Mastering Challenges of Cloud-Native Architectures

with Spring

Timo Salm
Master Solutions Architect, Broadcom

May 2025

vmware by Broadcom

About Me

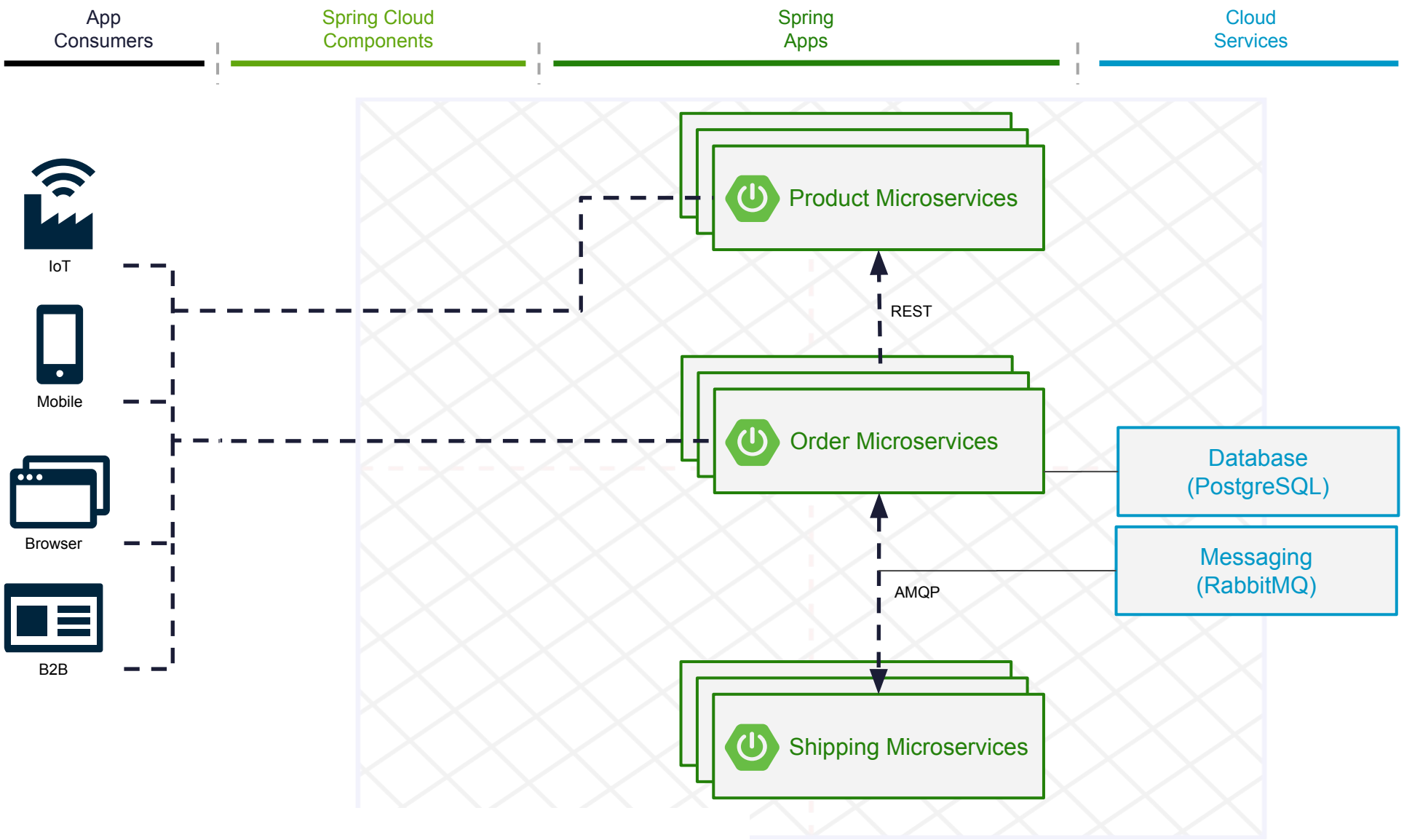
Timo Salm

Master Solutions Architect – EMEA

VMware Tanzu

- LinkedIn: https:/linkedin.com/in/timosalm/
- X (Twitter): @salmto
- GitHub: https://github.com/timosalm

A Typical Modern App Architecture



All Microservices:
Spring Boot

spring® by VMware Tanzu

Spring Framework

The Core Functionality

Spring Boot

Build Anything

Spring Cloud

Coordinate Anything

Code Clarity | Lower Complexity | Less Tech Debt |
Focus on Business Logic | Better Test Coverage | Faster Code Completion

https://12factor.net

The Twelve Factors

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
Store config in the environment

IV. Backing services
Treat backing services as attached resources

V. Build, release, run
Strictly separate build and run stages

VI. Processes
Execute the app as one or more stateless processes

VII. Port binding
Export services via port binding

VIII. Concurrency
Scale out via the process model

IX. Disposability
Maximize robustness with fast startup and graceful shutdown

X. Dev/prod parity
Keep development, staging, and production as similar as possible

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes

https://12factor.net

The Twelve Factors

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
Store config in the environment

IV. Backing services
Treat backing services as attached resources

V. Build, release, run
Strictly separate build and run stages

VI. Processes
Execute the app as one or more stateless processes

VII. Port binding
Export services via port binding

VIII. Concurrency
Scale out via the process model

IX. Disposability
Maximize robustness with fast startup and graceful shutdown

X. Dev/prod parity
Keep development, staging, and production as similar as possible

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes

https://12factor.net

The Twelve Factors

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
Store config in the environment

IV. Backing services
Treat backing services as attached resources

V. Build, release, run
Strictly separate build and run stages

VI. Processes
Execute the app as one or more stateless processes

VII. Port binding
Export services via port binding

VIII. Concurrency
Scale out via the process model

IX. Disposability
Maximize robustness with fast startup and graceful shutdown

X. Dev/prod parity
Keep development, staging, and production as similar as possible

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes

Configuration

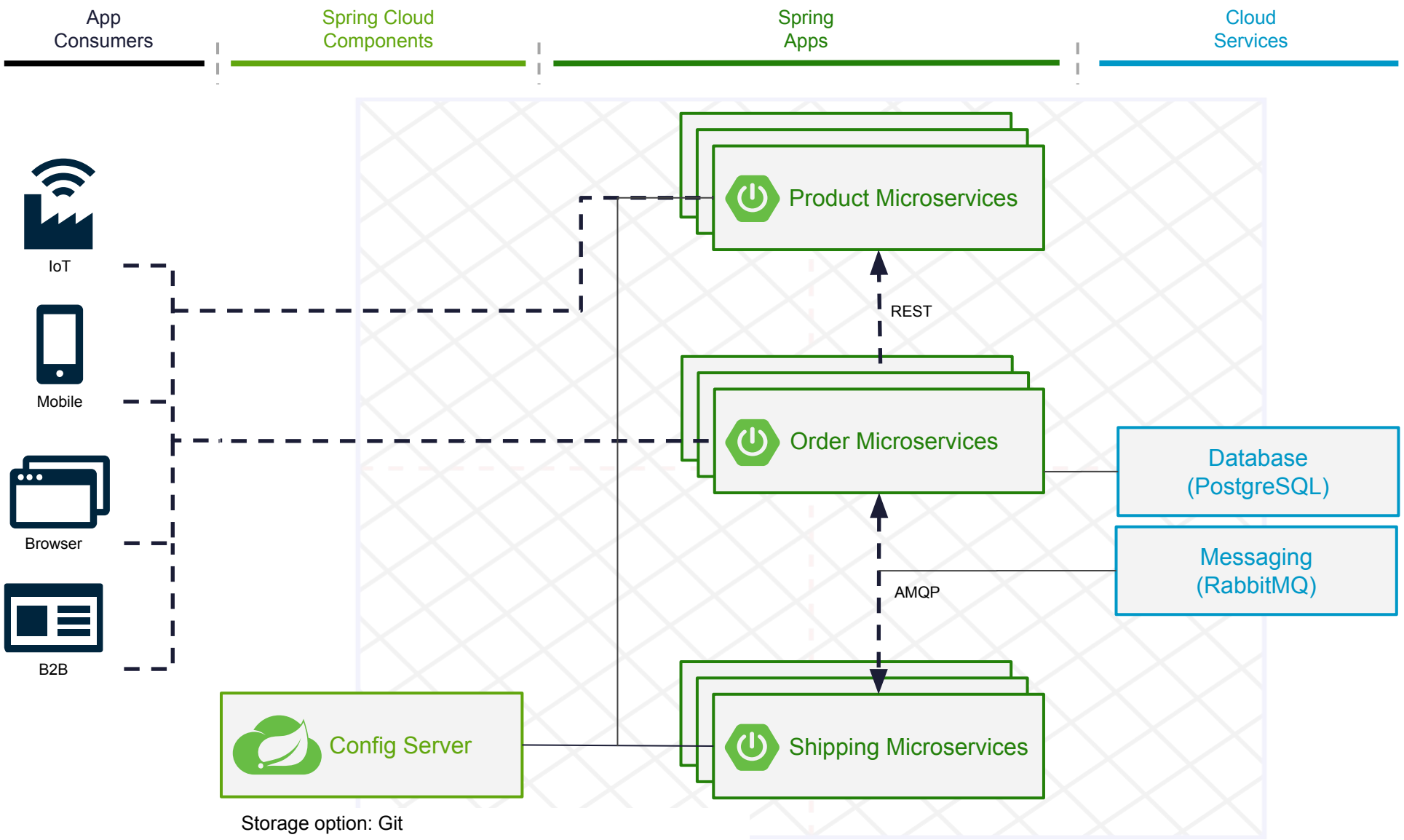
In cloud-native applications, configuration shouldn’t be bundled with code!

