Event-Driven Spring Workshop

Craig Walls
craig@habuma.com
Twitter: @habuma
GitHub: github.com/habuma/nfjs-samples

What to expect…

Event-Driven vs. Imperative Processing

Handling Events in Spring

What to expect…

Some live coding

Misteaks will be maid

There will be dogs

What we'll (try to) do...

11:00 - 12:30 :: Foundations

Introduction, Setting Up, Templates, Message-Driven Beans, RSocket

1:00 - 2:30 :: Going Higher

Spring Integration, Spring Cloud Stream, Spring Cloud DataFlow

All of this is tentative and flexible

Event-Driven Goals

React to events, no blocking

High throughput

Keep concerns separate

Focus on concerns, not infrastructure

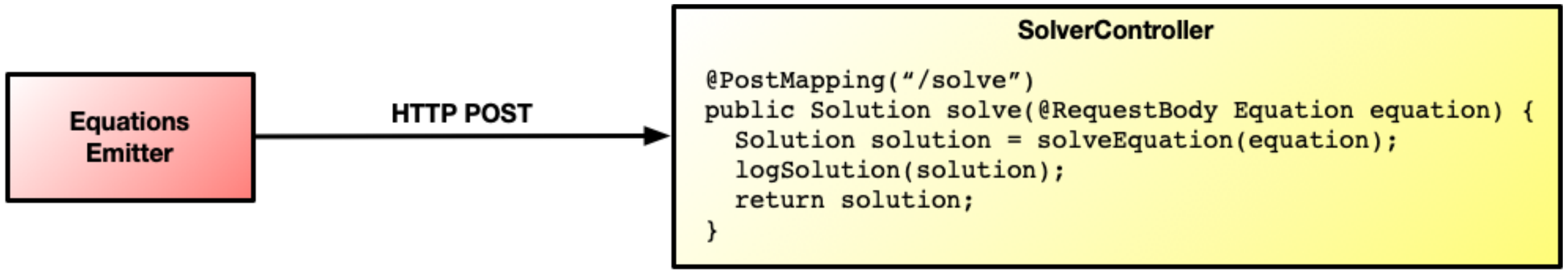
The Example “Problem”

Handle a message containing coefficients for a quadratic equation

Solve the equation

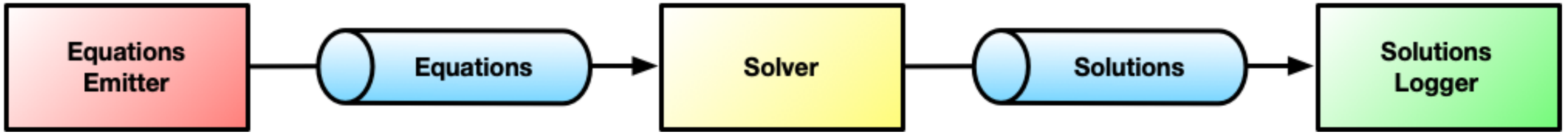
Log the solution

Non-Eventful Handling



Non-Eventful Handling

Eventful Handling



Event-Driven Options

Eventful Spring Options

Message templates

Message-driven beans

Spring Integration

Spring Cloud Stream

Spring Cloud DataFlow

Message Templates

Configuration...

```
spring:
  rabbitmq:
    host: my.rabbit.example.com
    port: 5672
    username: rabbituser
    password: l3tm31n
```

Message Templates

Sending messages…

```
@Autowired
RabbitTemplate rabbit;

...

public void sendSolution(Solution solution) {
  rabbit.convertAndSend("solutions", solution);
}
```

Message Templates

Receiving messages…

```
@Autowired
RabbitTemplate rabbit;

...

public Optional<Equation> receiveEquation() {
  Equation equation =
          (Equation) rabbit.receiveAndConvert("equations", -1);
  return Optional.ofNullable(equation);
}
```

