Do Auto and Manual Instrumentation In Apache SkyWalking

Wu Sheng

Apache SkyWalking original creator, PMC, Committer

Microsoft MVP

Wu Sheng

- What am I doing?
  - Apache SkyWalking(incubating)
  - SkyWalking multiple language agents
  - OpenSkywalking community
  - CNCF OpenTracing
  - W3C Trace Context Spec
  - Zipkin <-> SkyWalking integration
  - OpenCensus
  - Sharding-Sphere

SkyWalking APM

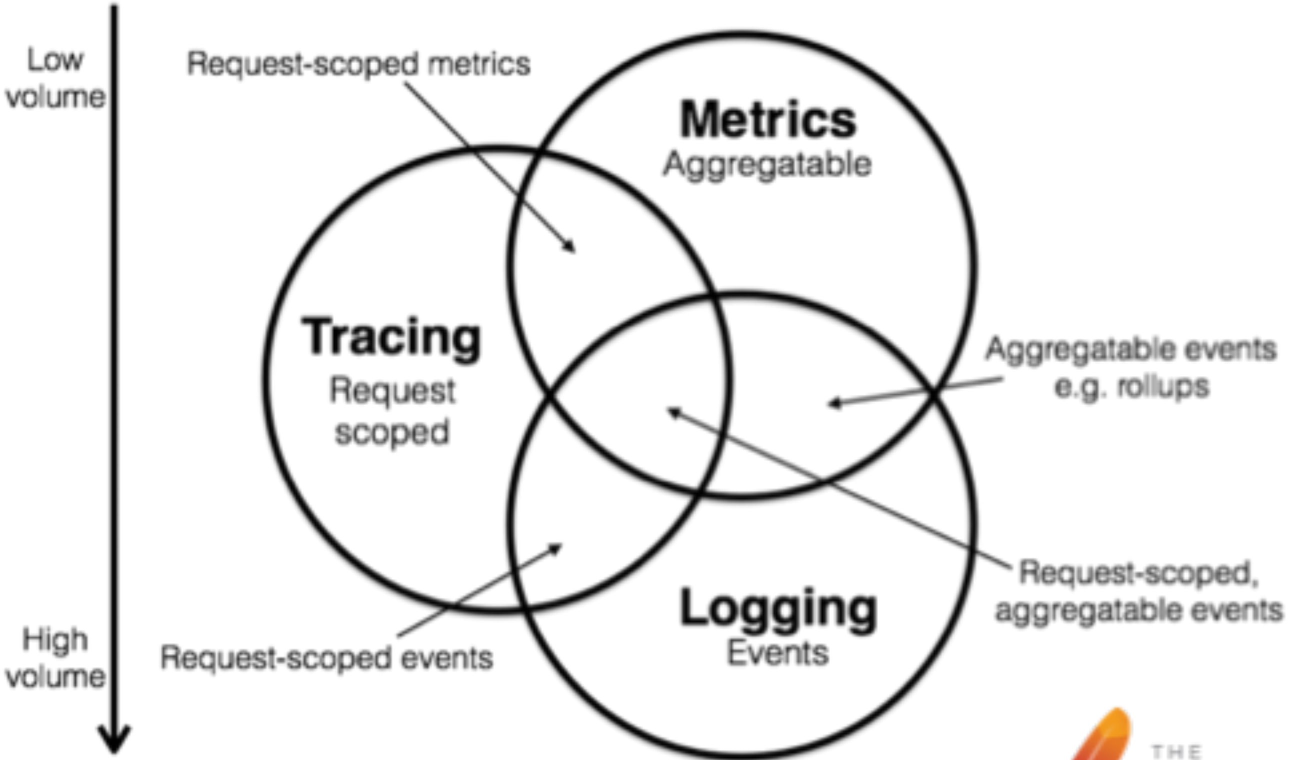
- Why create SkyWalking?
- Where did the name come from?