In the cloud, you have multiple applications, environments, and service instances — so configuration has to be flexible

Spring Cloud Config is designed to ease this burden

It delivers config straight to your apps and offers integration with multiple version control systems to keep your config safe

A Typical Modern App Architecture



All Microservices:
Spring Boot
Spring Cloud Config Client/Server

https://12factor.net

The Twelve Factors

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
Store config in the environment

IV. Backing services
Treat backing services as attached resources

V. Build, release, run
Strictly separate build and run stages

VI. Processes
Execute the app as one or more stateless processes

VII. Port binding
Export services via port binding

VIII. Concurrency
Scale out via the process model

IX. Disposability
Maximize robustness with fast startup and graceful shutdown

X. Dev/prod parity
Keep development, staging, and production as similar as possible

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes

Load Balancing

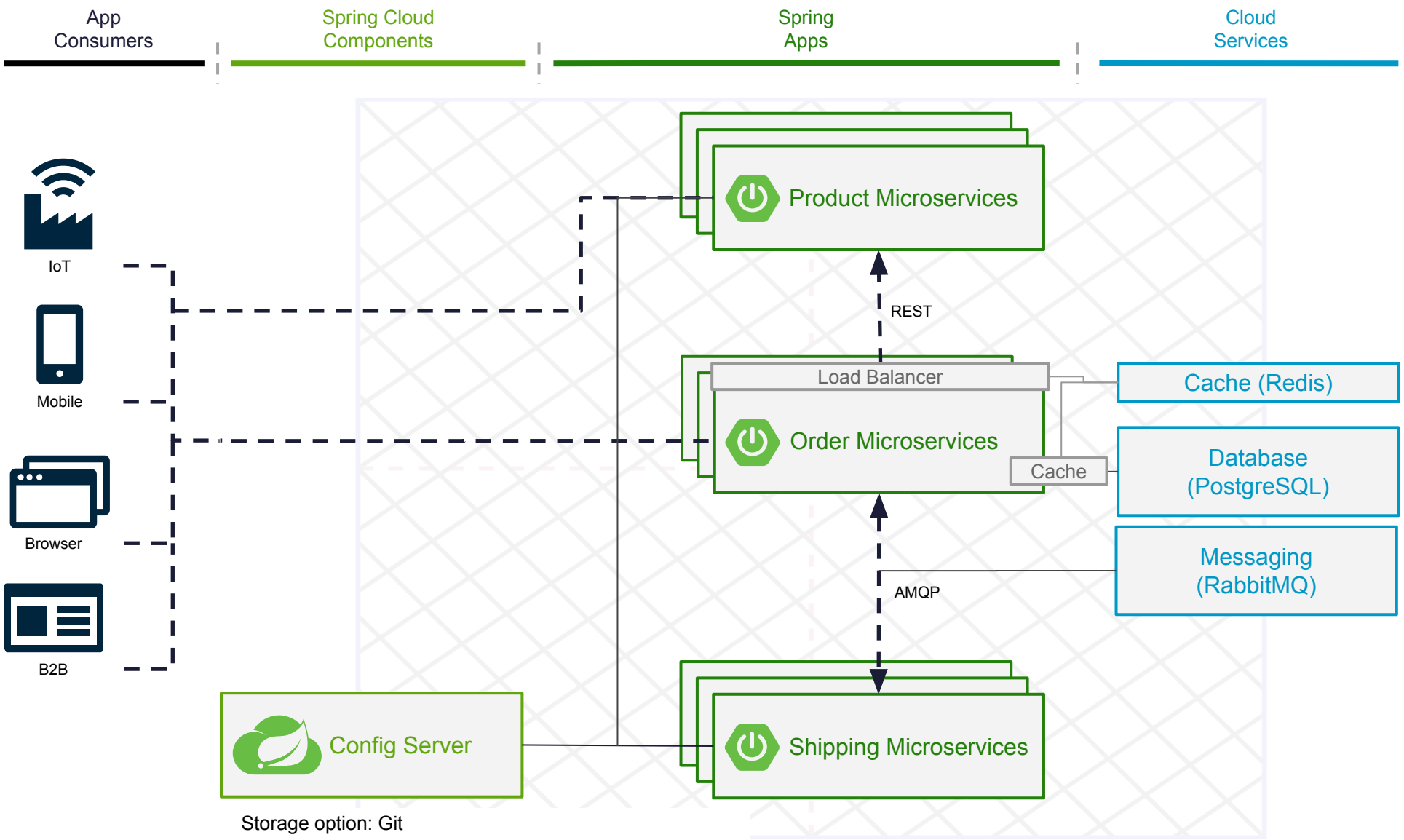
Great UXs need great responses. When your apps are under pressure, spreading the load helps smooth things out!

Spring Cloud Load Balancer can help your clients spread requests across multiple service instances

Integrates with `RestTemplate`, `RestClient`, and `WebClient`

Supports health checks, multiple caching options and Zone based balancing

A Typical Modern App Architecture



All Microservices:
Spring Boot
Spring Cloud Config Client/Server

Order Microservices:
+ Spring Cloud LoadBalancer

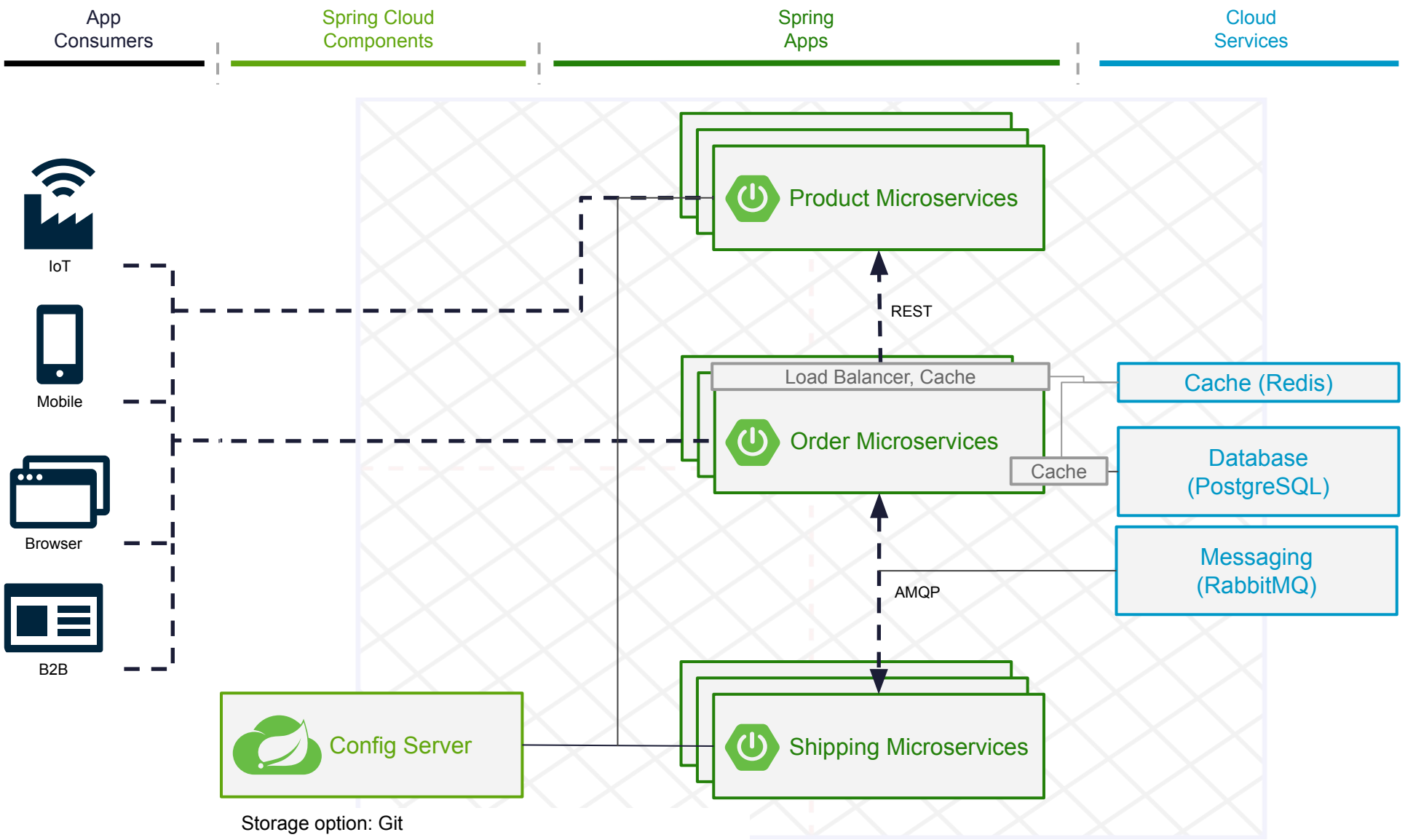
Caching

For example traditional databases are often too brittle or unreliable for use with microservices. That's why every modern distributed architecture needs a cache!

