Intelligent Cloud Operations
Part 4. Distributed Tracing Technologies

Definition (Gartner) [**AIOps**]

AIOps platforms utilize big data, modern machine learning and other advanced analytics technologies to directly and indirectly enhance IT operations (monitoring, automation and service desk) functions with proactive, personal and dynamic insight.

Prof. Jorge Cardoso
E-mail: jorge.cardoso@huawei.com
Intelligent Cloud Operations/SRE Dept.
Ireland and Munich Research Centers

2020

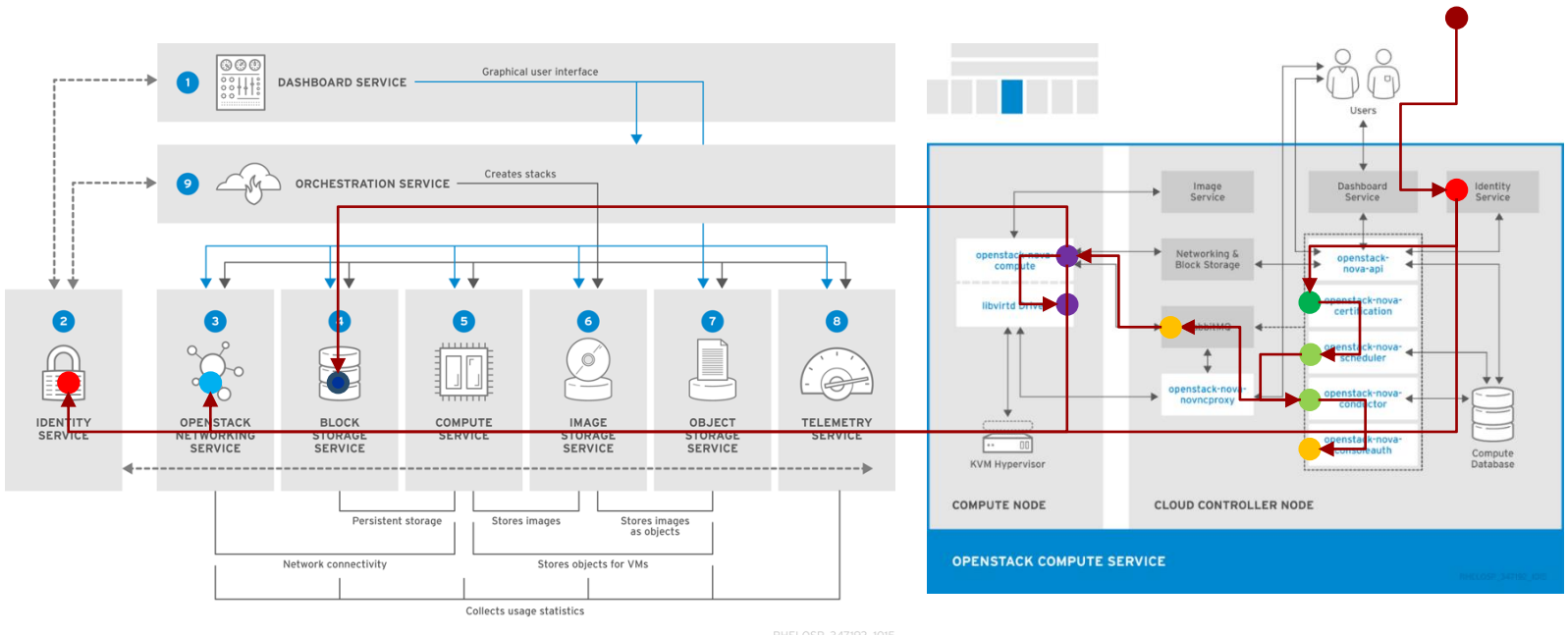
Technische Universität Berlin

HUAWEI

OpenStack Troubleshooting

Troubleshooting

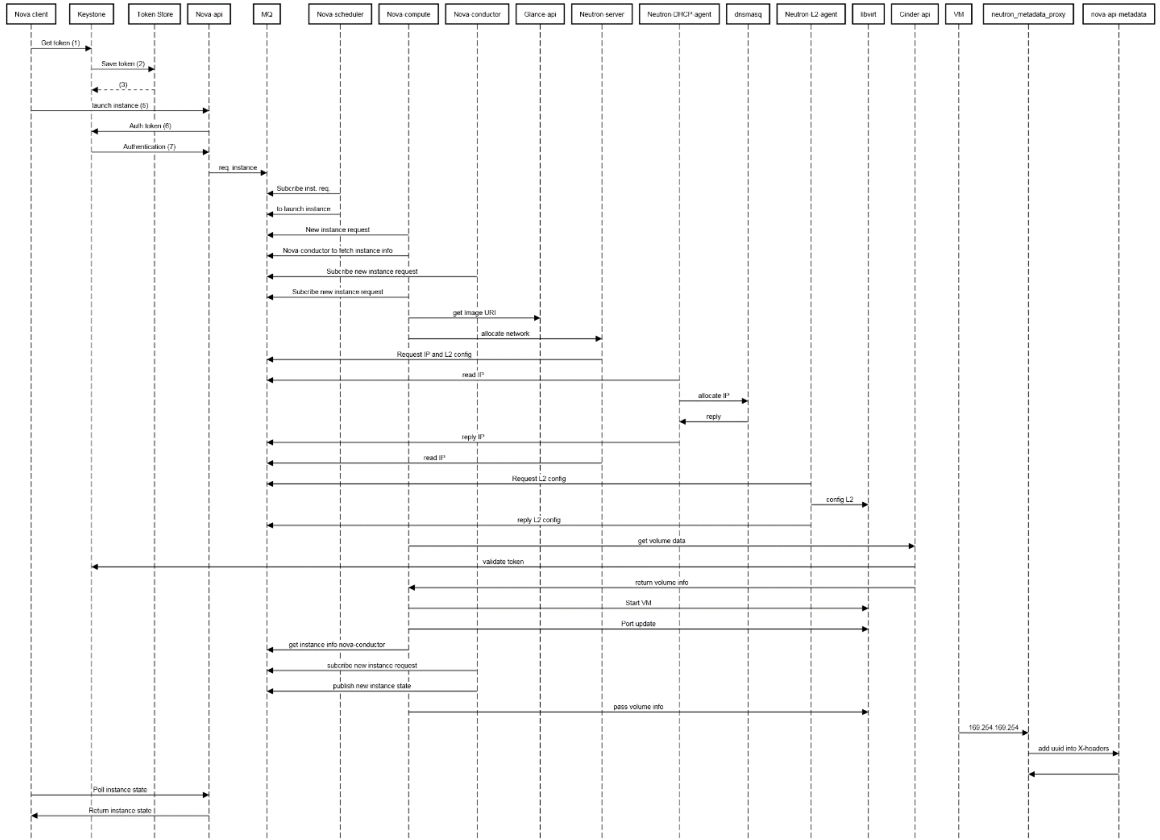
1. Find root cause issues in requests across several services / hosts
2. Benchmark different systems and identify performance bottlenecks
3. Reconstruct the workflow of service-to-service communications



https://access.redhat.com/documentation/en/red-hat-openstack-platform/8/paged/architecture-guide/chapter-1-components

