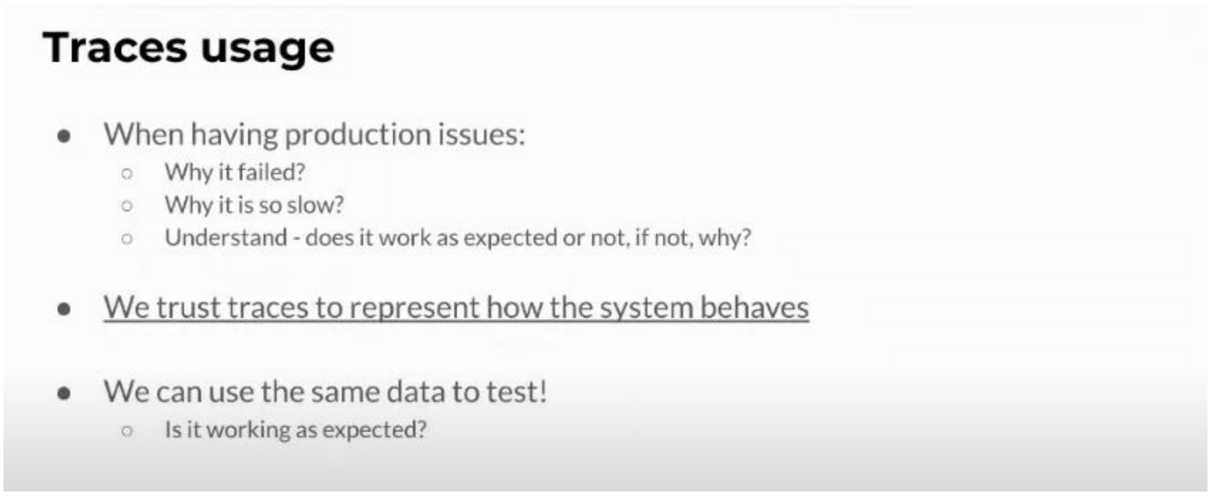
TESTING WITH OPENTELEMETRY

How we use TRACES?



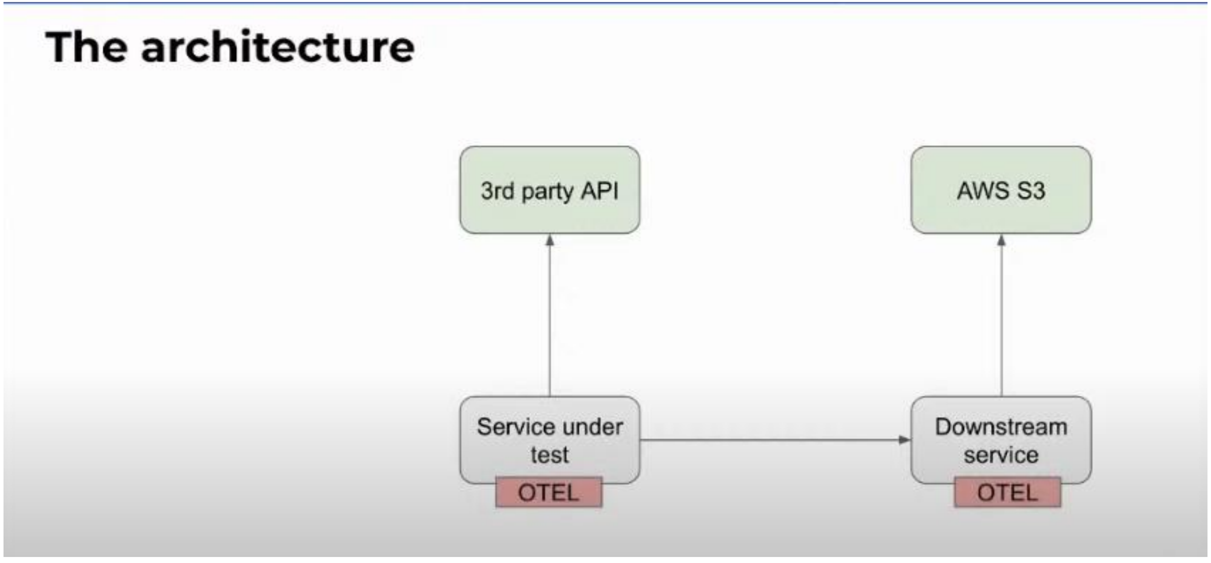
Traces usage

- When having production issues:
  - Why it failed?
  - Why it is so slow?
  - Understand - does it work as expected or not, if not, why?
- We trust traces to represent how the system behaves
- We can use the same data to test!
  - Is it working as expected?

Whenever we encounter any bug in production, a question arises, “why my application is behaving like this?”.

We look at traces to understand the issue. We trust this data to represent the system behaviour.

