Post-hoc systems architecture: An argument for microservices first

—

Anastas Stoyanovsky
Charles Gala
Aiton Goldman

IBM Watson
7 May 2019

IBM Watson / © 2018 IBM Corporation

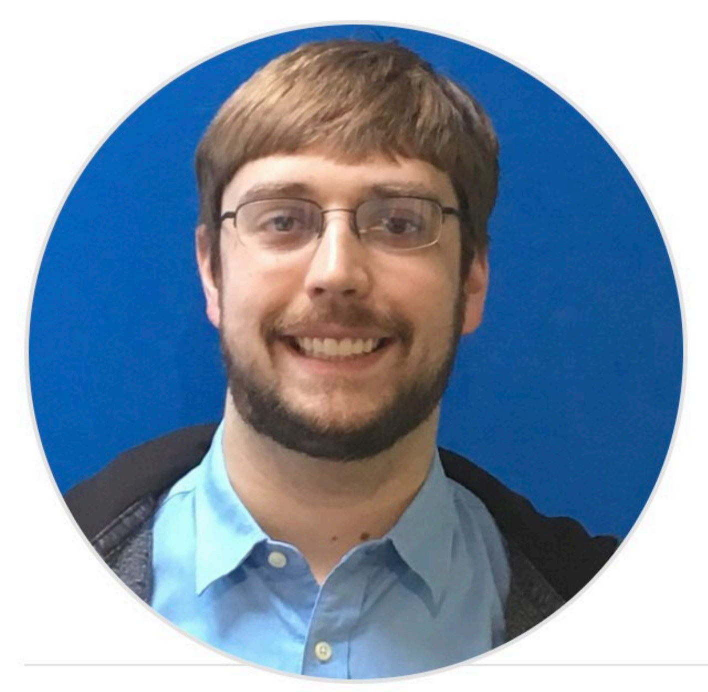
Introduction

© 2018 IBM Corporation

Who We Are

Anastas works in artificial intelligence and information retrieval for IBM Watson, working in collaboration with IBM Research. He holds an MSc in pure mathematics from Purdue University and a BSc in mathematics, computer science, and neuroscience from the University of Pittsburgh.

Who We Are



Charles Gala is a software engineer at IBM Watson, working on machine learning powered information retrieval for Watson Discovery Service. He has a research background in machine learning and pattern recognition techniques using live video. Charles has a master's degree in computer science and engineering from the Pennsylvania State University and a bachelor's degree in computer engineering from the University of Dayton.

Who We Are

Aiton Goldman is a Software Developer in IBM Watson focused on AI enhanced information retrieval solutions for the Watson Discovery Service. He has extensive experience with quality assurance and DevOps, developing extensive automated testing frameworks. Before his current role he worked at Honeywell, the Durand Lab at Carnegie Mellon University, and Transarc (developers of the Andrew Filesystem).,Aiton has a B.S in Information Decision Systems from Carnegie Mellon University, with a minor in Drama Production.

Outline

IBM Watson / © 2018 IBM Corporation

Outline

- **Introduction**
- **Background**
- **Problem Definition**
- **Proposed Approach**
- **Examples**
  - Information Retrieval Architecture
  - Machine Learning Train Architecture
- **Discussion**
- **Conclusion**

© 2018 IBM Corporation

Introduction

IBM Watson / © 2018 IBM Corporation

Opening Question

IBM Watson / © 2018 IBM Corporation

Opening Question

Should I design my new product
using a microservice architecture?

Opening Question

Should I design my new product
using a microservice architecture?

Depends who you ask!

© 2018 IBM Corporation

Rephrasing the Question

IBM Watson / © 2018 IBM Corporation

Rephrasing the Question

Under what circumstances do the benefits of starting with microservices significantly outweigh the risks?

A Starting Suggestion

Under what circumstances do the benefits of starting with microservices significantly outweigh the risks?

One scenario may be when development is happening concurrently with product design and/or with shifting external dependencies

© 2018 IBM Corporation

Background

IBM Watson / © 2018 IBM Corporation

Background

Let's start by reviewing the general arguments in either direction, starting with "Microservices first!"

IBM Watson / © 2018 IBM Corporation

e.g. https://martinfowler.com/articles/dont-start-monolith.html

Microservices First!

IBM Watson / © 2018 IBM Corporation

e.g. https://martinfowler.com/articles/dont-start-monolith.html

Microservices First!

Fast delivery of independent parts

IBM Watson / © 2018 IBM Corporation

e.g. https://martinfowler.com/articles/dont-start-monolith.html

Microservices First!

Fast delivery of independent parts

Easier to set boundaries during initial component creation
to prevent casual code coupling

e.g. https://martinfowler.com/articles/dont-start-monolith.html

Microservices First!

Fast delivery of independent parts

Easier to set boundaries during initial component creation
to prevent casual code coupling

It would be harder to later break up tightly coupled code
after starting with a monolith –
and you will probably have tightly coupled code

IBM Watson / © 2018 IBM Corporation

e.g. https://martinfowler.com/articles/dont-start-monolith.html

Monolith First!

e.g. https://martinfowler.com/bliki/MonolithFirst.html

Monolith First!

Higher premium to maintain microservices

IBM Watson / © 2018 IBM Corporation

e.g. https://martinfowler.com/bliki/MonolithFirst.html

Monolith First!

Higher premium to maintain microservices

“You ain’t gonna need it”

IBM Watson / © 2018 IBM Corporation

e.g. https://martinfowler.com/bliki/MonolithFirst.html

Monolith First!

Higher premium to maintain microservices

“You ain’t gonna need it”

High risk of getting bounded contexts wrong

IBM Watson / © 2018 IBM Corporation

e.g. https://martinfowler.com/bliki/MonolithFirst.html

Monolith First!

Higher premium to maintain microservices

“You ain’t gonna need it”

High risk of getting bounded contexts wrong

IBM Watson / © 2018 IBM Corporation

e.g. https://martinfowler.com/bliki/MonolithFirst.html

Shift of Perspective

© 2018 IBM Corporation

Shift of Perspective

If you accept that you'll get the bounded contexts wrong and budget for it, that is no longer a risk - it's just strategic technical debt

Shift of Perspective

If you accept that you'll get the bounded contexts wrong and budget for it, that is no longer a risk - it's just strategic technical debt

The question then becomes how to best intentionally take on that tech debt, and how to pay it off optimally

Problem Definition

Problem Definition

If we accept getting bounded contexts wrong as strategic technical debt:

Problem Definition

If we accept getting bounded contexts wrong as strategic technical debt:

How can we take advantage of microservice architecture to expedite initial release as much as possible,

IBM Watson / © 2018 IBM Corporation

Problem Definition

If we accept getting bounded contexts wrong as strategic technical debt:

How can we take advantage of microservice architecture to expedite initial release as much as possible,

while also minimizing the long-term cost of strategic technical debt incurred along the way?

Proposed Approach

Our Proposal

- When there is high uncertainty in product design or future requirements but development needs to start, it can be an effective strategy to start with a microservice architecture

Our Proposal

- When there is high uncertainty in product design or future requirements but development needs to start, it can be an effective strategy to start with a microservice architecture
- View incorrect bounded contexts as technical debt and not as risk
  - Do make best effort during design phase

Our Proposal

- When there is high uncertainty in product design or future requirements but development needs to start, it can be an effective strategy to start with a microservice architecture
- View incorrect bounded contexts as technical debt and not as risk
  - Do make best effort during design phase
- Address new requirements and use cases with new components (but judiciously)

Our Proposal

- When there is high uncertainty in product design or future requirements but development needs to start, it can be an effective strategy to start with a microservice architecture
- View incorrect bounded contexts as technical debt and not as risk
  - Do make best effort during design phase
- Address new requirements and use cases with new components (but judiciously)
- Ship fast while tracking strategic technical debt

Our Proposal

- When there is high uncertainty in product design or future requirements but development needs to start, it can be an effective strategy to start with a microservice architecture
- View incorrect bounded contexts as technical debt and not as risk
  - Do make best effort during design phase
- Address new requirements and use cases with new components (but judiciously)
- Ship fast while tracking strategic technical debt
- After release, address only the technical debt in mission-critical microservices
  - Focus on increasing maintainability and enabling development agility
  - Ignore technical debt in isolated, non-critical components until forced to

IBM Watson / © 2018 IBM Corporation

Risk Minimization

© 2018 IBM Corporation

Risk Minimization

- Avoid investing in features that see no usage but that could increase overhead

Risk Minimization

- Avoid investing in features that see no usage but that could increase overhead
- Decrease impact of unforeseen legal or platform requirements

Risk Minimization

- Avoid investing in features that see no usage but that could increase overhead
- Decrease impact of unforeseen legal or platform requirements
- Allow late phase addition of new functional requirements (though with higher debt)

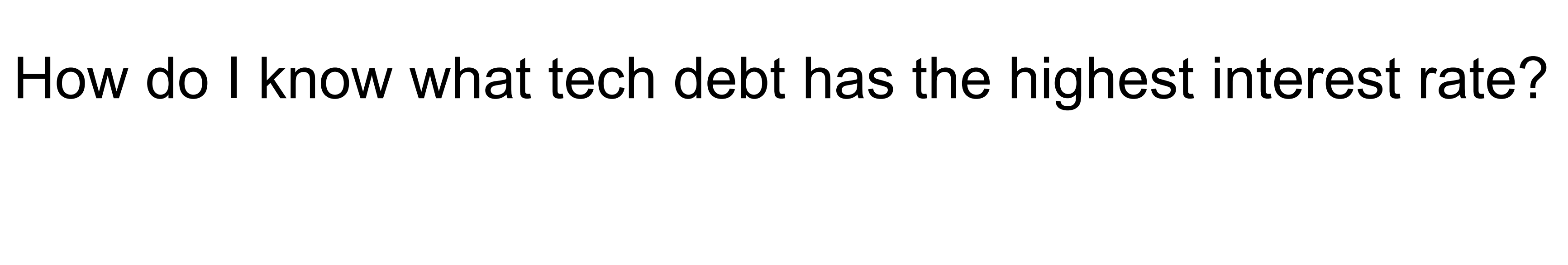
© 2018 IBM Corporation

Risk Minimization

- Avoid investing in features that see no usage but that could increase overhead
- Decrease impact of unforeseen legal or platform requirements
- Allow late phase addition of new functional requirements (though with higher debt)
- Avoid unintentional code coupling that can harm team velocity

© 2018 IBM Corporation

How do I know what tech debt has the highest interest rate?



IBM Watson / © 2018 IBM Corporation

How do I know what tech debt has the highest interest rate?

- We unintentionally carried out our proposed formal strategy

© 2018 IBM Corporation

How do I know what tech debt has the highest interest rate?

- We unintentionally carried out our proposed formal strategy
- Looking back empirically, we identify useful metrics for driving decisions on what debt to pay off

IBM Watson / © 2018 IBM Corporation

How do I know what tech debt has the highest interest rate?

- We unintentionally carried out our proposed formal strategy
- Looking back empirically, we identify useful metrics for driving decisions on what debt to pay off
  - Tech Debt
    - How much code needs to be refactored?
    - How hard is it to modify the codebase?

How do I know what tech debt has the highest interest rate?

- We unintentionally carried out our proposed formal strategy
- Looking back empirically, we identify useful metrics for driving decisions on what debt to pay off
  - Tech Debt
    - How much code needs to be refactored?
    - How hard is it to modify the codebase?
  - High versus Low Touch
    - How often does a component get modified?

Metrics

Metrics

Tech Debt Metrics

- Repository size
- Average number of lines modified per PR
- Percentage of production log output
- Test coverage

© 2018 IBM Corporation

Metrics

Tech Debt Metrics

- Repository size
- Average number of lines modified per PR
- Percentage of production log output
- Test coverage

High / Low Touch Metrics

- Commits per PR
- Comments per PR
- Mean, standard deviation of time between commits

Metrics

Tech Debt Metrics

- Repository size
- Average number of lines modified per PR
- Percentage of production log output
- **Test coverage**

High / Low Touch Metrics

- Commits per PR
- Comments per PR
- **Mean, standard deviation of time between commits**

IBM Watson / © 2018 IBM Corporation

Case Study

- We will use ourselves as a case study

IBM Watson / © 2018 IBM Corporation

Case Study

- We will use ourselves as a case study
- We informally adopted the approach we suggest here, over several years

Case Study

- We will use ourselves as a case study
- We informally adopted the approach we suggest here, over several years
- We give two examples of how this informal approach manifested

Case Study

- We will use ourselves as a case study
- We informally adopted the approach we suggest here, over several years
- We give two examples of how this informal approach manifested
- We will identify commonalities across those examples

Case Study

- We will use ourselves as a case study
- We informally adopted the approach we suggest here, over several years
- We give two examples of how this informal approach manifested
- We will identify commonalities across those examples
- After each example, using benefit of hindsight, we identify basic metrics to help make this approach both formal and intentional

Example: Information Retrieval Architecture

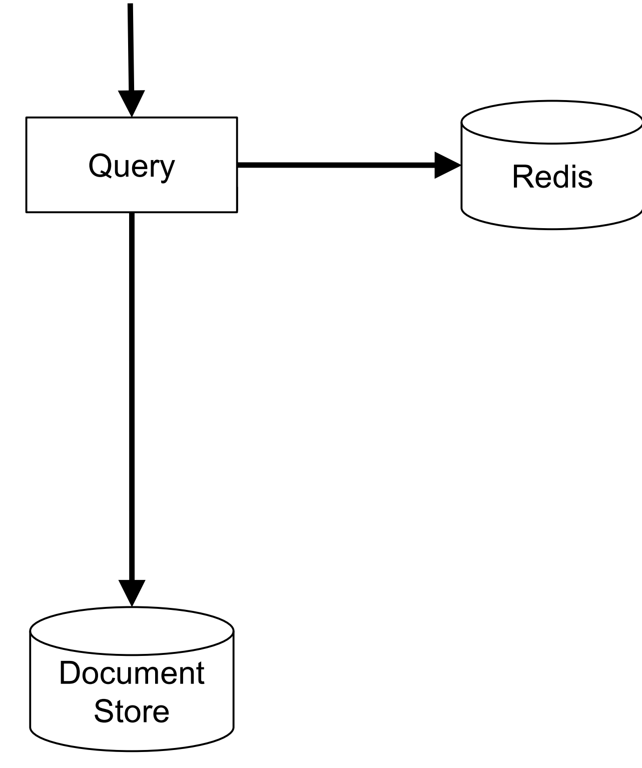
Background

- Our team's MVP was to improve an existing cloud search offering MVP using both machine learning and enhanced information retrieval methods
- Our cloud organization was developing as we were implementing
- New features and use cases were identified, often well after development started