Troubleshooting
Workflow for VM Creation

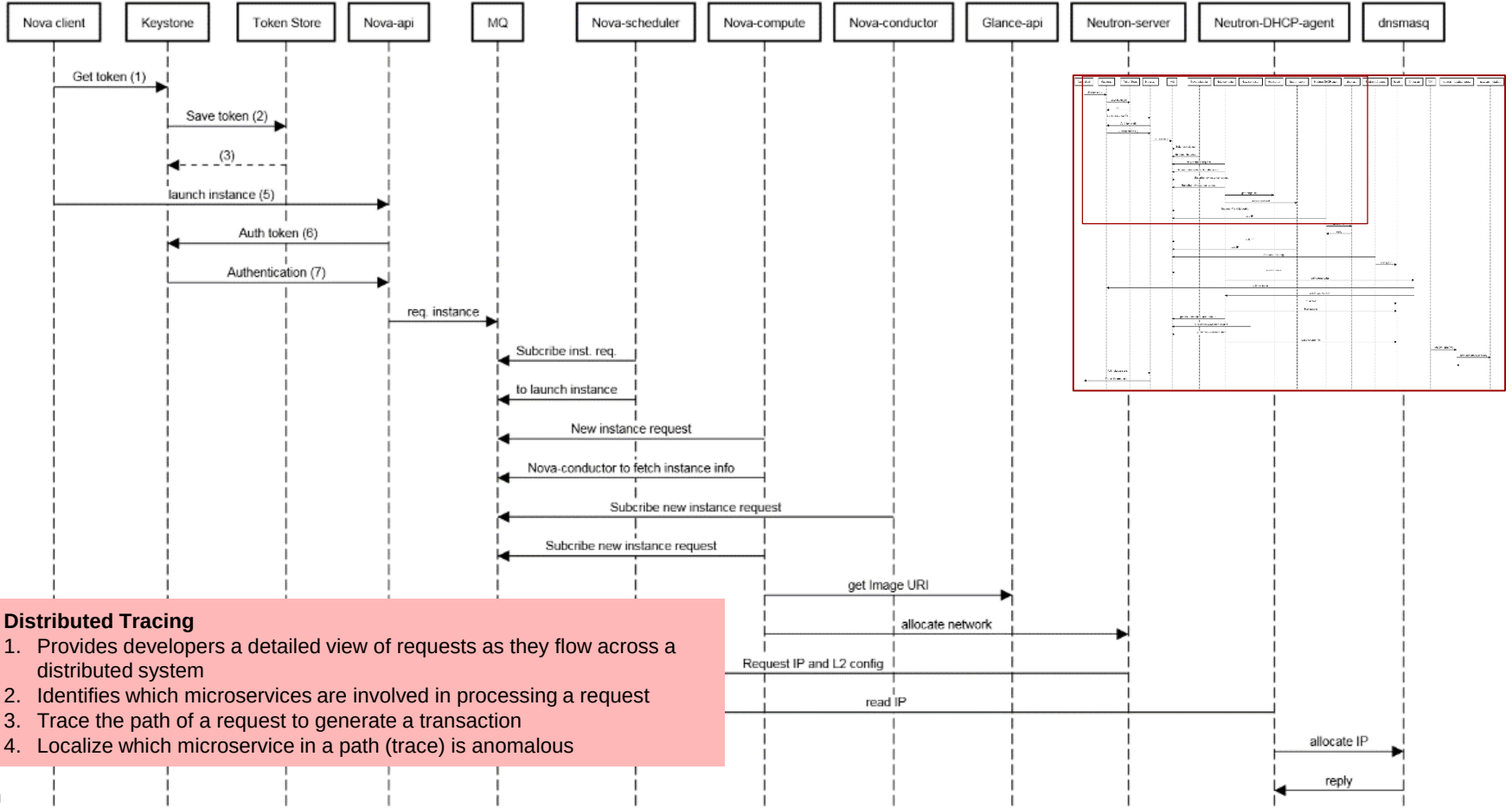
```
Nova client-> Keystone: Get token (1)
Keystone-> Token Store: Save token (2)
Token Store--> Keystone: (3)
Keystone-->Nova client: Auth token (4)

Nova client-> Nova-api: launch instance (5)
Nova-api-> Keystone: Auth token (6)
Keystone->Nova-api: Authentication (7)

Nova-api->MQ: req. instance

Nova-scheduler->MQ: Subcribe inst. req.
Nova-scheduler->MQ: to launch instance

Nova-compute->MQ: New instance request
Nova-compute->MQ: Nova-conductor to fetch instance info
Nova-conductor->MQ: Subcribe new instance request

Nova-compute->MQ: Subcribe new instance request

Nova-compute-> Glance-api: get Image URI
Glance-api->-Nova-compute: Return image URI

Nova-compute->Neutron-server: allocate network
Neutron-server->MQ: Request IP and L2 config

Neutron-DHCP-agent->MQ: read IP
Neutron-DHCP-agent->dnsmasq: allocate IP
dnsmasq->Neutron-DHCP-agent: reply
Neutron-DHCP-agent->MQ: reply IP

Neutron-server->MQ:read IP
Neutron-L2-agent->MQ: Request L2 config
Neutron-L2-agent->libvirt: config L2
Neutron-L2-agent->MQ: reply L2 config
Neutron-server->-Nova-compute: net info

Nova-compute->Cinder-api: get volume data
Cinder-api->Keystone: validate token
Keystone-->-Cinder-api: updated auth headers with roles and acl
Cinder-api->Nova-compute: return volume info

Nova-compute->libvirt: Start VM
Nova-compute->libvirt: Port update

Nova-compute->MQ: get instance info nova-conductor
Nova-conductor->MQ: subcribe new instance request
Nova-conductor->MQ: publish new instance state

Nova-compute->libvirt: pass volume info

VM->neutron_metadata_proxy: 169.254.169.254
neutron_metadata_proxy->nova-api-metadata: add uuid into X-headers
nova-api-metadata->neutron_metadata_proxy:
neutron_metadata_proxy->-VM-instance: return metadata

Nova client->Nova-api: Poll instance state
Nova-api-> Nova client: Return instance state
```



https://sequencediagram.org/

https://docs.openstack.org/install-guide/get-started-logical-architecture.html

Troubleshooting
Workflow for VM Creation



Distributed Tracing

1. Provides developers a detailed view of requests as they flow across a distributed system
2. Identifies which microservices are involved in processing a request
3. Trace the path of a request to generate a transaction
4. Localize which microservice in a path (trace) is anomalous

Tracing Services/Systems Concepts

Trace

- A transaction which captures the path that a request follows across a distributed system. It is **tree** or a directed acyclic graph (DAG) of **spans**

Span

- Spans represents an individual unit of work done in a distributed system. Spans are related to one another through a parent-child causal relationship.

Root Span

- The first span in a trace. The root span duration often represents the duration of the entire trace

Context propagation

- Span can be correlated together by propagating a context across microservices. The context contains a request id which identifies the trace to which it belongs.

