# Processing CloudEvents with Spring and Knative

## A quick tour

Thomas Risberg

vmware

# Agenda

## Looking back

Spring Framework 1.0 released first day of Spring in 2004

## Building Blocks

Spring Boot - *Convention over Configuration for Spring*

Spring Cloud - *Create Cloud Native Apps*

Cloud Native Buildpacks - *Build source code into OCI images*

CloudEvents - *Specification for describing event data*

Knative Serving - *Serverless on Kubernetes*

Knative Eventing - *Event Delivery*

## Building our first CloudEvents app

Step-by-step

We've come a long way in 16 years!

Spring

java/j2ee Application Framework

Home
Mission Statement
Downloads
Documentation
Demo/Tutorial

License
Buttons
Source Forge Project
Mailing Lists
Discussion/Help Forums
JIRA Issue Tracking

**Other languages**

SpringFramework 中文论坛 首页

Spring Pad
Light-weightコンテナ
Spring FrameworkのWiki

Home

## SPRING IS HERE!

We are delighted to announce the arrival of the

**Spring Framework 1.0 Final Release**

Thanks to all contributors and early adopters that have followed our 1.0 milestones and release candidates: Spring wouldn't be as mature as it is without you! Read more here [2004-03-24]

vmware

# Guiding Principles

**Spring Framework** aimed to make it easier for developers to focus on solving business problems rather than infrastructure issues.

Spring started when applications where often deployed to J2EE application servers like WebLogic or WebSphere.

The same principles can be applied to new deployment environment like Kubernetes.

**Spring Boot** brought Convention over Configuration to Spring apps.

Spring Boot combined with **Spring Cloud** made it easy to write cloud native micro services.

# Building Blocks

## What we need to build and run our app

# Spring Boot

**Spring Boot** brought Convention over Configuration to Spring apps.

Spring Boot provides: auto-configuration, starters, actuator, test and more …

And now your apps can fit in a tweet:

# Spring Cloud Function

**Spring Cloud** brings common features to Cloud Native apps.

Like: distributed configuration, service registration/discovery, routing, circuit breakers, function support and much more …

We'll use the function support from Spring Cloud Function:

```
@Bean
public Function<Message<JsonNode>, Message<String>> fun() {
    return (in) -> {
        CloudEvent<AttributesImpl, SpringEvent> cloudEvent = CloudEventMapper.convert(in, SpringEvent.class);
        String results = "Processed: " + cloudEvent.getData();
        log.info(results);
        return MessageBuilder.withPayload(results).build();
    };
}
```

# Cloud Native Buildpacks

**Cloud Native Buildpacks** “pluggable, modular tools that translate source code into OCI images”

Helps manage apps at scale with automated delivery of both OS-level and application-level dependency upgrades.

We use the **pack** CLI to build our images with the “**cloudfoundry/cnb:cflinuxfs3**“ builder

# CloudEvents

**CloudEvents** “A specification for describing event data in a common way”

*“Events are everywhere, yet event publishers tend to describe events differently.”*

Why CloudEvents?

- Consistency
- Accessibility
  - SDKs for
    - Go
    - JavaScript
    - Java
    - C#
    - Ruby
    - Python
- Portability

```
Content-Type: application/json
ce-specversion: 1.0
ce-type: myevent
ce-id: 1234-1234-1234
ce-source: example.com

{
  "specversion": "1.0",
  "type": "coolevent",
  "id": "xxxx-xxxx-xxxx",
  "source": "bigco.com",
  "data": { ... }
}
```

# CloudEvent type used for demo

JSON schema used for the demo.

We generate a Java class using the **jsonschema2pojo** Maven plugin.

We parse events and create instances of **CloudEvent** class from the CloudEvent SDK for Java.

```
{
    "$schema": "http://json-schema.org/draft-07/schema#",
    "title": "SpringEvent",
    "description" : "This is the schema for the SpringEvent type.",
    "type": "object",
    "properties": {
        "releaseDate": {
            "type": "string",
            "format": "date-time"
        },
        "releaseName": {
            "type": "string"
        },
        "version": {
            "type": "string"
        }
    },
    "additionalProperties": false
}
```