SkyWalking APM

- Why need auto instrumentation?
- Most commercial APMs are AUTO

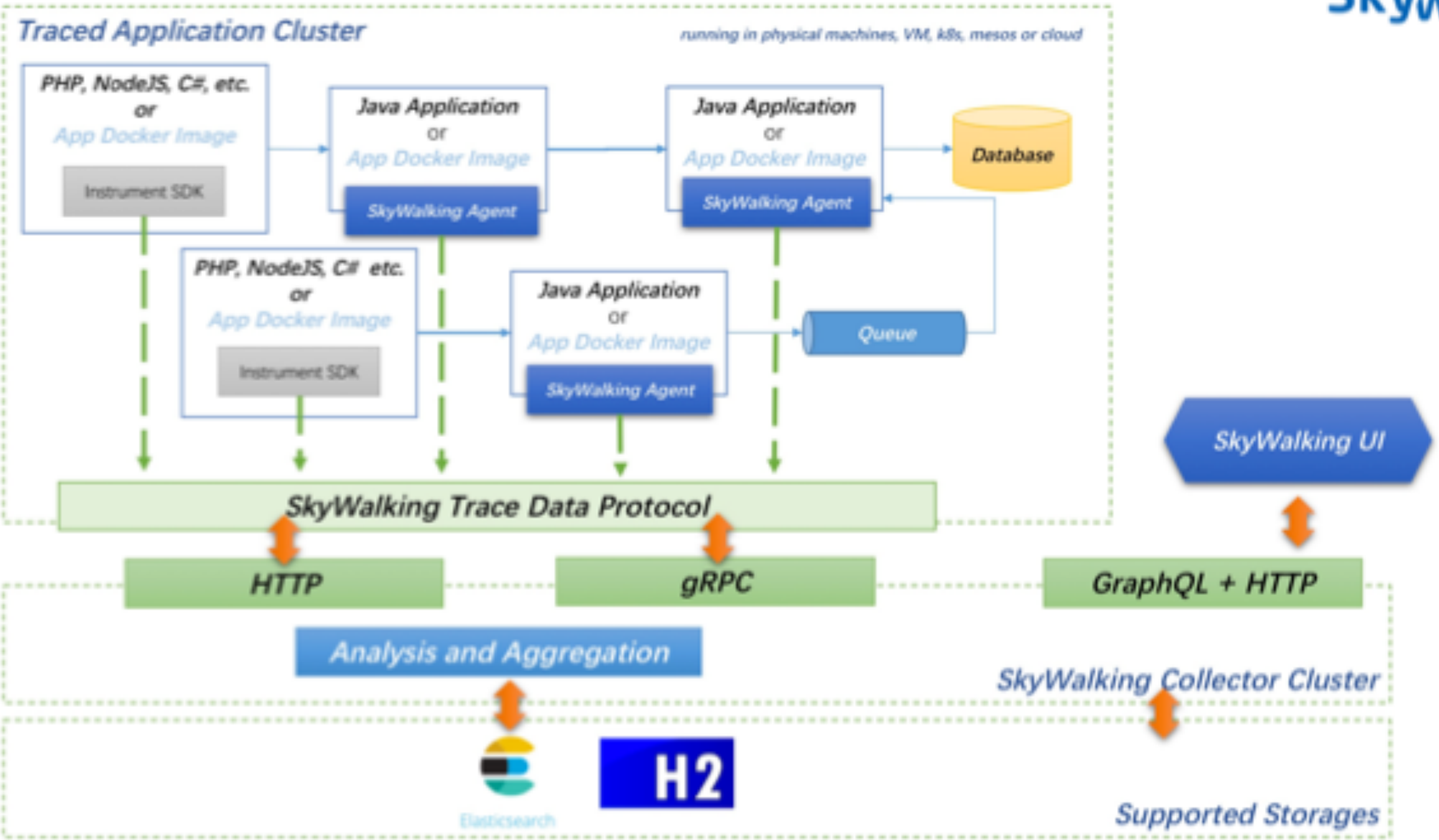
SkyWalking APM

- Differences between APM and tracing



How does SkyWalking work?

Skywalking



Why still need manual Instrumentation

- Instrumentation under conditions, to reduce performance payload
- Controlled by developers, rather than APM
- Across thread
  - Thread pool
  - Task assignment
  - Coroutine
  - Goroutine
- Low level transports, TCP