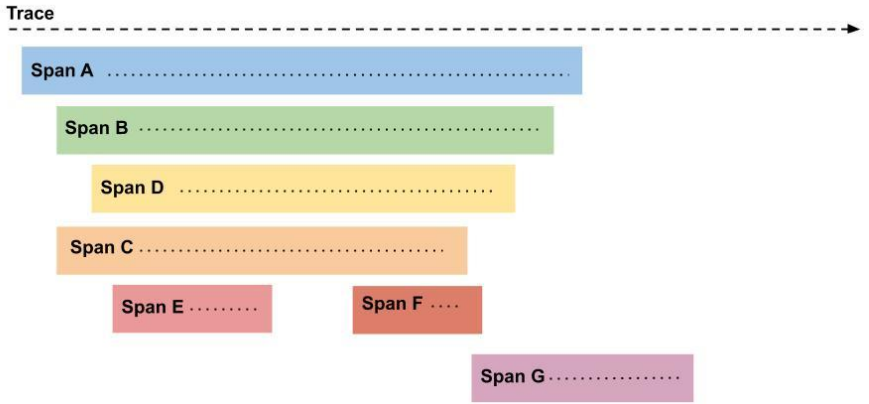


Figure. A trace is composed of spans. The root span is A. Spans B and C are children of span A.

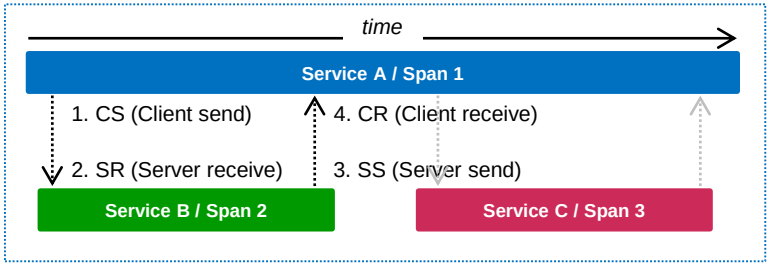
Span Content

- *Operation name*. API call that created the span
- *Start/finish timestamp*: Start and finish time of the operation
- *Tags*: Information injected into the span
- *SpanContext*: Span metadata transported across span boundaries

Tracing Services/Systems Concepts

Context Propagation

1. Create a new request_id when a user request is received by a service at the boundary of the distributed system
2. Store the request_id in a local context object and other metadata
3. Propagate the context across the distributed as the user request is processed
4. Service create spans and place key-value pairs describing the service/operation processing inside, along with the request_id



- **Client Send (CS)**: timestamp when client initiated the request
- **Server Receive (SR)**: timestamp when server receives the request
- **Server Send (SS)**: timestamp when server sends back the response
- **Client Receive (CR)**: timestamp when client receives back the response

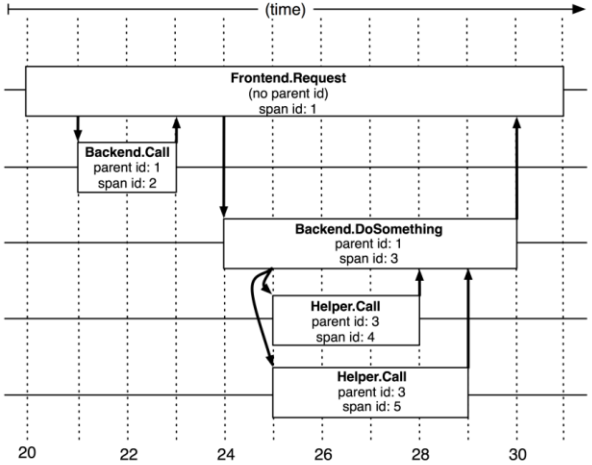


Figure. The causal and temporal relationships between five spans in a Dapper trace tree (from Dapper, a Large-Scale Distributed Systems Tracing Infrastructure)

Commercial/Open Source Solutions
Tracing Services Systems, and Standards

- **Tracing Services**
  - *Google Stackdriver (cloud.google.com/trace)*
  - *Amazon AWS X-Ray (aws.amazon.com/xray)*
  - Lightstep (lightstep.com)
- **Tracing Systems**
  - Twitter Zipkin (zipkin.io)
  - Uber Jaeger
  - OSProfiler
  - Pivot Tracing (pivottracing.io)
- **Tracing Standards**
  - opentelemetry.io
  - OpenTracing
  - OpenCensus