# Knative

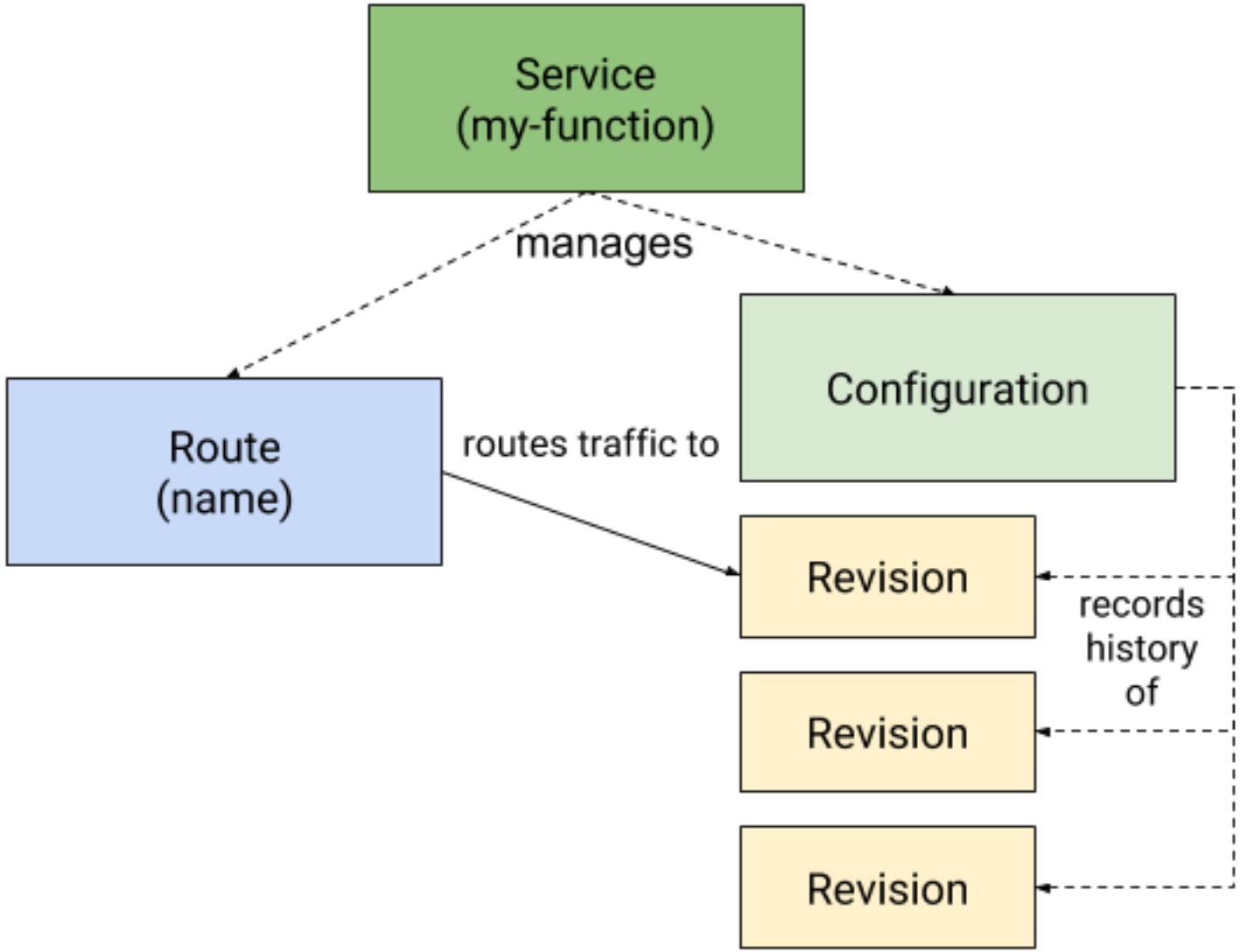
**Knative** “Make your developers more productive”

*“Knative components build on top of Kubernetes, abstracting away the complex details and enabling developers to focus on what matters.”*