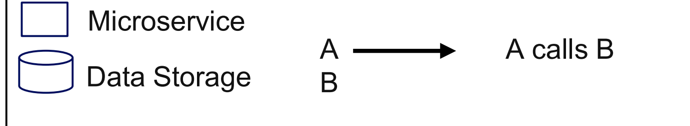
Shipping Quickly
(or: Accumulating Technical Debt)

© 2018 IBM Corporation

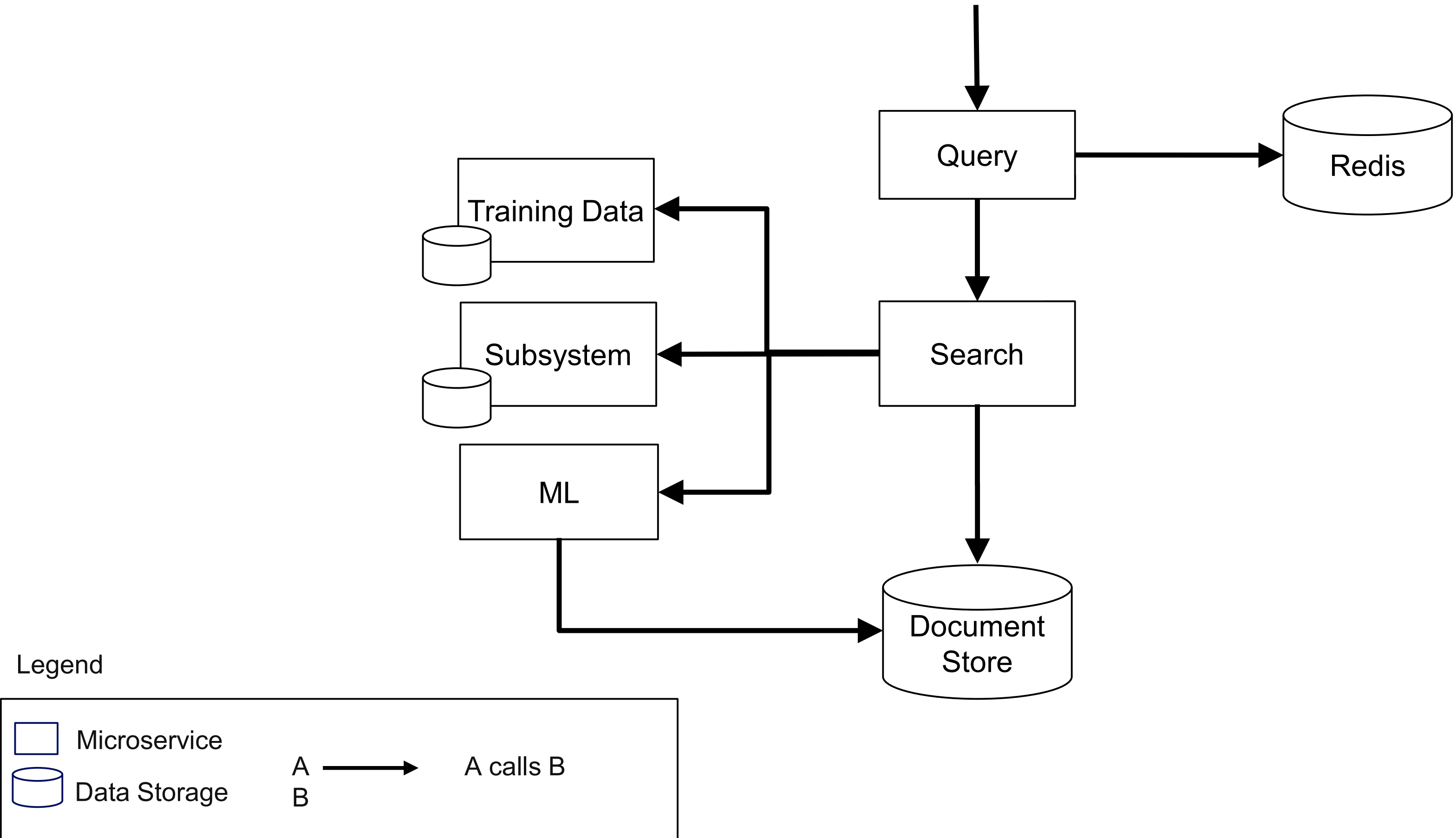
Product MVP



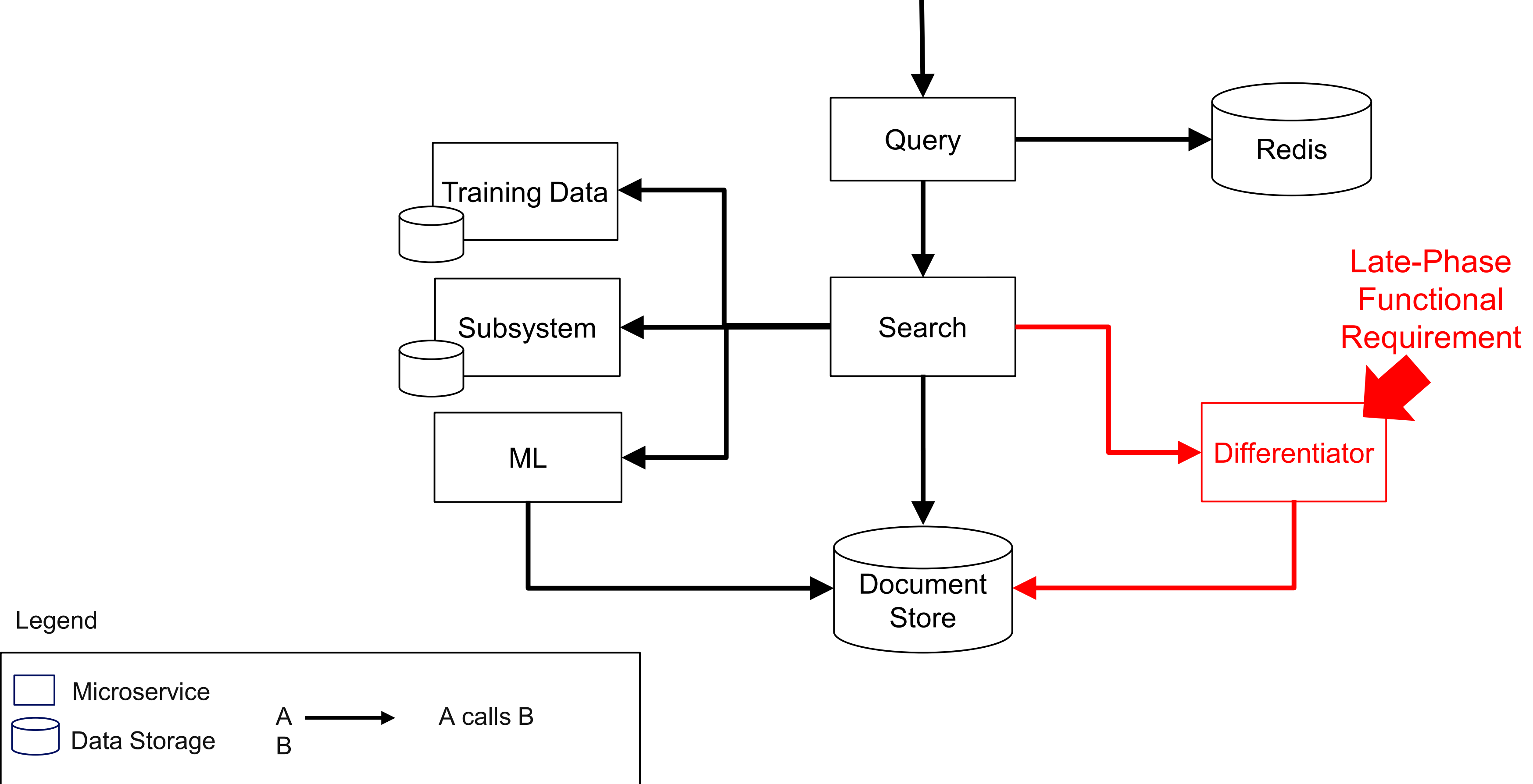
Legend



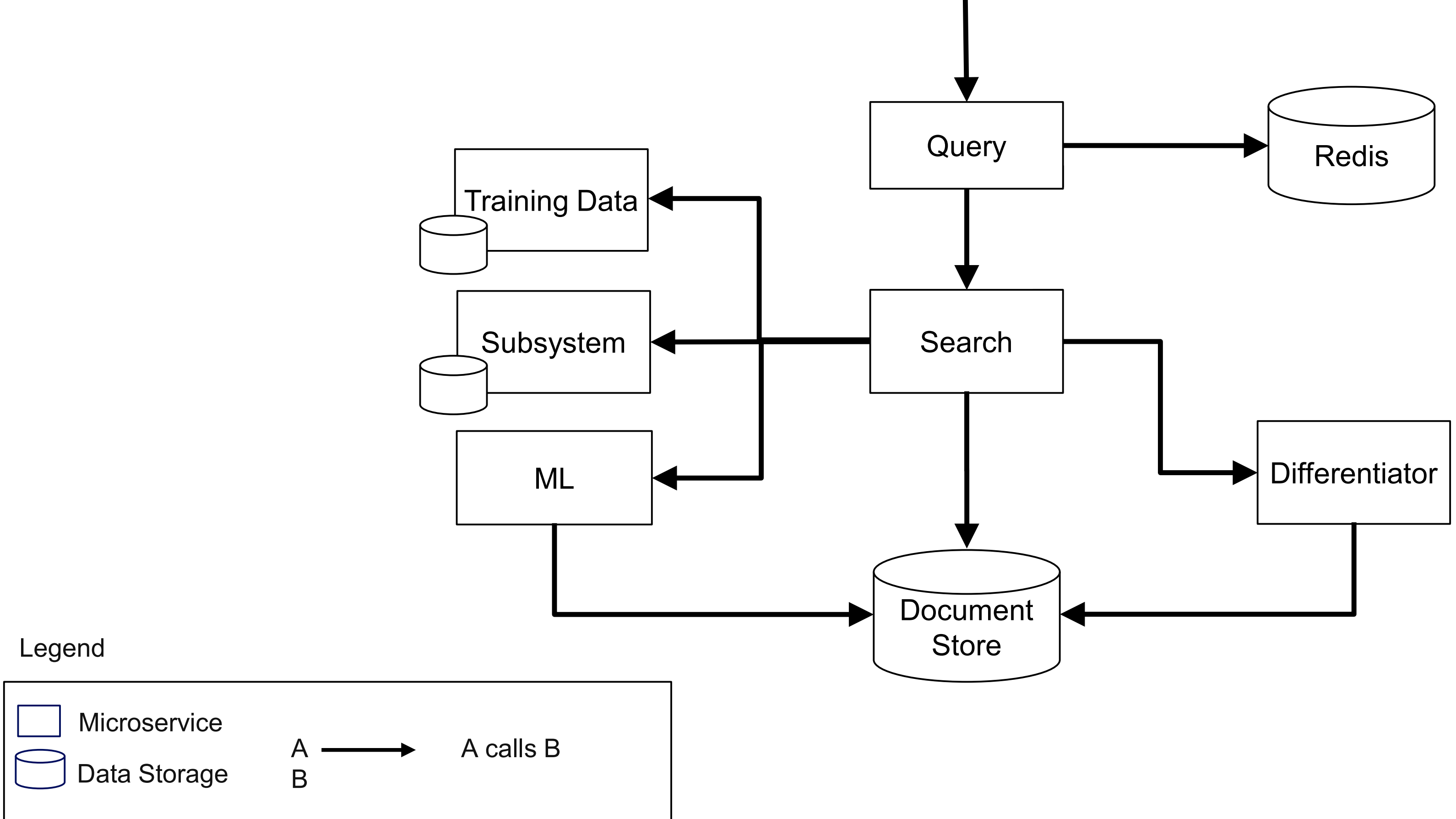
Our Team's Intended MVP



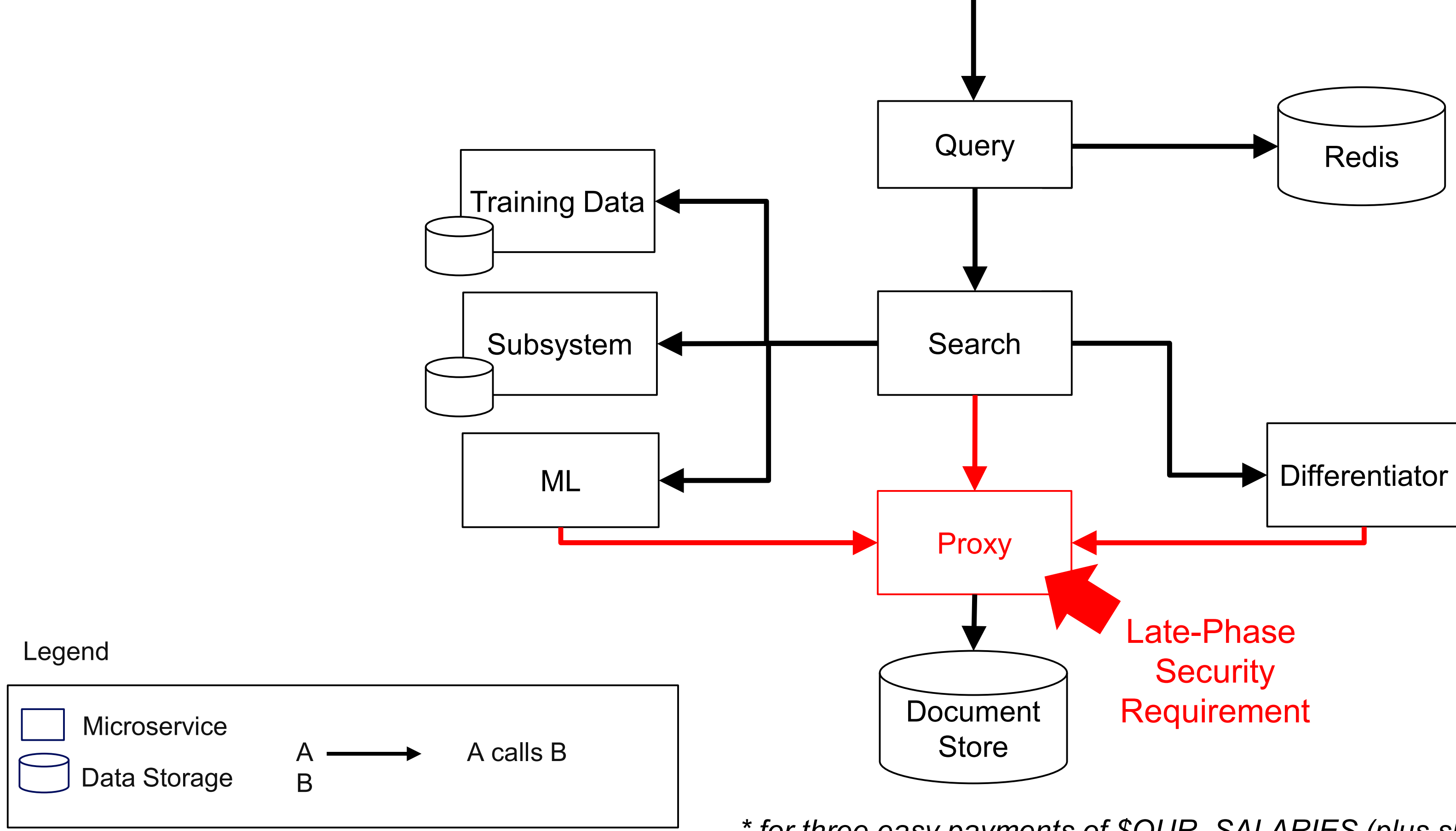
Our Team's Actual MVP



Our Team's Actual MVP

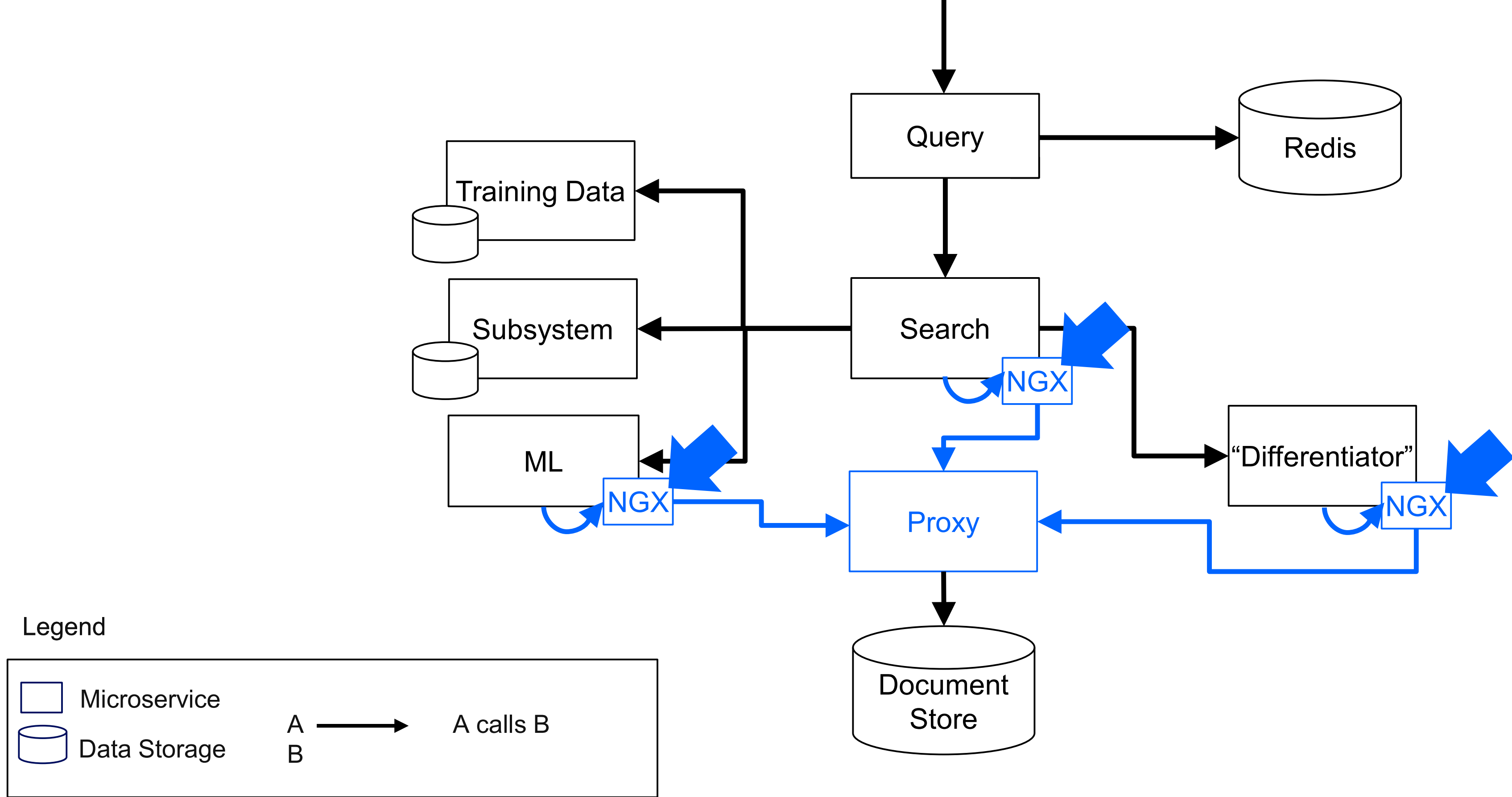


But wait! There's more!*

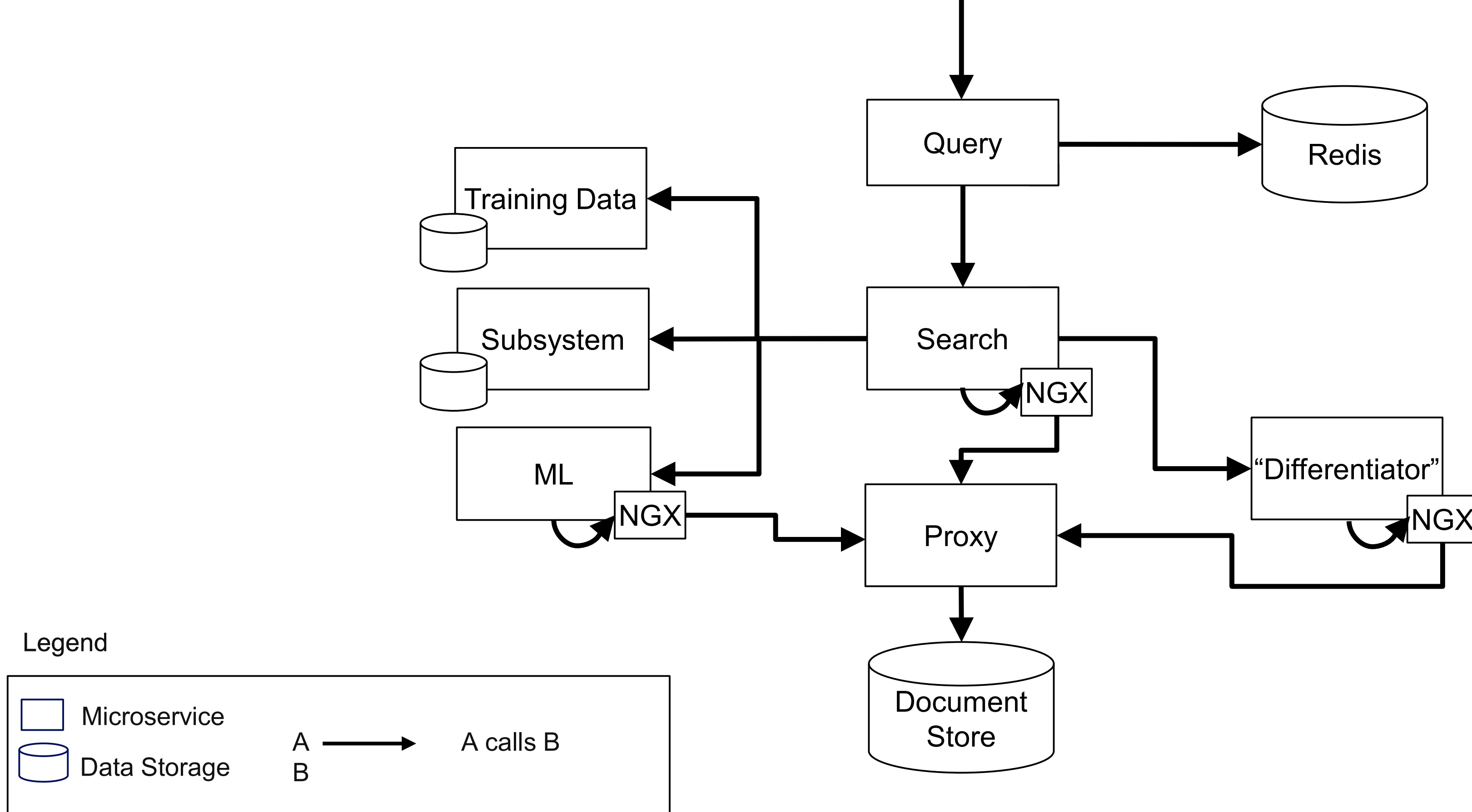


* *for three easy payments of $OUR_SALARIES (plus shipping and handling)*

Solve it with architecture!



Solve it with architecture!



Summary

- Differentiating use case was added late and was implemented with a new component
  - Isolated significant technical debt and shipped
- New platform requirements were met with a new deployable component
  - Solution was contained in one sidecar deployed in front of multiple components subject to the requirements
- Shared concerns were identified between two components, *Search* and *ML*
  - One component was low-touch; we were able to ignore this problem until much later
- This process can be characterized as an **expansion phase**