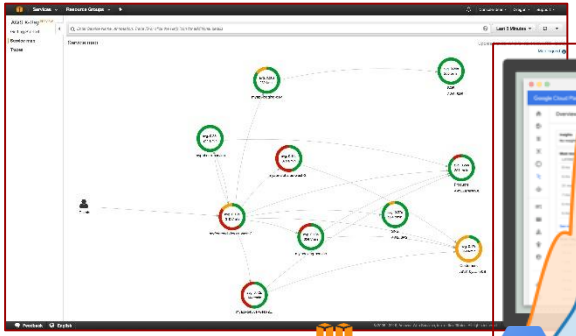


Fig. Amazon AWS X-Ray

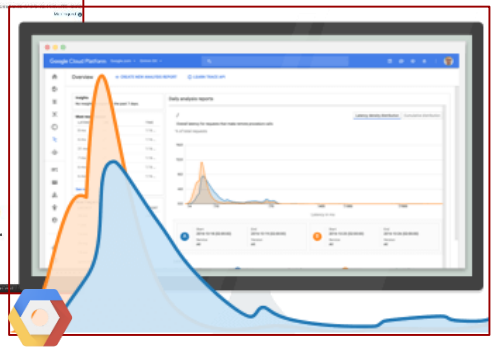


Fig. Google Cloud Trace

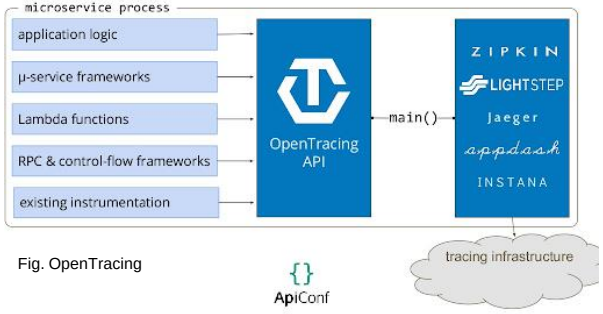
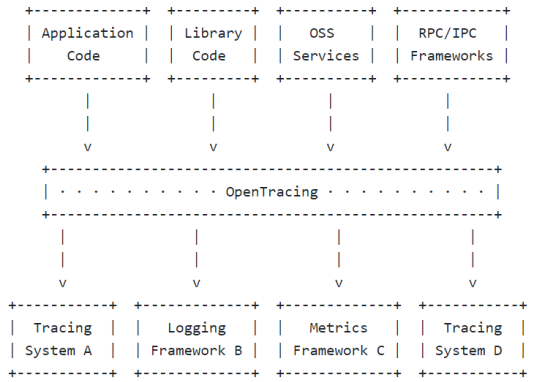


Fig. OpenTracing

OpenTracing provides a consistent, expressive, vendor-neutral APIs for popular platforms [1]

2019: The open source distributed tracing projects OpenCensus and OpenTracing were merged into anew project called OpenTelemetry



[1] http://opentracing.io/documentation/
[2] https://github.com/opentracing/specification/blob/master/specification.md

Tracing Services/Systems
Stackdriver, X-Ray, Jaeger

Stackdriver (Google) Characteristics

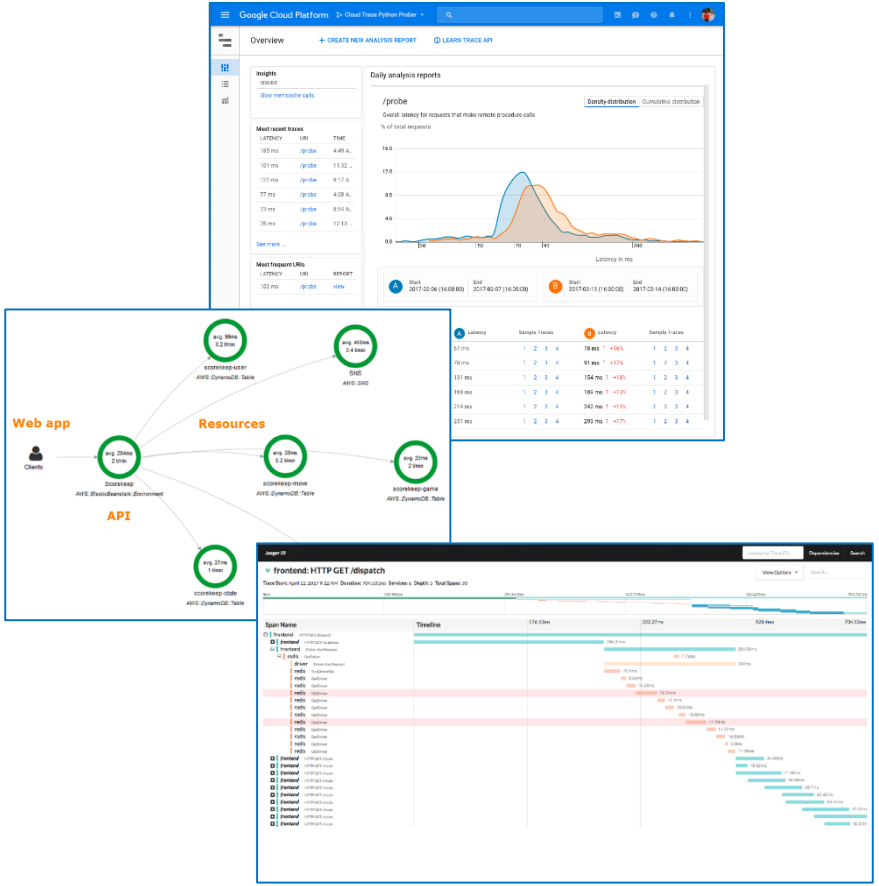
- Support for Python and gRPC
- Support Zipkin traces with forwarding to Stackdriver
- Easy to install: start Docker image on Compute Engine and view traces of a sample app within a few of minutes

X-Ray (AWS) Characteristics

- Support for OpenCensus but not OpenTracing
- Support for Python, Node, Java, and .NET apps (X-Ray SDK pushes metrics to a local collector).
- Sampling rates configuration based on different objects (e.g. service types like EC2 or Beanstock)