Highlights

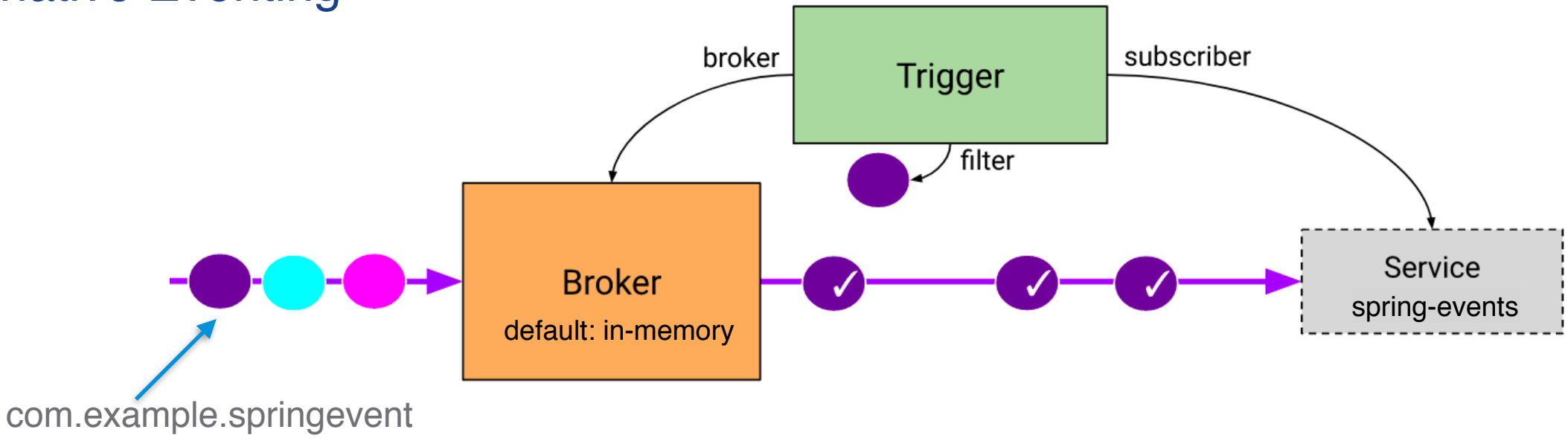
- Focused API with higher level abstractions for common app use-cases. Stand up a scalable, secure, stateless service in seconds.
- Loosely coupled features let you use the pieces you need.
- Pluggable components let you bring your own logging and monitoring, networking, and service mesh.
- Knative is portable: run it anywhere Kubernetes runs, never worry about vendor lock-in.
- Idiomatic developer experience, supporting common patterns such as GitOps, DockerOps, ManualOps.
- Knative can be used with common tools and frameworks such as Django, Ruby on Rails, Spring, and many more.

# Knative Serving




```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: spring-events
  namespace: default
spec:
  template:
    spec:
      containers:
        - image: spring-events
```

# Knative Eventing




```
apiVersion: eventing.knative.dev/v1alpha1
kind: Trigger
metadata:
  name: spring-events
  annotations:
    knative-eventing-injection: enabled
spec:
  filter:
    attributes:
      type: com.example.springevent
  subscriber:
    ref:
     apiVersion: v1
     kind: Service
     name: spring-events
```

# Building our first CloudEvents app

## Putting it all together

# The Developer Experience

should be application-centric and team-centric, not infrastructure-centric

# Time for some code …

```
@Bean
public Function<Message<JsonNode>, Message<String>> fun() {
    return (in) -> {
        CloudEvent<AttributesImpl, SpringEvent> cloudEvent = CloudEventMapper.convert(in, SpringEvent.class);
        String results = "Processed: " + cloudEvent.getData();
        log.info(results);
        return MessageBuilder.withPayload(results).build();
    };
}
```

## Initialize a project

Initialize a Spring Boot function application from start.spring.io:

```
APPNAME=spring-events
curl https://start.spring.io/starter.tgz \
 -d dependencies=webflux,actuator,cloud-function \
 -d language=java \
 -d javaVersion=11 \
 -d type=maven-project \
 -d groupId=com.example \
 -d artifactId=${APPNAME} \
 -d name=${APPNAME} \
 -d packageName=com.example.${APPNAME} \
 -d baseDir=${APPNAME} | tar -xzvf -
cd ${APPNAME}
```

## Add the function code

Add CloudEvents API as a dependency in `pom.xml`:

```
        <dependency>
            <groupId>io.cloudevents</groupId>
            <artifactId>cloudevents-api</artifactId>
            <version>1.2.0</version>
        </dependency>
```

# Step-by-step guide:

https://github.com/trisberg/spring-knative-cloudevents-2020/blob/master/spring-knative-cloudevents.adoc

# Resources

**Spring Framework:** https://spring.io/projects/spring-framework

**Spring Boot:** https://spring.io/projects/spring-boot

**Spring Cloud Function:** https://spring.io/projects/spring-cloud-function

**Cloud Native Buildpacks:** https://buildpacks.io/

**Knative:** https://knative.dev/

**Skaffold:** https://skaffold.dev/

# Thank You