Manual instrumentation APIs

SkyWalking

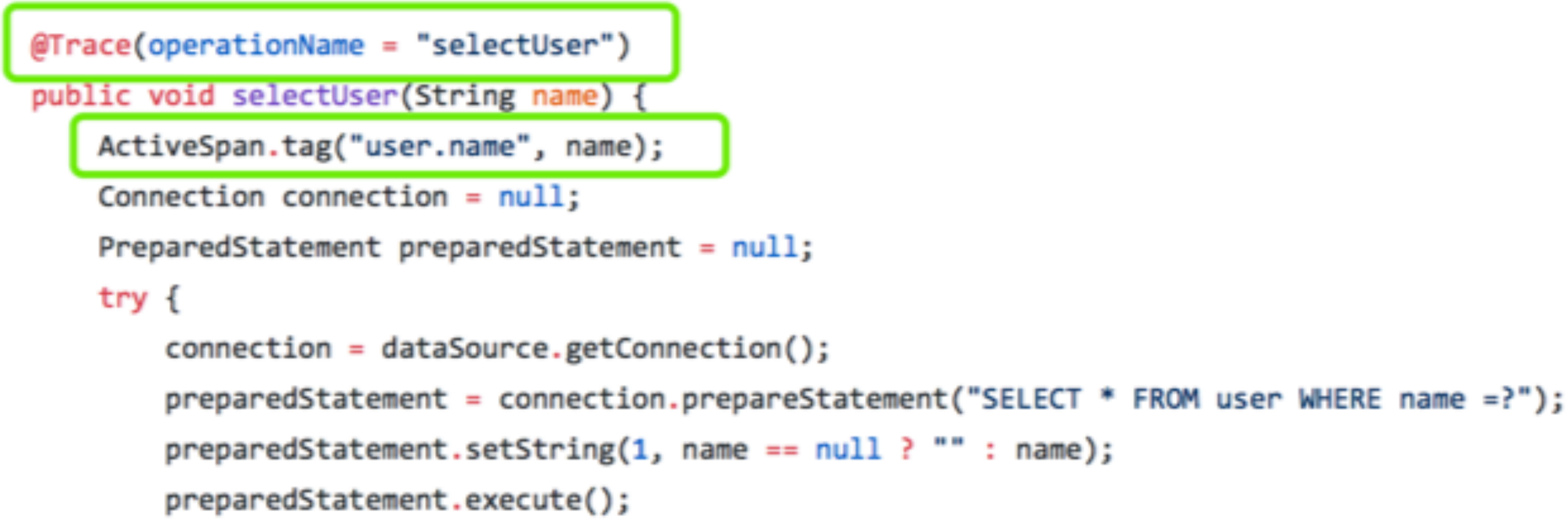
Manual Instrumentation Library

OpenTracing

Manual Instrumentation APIs

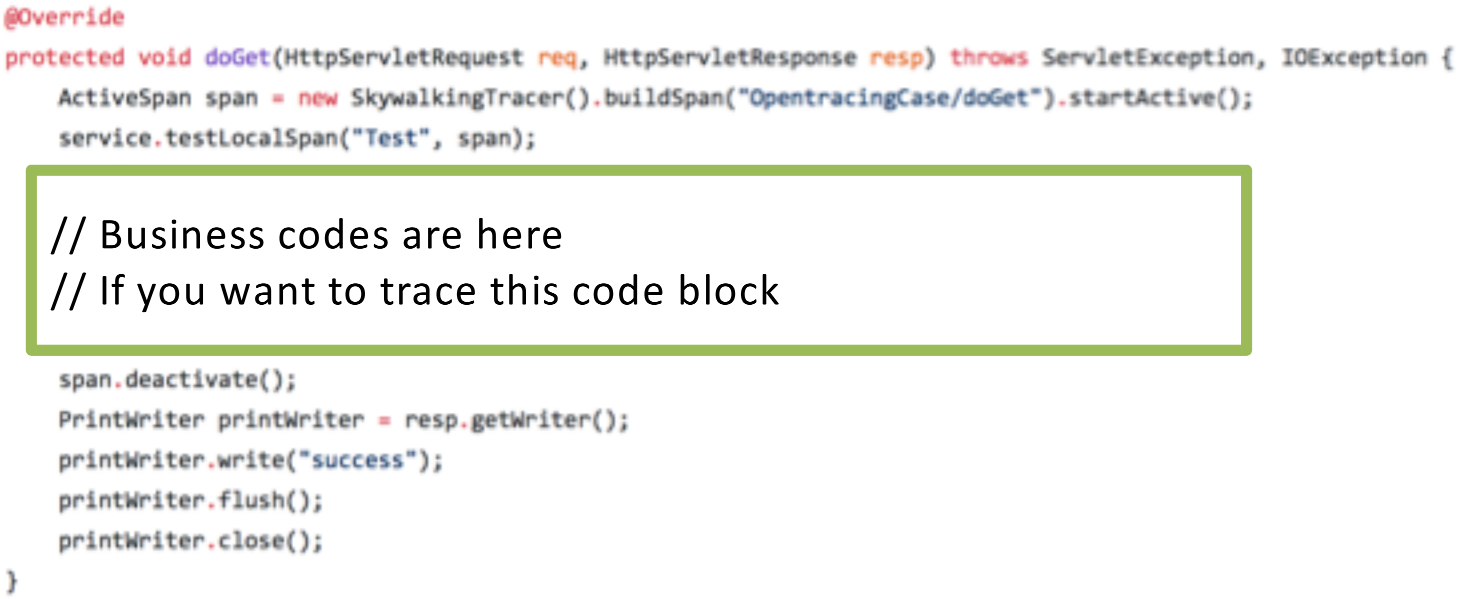
SkyWalking trace toolkit

- Trace any method with only annotation required
- Tag supported



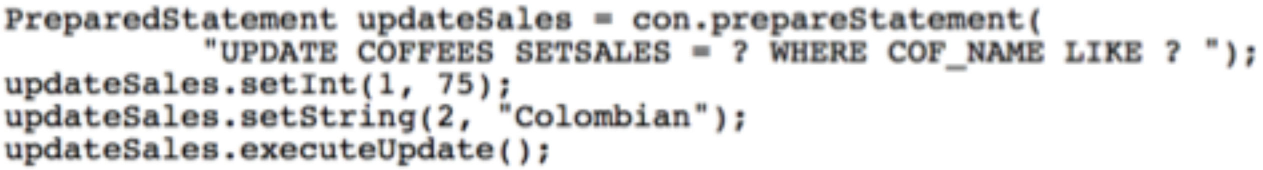
```
@Trace(operationName = "selectUser")
public void selectUser(String name) {
    ActiveSpan.tag("user.name", name);
    Connection connection = null;
    PreparedStatement preparedStatement = null;
    try {
        connection = dataSource.getConnection();
        preparedStatement = connection.prepareStatement("SELECT * FROM user WHERE name =?");
        preparedStatement.setString(1, name == null ? "" : name);
        preparedStatement.execute();
```

OpenTracing APIs bridge



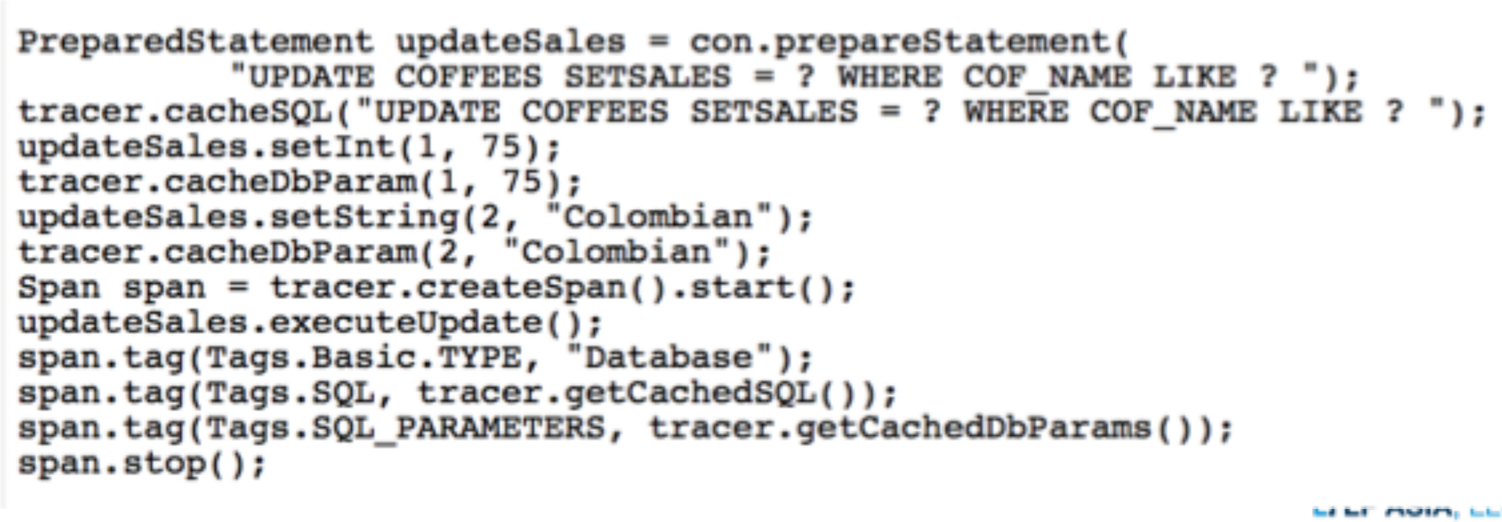
```java
@Override
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
    ActiveSpan span = new SkywalkingTracer().buildSpan("OpentracingCase/doGet").startActive();
    service.testLocalSpan("Test", span);

// Business codes are here
// If you want to trace this code block

    span.deactivate();
    PrintWriter printWriter = resp.getWriter();
    printWriter.write("success");
    printWriter.flush();
    printWriter.close();
}
```