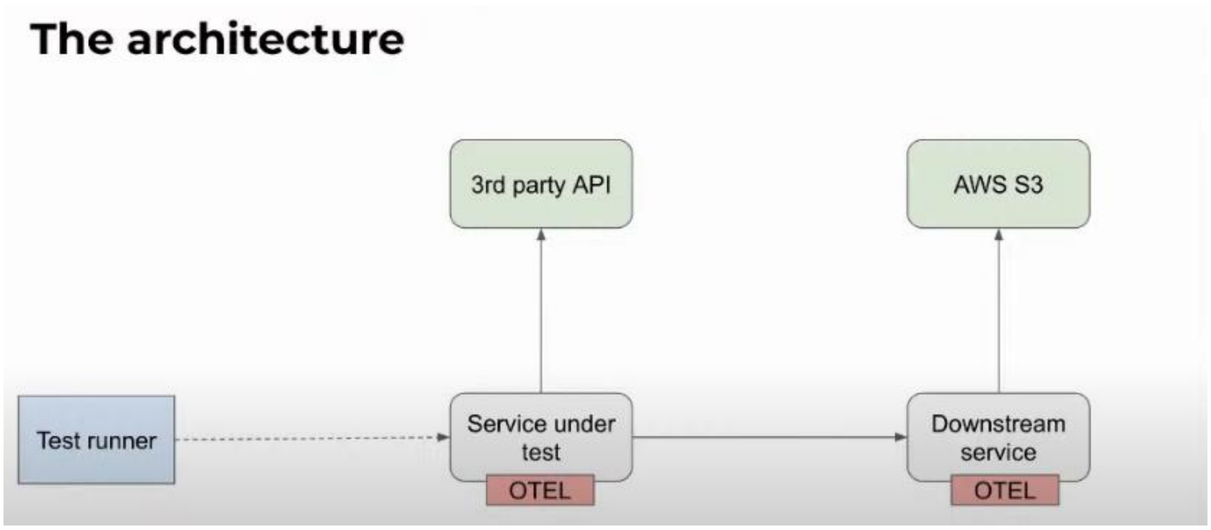
TRACE BASED TESTING



The architecture

We have service under test, to check whether it can be pushed to Production. And that service uses 1 downstream service or more and both have Otel installed.

The architecture

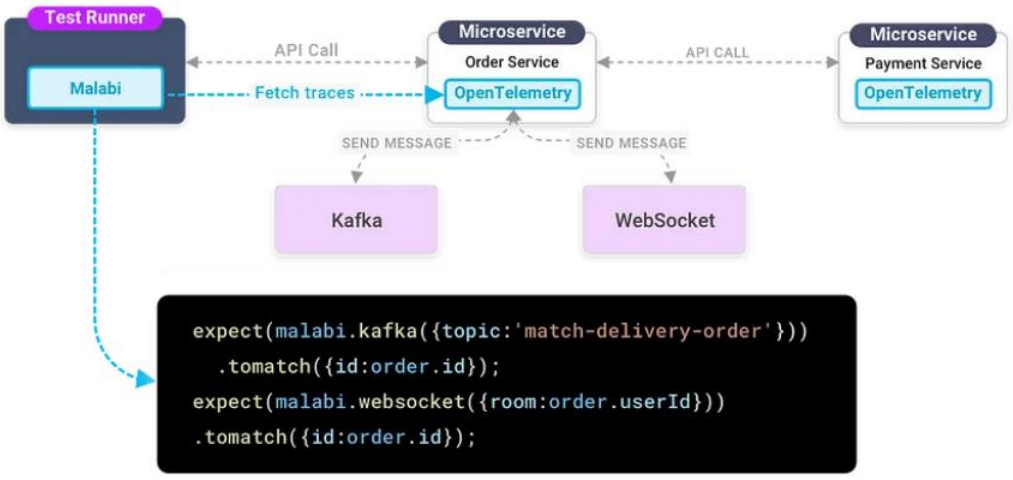


We have a test runner maybe sending some API or some UI that communicates with the service under test so.

Open Telemetry will come into play only when we deal with real data.

Mock data is not a valid testing way for open Telemetry.

For Trace Based Testing we can use “MALABI”



```
expect(malabi.kafka({topic:'match-delivery-order'}))
  .tomatch({id:order.id});
expect(malabi.websocket({room:order.userId}))
.tomatch({id:order.id});
```

Traces gathered in the OpenTelemetry SDK are retrieved via the Malabi library, which is incorporated into the Test Runner. To accomplish this, a custom exporter that saves the trace in memory is added to the OTEL SDK (the exporter is already given by Malabi) by using the Open Telemetry SDK or adding the custom exporter manually.

As of right now, Malabi can validate spans made by the service that is being tested (shown as the order service in the above diagram). We will also need to store those spans and start up some backend (OTEL Collector, Jaeger) to support downstream services and async message brokers.

We can simply gain a deeper understanding of the internal workings of the workflow by gathering the output traces from each test and providing them to the developer during the assertion phase.

MALABI USE CASES

- ➔ Internal Service Validation,
- ➔ Downstream Services,
- ➔ Messaging Systems,
- ➔ UI Testing
- ➔ Async Operations