The Spring Framework provides support for transparently adding caching to an application

The cache abstraction does not provide an actual store. Examples for Cache providers that are supported out of the box are EhCache, Hazelcast, Couchbase, Redis and Caffeine. Other providers like VMware Tanzu GemFire can also be used with minimal configuration

A Typical Modern App Architecture



All Microservices:
Spring Boot
Spring Cloud Config Client/Server

Order Microservices:
+ Spring Cloud LoadBalancer
+ Spring Cache Abstraction

Circuit Breaker

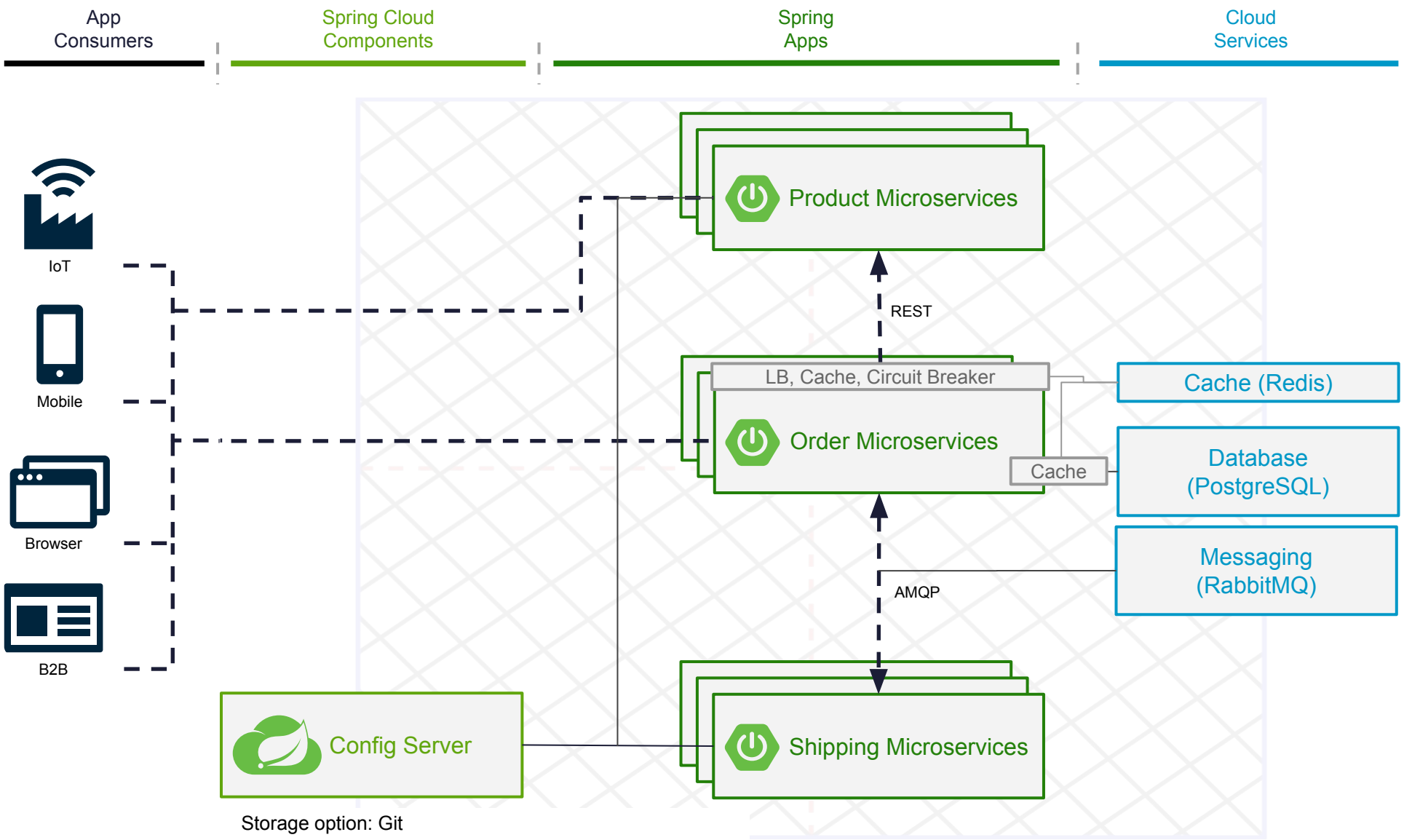
Distributed systems can be unreliable - requests might timeout or fail completely ...

Circuit breakers mitigate this problem using sensible defaults and reliable fallbacks in case of emergency.

Spring Cloud Circuit Breaker gives you the choice of two popular open-source options:

- Resilience4J
- Spring Retry
- Sentinel

A Typical Modern App Architecture



All Microservices:
Spring Boot
Spring Cloud Config Client/Server

Order Microservices:
+ Spring Cloud LoadBalancer
+ Spring Cache Abstraction
+ Spring Cloud Circuit Breaker

https://12factor.net

The Twelve Factors

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
Store config in the environment

IV. Backing services
Treat backing services as attached resources

V. Build, release, run
Strictly separate build and run stages

VI. Processes
Execute the app as one or more stateless processes

VII. Port binding
Export services via port binding

VIII. Concurrency
Scale out via the process model

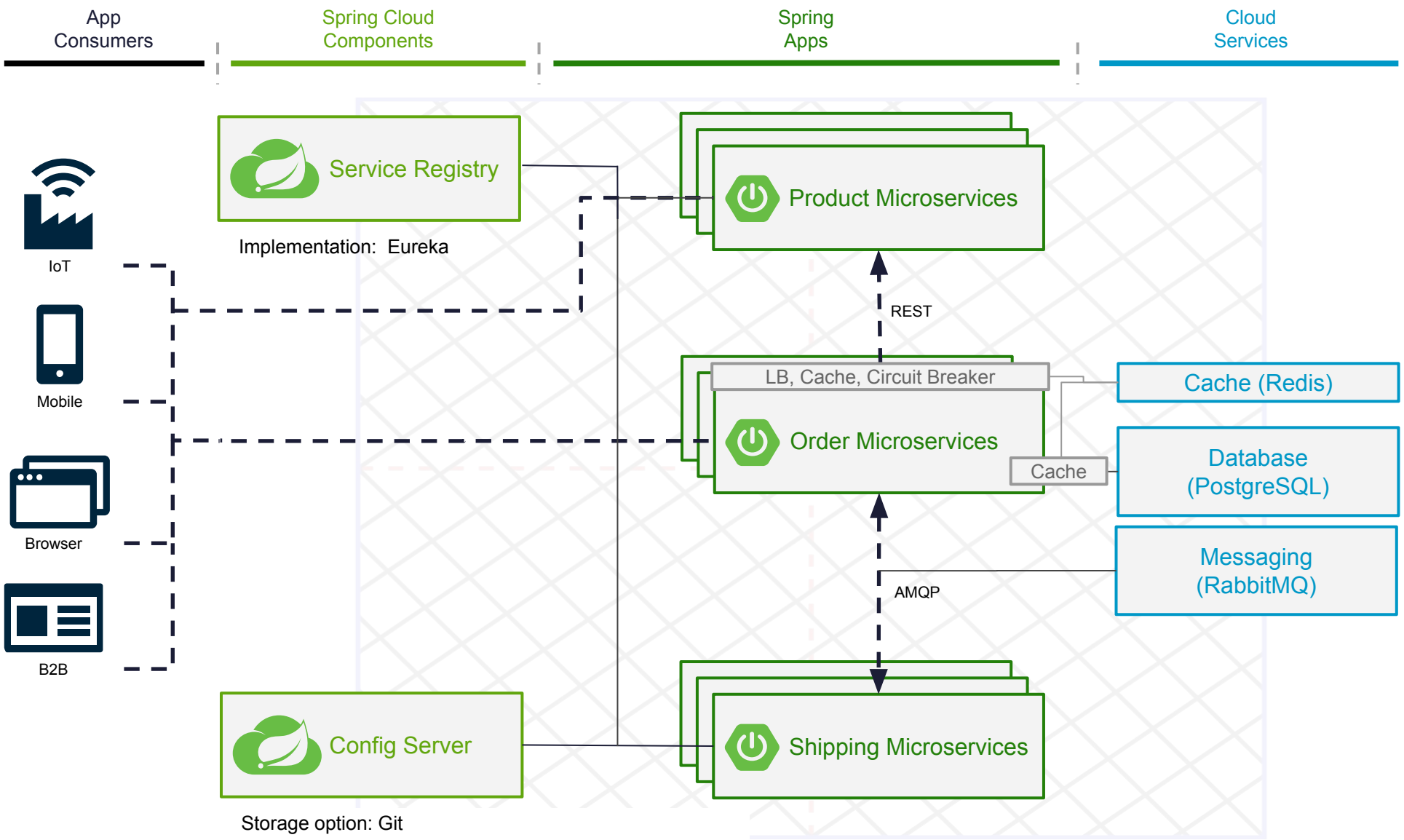
IX. Disposability
Maximize robustness with fast startup and graceful shutdown

