



KubeCon + CloudNativeCon Seattle

Distributed Tracing in
Serverless Systems

Nitzan Shapira, Epsagon



> whoami



Nitzan Shapira (@nitzanshapira)

Software engineer > 12 years

Co-Founder, CEO at Epsagon



Tel Aviv



Things to discuss

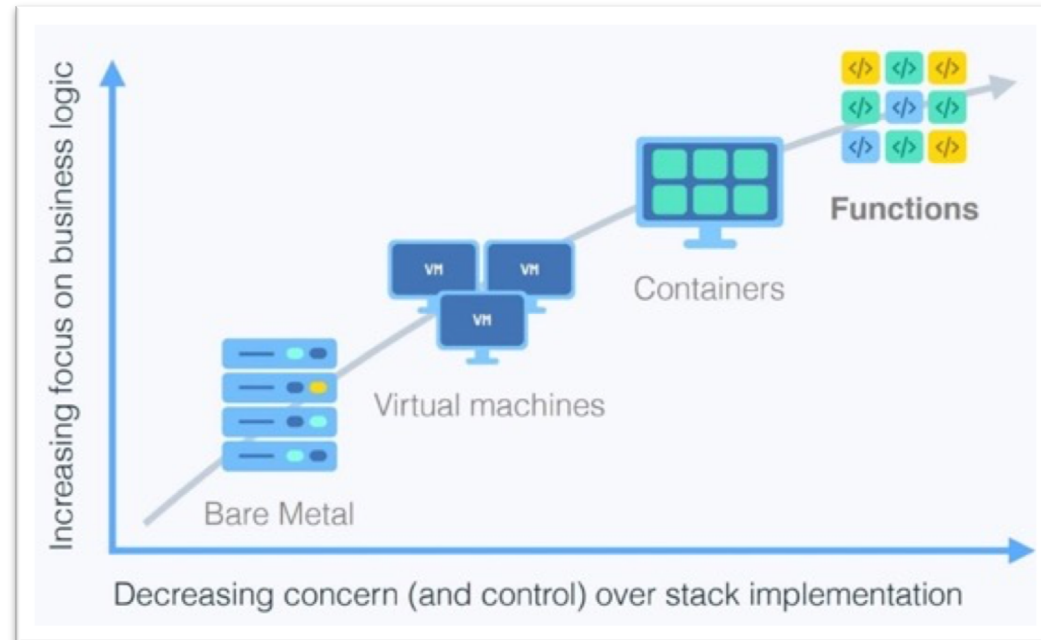
What is serverless? How is it different?

What is observability for serverless?

How can distributed tracing help?

How will it help my job?

What is serverless?



[Compute-as-a-Service]
FaaS: Function-as-a-Service
CaaS: Container-as-a-Service

+

Managed services (APIs)

=

Don't manage infrastructure

Focus on business logic



Why serverless?

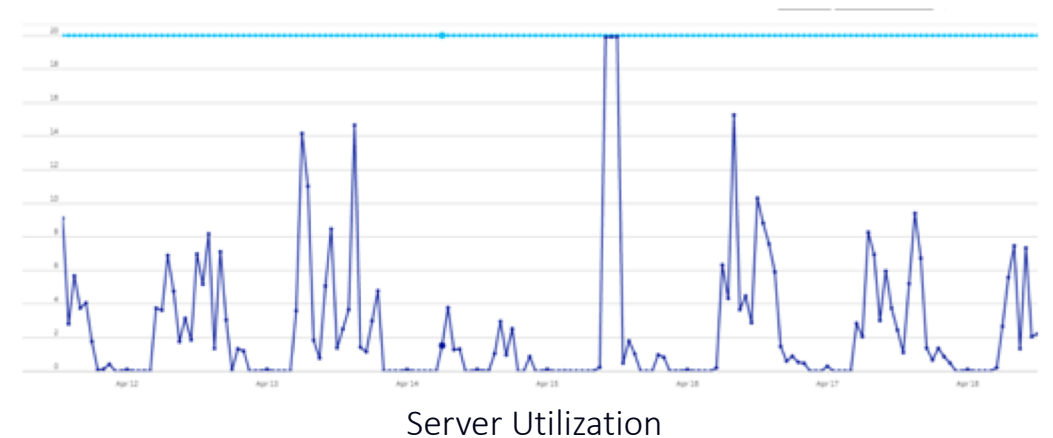
Pay-per-use: reduces cloud compute cost by 90%

Out-of-the-box **auto-scaling**

DevOps → LowOps

++Developer velocity

Focus on **business logic** – iterate faster



The limitations of FaaS



Limited memory



Limited running time



Stateless



Cold starts

+ concurrency limit
+ some others...

