Microservices AntiPatterns and Pitfalls

Self-Assessment Workbook

Mark Richards
Independent Consultant
Hands-on Software Architect
Published Author / Conference Speaker

http://www.wmrichards.com
https://www.linkedin.com/in/markrichards3
@markrichardssa

Self Assessment Workbook

Analysis

1. Map out the communication paths between services. How much communication is there between services?
2. List the operations for each service. Is there high cohesion between operations for each service?

Goals

Minimize inter-service communication

Strive for high cohesion between operations in a service

Reference: *grains of sand pitfall*

Self Assessment Workbook

Analysis

1. Identify and document the business drivers and reasons for using a microservices architecture

Goals

Always make design and programming decisions within the context of business drivers

Reference: *developer without a cause pitfall*

Self Assessment Workbook

Analysis

1. List the technical and business pain points in your current application and environment.
2. List the reasons why you are considering using microservices
3. Do the characteristics and advantages of microservices address your pain points and business needs?

Goals

Make sure microservices is the right architecture style for your situation and business needs

Reference: *jump on the bandwagon pitfall*

Self Assessment Workbook

Analysis

1. Identify the context of each request (e.g., order number, customer id, confirmation number, etc.).
2. Create a logging API wrapper around you current logging tool

Goals

Consolidate logs from multiple services to identify the request flow of a specific request

Reference: *logging can wait pitfall*

Self Assessment Workbook

Analysis

1. Create a mapping of business requests to microservices
2. Identify those requests that require multiple services. Are ACID transactions required? If so consolidate services.
3. List the service databases requiring eventual consistency

Goals

Identify which requests require an ACID transaction

Identify how and when you will do eventual consistency

Reference: *using too much acid pitfall*

Self Assessment Workbook

Analysis

1. Where is the version documentation for each contract?
2. Document the procedures for communicating contract changes
3. How many versions for each contract do you support?
4. Do you have procedures defined for deprecating older versions?

Goals

Support backwards compatibility with your services and ensure effective communications when contract changes are made.

Reference: *static contract pitfall*

Self Assessment Workbook

Analysis

1. Identify each service owner and the services they own
2. Document the communication procedures between service owners

Goals

Associate services with specific service owners

Ensure service owners effectively communicate with one another

Reference: *service orphan pitfall*

Self Assessment Workbook

Analysis

1. Establish the average latency time for your remote access calls
2. Determine which requests need to be optimized based on multiple network hops

Goals

Understand your network latency for service access

Reference: *are we there yet pitfall*

Self Assessment Workbook

Analysis

1. Identify requests that can leverage asynchronous processing
2. Determine if you have broadcast capability needs
3. Determine if you have remote request transaction needs

Goals

Understand your remote processing capability needs

Improve performance through messaging capabilities

Reference: *give it a rest pitfall*

Self Assessment Workbook

Analysis

1. List the operations and hooks within your service template. Are there operations that can be added to the template?
2. Are development teams using the same base image and service template?

Goals

Encapsulate as much common framework code as possible in your service templates and base images to increase productivity and consistency when creating services.

Reference: *dare to be different pitfall*

Self Assessment Workbook

Analysis

1. Map services to existing data tables to identify ownership
2. Identify data overlap and dependencies between services
3. Develop functional migration plan for each service
4. Develop separate data migration plan for each service

Goals

Avoid frequent data migrations

Identify data sharing needs to help determine service granularity

Reference: *data-driven migration anti-pattern*

Self Assessment Workbook

Analysis

1. Identify the staging iterations in your iteration plan
2. Identify dependencies between staging and functional iterations
3. Identify opportunities for parallel staging tasks

Goals

Reduce staging iterations

Deliver business functionality faster

Reference: *all the world's a stage anti-pattern*

Self Assessment Workbook

Analysis

1. Identify requests requiring transformations
2. Identify requests requiring orchestration
3. Identify requests that access third-party systems
4. Analyze use of messaging microservices

Goals

Provide messaging capabilities without the use of an integration hub

Favor choreography over orchestration

Reference: *hop on the bus anti-pattern*

Self Assessment Workbook

Analysis

1. Document your strategy for reacting to server responsiveness
2. Identify opportunities for using the circuit breaker pattern
3. Externalize your timeout values through configuration properties or external data-driven values

Goals

Reduce the time to determine if a service is non-responsive

Avoid timing-out requests when the service has completed

Reference: *the timeout anti-pattern*

Self Assessment Workbook

Analysis

1. List your reporting requirements and the timeliness of reports
2. Can you maintain a separate reporting database or data warehouse?
3. List the data required for reporting from each service

Goals

Maintain a strong bounded context for each service and its corresponding data and provide timely data for reporting

Reference: *reach-in reporting anti-pattern*

Self Assessment Workbook

Analysis

1. Identify shared library and shared module dependencies
2. Document your versioning strategy for shared libraries
3. Break apart shared libraries into smaller context-based libraries (e.g.,. security.jar, conversion.jar, etc.)

Goals

Reduce the number of shared modules between services

Create fine-grained shared libraries to control change

Reference: *i was taught to share anti-pattern*