How auto instrumentation works?



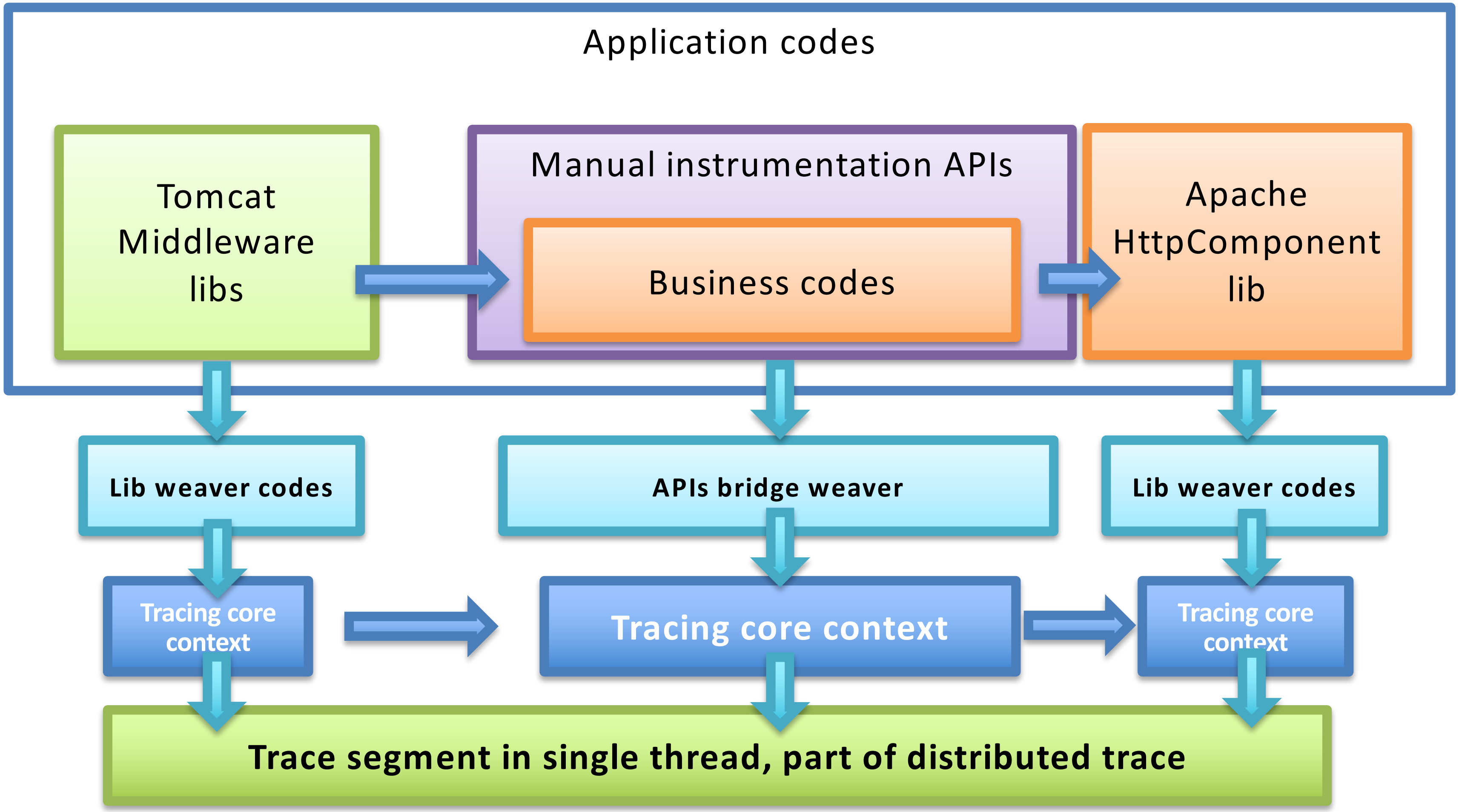
```
PreparedStatement updateSales = con.prepareStatement(
          "UPDATE COFFEES SETSALES = ? WHERE COF_NAME LIKE ? ");
updateSales.setInt(1, 75);
updateSales.setString(2, "Colombian");
updateSales.executeUpdate();
```

- **Running with Instrumentation, Pseudocode Only**
- Auto-instrumentation mechanism
    - AOP(Aspect Oriented Programming) in byte-code level.
    - Manipulate codes at runtime.