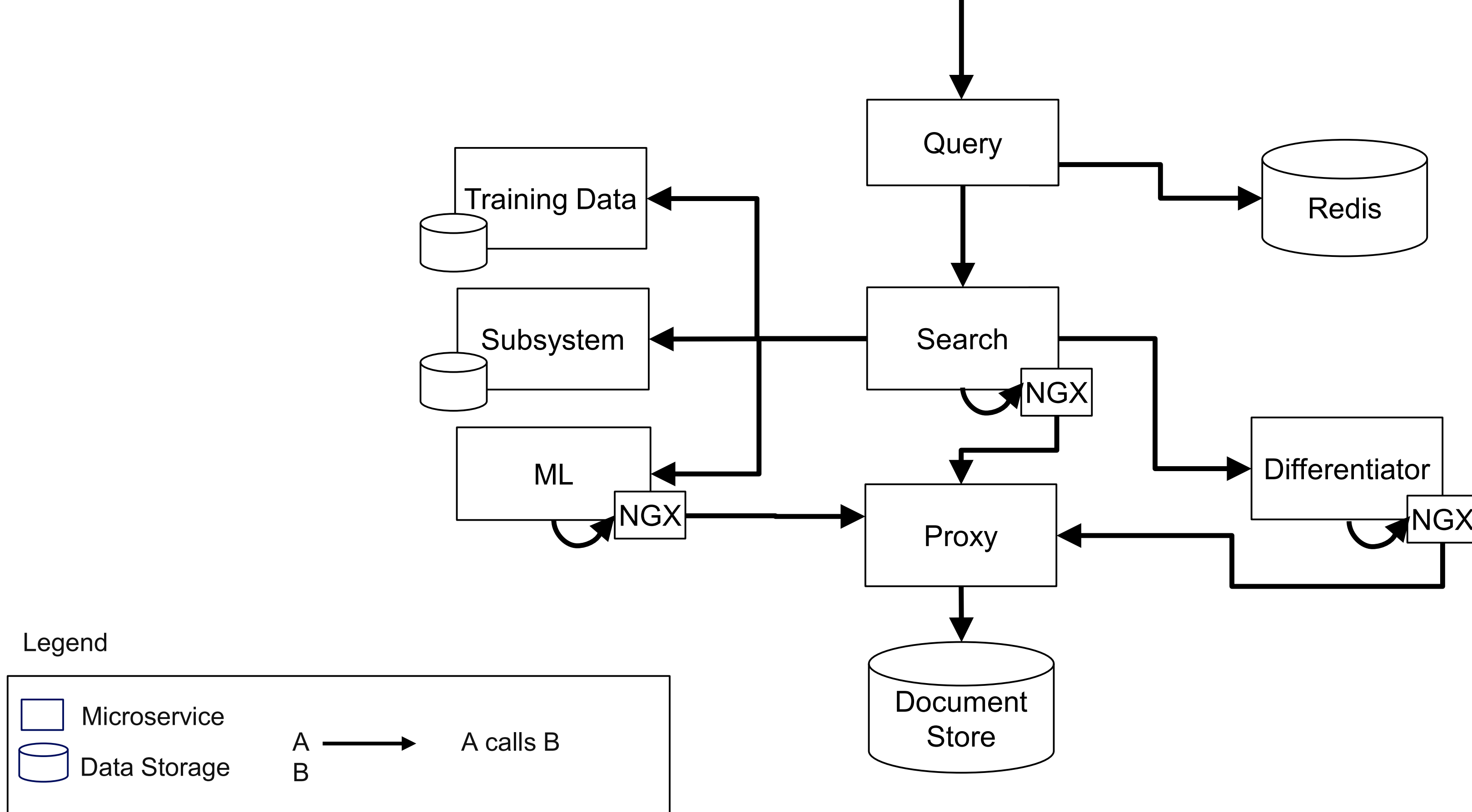
Paying off Tech Debt

© 2018 IBM Corporation

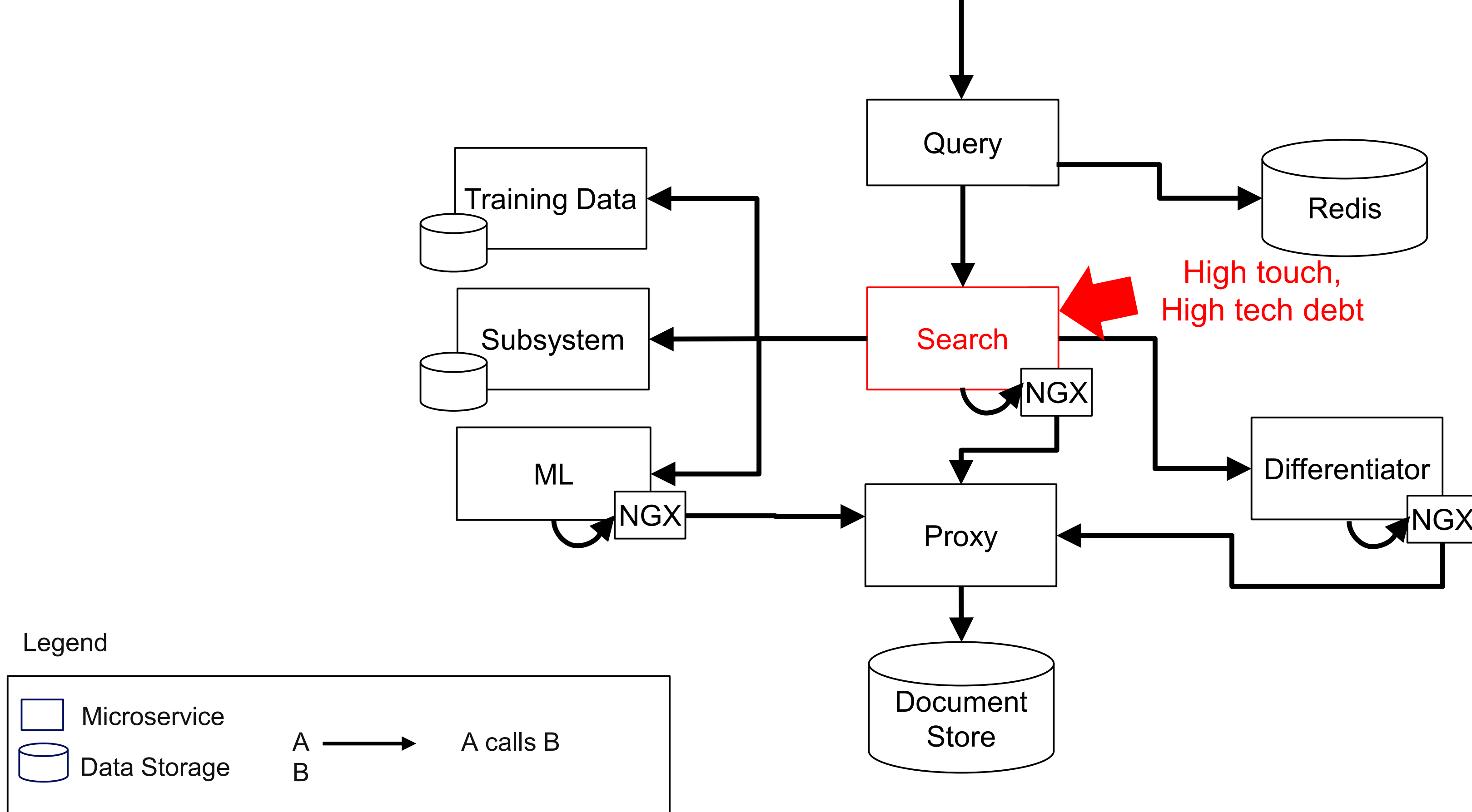
Paying off Tech Debt

- Large amount of tech debt in the mission-critical *Search* component
  - Significantly hampered further feature development
- Large amount of tech debt in the *Differentiator* component, which did not see much usage
  - Did not require changes often
- Incorrect bounded contexts across *Search*, *ML* components was technical debt
  - Ignored until new feature development required similar changes to both components
  - At that time, we merged those services

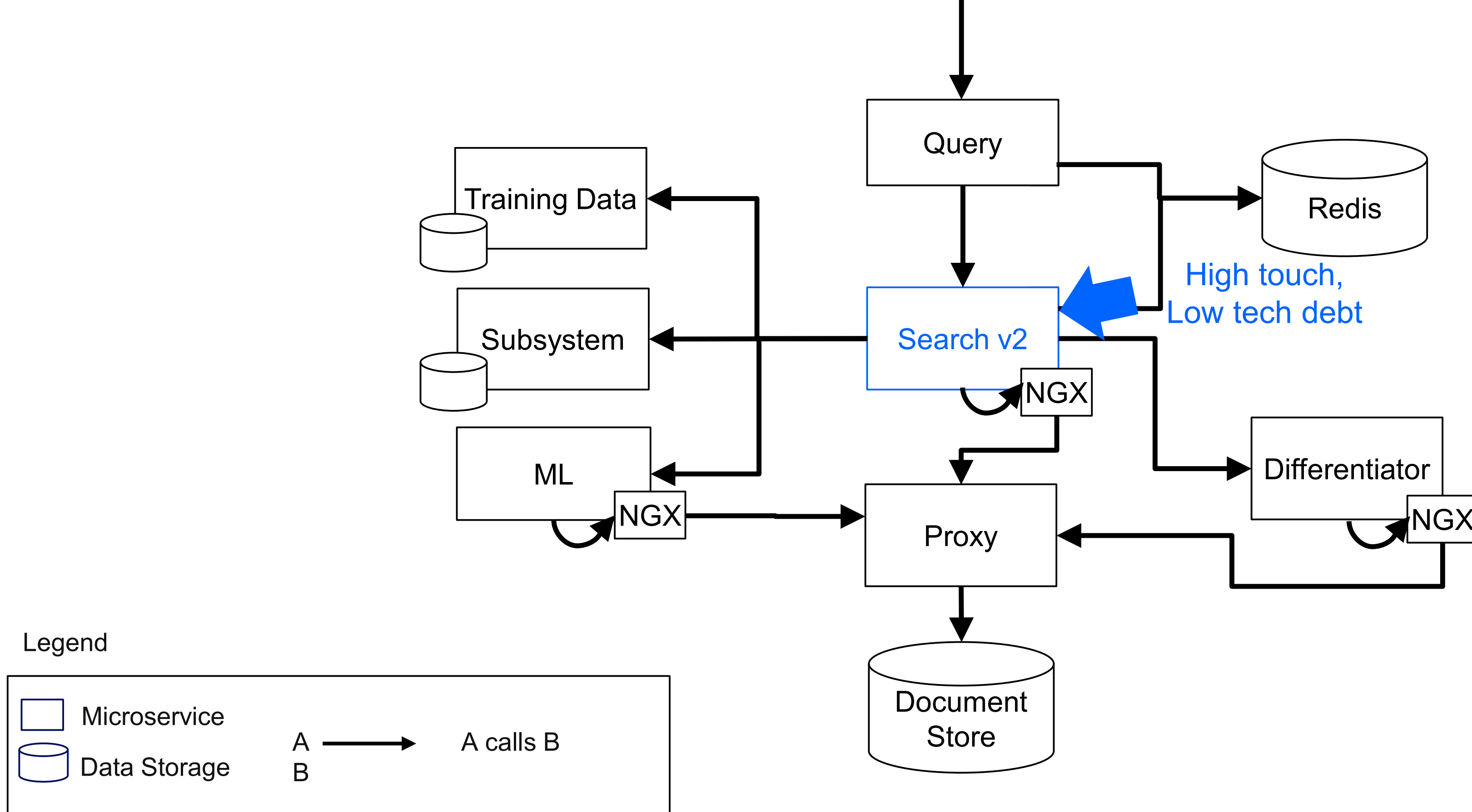
Identifying Tech Debt: Rewriting



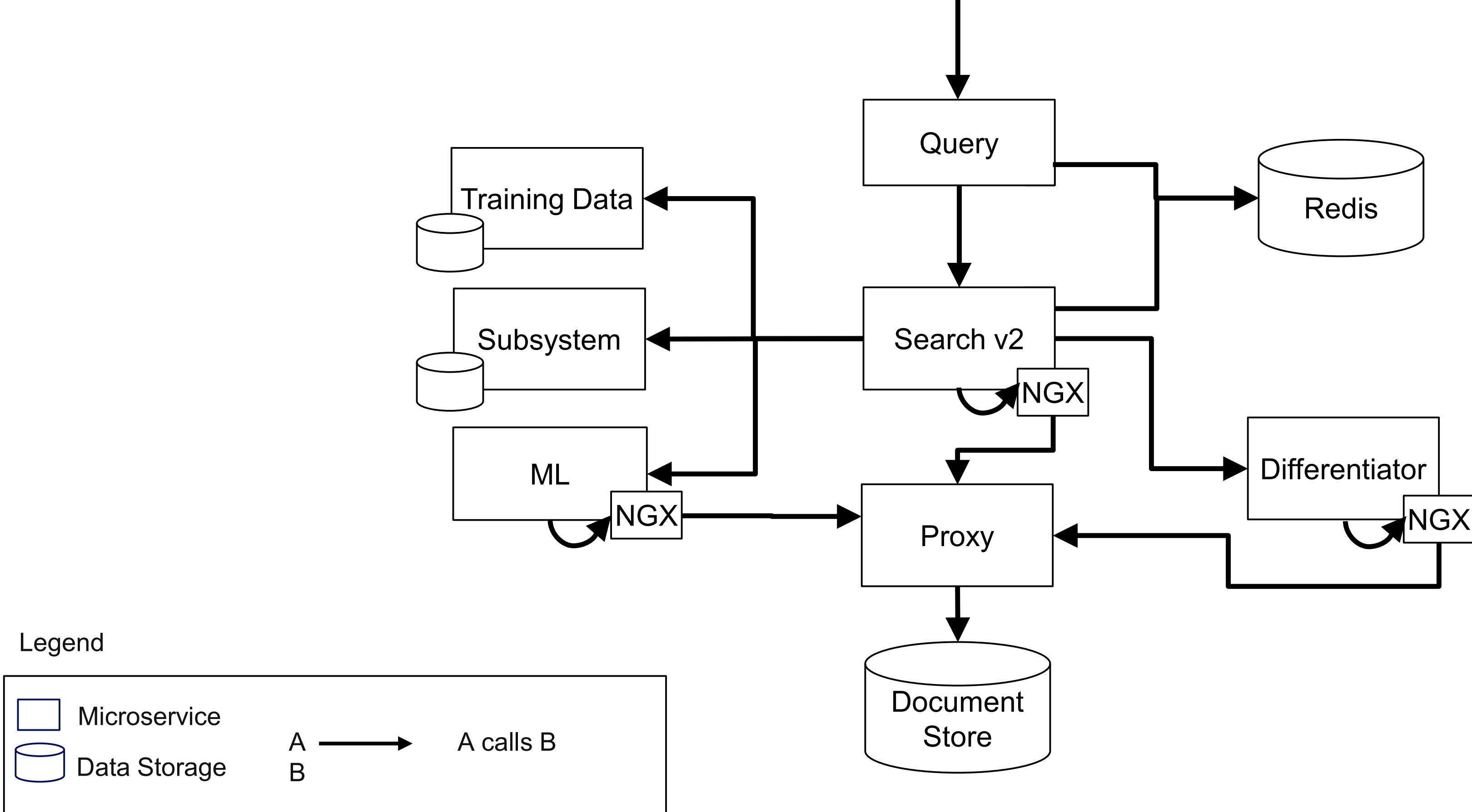
Identifying Tech Debt: Rewriting



New and Improved Search!



New and Improved Search!