X. Dev/prod parity
Keep development, staging, and production as similar as possible

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes

A Typical Modern App Architecture



All Microservices:
Spring Boot
Spring Cloud Config Client / Server
Spring Cloud Netflix (Service Discovery)

Order Microservices:
+ Spring Cloud LoadBalancer
+ Spring Cache Abstraction
+ Spring Cloud Circuit Breaker

https://12factor.net

The Twelve Factors

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
Store config in the environment

IV. Backing services
Treat backing services as attached resources

V. Build, release, run
Strictly separate build and run stages

VI. Processes
Execute the app as one or more stateless processes

VII. Port binding
Export services via port binding

VIII. Concurrency
Scale out via the process model

IX. Disposability
Maximize robustness with fast startup and graceful shutdown

X. Dev/prod parity
Keep development, staging, and production as similar as possible

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes

https://12factor.net

The Twelve Factors

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
Store config in the environment

IV. Backing services
Treat backing services as attached resources

V. Build, release, run
Strictly separate build and run stages

VI. Processes
Execute the app as one or more stateless processes

VII. Port binding
Export services via port binding

VIII. Concurrency
Scale out via the process model

IX. Disposability
Maximize robustness with fast startup and graceful shutdown

X. Dev/prod parity
Keep development, staging, and production as similar as possible

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes

Enhancing Runtime Efficiency

Several technologies can be leveraged to achieve performance gains:

- GraalVM Native Images
- Project CraC
- Class Data Sharing (CDS)
- AOT (Ahead-of-Time) Cache (JDK 24)
- Project Leyden (JDK 25+)

Each of the technologies has tradeoffs that should be considered depending on application needs

Supported in Spring Framework 6.X and Spring Boot 3.X

Performance Comparison

Java / Spring Boot	Mode	Startup time (ms)	Time change (%)	Memory used (MB)	Memory change (%)
1.8.0_442 / 2.7.3	Exec JAR	3.440	0.0	495.75	0.0
21.0.6 / 3.4.5	Exec JAR	3.790	+10.2	442.70	-9.1
21.0.6 / 3.4.5	Unpacked*	3.01	-7.0	418.11	-15.6
21.0.6 / 3.4.5	**GraalVM**	**0.39**	**-88.0**	**185.93**	**-37,5**
21.0.6 / 3.4.5	Project CraC	0.55	-82,9	367.23	-14,6
21.0.6 / 3.4.5	Spring AOT	3.34	3.40	436.37	-6,3
21.0.6 / 3.4.5	CDS	1.82	-43.8	402.05	-18,9
21.0.6 / 3.4.5	CDS + Spring AOT	1.52	-55.8	409.98	-17,3
24, Leyden EA	**AOT code compilation + Spring AOT**	**1.2**	**-65.1**	**426**	**-14.07**

Technology Comparison

	Startup	Warmup	Optimal peak throughput	Memory consumption	Compilation	Compatibility issues	Security	Container size vs JVM
Native executable with GraalVM	Instant	Instant	Training run with Oracle GraalVM	Reduced	Heavy and slow	Reachability metadata needed	✓	Smaller
JVM with Project CRaC	Instant	Instant with Training run	✓	No gain	Fast	Restoration Linux only	Secret leaking in snapshots	Bigger
JVM with CDS	1.5x faster 2x with Spring AOT	No gain	✓	Slightly reduced	Fast	Training run side effects	✓	Bigger
JVM with AOT cache [1] + compiled code	3x faster 4x with Spring AOT	Fast with training run in Java 25+?	✓	Slightly reduced in Java 25? [2]	Fast	Training run side effects	✓	Slightly bigger [3]

[1] Available as of Java 24 without code AOT compilation
[2] Reduced memory consumption has not yet been a focus of Project Leyden, but we expect same benefits than CDS
[3] AOT cache, unlike CDS, allows to remove the built-in JDK CDS archive while still keeping the same performance benefits

https://12factor.net

The Twelve Factors

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
Store config in the environment

IV. Backing services
Treat backing services as attached resources

V. Build, release, run
Strictly separate build and run stages

VI. Processes
Execute the app as one or more stateless processes

VII. Port binding
Export services via port binding

VIII. Concurrency
Scale out via the process model

IX. Disposability
Maximize robustness with fast startup and graceful shutdown

X. Dev/prod parity
Keep development, staging, and production as similar as possible

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes

Example for a Simple vs an Optimized Container Image

```
FROM: openjdk:8-jdk-alpine

EXPOSE 8080

COPY target/*.jar app.jar

ENTRYPOINT ["java","-jar","/app.jar"]
```

Example for a Simple vs an Optimized Container Image

```
FROM: openjdk:8-jdk-alpine

EXPOSE 8080

COPY target/*.jar app.jar

ENTRYPOINT ["java","-jar","/app.jar"]
```

```
FROM eclipse-temurin:21-jdk-alpine as builder

WORKDIR application

ARG JAR_FILE=target/*.jar

COPY ${JAR_FILE} application.jar

RUN java -Djarmode=layertools -jar application.jar extract


FROM eclipse-temurin:21-jre-alpine

WORKDIR application

COPY --from=builder application/dependencies/ ./

COPY --from=builder application/spring-boot-loader/ ./

COPY --from=builder application/snapshot-dependencies/ ./

COPY --from=builder application/application/ ./

ENTRYPOINT ["java",
"org.springframework.boot.loader.JarLauncher"]
```

Secure and Up-to-Date Containers

Buildpacks.io

Cloud Native Buildpacks transform your application source code to images that can run on any cloud

- Portability via the OCI standard
- Ensure apps meet security and compliance requirements
- Perform upgrades with minimal effort and intervention
- CNCF Incubation

Spring Boot supports building containers with Cloud Native Buildpacks via a Maven and Gradle plugin

```
./gradlew bootBuildImage

./mvnw spring-boot:build-image
```

https://12factor.net

The Twelve Factors

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
Store config in the environment

IV. Backing services
Treat backing services as attached resources

V. Build, release, run
Strictly separate build and run stages

VI. Processes
Execute the app as one or more stateless processes

VII. Port binding
Export services via port binding

VIII. Concurrency
Scale out via the process model

IX. Disposability
Maximize robustness with fast startup and graceful shutdown

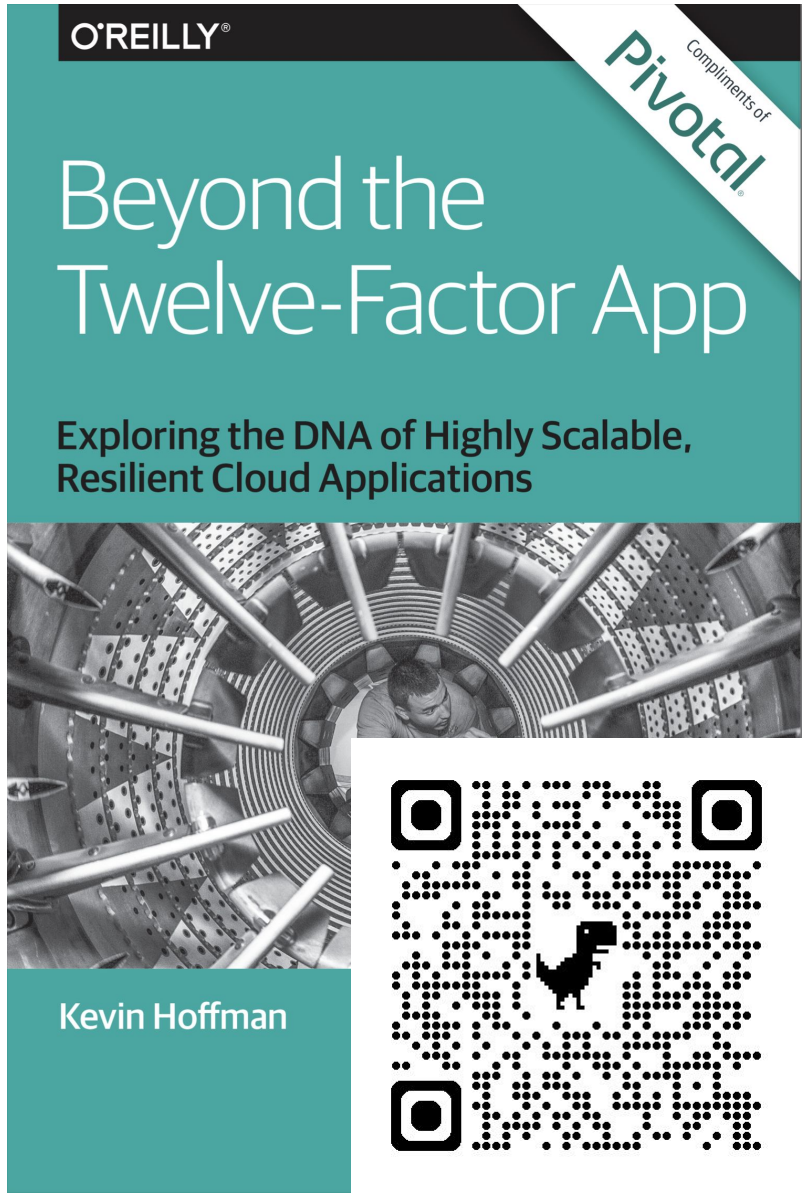
X. Dev/prod parity
Keep development, staging, and production as similar as possible