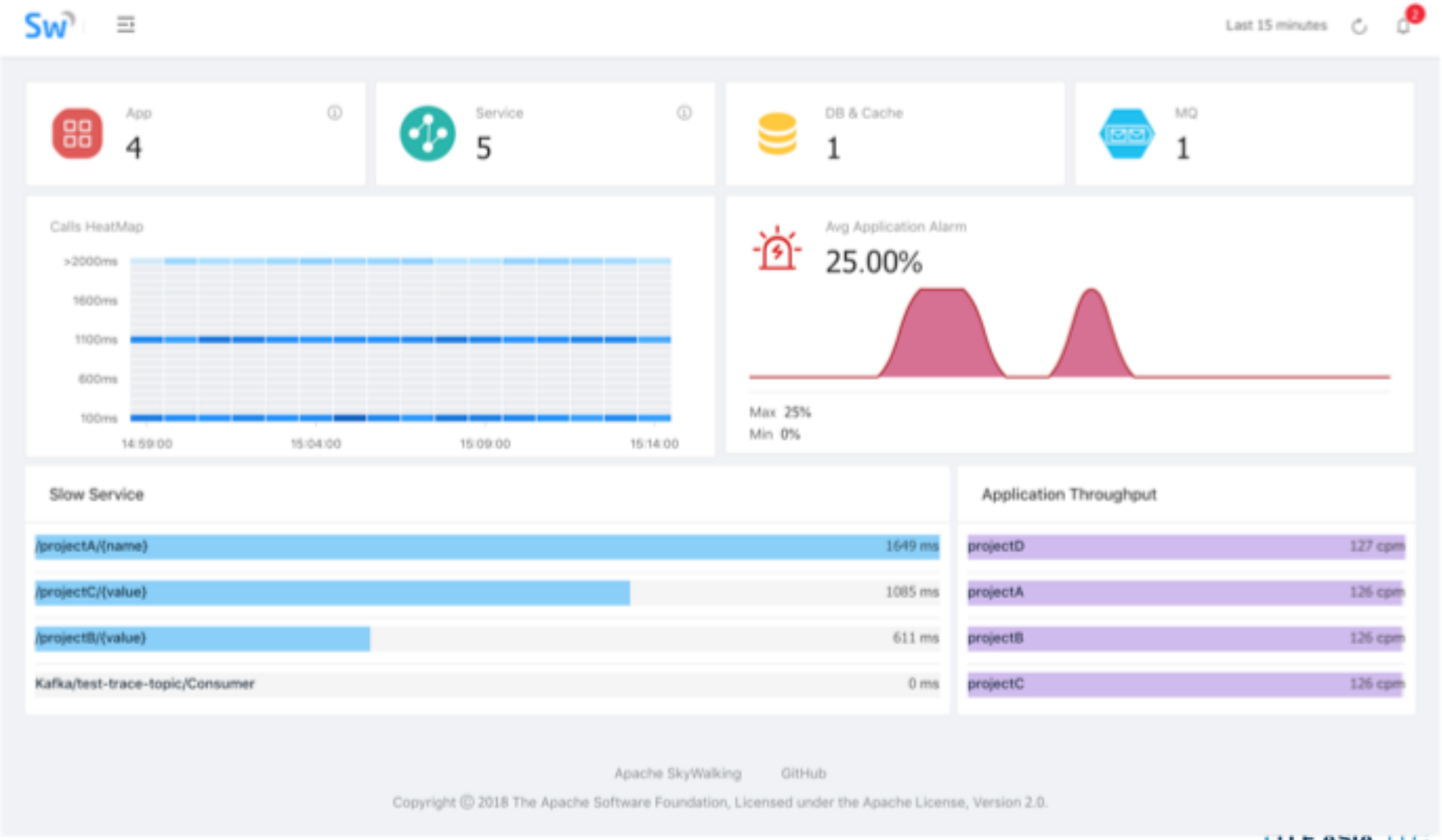
```
PreparedStatement updateSales = con.prepareStatement(
          "UPDATE COFFEES SETSALES = ? WHERE COF_NAME LIKE ? ");
tracer.cacheSQL("UPDATE COFFEES SETSALES = ? WHERE COF_NAME LIKE ? ");
updateSales.setInt(1, 75);
tracer.cacheDbParam(1, 75);
updateSales.setString(2, "Colombian");
tracer.cacheDbParam(2, "Colombian");
Span span = tracer.createSpan().start();
updateSales.executeUpdate();
span.tag(Tags.Basic.TYPE, "Database");
span.tag(Tags.SQL, tracer.getCachedSQL());
span.tag(Tags.SQL_PARAMETERS, tracer.getCachedDbParams());
span.stop();
```