"Mistakes" in our bounded contexts

Query

Redis

Training Data

Subsystem

Shared Concerns

Search v2

NGX

ML

NGX

Differentiator

NGX

Proxy

Document Store

Legend

Microservice

Data Storage

A B — A calls B

"Mistakes" in our bounded contexts

Query

Training Data

Redis

Subsystem

Consolidated Concerns

Search + ML

NGX

Differentiator

NGX

Proxy

Document Store

Legend

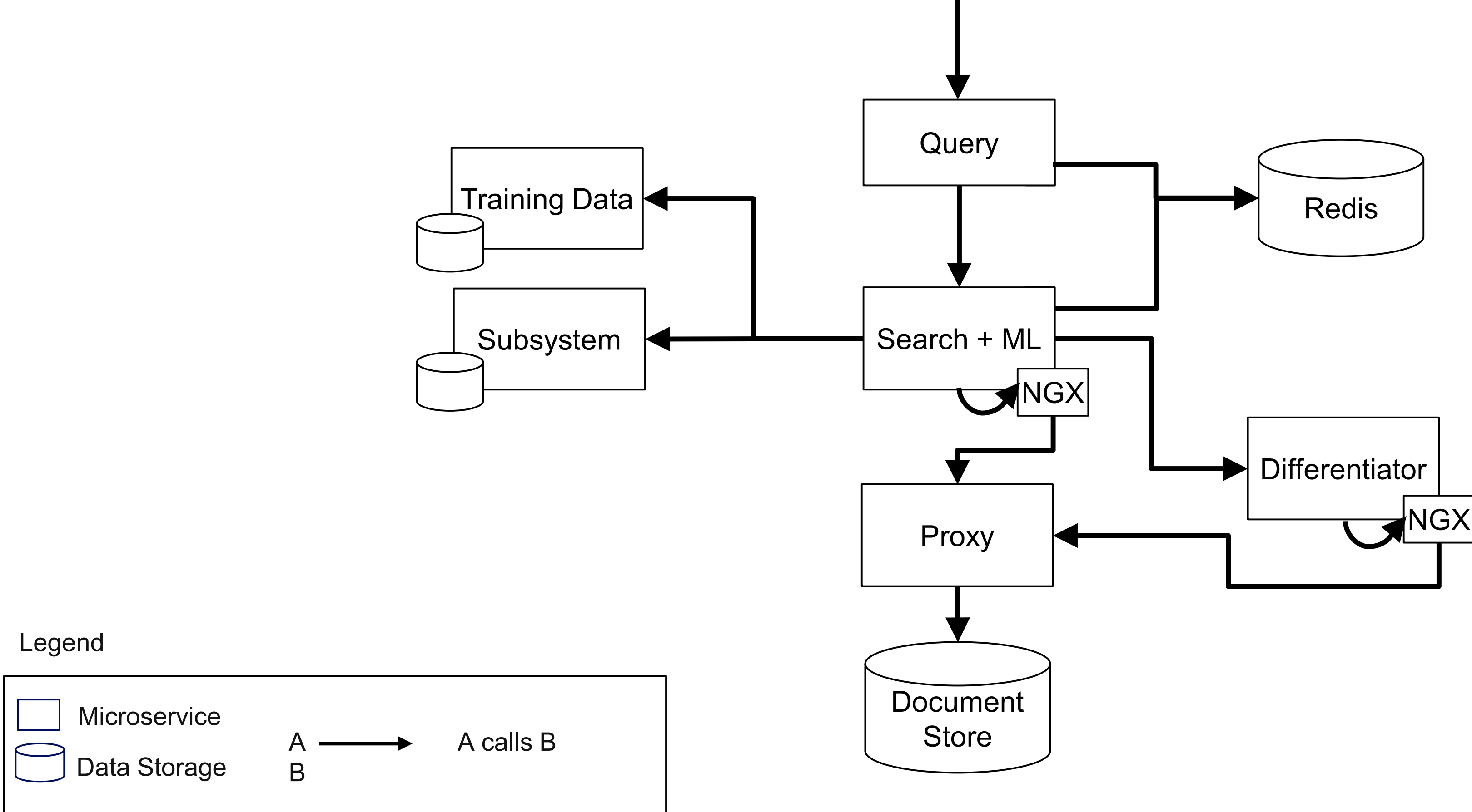
Microservice

Data Storage

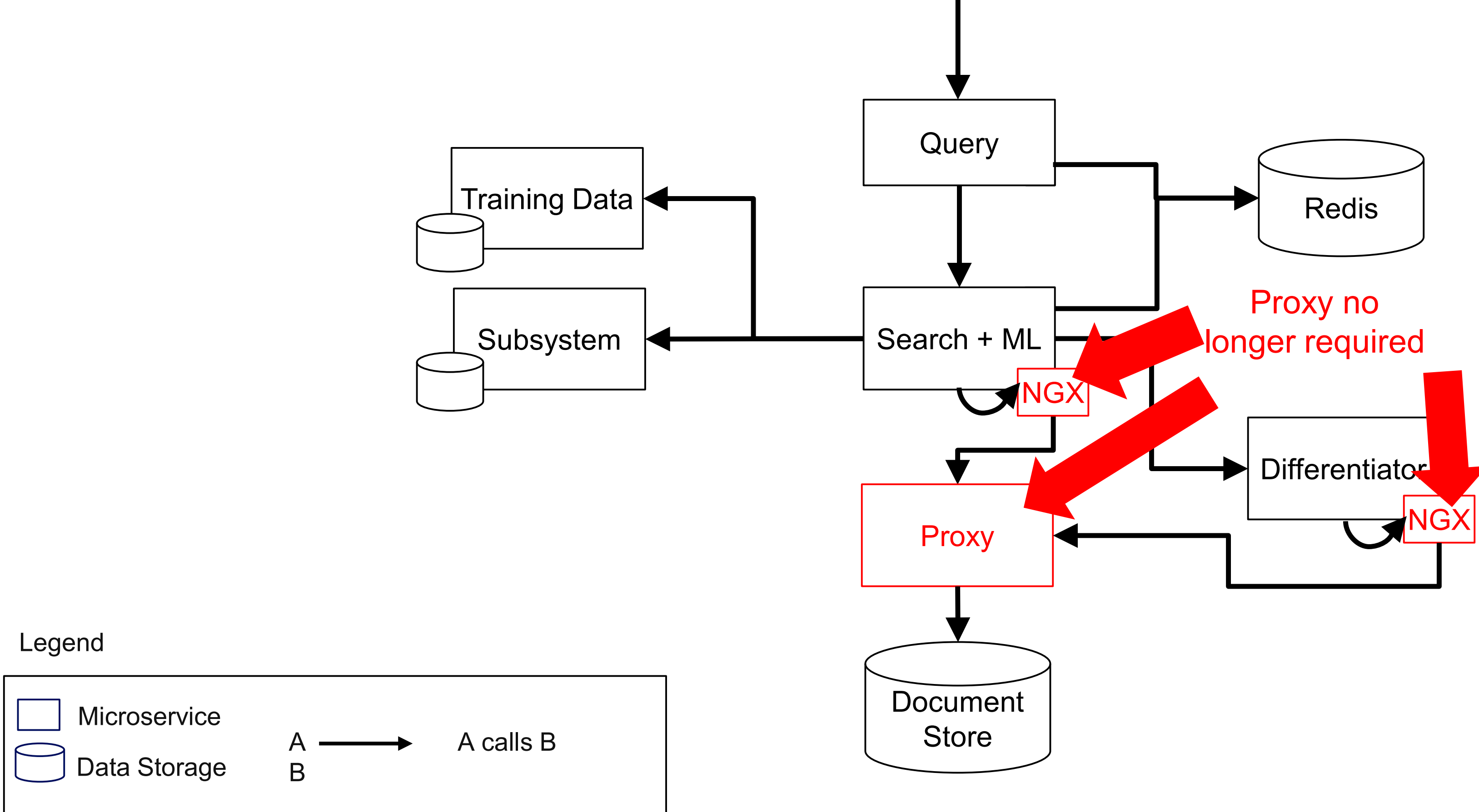
A B

A calls B

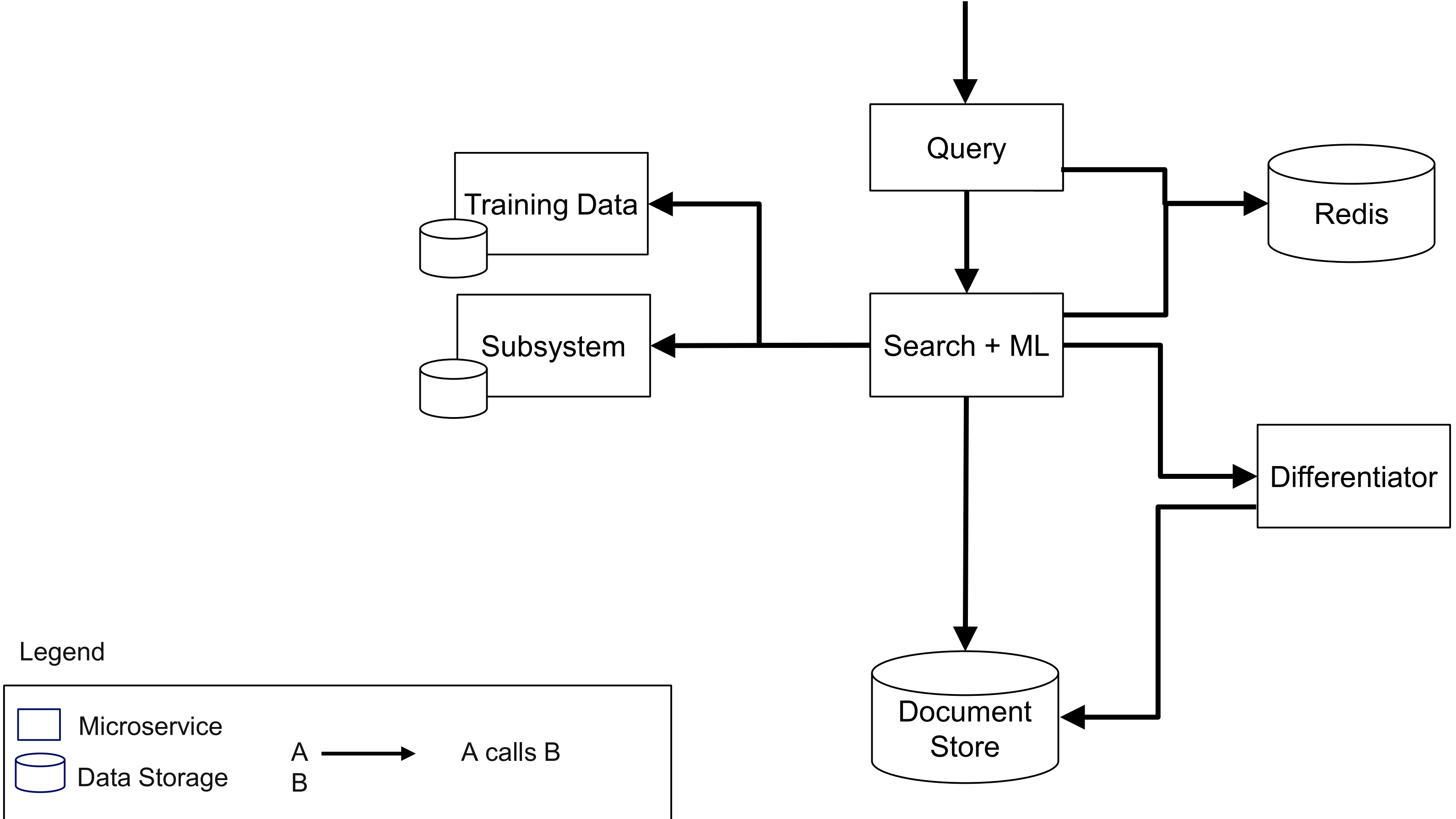
Merge the two together



Paying off Tech Debt: Deleting



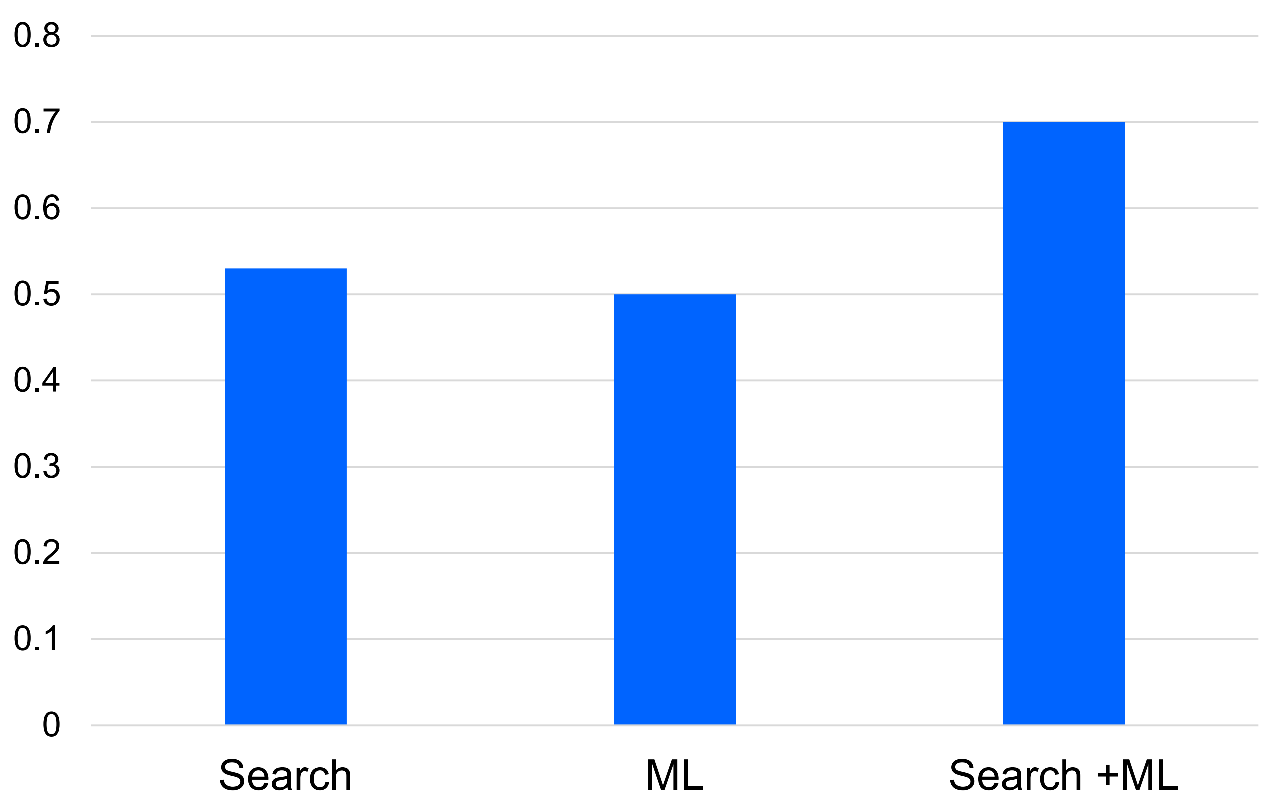
Achieving Relative Stability



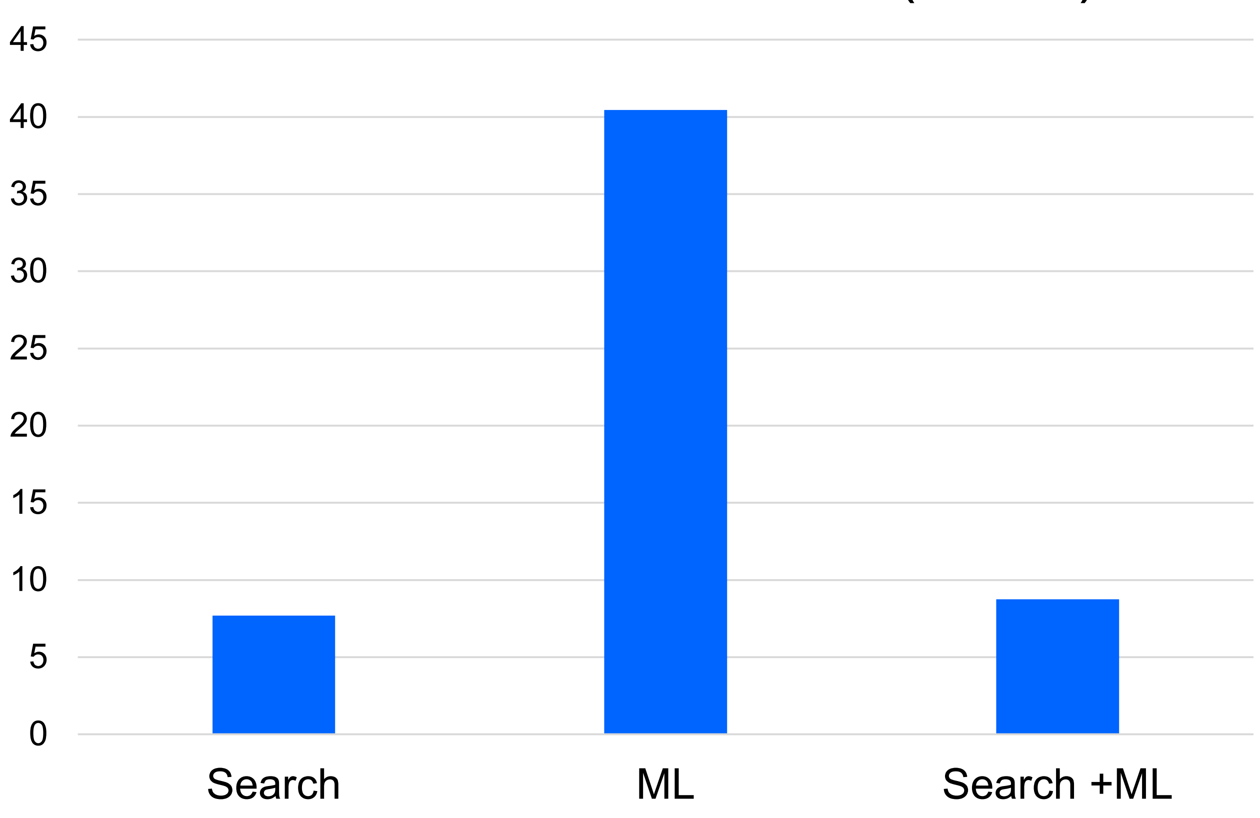
Metrics: High Touch, Paid Off Tech Debt

Component	Number of PRs (z-score)	Unit Test Coverage	Hours between Commits (Mean)	Hours Between Commits (STD)
Search	1.09	0.53	7.70	39.03
ML	**-0.11**	0.50	**40.44**	**317.65**
Search +ML	0.89	**0.70**	8.75	46.14

