- Data Partitioning
- Rolling Upgrades
- Automated Rollback
- Service Fabric
- Low Latency
- Stateful services
- Placement
- Constraints
- Health
- Monitoring
- Container Orchestration & lifecycle management
- Self-healing
- Replication & Failover

Stateless Services
- High Availability
- Simple programming models
- Hybrid Operations
- High Density
- Hyper-Scale
- Data Partitioning
- Rolling Upgrades
- Automated Rollback
- Service Fabric
- Low Latency
- Stateful services
- Placement
- Constraints
- Health
- Monitoring
- Container Orchestration & lifecycle management
- Self-healing
- Replication & Failover

Azure

Private Clouds

Comparing stateful programming models

**Reliable Service API**
- Build stateless services using existing technologies such as ASP.NET
- Build stateful services using reliable collections
- Manage the concurrency and granularity of state changes using transactions
- Communicate with services using the technology of your choice (e.g. WebAPI, WCF)

**Reliable Actor API**
- Build reliable stateless and stateful objects with a virtual Actor Programming Model
- Suitable for applications with multiple independent units of state and compute
- Automatic state management and turn based concurrency (single threaded execution)
Actors vs Services

<table>
<thead>
<tr>
<th>Actor</th>
<th>Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>State analogy</td>
<td>‘Entity graph’</td>
</tr>
<tr>
<td>Scope</td>
<td>Can’t work alone</td>
</tr>
<tr>
<td>Scale</td>
<td>Scale in millions of instances</td>
</tr>
<tr>
<td>Lifecycle</td>
<td>Active when needed</td>
</tr>
<tr>
<td>Concurrency</td>
<td>Single-threaded access</td>
</tr>
<tr>
<td>State</td>
<td>Change state</td>
</tr>
<tr>
<td>Communication</td>
<td>Fixed</td>
</tr>
<tr>
<td>External Access</td>
<td>Through Service</td>
</tr>
</tbody>
</table>

Service partitioning

- Services can be partitioned for scale-out
- You can choose your own partitioning scheme
- Service partitions are stripped across machine in the cluster
Demo

**AZURE SERVICE FABRIC**

Hosting your services

- Hosting in application container
- High density hosting
- One service per host

VSH11 - Exploring Microservices in a Microsoft Landscape - Marcel de Vries
Services up and running

OS
- Core OS
- Windows 10
- ubuntu

Virtual machines
- VSH11

PaaS
- Microsoft Azure
- Amazon Web Services

Container runtimes
- docker
- Rocket

Relation between number of services and complexity

Complexity

#services
Effect of complexity on system qualities

Is a Microservice architecture a good choice for the system you're working on?

It depends!

Source: http://martinfowler.com/articles/microservices.html
Conclusion

• Micro services is the new hot thing
• Still maturing, lot to be learned and discovered
• Might not be for everybody
• Think of a strategy to implement your Microservices
• Migrate gradually
• Evaluate specialized hosting platforms and frameworks and technologies

@marcelv
Fluentbytes.com
mdevries@xpirit.com
bit.ly/XpiritMagazine

Test Automation with CodedUI

Testing Web Applications with CodedUI