Message Templates

Semi-blocking; polling-like in nature

No clear way to break processing into distinct operations

Configuration-coupled to Message Broker

JMS, RabbitMQ, or Kafka

Message Driven Beans

```
@RabbitListener(queues = "equations")
@SendTo("solutions")
public Solution handleMessage(Equation equation) {
  Solution solution = solver.solve(equation);
  return solution;
}
```

```
@RabbitListener(queues = "solutions")
public void handleMessage(Solution solution) {
  log.info("SOLUTION:  " + solution);
}
```

Message Driven Beans

Reactive; queue-watching is handled by framework

No clear way to break processing into distinct operations

Coupled to message broker; Both consumer and producer must agree on broker

JMS, RabbitMQ, or Kafka

Spring Integration

```
@Bean
public IntegrationFlow flow(
    ConnectionFactory connectionFactory,
    AmqpTemplate amqpTemplate,
    QuadraticSolver solver) {
  return IntegrationFlows
      .from(new AmqpMessageSource(connectionFactory, "equations"),
          c->c.poller(Pollers.fixedRate(1000)))
      .<Equation, Solution>transform(equation -> solver.solve(equation))
      .handle(Amqp.outboundGateway(amqpTemplate).routingKey("solutions"))
      .get();
}
```

Spring Integration

Reactive

Processing can be split into distinct operations

Loosely coupled to message broker

Events are not necessarily sourced from a Message broker

Spring Cloud Stream

```
@Bean
public Function<Equation, Solution> solve(QuadraticSolver solver) {
  return equation -> solver.solve(equation);
}
```

```
spring:
  cloud:
    stream:
      bindings:
        solve-in-0:
          destination: equations
          group: quadratic
        solve-out-0:
          destination: solutions
          group: quadratic
```

Spring Cloud Stream

```
@Bean
public Consumer<Solution> logSolution() {
  return solution -> {
    log.info("SOLUTION:   " + solution);
  };
}
```

```
spring:
  cloud:
    stream:
      bindings:
        logSolution-in-0:
          destination: solutions
          group: quadratic
```

Spring Cloud Stream

Reactive

Processing can be easily split into distinct operations

Message-broker is the binding

Events are not necessarily sourced from a Message broker

Spring Cloud DataFlow

Out-of-the-box applications

Sources

file
ftp
gemfire
gemfire-cq
http
jdbc
jms
load-generator
loggregator
mail
mongodb
mqtt
rabbit
s3
sftp
sftp-dataflow
syslog
tcp
tcp-client
time
trigger
twitterstream

Processors

aggregator
bridge
counter
filter
groovy-filter
groovy-transform
grpc
header-enricher
httpclient
image-recognition
object-detection
pmml
pose-estimation
python-http
python-jython
scriptable-transform
splitter
tcp-client
tensorflow
transform
twitter-sentiment

Sinks

cassandra
counter
file
ftp
gemfire
gpfdist
hdfs
jdbc
log
mongodb
mqtt
pgcopy
rabbit
redis-pubsub
router
s3
sftp
task-launcher-dataflow
tcp
throughput
websocket