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes

Taking into account the changes in priority order, definition, and additions, this book describes the following facets of cloud-native applications:

1. One codebase, one application
2. API first
3. Dependency management
4. Design, build, release, and run
5. Configuration, credentials, and code
6. Logs
7. Disposability
8. Backing services
9. Environment parity
10. Administrative processes
11. Port binding
12. Stateless processes
13. Concurrency
14. Telemetry
15. Authentication and authorization



https://www.vmware.com/docs/ebook-beyond-the-12-factor-app

Distributed Tracing

Debugging distributed applications is a complex and time consuming chore!

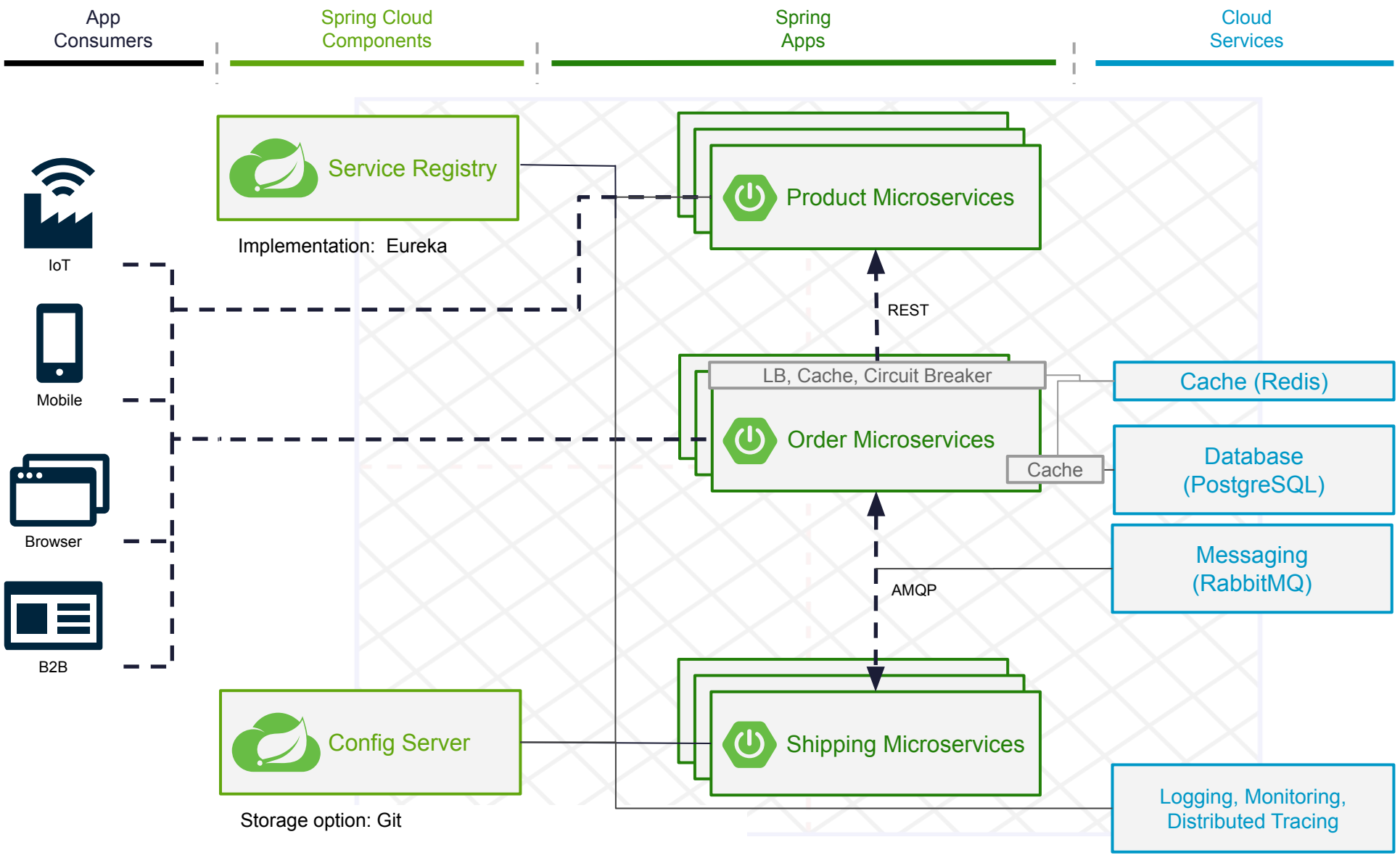
For any given failure or poor experience, you'll need to piece together traces from multiple independent microservices.

Spring Boot Actuator provides dependency management and auto-configuration for Micrometer Tracing, a facade for popular tracer libraries.

It ships auto-configuration for the following tracers:

- OpenTelemetry with Zipkin, or OTLP
- OpenZipkin Brave with Zipkin

A Typical Modern App Architecture

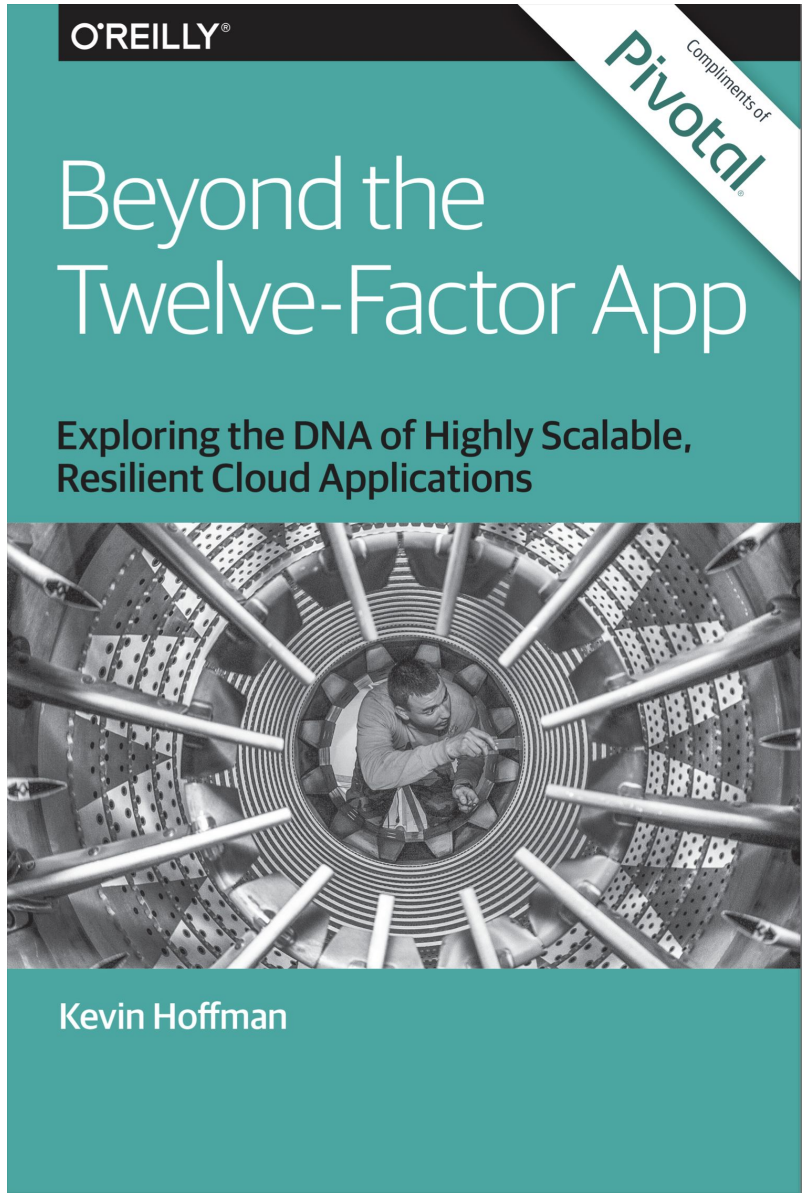


All Microservices:
Spring Boot
Spring Cloud Config Client/Server
Spring Cloud Netflix (Service Discovery)