Jaeger (Uber) Characteristics

- Supports OpenTracing
- Send traces to a local agent via UDP, who sends traces to a collector
- Supports Cassandra for trace storage



Tracing Services/Systems
Zipkin

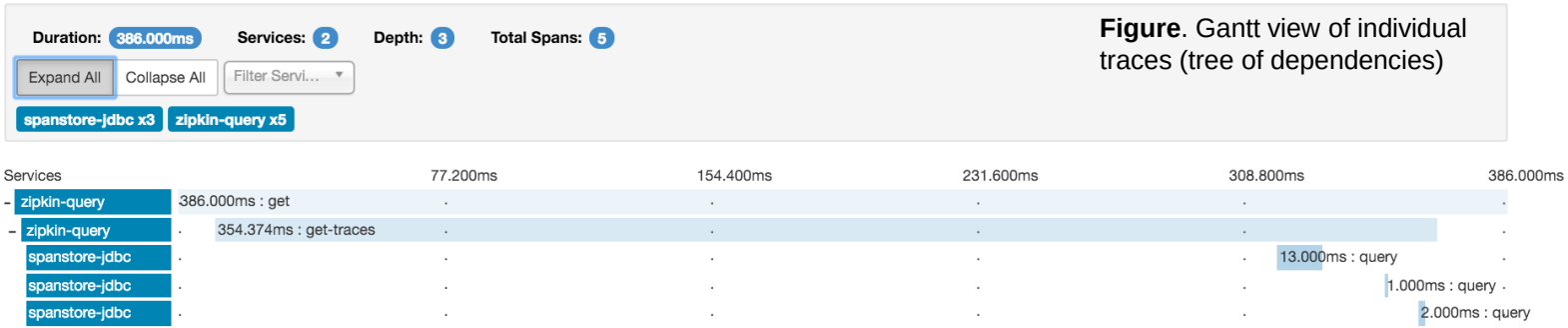


Figure. Gantt view of individual traces (tree of dependencies)

Zipkin Characteristics

- Supports Cassandra, ElasticSearch, and MySQL
- Implementations in Java, Go, JavaScript, Ruby, and Scala.
- Instrumented apps send data to a remote collector via HTTP, Kafka, and Scribe
- Python libraries: py_zipkin, pyramid_zipkin, swagger_zipkin, and flask-zipkin

Figure. *Example* of how to add instrumentation when using https://github.com/Yelp/py_zipkin

```
from py_zipkin.zipkin import zipkin_span

def some_function(a, b):
    with zipkin_span(
        service_name='my_service',
        span_name='my_span_name',
        transport_handler=some_handler,
        port=42,
        sample_rate=0.05, # Value between 0.0 and 100.0
    ):
        do_stuff(a, b)
```

Tracing Services/Systems
Jaeger/OpenTracing Key Constructs

Install Jaeger

```
$ docker run -d -p5775:5775/udp -p6831:6831/udp -p6832:6832/udp -p5778:5778 -p16686:16686 -p14268:14268 -p9411:9411 jaegertracing/all-in-one:0.8.0
$ pip install jaeger-client
```

1. Configure Tracer

```
import logging
from jaeger_client import Config

def init_tracer(service):
    logging.getLogger('').handlers = []
    logging.basicConfig(format='%(message)s',
                        level=logging.DEBUG)

    config = Config(
        config={
            'sampler': {
                'type': 'const',
                'param': 1,
            },
            'logging': True,
        },
        service_name=service,
    )

    # this call also sets opentracing.tracer
    return config.initialize_tracer()
```

2. Initialize Tracer

```
tracer = init_tracer('service-name')
```

3. Create Spans

```
with tracer.start_span('span-1') as span1:
    span1.set_tag('tag-1', '001')
    with tracer.start_span('span-2', child_of=span1) as span2:
        span2.set_tag('tag-2', '002')
```

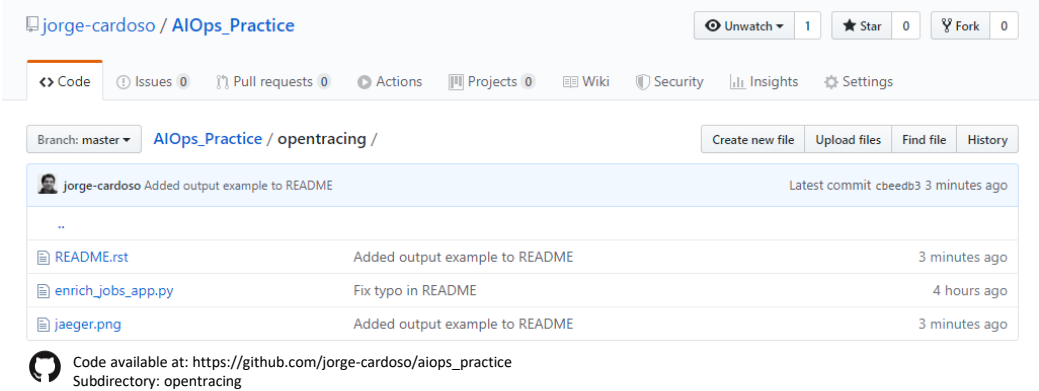
4. Tracing HTTP requests

```
with tracer.start_span('get-python-jobs') as span:
    homepages = []
    res = requests.get('https://jobs.github.com/'
                       'positions.json?description=python')
    span.set_tag('jobs-count', len(res.json()))
    for result in res.json():
        with tracer.start_span(result['company'],
                               child_of=span) as site_span:
            print('Getting website for %s' % result['company'])
            try:
                homepages.append(
                          requests.get(result['company_url']))
                site_span.set_tag('request-type', 'Success')
            except:
                print('Unable to get site for %s' %
                       result['company'])
                site_span.set_tag('request-type', 'Failure')
```