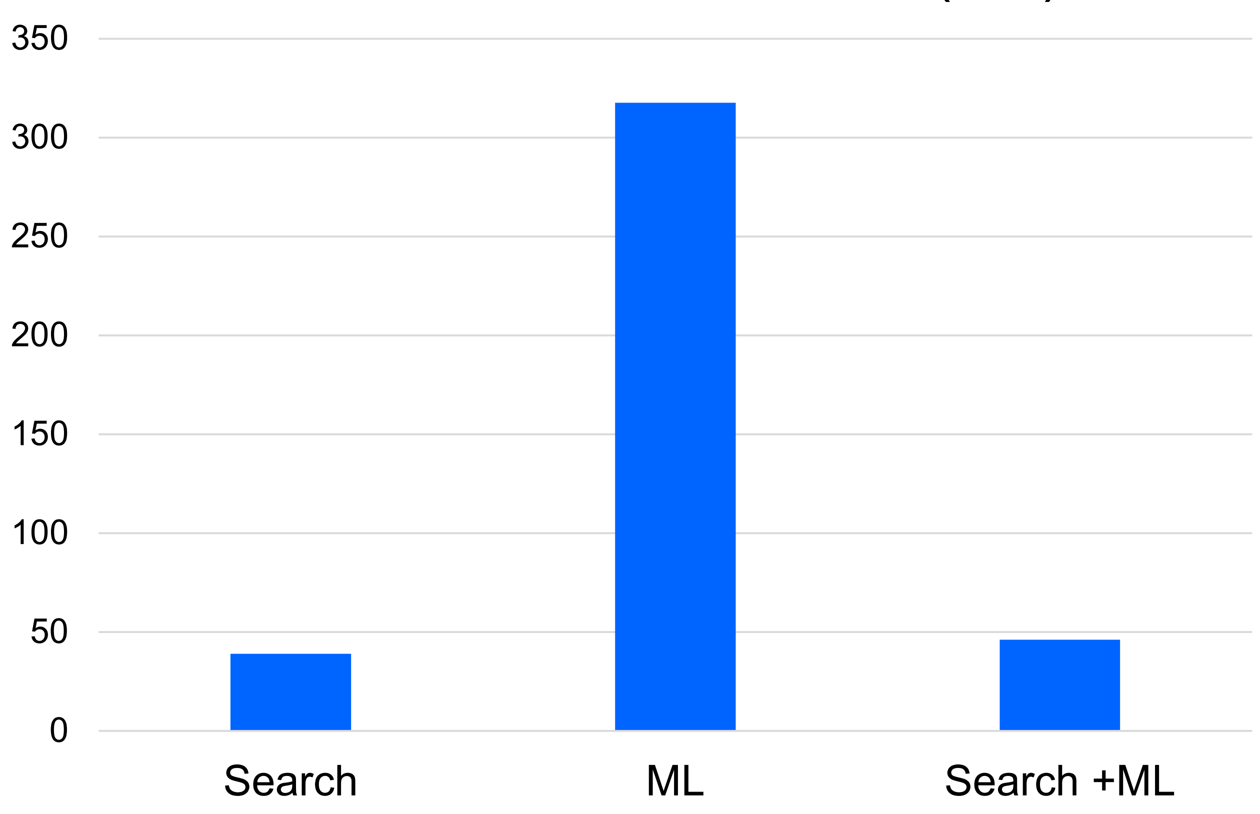
IBM Watson / © 2018 IBM Corporation



IBM Watson / © 2018 IBM Corporation



IBM Watson / © 2018 IBM Corporation



IBM Watson / © 2018 IBM Corporation

Summary

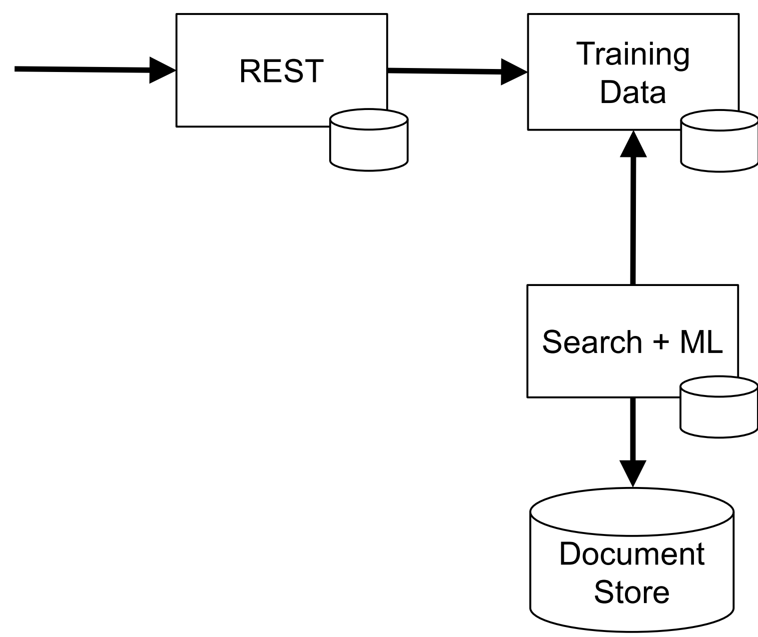
- Isolation of technical debt per component allowed us to focus on technical debt in mission critical components
  - If we had shipped with a monolith, code coupling may have made paying this off much more expensive
- Incorrect bounded contexts were addressed as technical debt when they became a problem
- Removal of platform requirement was addressed by simply removing a deployable component
- This process can be characterized as a **contraction phase**

Example: Machine Learning Training Architecture

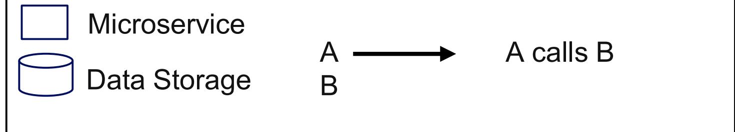
Background

- Recall that part of our team's MVP was to introduce machine learning capabilities
- We inherited a legacy service that was ported from one platform (where it had been ported from another)
  - Very high technical debt and difficult to work in
  - Won't be focusing on this, as we are discussing greenfield development and not legacy refactoring
- One of our requirements was to automate the manual training process for that legacy product

