

Processing CloudEvents with Spring and Knative

A quick tour

Thomas Risberg

About Me

Thomas Risberg (@trisberg)

Member of the Spring engineering team at Pivotal

Contributing to projectriff, serverless functions for stream and event processing

Joined the Spring Framework open source project in 2003 working on JDBC support

Co-authored “Professional Java Development with the Spring Framework” together with Rod Johnson, Juergen Hoeller, Alef Arendsen, and Colin Sampaleanu published by Wiley in 2005



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コンテンツ

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]

Building Blocks

What we need to build and run our app

Knative

Knative “Make your developers more productive”

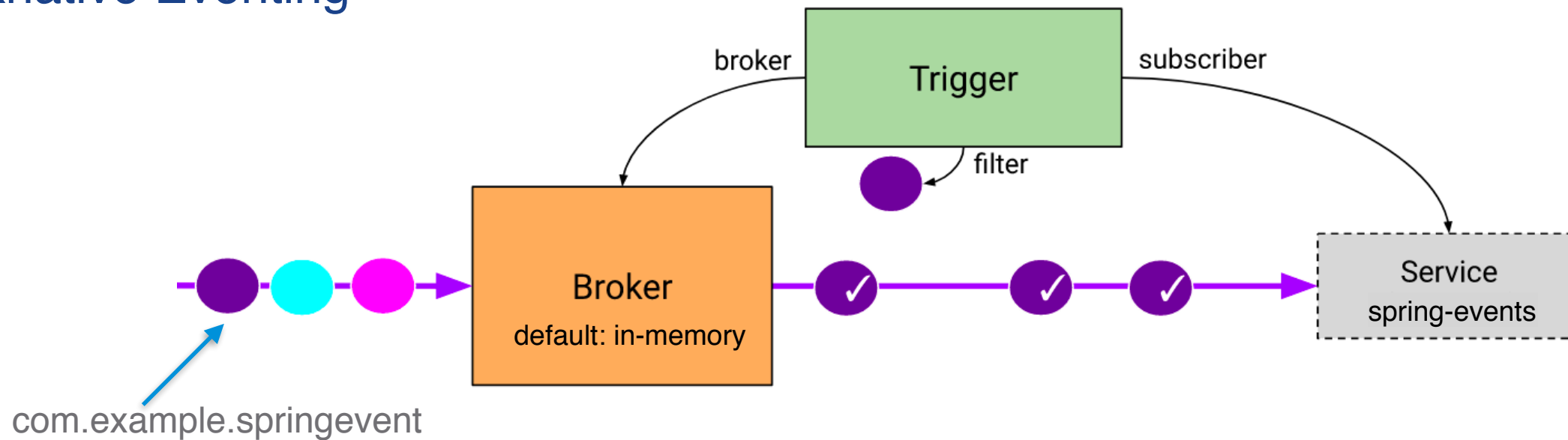
“Knative components build on top of Kubernetes, abstracting away the complex details and enabling developers to focus on what matters.”

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.

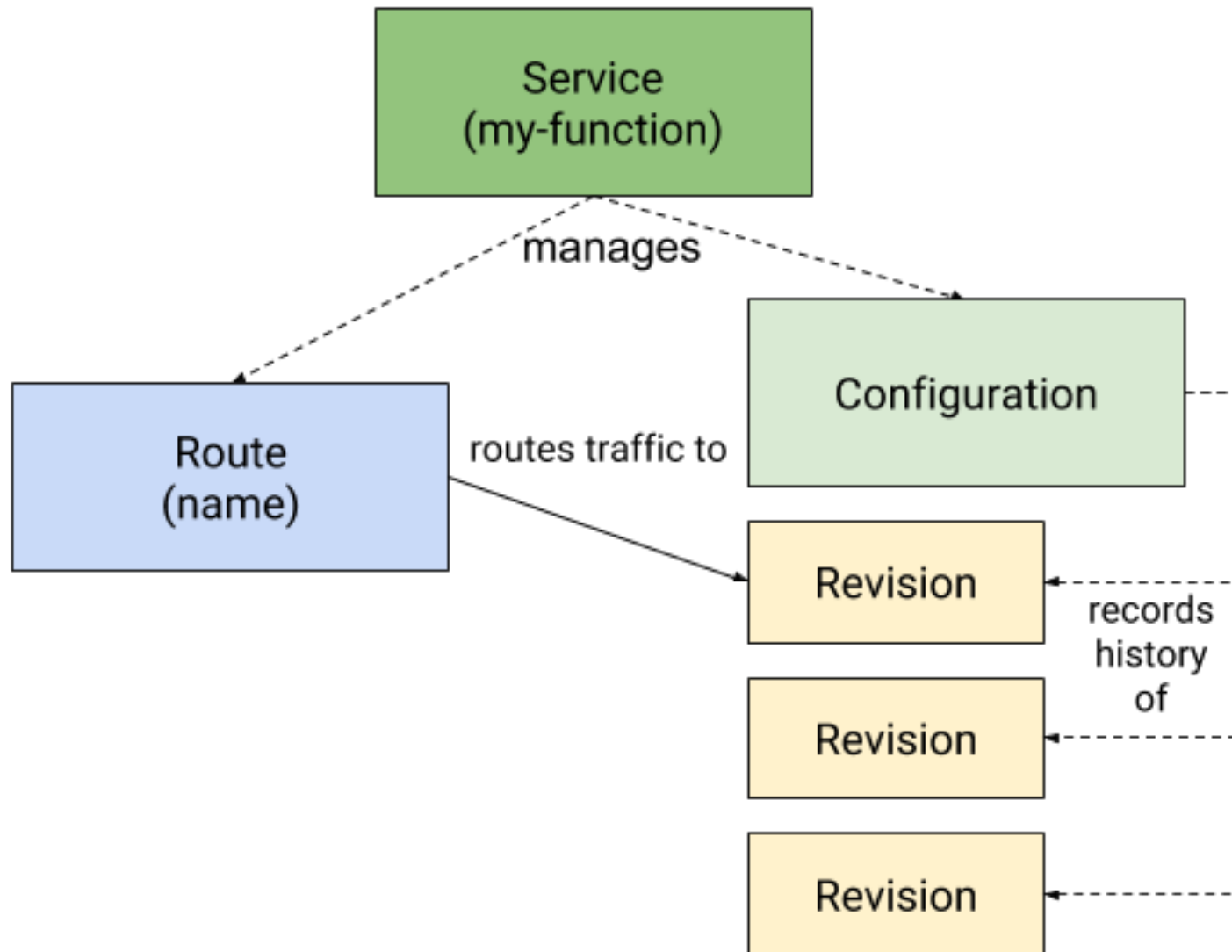


Knative Eventing



```
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
```

Knative Serving



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

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
}
```

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



Future Developments for Functions on Knative?

Knative Functions - Working Group Proposal



Reduce the friction of discovering, consuming, and producing events for developers using Knative Serving and Eventing.

Join Knative Users Group to follow this working group proposal.

Projectriff



Open source platform for building and running Functions, Applications, and Containers on Kubernetes and Knative Serving.

Provides Cloud Native Buildpack for adding streaming capable function invokers to Java and Node.js.

Uses the **kpack** to build images with “**projectriff/builder**”.

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();
    };
}
```

Step-by-step guide:

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

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>
```


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/>

Knative Users Group: <https://groups.google.com/forum/#!forum/knative-users>

Skaafold: <https://skaafold.dev/>

Projectriff: <https://projectriff.io>

Presentation

Step-by-step guide and slides: <https://github.com/trisberg/spring-knative-cloudevents-2020>



Thank You