Example extracted from https://opentracing.io/guides/python/quickstart/

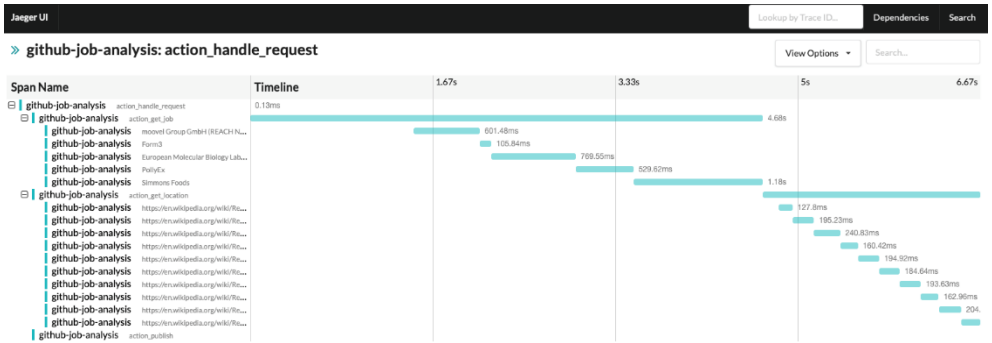
Tracing Services/Systems
Jaeger/OpenTracing Simple Application

1. GitHub example of using tracing



Code available at: https://github.com/jorge-cardoso/aiops_practice
Subdirectory: opentracing

3. Traces generated by the application



2. Output of the application

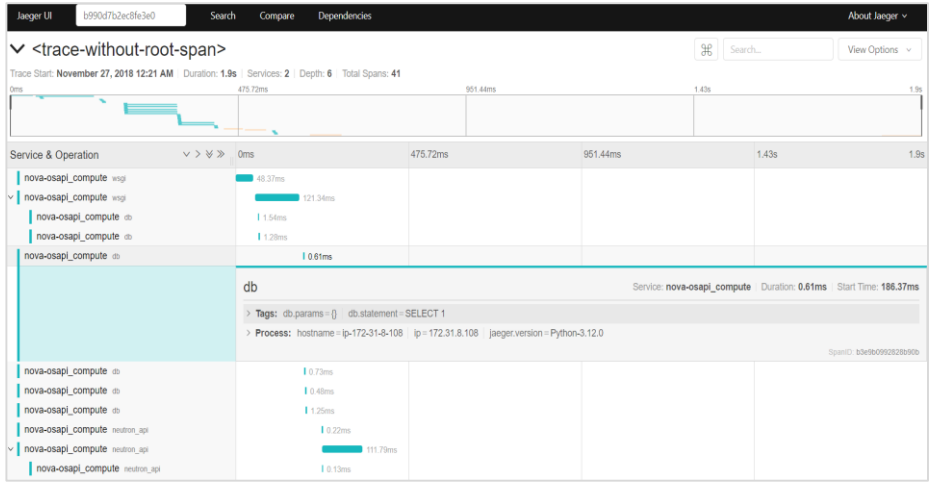
$ python enrich_jobs_app.py

Getting website for moovel Group GmbH (REACH NOW)….
Unable to get site for moovel Group GmbH (REACH NOW)
Getting website for Form3: https://form3.tech/
Getting website for European Molecular Biology …..
Getting website for PollyEx: https://www.pollyex.com/
Getting website for Simmons Foods: ……
Input to process: [('Form3', 'Remote'), ('European Molecular…
Getting link: https://en.wikipedia.org/wiki/Remote_control
Getting link: https://en.wikipedia.org/wiki/Remote_Desktop_Protocol
Getting link: https://en.wikipedia.org/wiki/Remote_sensing
Getting link: https://en.wikipedia.org/wiki/Remote_viewing
Getting link: https://en.wikipedia.org/wiki/Remotely.....
Getting link: https://en.wikipedia.org/wiki/Remote_Desktop_Services
Getting link: https://en.wikipedia.org/wiki/Remote_control_animal
Getting link: https://en.wikipedia.org/wiki/Remote_procedure_call
Getting link: https://en.wikipedia.org/wiki/Remote_keyless_system