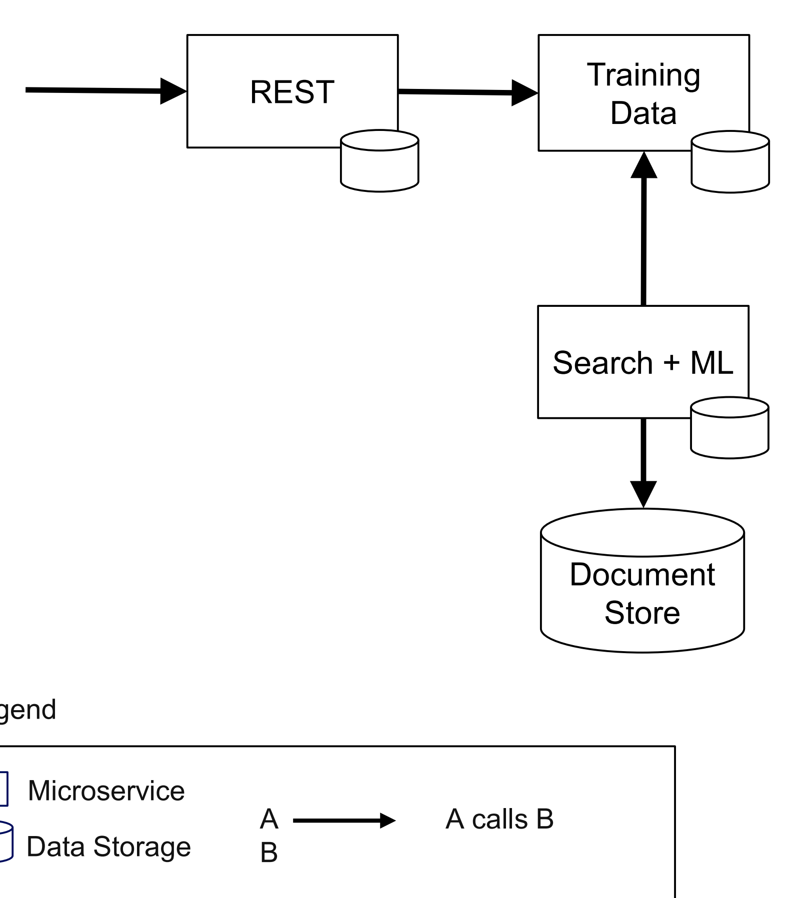
Training View of the Architecture

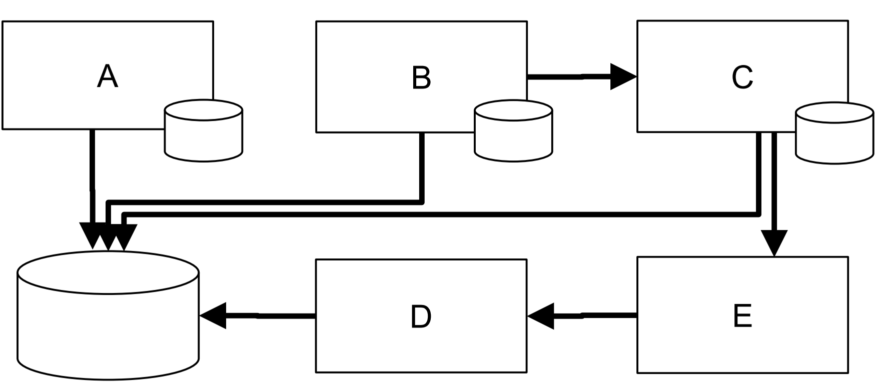


Legend

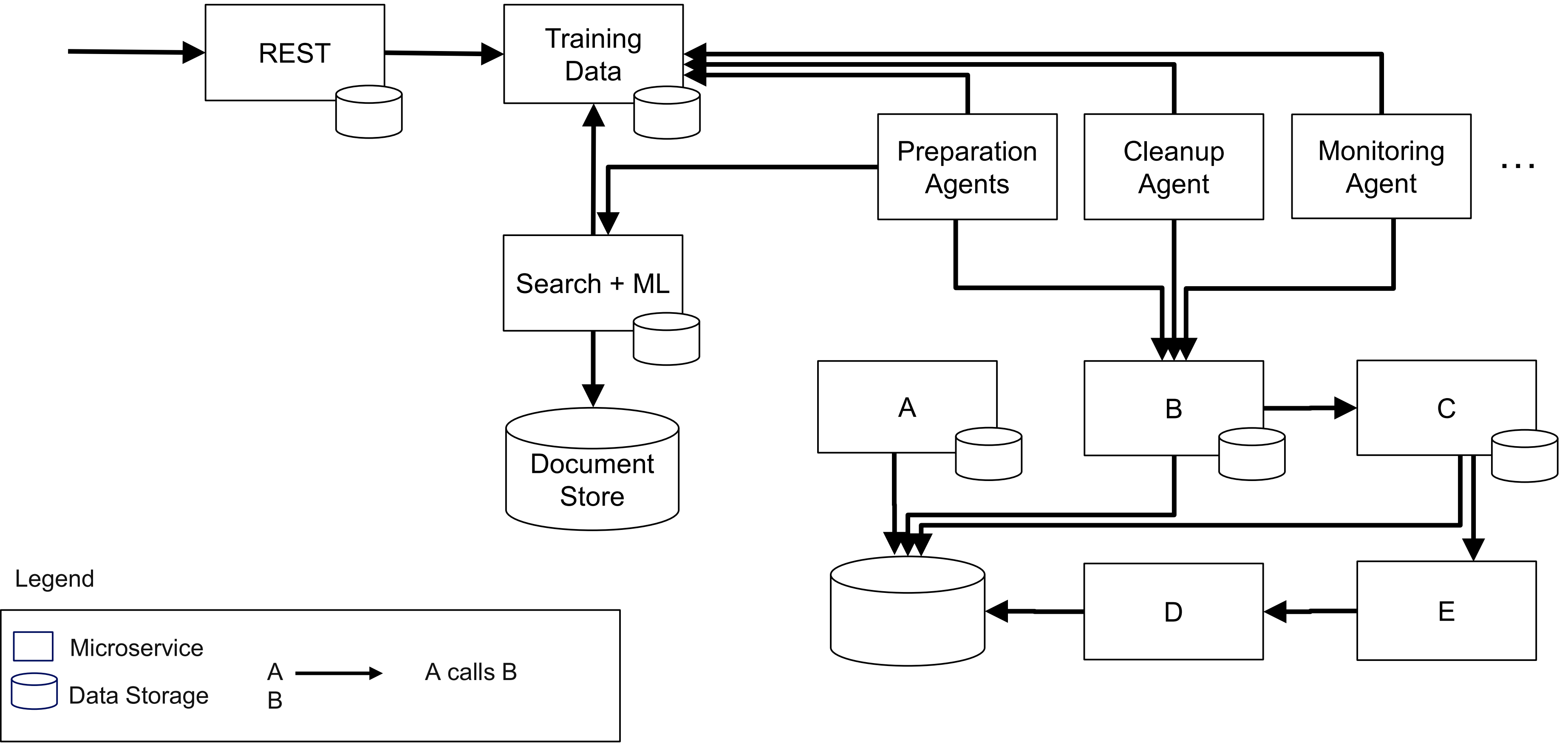


New Legacy System to Manage

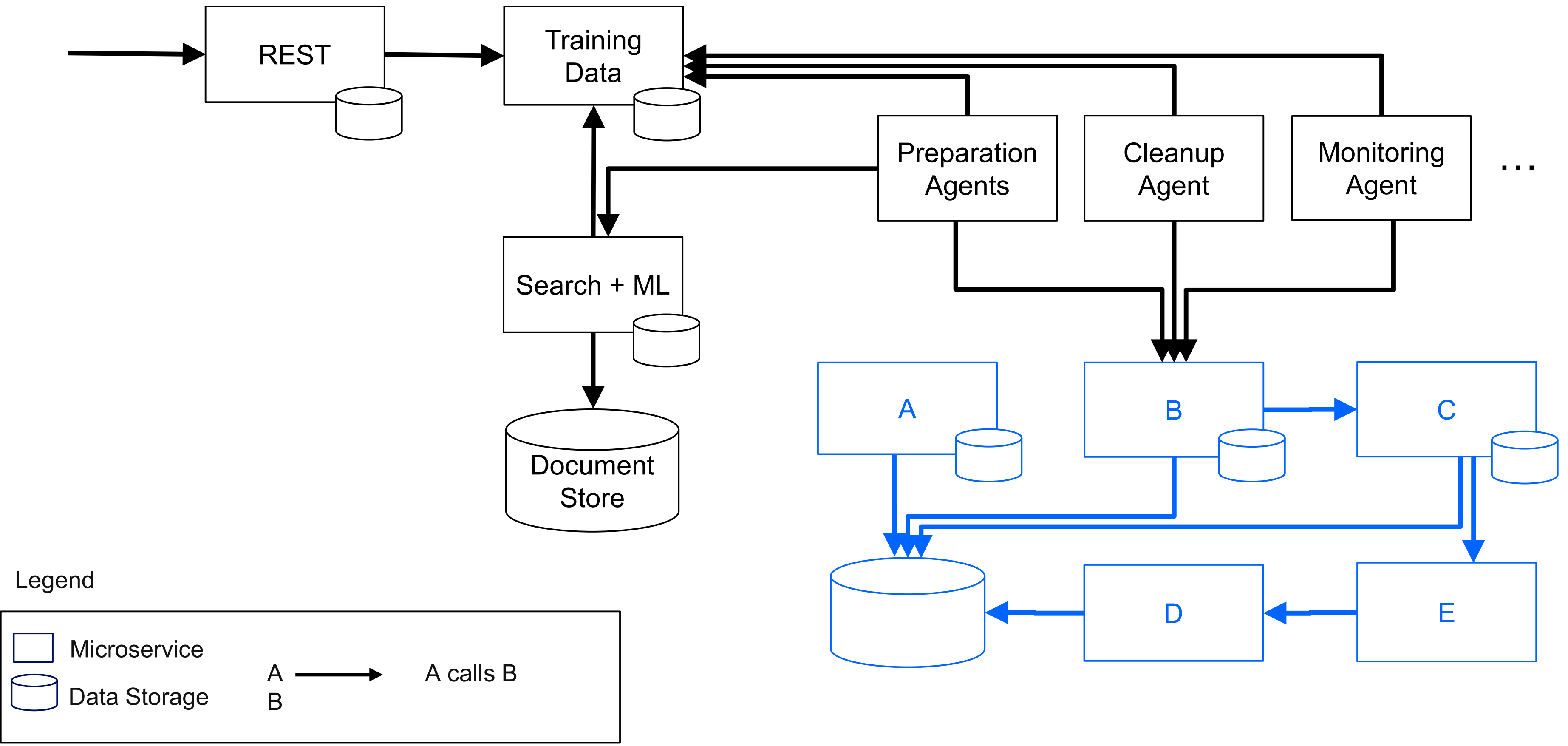




Agents to Process Training Data



Agents to Process Training Data



Agents to Process Training Data

REST

Training Data

Search + ML

Document Store

Preparation Agents

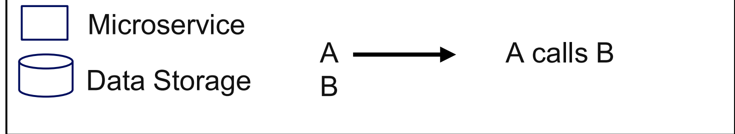
Cleanup Agent

Monitoring Agent

…

Legacy

Legend



Agents to Process Training Data

REST

Training Data

Preparation Agents

Cleanup Agent

Monitoring Agent

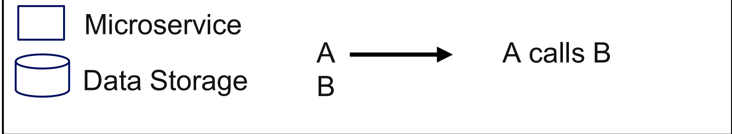
…

Search + ML

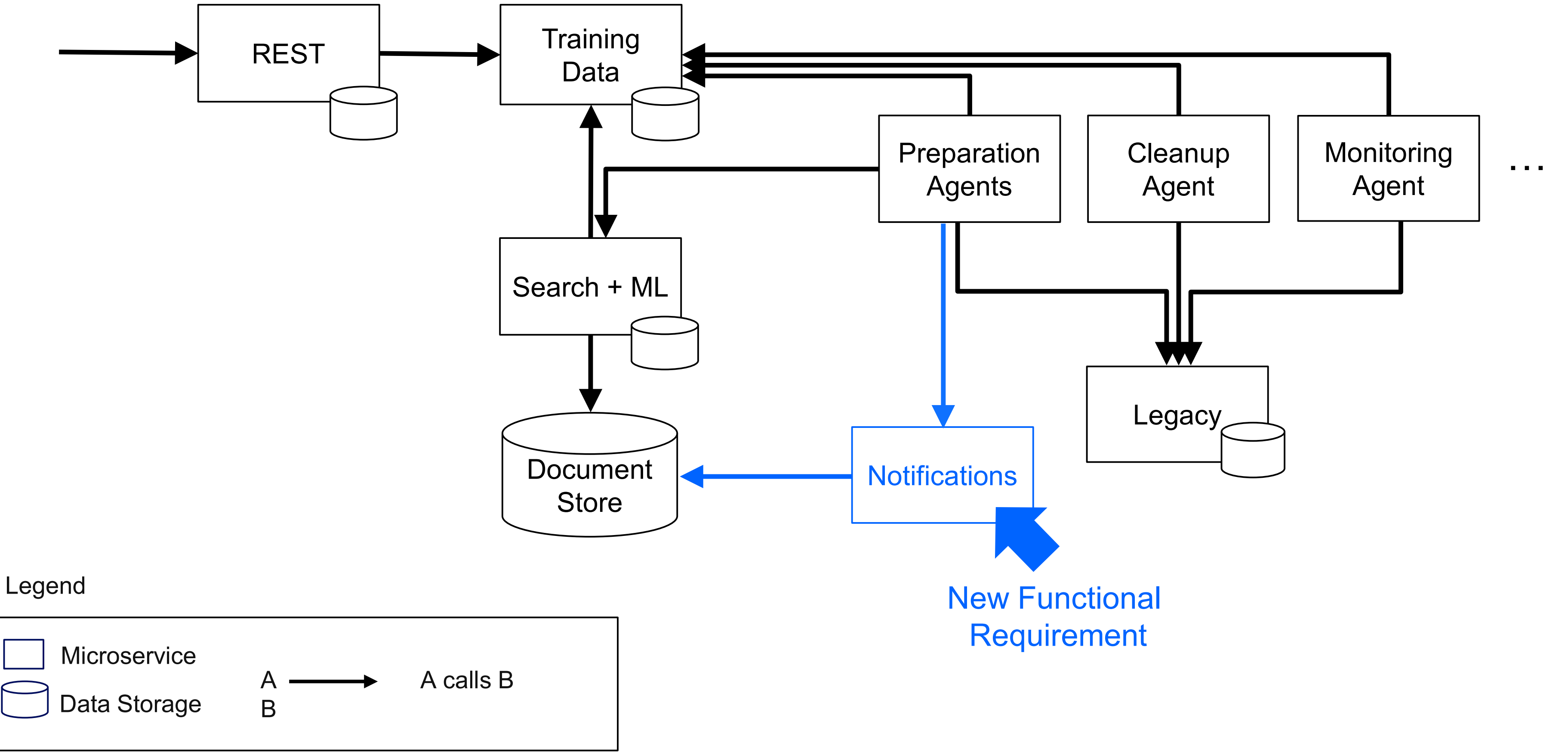
Document Store

Legacy

Legend



New requirement = new service



New requirement = new service

REST

Training Data

Preparation Agents

Cleanup Agent

Monitoring Agent

…

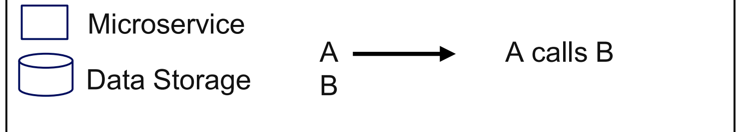
Search + ML

Legacy

Document Store

Notifications

Legend



Interlude

- We introduced several “agents” to automate a previously manual training process
  - We knew that had shared concerns; we shipped fast and labeled that as tech debt
- We again addressed a late-phase requirement with a new component
  - Its tech debt stayed isolated and unaddressed, years later

Opportunities for improvement

REST

Training Data

Preparation Agents

Cleanup Agent

Monitoring Agent

...

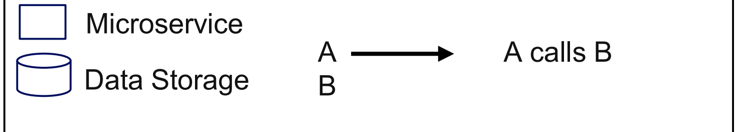
Search + ML

Legacy

Document Store

Notifications

Legend



Opportunities for improvement

Shared code and interactions between agents

REST

Training Data

Search + ML

Document Store

Preparation Agents

Monitoring Agent

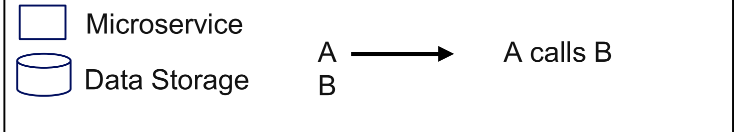
Cleanup Agent

...

Notifications

Legacy

Legend



Opportunities for improvement

Shared code and interactions between agents

REST

Training Data

Training Agents

Monitoring Agent

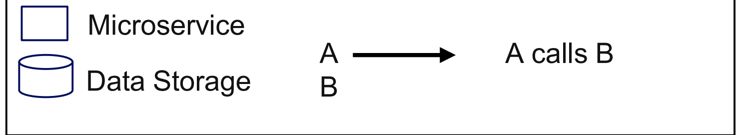
Search + ML

Legacy

Document Store

Notifications

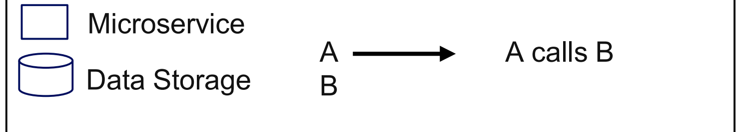
Legend



Opportunities for improvement

REST
Training Data
Training Agents
Monitoring Agent
Search + ML
Legacy
Document Store
Notifications

Legend



Shared codebase for training agents

Isolated,
Low Touch
Tech Debt

REST

Training Data

Training Agents

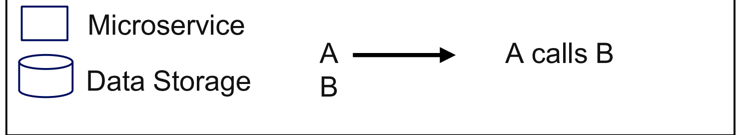
Monitoring Agent

Search + ML

Legacy

Document Store

Notifications



Shared codebase for training agents

REST

Training Data

Training Agents

Monitoring Agent

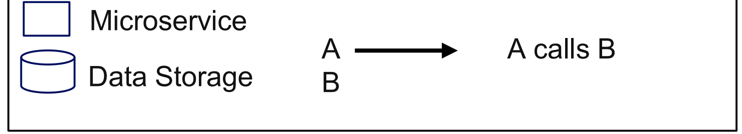
Search + ML

Legacy

Document Store

Notifications

Legend



GDPR

GDPR!

REST

Training Data

Training Agents

Monitoring Agent

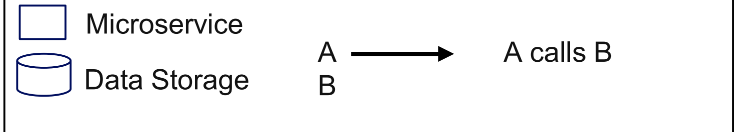
Search + ML

Subsystem

Document Store

Notifications

Legend



GDPR Managed!

REST

Training Data

Training Agents

Monitoring Agent

New Requirement, New Component

GDPR as a Service

Search + ML

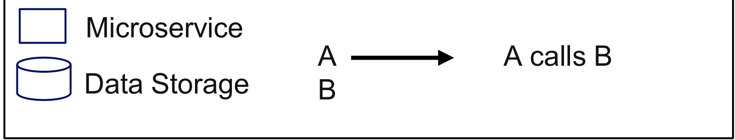
Subsystem

Message Queue

Document Store

Notifications

Legend



GDPR Managed!