Manual instrumentation on agent



APM from trace

- Overview the whole cluster



App 4

Service 5

DB & Cache 1

MQ 1

Avg Application Alarm

25.00%

Max 25%

Min 0%

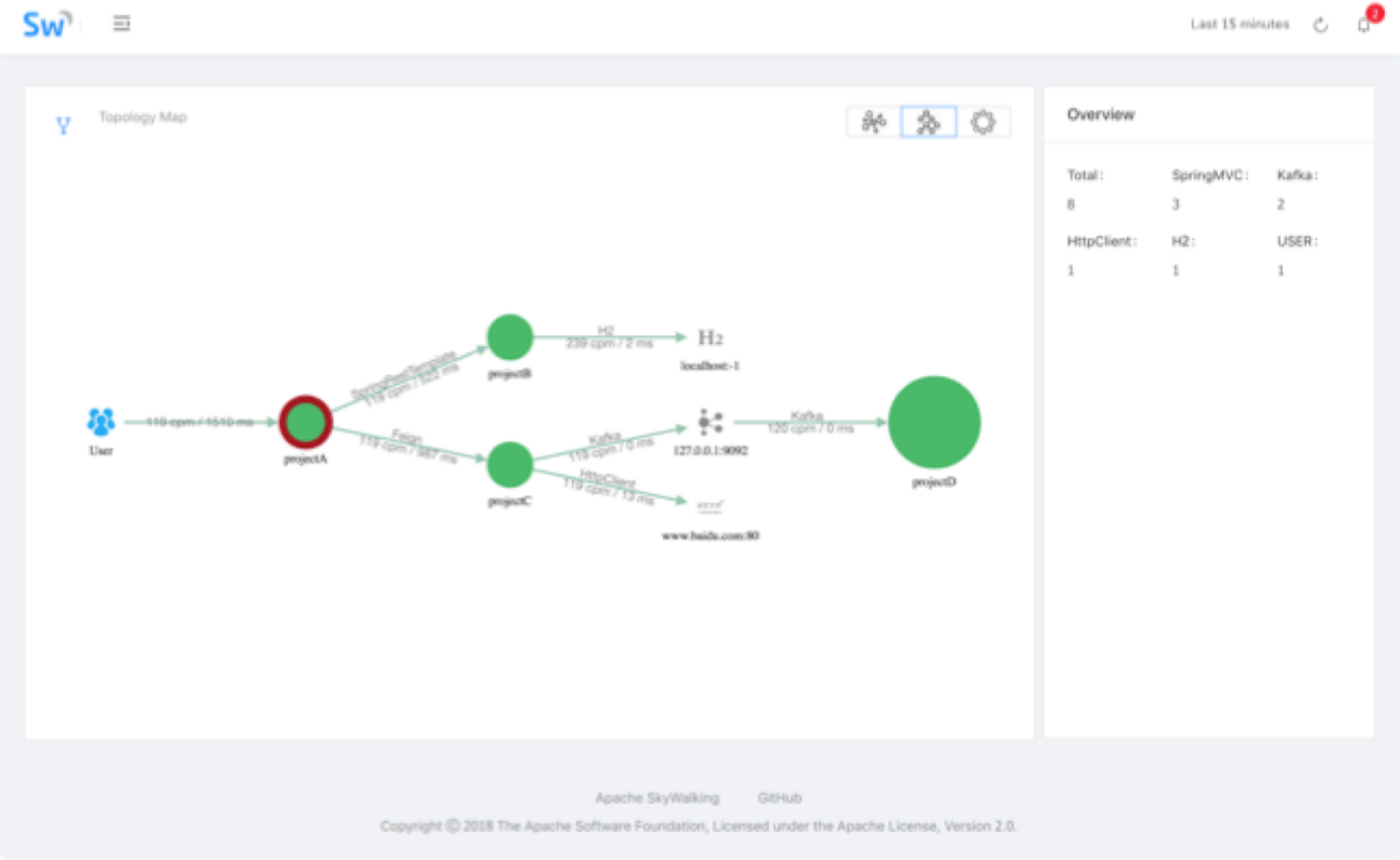
Slow Service	
/projectA/{name}	1649 ms
/projectC/{value}	1085 ms
/projectB/{value}	611 ms
Kafka/test-trace-topic/Consumer	0 ms

Application Throughput	
projectD	127 cpm
projectA	126 cpm
projectB	126 cpm
projectC	126 cpm