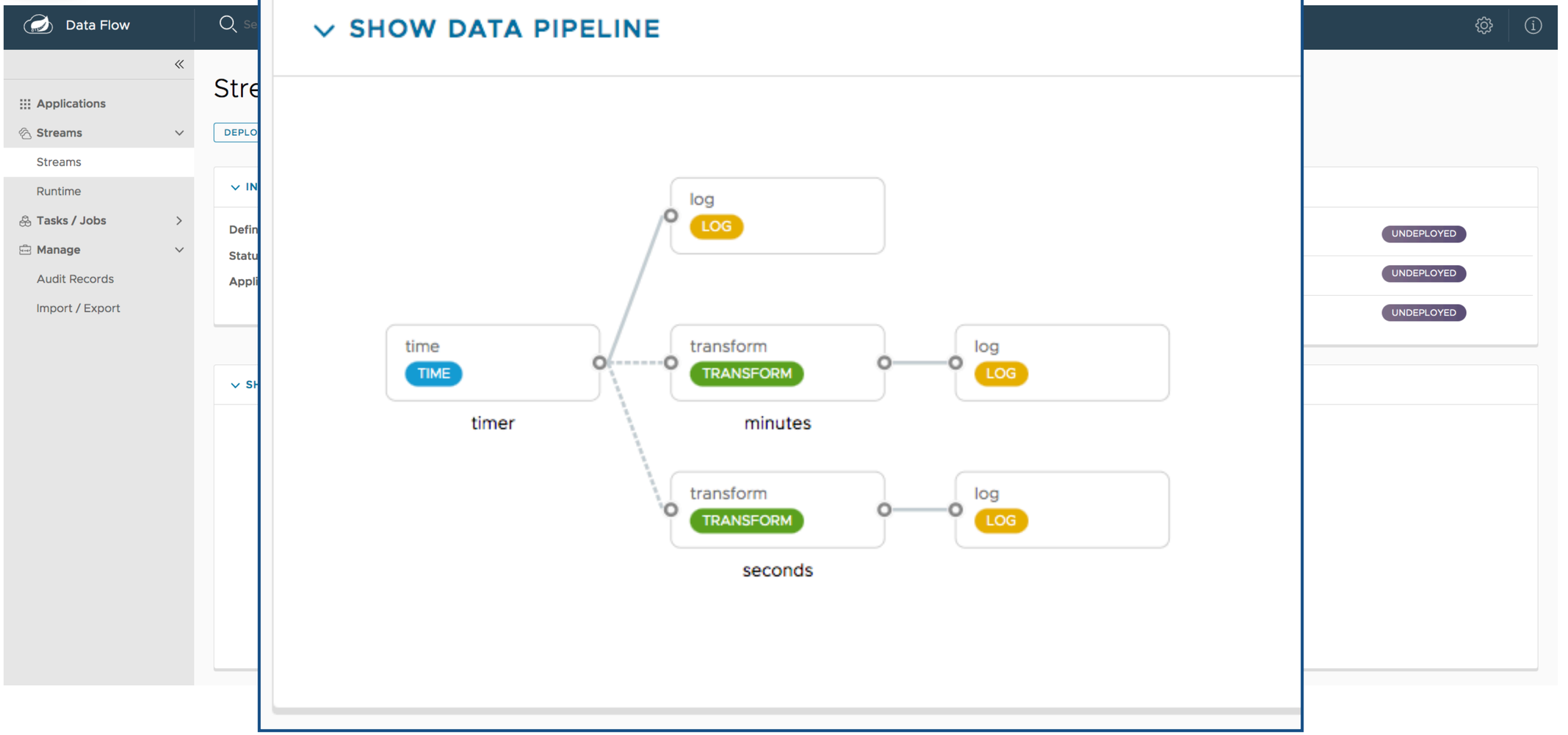
Tasks

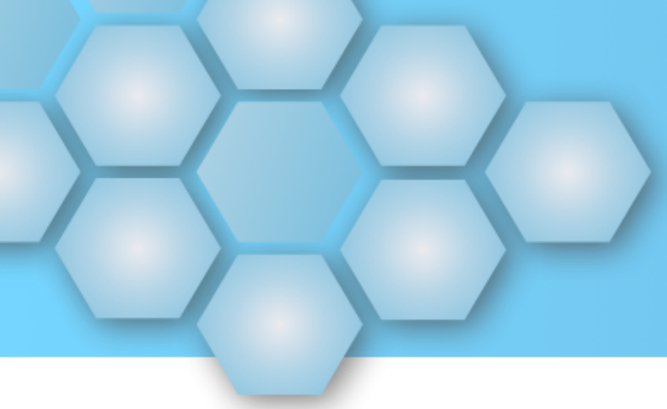
compound-task-runner
timestamp
timestamp-batch

Define streams with DSL

```
http --port=9000 | transform --expression 'hello-world' | log
```

Using the Dashboard





Spring Cloud DataFlow

Reactive

Processing can be easily split into distinct operations

Message-broker is the binding

Events are not necessarily sourced from a Message broker

A platform on which to build and deploy event-driven microservices

Thank you!

Check out my books:
http://www.habuma.com/sia5_nfjs.html
Discount code (50% off): tsspring