Order Microservices:
+ Spring Cloud LoadBalancer
+ Spring Cache Abstraction
+ Spring Cloud Circuit Breaker

Taking into account the changes in priority order, definition, and additions, this book describes the following facets of cloud-native applications:

1. One codebase, one application
2. API first
3. Dependency management
4. Design, build, release, and run
5. Configuration, credentials, and code
6. Logs
7. Disposability
8. Backing services
9. Environment parity
10. Administrative processes
11. Port binding
12. Stateless processes
13. Concurrency
14. Telemetry
15. Authentication and authorization



https://tanzu.vmware.com/content/ebooks/beyond-the-12-factor-app

Gateway

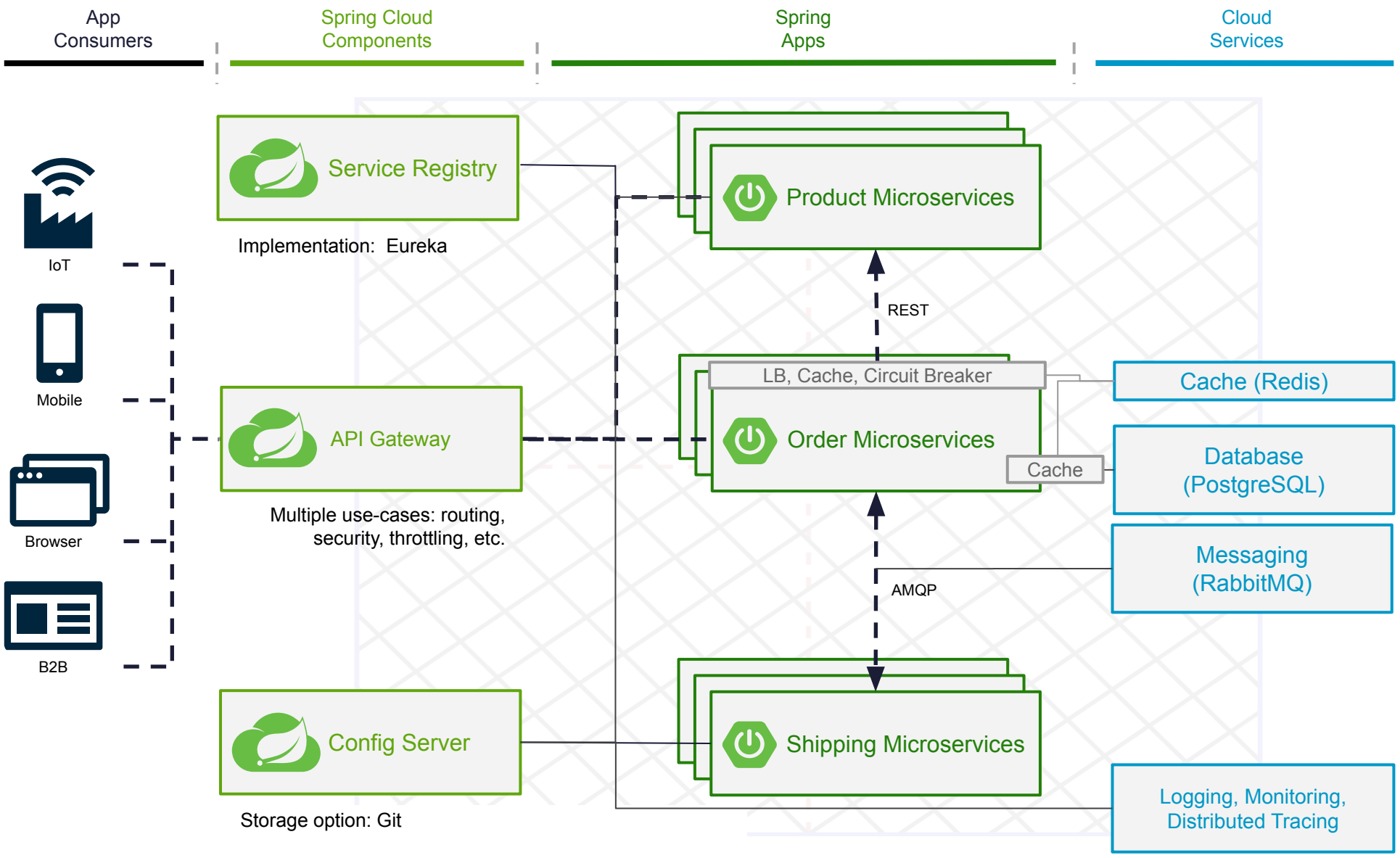
With so many APIs in play, developers need an
API Gateway that they can control!

Spring Cloud Gateway puts developers in control of APIs:

- Securing and hiding services
- Routing and filtering messages
- Handling load
- And much more …

Manage your config in regular version-control.
Roll-out your changes instantly - no tickets, no downtime!

A Typical Modern App Architecture



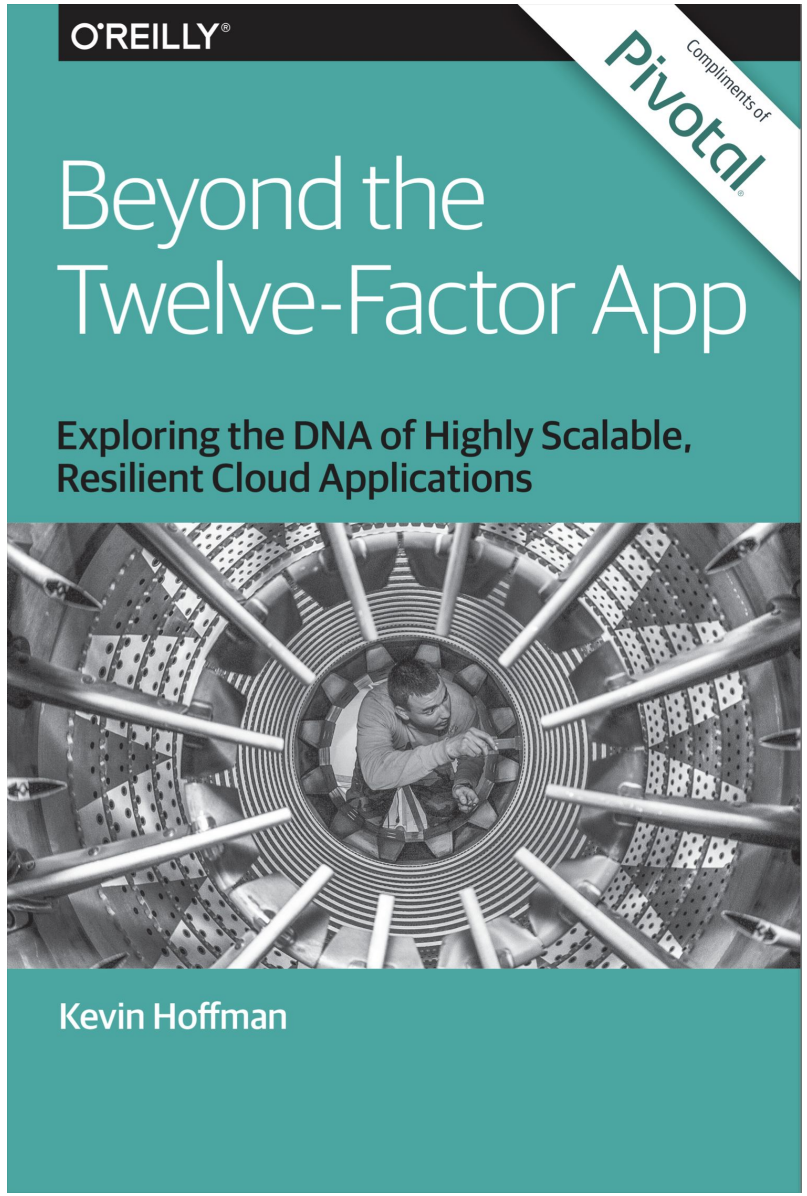
All Microservices:
Spring Boot
Spring Cloud Config Client/Server
Spring Cloud Netflix (Service Discovery)

Order Microservices:
+ Spring Cloud LoadBalancer
+ Spring Cache Abstraction
+ Spring Cloud Circuit Breaker

Gateway
+ Spring Cloud Gateway

Taking into account the changes in priority order, definition, and additions, this book describes the following facets of cloud-native applications:

1. One codebase, one application
2. API first
3. Dependency management
4. Design, build, release, and run
5. Configuration, credentials, and code
6. Logs
7. Disposability
8. Backing services
9. Environment parity
10. Administrative processes
11. Port binding
12. Stateless processes
13. Concurrency
14. Telemetry
15. Authentication and authorization



https://tanzu.vmware.com/content/ebooks/beyond-the-12-factor-app

Authentication and Authorization

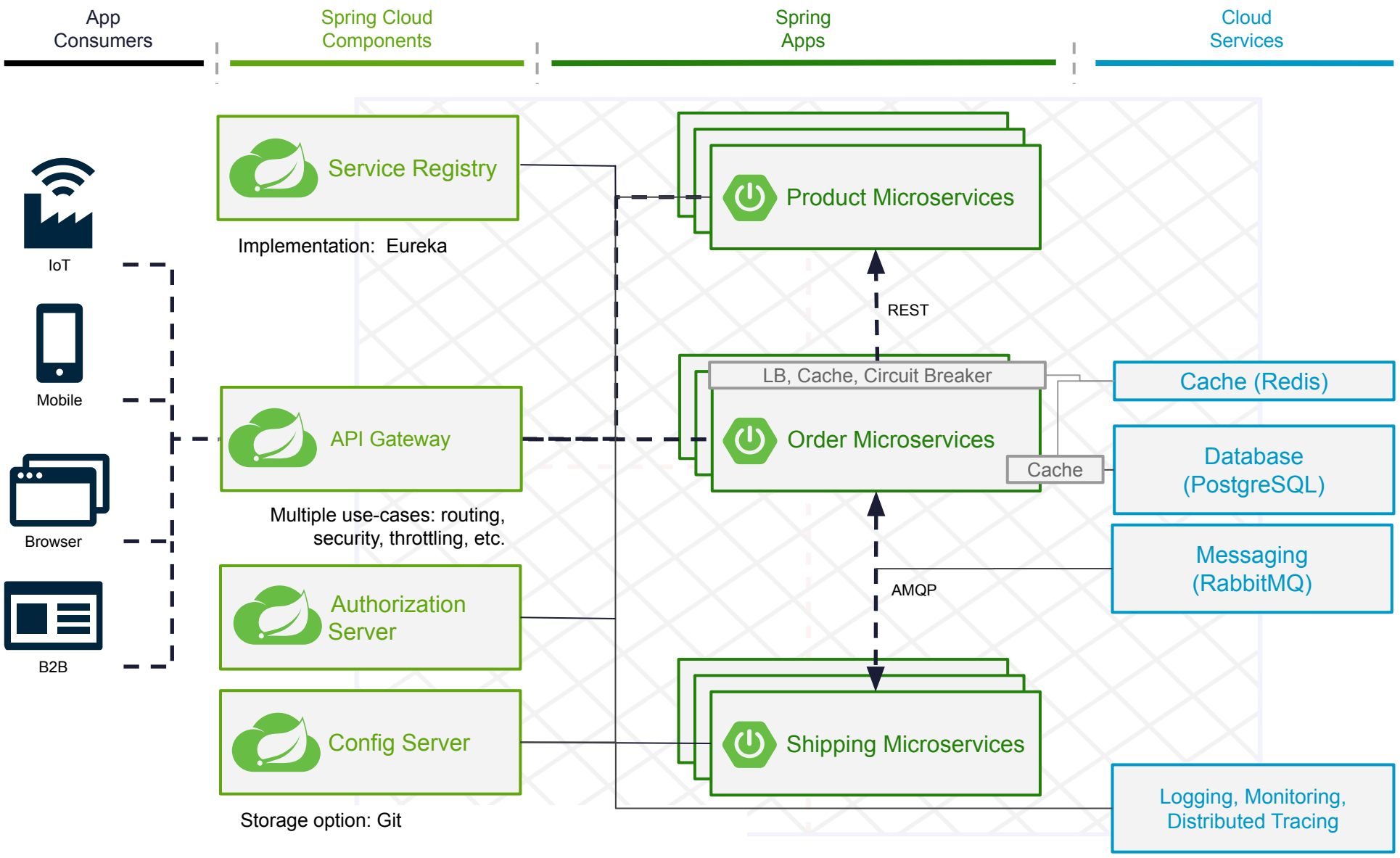
Security is a vital part of any application and cloud environment!

OAuth 2 is an authorization framework granting clients access to protected resources via an authorization server.

To make the application secure, you can simply add Spring Security as a dependency.
By adding the Spring Security OAuth 2 Client, it will secure your app with OAuth 2 by default.

Spring Authorization Server delivers OAuth 2 Authorization Server support to the Spring community.

A Typical Modern App Architecture



All Microservices:
Spring Boot
Spring Cloud Config Client/Server
Spring Cloud Netflix (Service Discovery)
Spring Security (OAuth 2 Client)

Order Microservices:
+ Spring Cloud LoadBalancer
+ Spring Cache Abstraction
+ Spring Cloud Circuit Breaker

Gateway
+ Spring Cloud Gateway

Streaming Data

When you're working with streaming data, you need three key abstractions to simplify your code:

- Binders to integrate messaging systems (Kafka, RabbitMQ, SQS, etc.)
- Bindings to bridge the gap between messaging systems and code
- Messages to provide structure for data

Spring Cloud Stream delivers them all. Spring Cloud Stream also handles provisioning, content conversion, error handling, config management, consumer groups, partitioning, monitoring, and more

Serverless & Functions

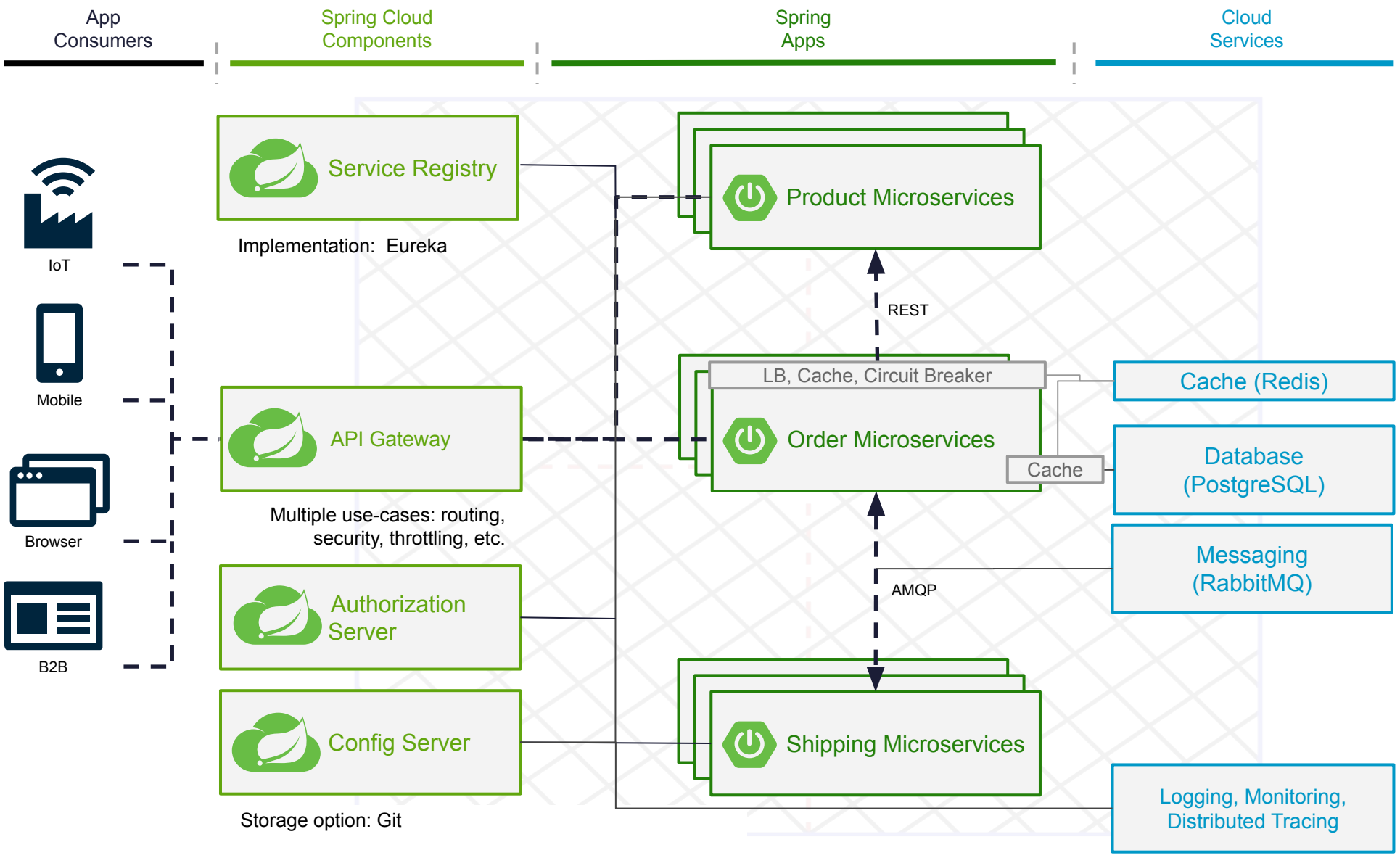
Vendor lock-in is a concern for many, so why not decouple your functions from your provider?