Apache SkyWalking GitHub

APM from trace

- Topology detected



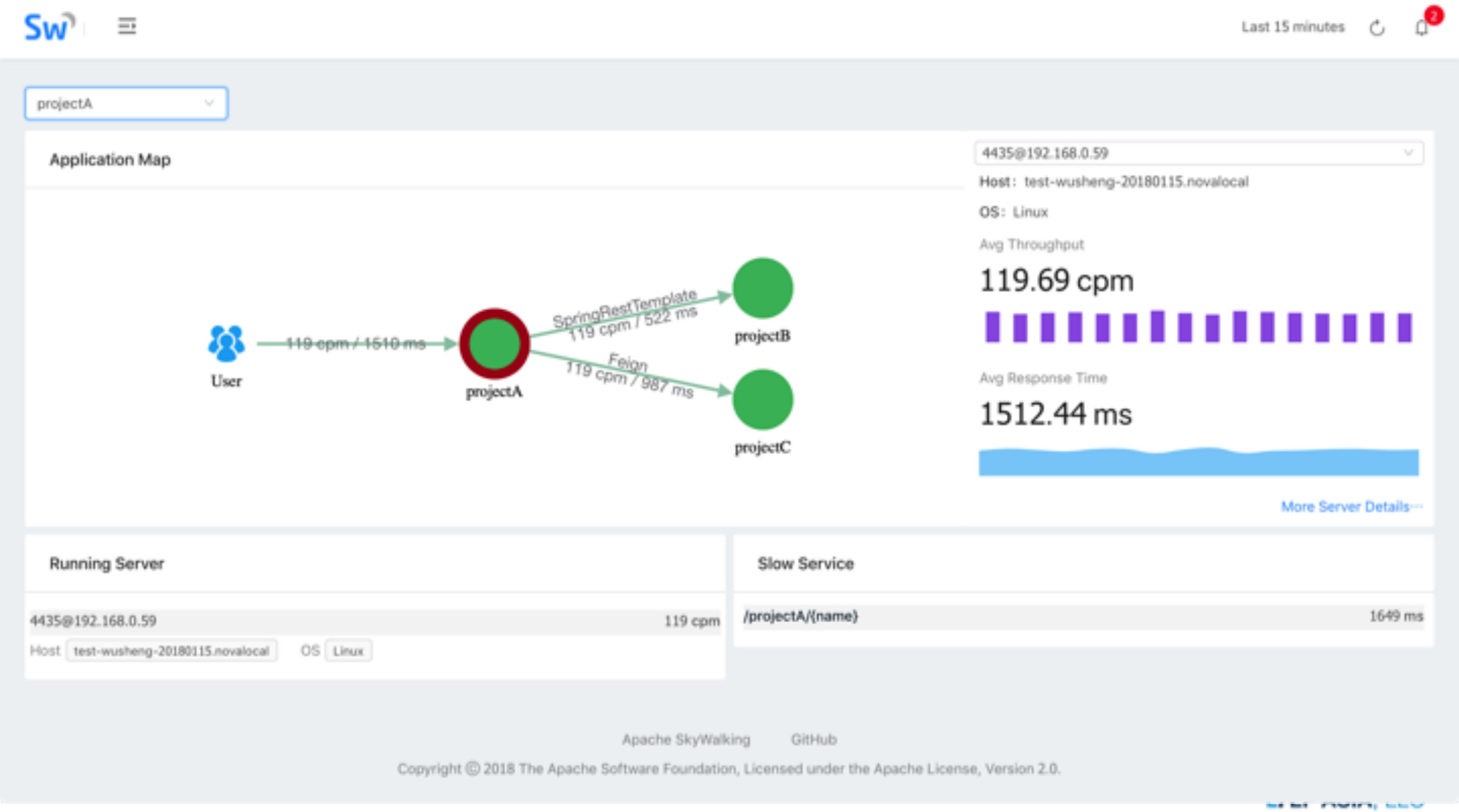
Overview

Total :	SpringMVC :	Kafka :
8	3	2
HttpClient :	H2 :	USER :
1	1	1

Apache SkyWalking GitHub

APM from trace

- Metrics of every application and its instance



4435@192.168.0.59

Host: test-wusheng-20180115.novalocal

OS: Linux

Avg Throughput

119.69 cpm

Avg Response Time

1512.44 ms

Running Server

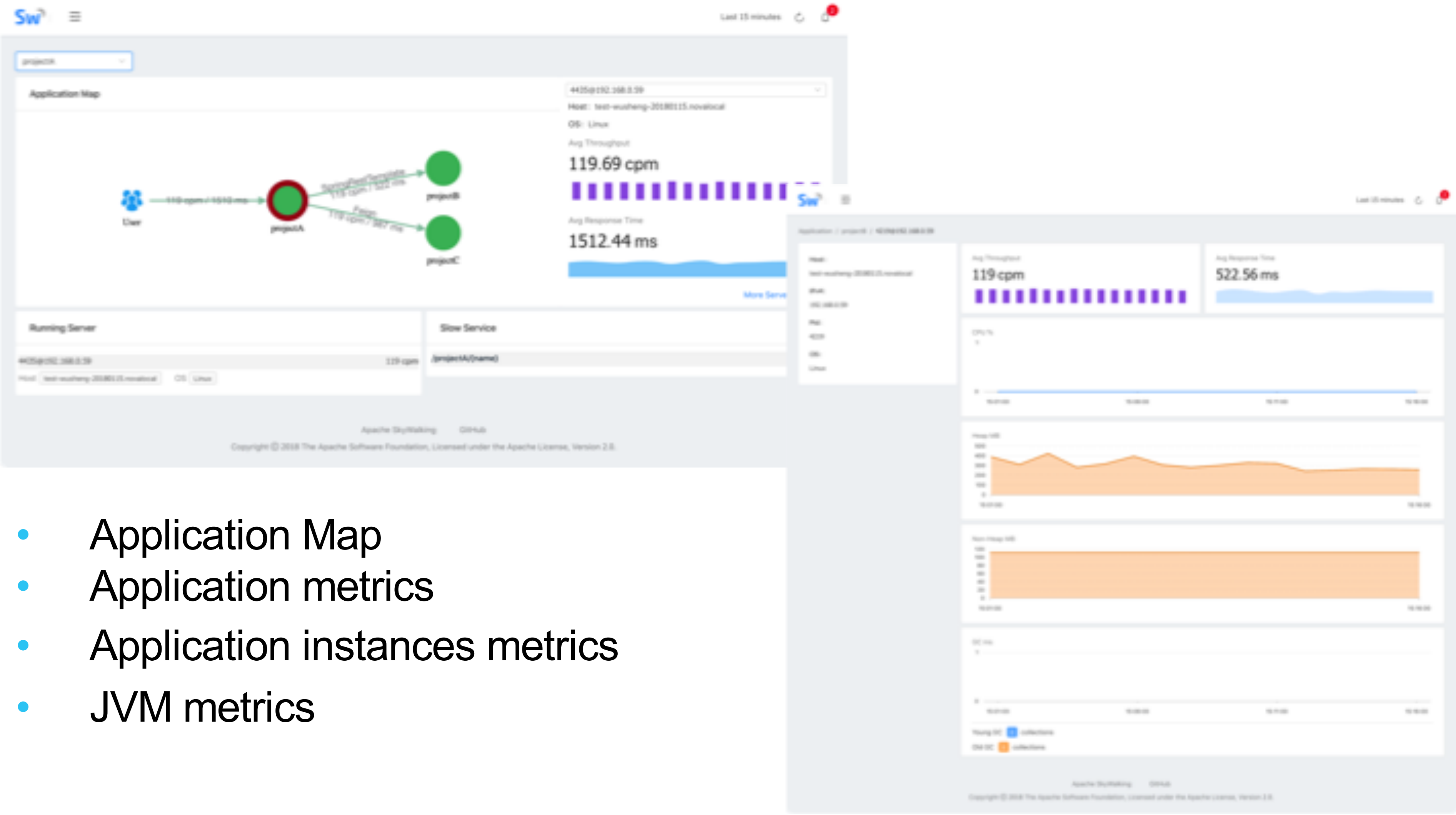
4435@192.168.0.59 119 cpm

Host test-wusheng-20180115.novalocal OS Linux

Slow Service

/projectA/{name} 1649 ms

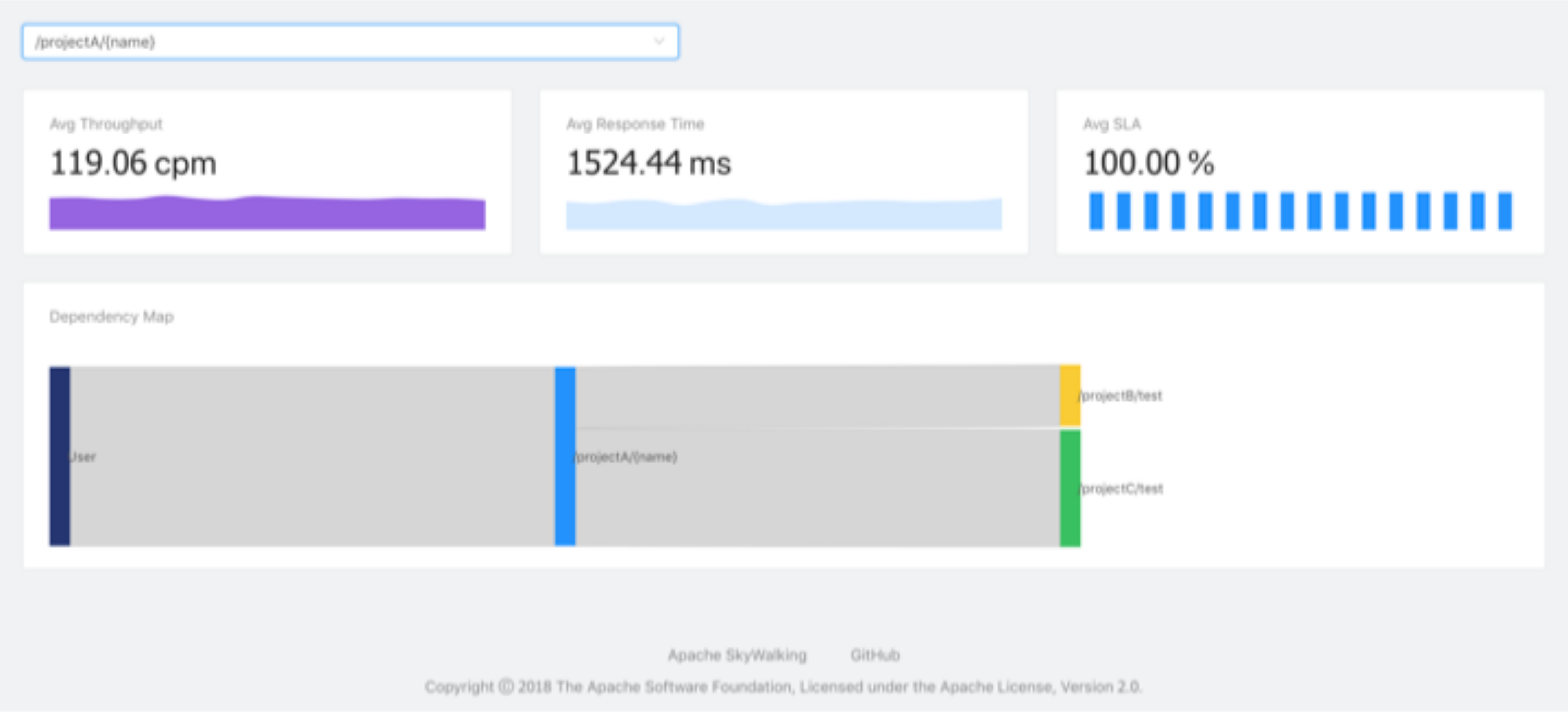
APM from trace



- Application Map
- Application metrics
- Application instances metrics
- JVM metrics

APM from trace

- Metrics of Rest and RPC services



/projectA/{name}

Avg Throughput
119.06 cpm

Avg Response Time
1524.44 ms

Avg SLA
100.00 %

Dependency Map

Apache SkyWalking   GitHub

Cool things in 5.x series

- Native metric data support. Traces become optional
- Zipkin -> SkyWalking collector -> visualization
- OpenCensus -> SkyWalking reporter

SkyWalking community

- GitHub
  - Java agent, collector, UI: https://github.com/apache/incubator-skywalking
  - .NET Core agent: https://github.com/OpenSkywalking/skywalking-netcore
  - Node.js server side agent: https://github.com/OpenSkywalking/skywalking-nodejs
- Gitter: https://gitter.im/OpenSkywalking/Lobby
- Mail list: dev@skywalking.apache.org

LINUXCON

containercon

CLOUDOPEN

CHINA 中国

THINK OPEN

开放性思维

Skywalking

LF ASIA, LLC