REST

Training Data

Training Agents

Monitoring Agent

GDPR as a Service

Search + ML

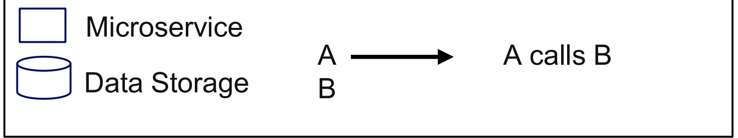
Subsystem

Message Queue

Document Store

Notifications

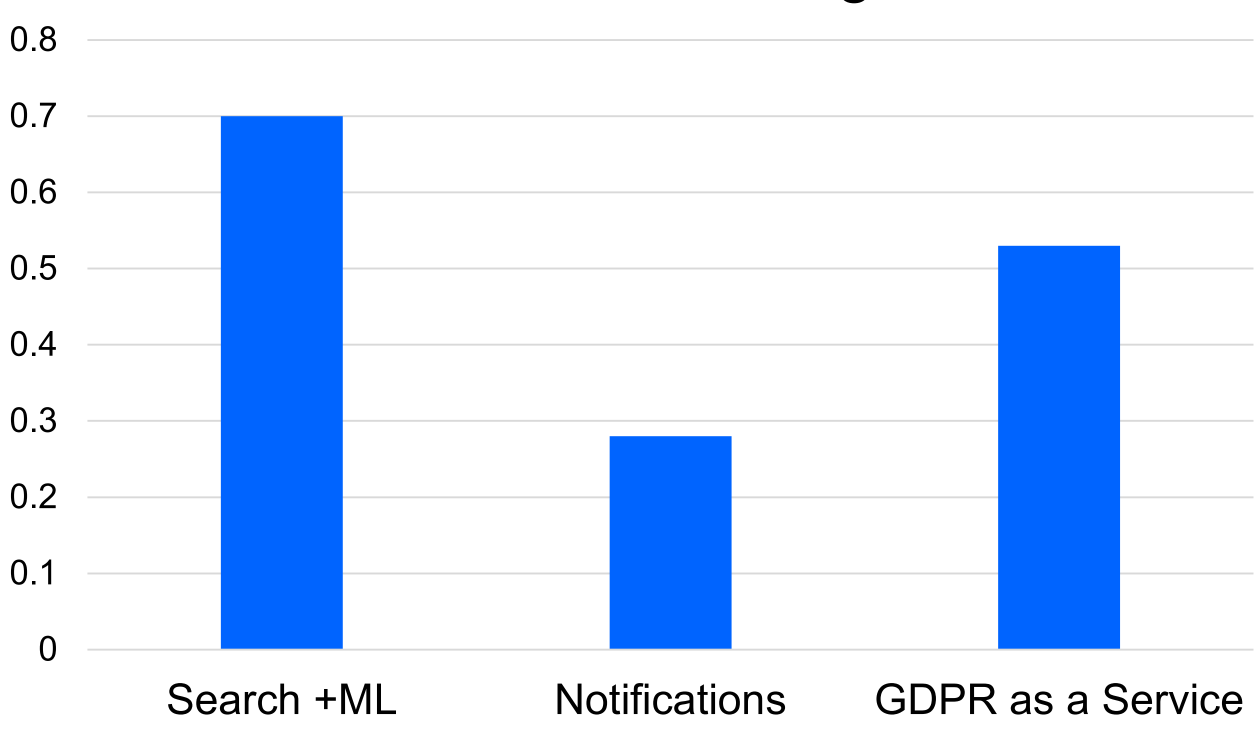
Legend



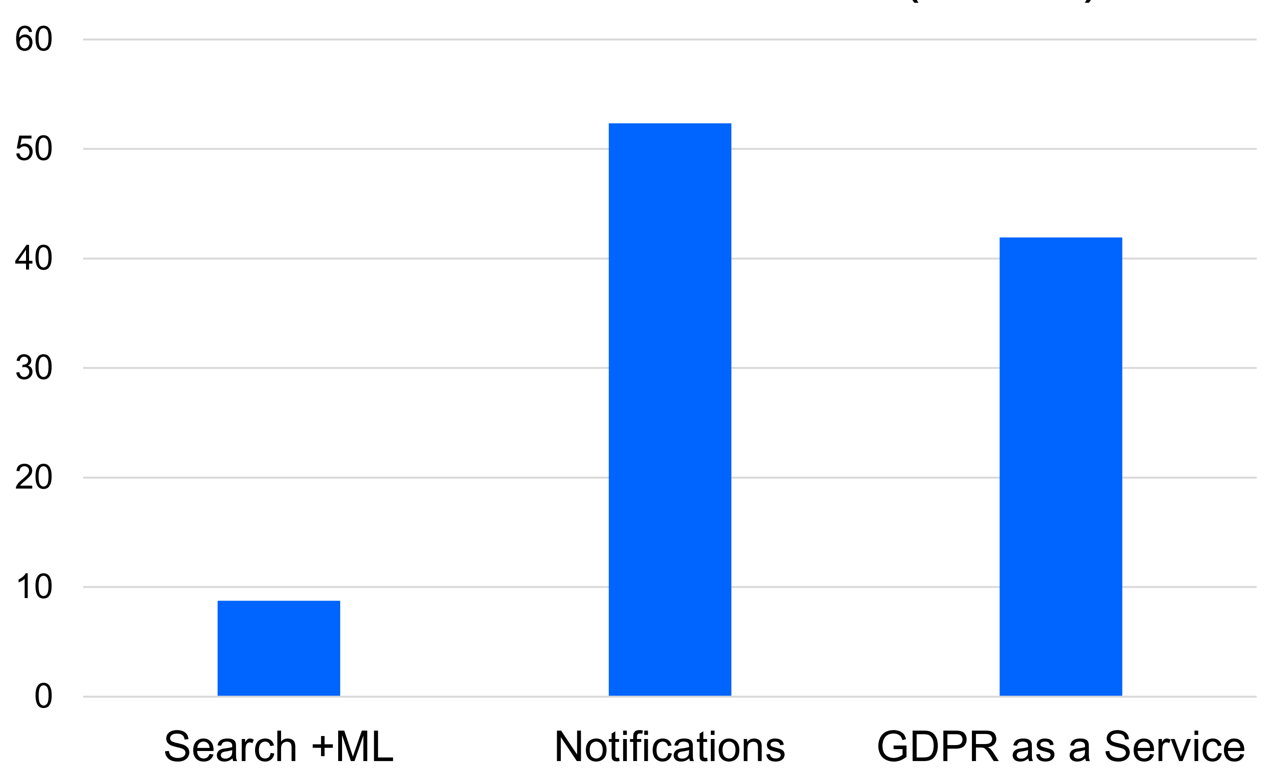
Metrics

Component	Number of PRs (z-score)	Unit Test Coverage	Hours between Commits (Mean)	Hours between Commits (STD DEV)
Search +ML	0.89	0.70	8.75	46.14
Notifications	**-0.68**	**0.28**	**52.35**	**301.40**
GDPR as a Service	**-1.25**	**0.53**	**41.92**	**249.44**

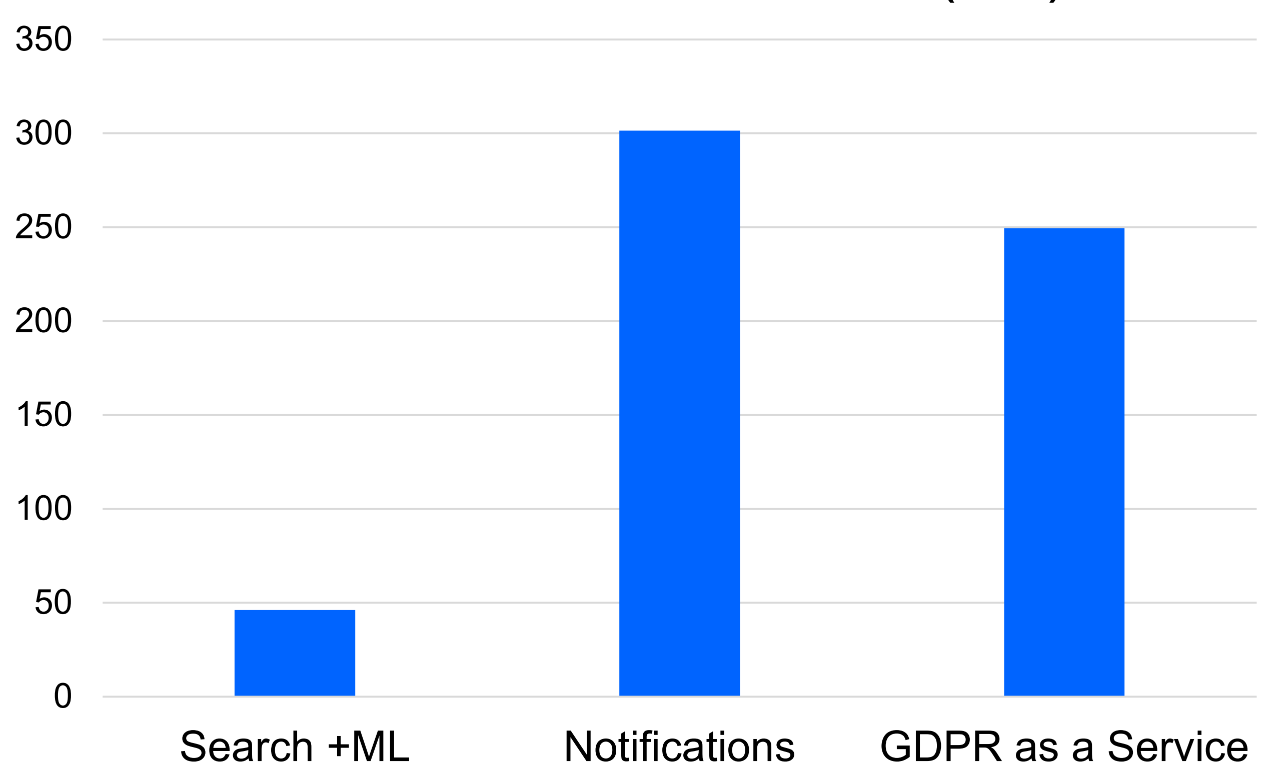
IBM Watson / © 2018 IBM Corporation



IBM Watson / © 2018 IBM Corporation



IBM Watson / © 2018 IBM Corporation



IBM Watson / © 2018 IBM Corporation

Summary

- Met new requirements with new components (**expansion phase**)
  - Including GDPR
  - Isolated low-touch tech debt
- Paid off high touch, high-debt tech debt (**contraction phase**)
  - Extracted common concerns into architecturally hoisted* shared library

* Keeling, M., Runde, J., & Gala, C. J. (2018). *Architectural Hoisting: Or How I Learned to Stop Writing Breaking Code and Love the Architecture.* SATURN

Discussion

IBM Watson / © 2018 IBM Corporation

Summary

- New requirements with met with new components
  - Strategic technical debt accepted in order to meet deadlines
- High-touch, high tech debt services were rewritten/combined
  - Increase in unit test coverage shows increased maintainability, decreased tech debt
- Low-touch, high tech debt services remained isolated
  - Large standard deviations in commit intervals shows modifications rarely necessary
- The overall process manifested in expansion/contraction cycles

Mistakes Made

- Started collecting application runtime metrics far too late
  - Revealed many problems as we added more
- Depended on customers to find our problems for us
  - Massive support cost
  - Especially when people started *using* our product

Advantages

- Ship fast
- Isolate tech debt
- Keep up with evolving product design
- Meet new requirements quickly
- Quick iterations per component
- Inexperienced team members can focus on small codebases

Disadvantages

- Experienced architect is probably essential
- Pay the microservice “premium”
  - Higher deployment, management overhead
- High support costs at first
  - On-call was hell for a while
- Have to *actually* pay off tech debt
  - Took a lot of convincing of product owner
  - Functional development mostly halted
  - Took us 3-4 months to get to a healthy state

Conclusion

© 2018 IBM Corporation

Conclusion

- Both general perspectives on starting with microservices have correct points
- By viewing getting bounded contexts wrong as strategic technical debt instead of as risk, starting with microservices can allow one to keeping up with rapidly evolving product design

© 2018 IBM Corporation

Conclusion

- Both general perspectives on starting with microservices have correct points
- By viewing getting bounded contexts wrong as strategic technical debt instead of as risk, starting with microservices can allow one to keeping up with rapidly evolving product design
- The key elements of this approach manifest as an expansion/contraction cycle:
  - Address new requirements with new components
  - Isolate technical debt
  - Pay off technical debt only in mission-critical components

Conclusion

- Both general perspectives on starting with microservices have correct points
- By viewing getting bounded contexts wrong as strategic technical debt instead of as risk, starting with microservices can allow one to keeping up with rapidly evolving product design
- The key elements of this approach manifest as an expansion/contraction cycle:
  - Address new requirements with new components
  - Isolate technical debt
  - Pay off technical debt only in mission-critical components
- Several simple preemptive measures could have avoided costly mistakes
  - e.g. instrument early

Conclusion

- Both general perspectives on starting with microservices have correct points
- By viewing getting bounded contexts wrong as strategic technical debt instead of as risk, starting with microservices can allow one to keeping up with rapidly evolving product design
- The key elements of this approach manifest as an expansion/contraction cycle:
  - Address new requirements with new components
  - Isolate technical debt
  - Pay off technical debt only in mission-critical components
- Several simple preemptive measures could have avoided costly mistakes
  - e.g. instrument early
- We think this strategy could be more generally applied against other types of uncertainty

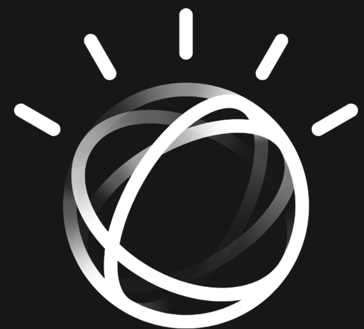
Acknowledgements

© 2018 IBM Corporation

Acknowledgements

- Michael Keeling, who started our team and trained us
- Joe Runde, for valuable feedback during the composition process
- William Chaparro, for encouraging and supporting us

© 2018 IBM Corporation



IBM Watson / © 2018 IBM Corporation

Supplementary Materials

Metrics (Comprehensive)

Component	Number of PRs (z-score)	Number of Commits (z-score)	Number of Comments (z-score)	Refactor Size (z-score)	Repository Size (z-score)	Percentage Code Refactored (z-score)	Log Count (z-score)	Unit Test Coverage	Time between Commits (Mean)	Time between Commits (STD DEV)
Search	1.09	0.09	0.83	0.13	0.06	-0.78	NA	0.53	7.70	39.03
Search + ML	0.89	1.25	0.81	1.88	1.26	-0.87	2.84	0.70	8.75	46.14
ML	-0.11	-1.31	0.16	-0.37	-0.45	-0.34	NA	0.50	40.44	317.65
Differentiator	0.39	0.06	-0.38	0.35	-0.35	-0.21	-0.28	0.58	26.04	139.32
REST	0.44	0.24	0.58	0.43	0.70	-0.92	-0.33	0.37*	16.32	61.58
Training Data	2.31	0.15	0.68	0.07	2.71	-1.09	-0.15	0.56	13.26	51.80
Preparation Agent	-0.34	-1.08	0.88	-0.47	-0.53	-0.04	NA	0.73	26.38	109.92
Training Agents	-0.27	0.59	0.80	0.06	-0.46	0.01	-0.35	0.68	26.06	93.90
Monitoring Agent	-0.80	-1.36	-1.34	-1.12	-0.65	0.23	-0.35	0.55	77.12	282.87
Notifications	-0.68	-0.15	0.16	-0.83	-0.53	-0.38	-0.35	0.28	52.35	301.40
GDPR	-1.25	1.56	-0.64	1.84	-0.43	0.92	-0.35	0.53	41.92	249.44

* The majority of the REST component's testing was in Cucumber tests, instead of unit tests, because it is a REST façade.

IBM Watson / © 2018 IBM Corporation

Metrics (Highlighted)

Component	Number of PRs (z-score)	Log Output Percentage (z-score)	Unit Test Coverage	Hours between Commits (Mean)	Hours between Commits (STD)
Search	**1.09**	NA	**0.53**	**7.70**	**39.03**
Search+ML	0.89	**2.84**	**0.70**	**8.75**	**46.14**
ML	-0.11	NA	**0.50**	**40.44**	**317.65**
Differentiator	0.39	-0.28	0.58	26.04	139.32
REST	0.44	-0.33	0.37*	16.32	61.58
Training Data	**2.31**	-0.15	0.56	13.26	51.80
Preparation Agent	-0.34	NA	0.73	26.38	109.92
Training Agents	-0.27	-0.35	0.68	26.06	93.90
Monitoring Agent	**-0.80**	-0.35	0.55	**77.12**	**282.87**
Notifications	**-0.68**	-0.35	**0.28**	**52.35**	**301.40**
GDPR as a Service	**-1.25**	-0.35	0.53	**41.92**	**249.44**

IBM Watson / © 2018 IBM Corporation

* The majority of the REST component's testing was in Cucumber tests, instead of unit tests, because it is a REST façade.