Spring Cloud Function lets you write functions once and run anywhere with familiar Spring APIs.

Function chaining lets you create sophisticated capabilities with ease

Multiple function inputs and outputs allow you to merge, join, and build other advanced use cases

A Typical Modern App Architecture



All Microservices:
Spring Boot
Spring Cloud Config Client/Server
Spring Cloud Netflix (Service Discovery)
Spring Security (OAuth 2 Client)

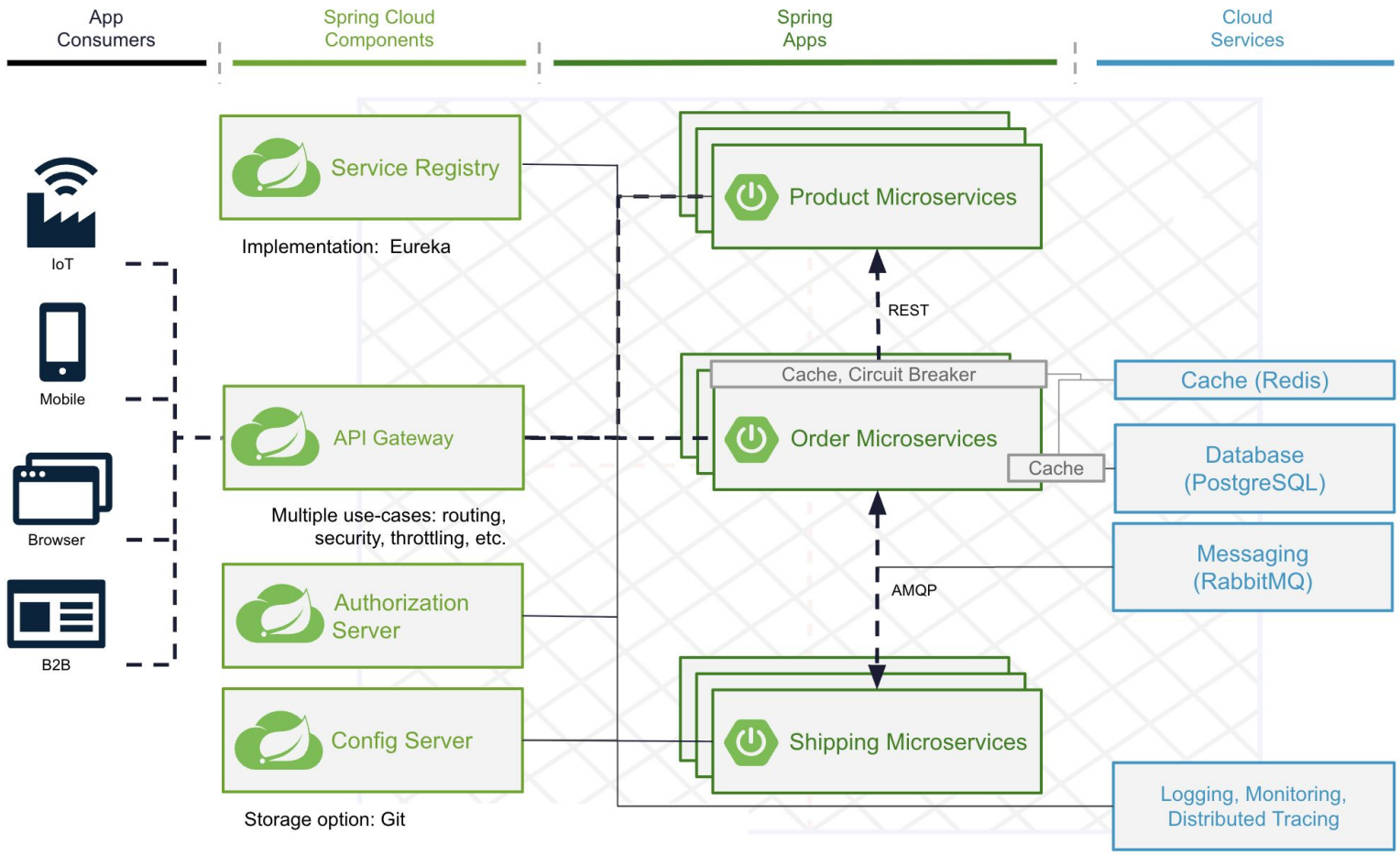
Order Microservices:
+ Spring Cloud LoadBalancer
+ Spring Cache Abstraction
+ Spring Cloud Circuit Breaker

Shipping Microservices:
+ Spring Cloud Stream
+ Spring Cloud Function

Gateway
+ Spring Cloud Gateway

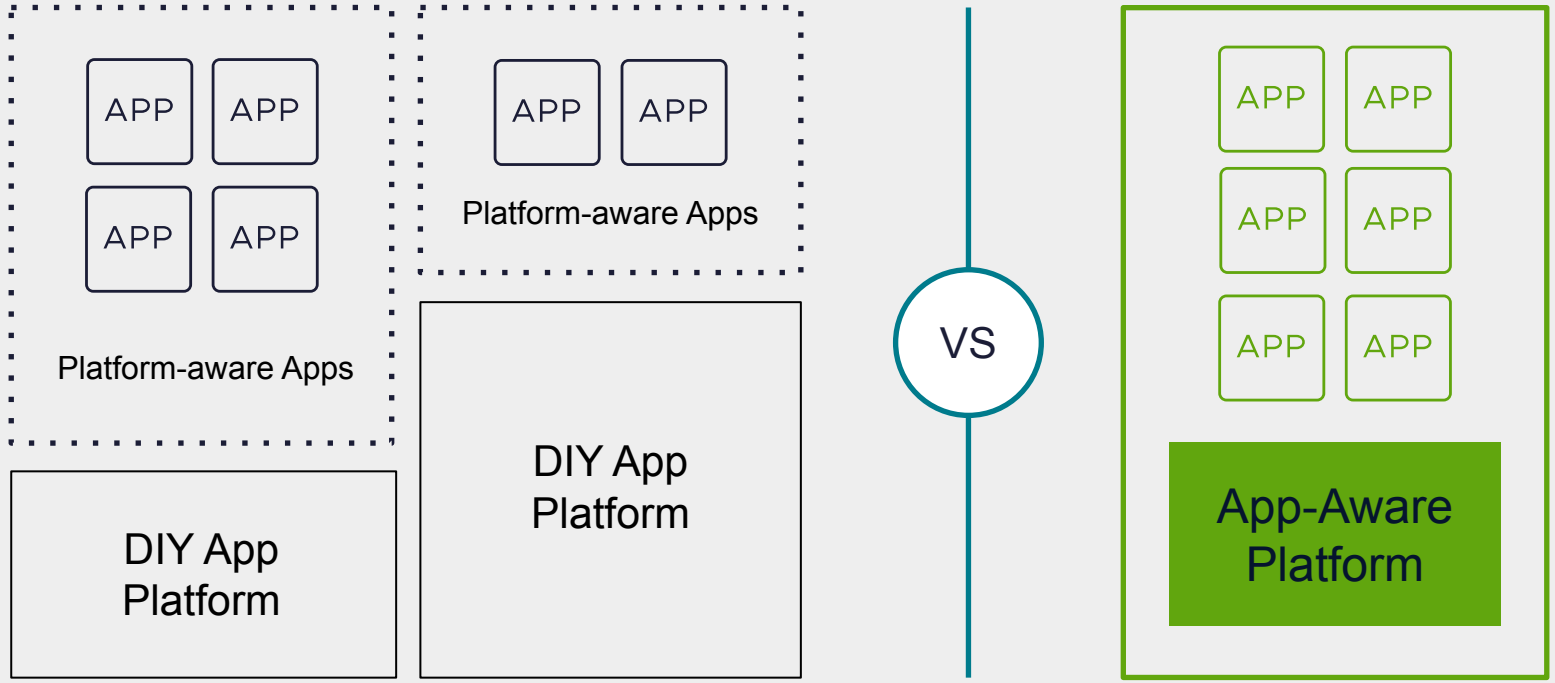
Common challenges

- High effort required to manage cloud infrastructure for microservices
- Application lifecycle is difficult to manage
- Painful to troubleshoot application issues



The Benefits of an App-Aware Platform

Removing the burden from the developers



Developer focuses on defining requirements for app

Enables shift-left model with no burden to developer

Full portability from development to production

Platform best-practice are automatically applied

Thank You

https://github.com/timosalm/modern-app-arch-spring-cloud

vmware by Broadcom