https://github.com/jorge-cardoso/aiops_practice.git

OSprofiler

- Generate a trace for API requests
- Categories: WSGI, RPC, DB calls
- Call hierarchy
- Time spent in each services/methods
- Projects/services
- Logging/debugging information
- Reports in HTML, JSON, DOT
- Data store: MongoDB, Redis, Loginsight, Ceilometer, Monasca, OpenTracing



openstack

Uber JAEGER

Tracing for OpenStack
JSON example


```
{
  "data": [
    {
      "traceID": "9f2b949d93df62ba",
      "spans": [
        {
          "traceID": "9f2b949d93df62ba",
          "spanID": "ab6d567ee059e105",
          "flags": 1,
          "operationName": "db",
          "references": [
            {
              "refType": "CHILD_OF
              "traceID": "9f2b949d9
              "spanID": "b3e1a556d
            }
          ],
          "startTime": 15432747577
          "duration": 11414,
          "tags": [
            {
              "key": "db.params",
              "type": "string",
              "value": "{}"
            },
          "logs": [],
          "processID": "p1",
          "warnings": null
        },
        {
          "traceID": "9f2b949d93df62ba",
          "spanID": "890dab5d58860e56",
          "flags": 1,
          "operationName": "db",
          "references": [
            {
              "refType": "CHILD_OF",
```


Levels	Duration	Type	Project	Service	Host	Details
− 0	524 ms	total	n/a	n/a	n/a	Details
1	12 ms	wsgi	keystone	main	vinhnt-LAB	Details
+ 1	106 ms	wsgi	keystone	main	vinhnt-LAB	Details
− 1	322 ms	wsgi	glance	api	vinhnt-LAB	Details
+ 2	44 ms	wsgi	keystone	admin	vinhnt-LAB	Details
2	1 ms	db	glance	api	vinhnt-LAB	Details
2	168 ms	db	glance	api	vinhnt-LAB	Details
− 1	64 ms	wsgi	glance	api	vinhnt-LAB	Details
+ 2	54 ms	wsgi	keystone	admin	vinhnt-LAB	Details

https://docs.openstack.org/install-guide/get-started-logical-architecture.html

Tracing for OpenStack
Enabling OSProfiler

- 1. Edit devstack/local.conf

- For Redis add
  - enable_plugin osprofiler https://git.openstack.org/openstack/osprofiler master
    OSPROFILER_COLLECTOR=redis
- For Jaeger add
  - enable_plugin osprofiler https://git.openstack.org/openstack/osprofiler refs/changes/67/611067/4
  - OSPROFILER_BRANCH=refs/changes/67/611067/4
  - OSPROFILER_COLLECTOR=jaeger

- 2. Run devstack/stack.sh
- 3. Prepare the CLI
  - $ cd /opt/stack/devstack
  - $ source openrc admin admin

- 4. Run the openstack commands by appending --os-profile SECRET_KEY in the end of command for example:
  - $ openstack volume list --os-profile SECRET_KEY
  - $ openstack image list --os-profile SECRET_KEY

More details
- https://github.com/openstack/osprofiler
- https://docs.openstack.org/osprofiler/latest/

The OSProfiler library is an official project which enables to trace calls made to OpenStack. This enables to understand the workflow supporting calls and identify which types of calls are made inside OpenStack

Instead of having to go through the whole code and adding instrumentation points near HTTP/RPC/DB calls, osprofiler is already integrated in all of the main projects of OpenStack (Nova, Neutron, Keystone, Glance etc..).

- 5. To view the trace

- For Redis
  - $ osprofiler trace show --connection-string redis://localhost:6379 --html <trace-id> --out <some_name>.html
  - Copy that file to your local laptop and open in browser
  - You can use JSON format as well
- For Jaeger
  - Check Jaeger UI at http://VM_IP:16686
  - Use the shortened traceid printed in the search box of Jaeger UI and search for the trace

Distributed Tracing
References

Systems

- **OpenZipkin: A distributed tracing system:** https://zipkin.io
- **AWS X-Ray Distributed Tracing System**: https://aws.amazon.com/xray/
- **Jaeger: open source, end-to-end distributed tracing:** https://www.jaegertracing.io

Papers

- **Dapper, a Large-Scale Distributed Systems Tracing Infrastructure**: https://ai.google/research/pubs/pub36356
- **Facebook Canopy**: https://cs.brown.edu/~jcmace/papers/kaldor2017canopy.pdf
- **So, you want to trace your distributed system? Key design insights from years of practical experience**: http://www.pdl.cmu.edu/PDL-FTP/SelfStar/CMU-PDL-14-102.pdf

Thank you.

Bring digital to every person, home and organization for a fully connected, intelligent world.

Copyright©2019 Huawei Technologies Co., Ltd.
All Rights Reserved.

The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. Huawei may change the information at any time without notice.