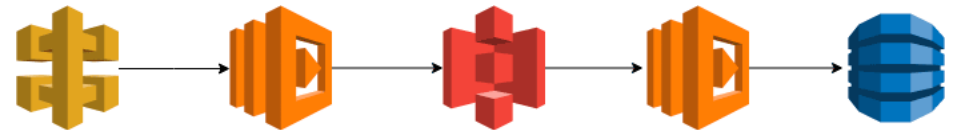
The properties of serverless applications

Serverless is micro-services

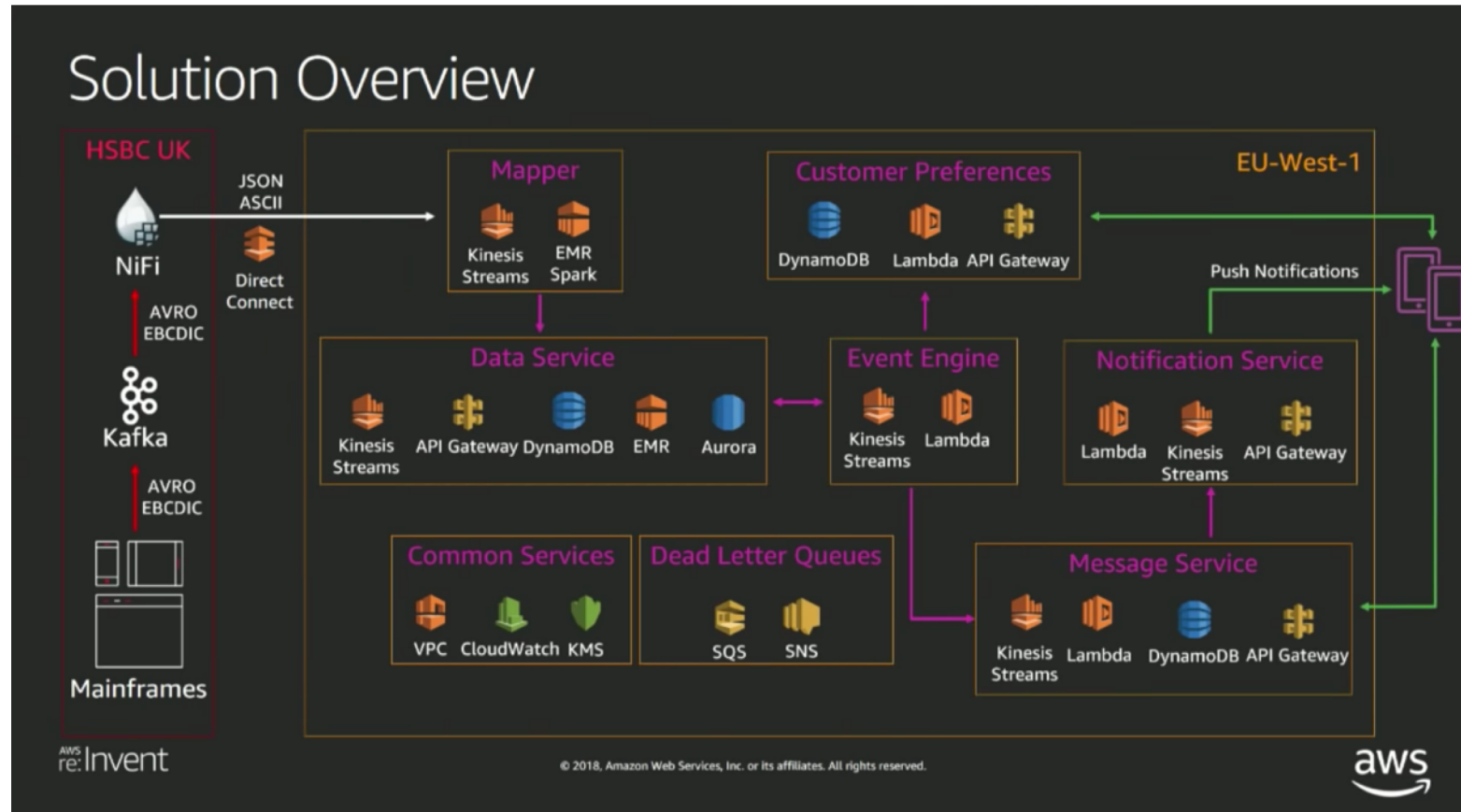
Serverless applications are

- Highly distributed
- Highly event-driven

Utilizing managed services via **APIs** is key

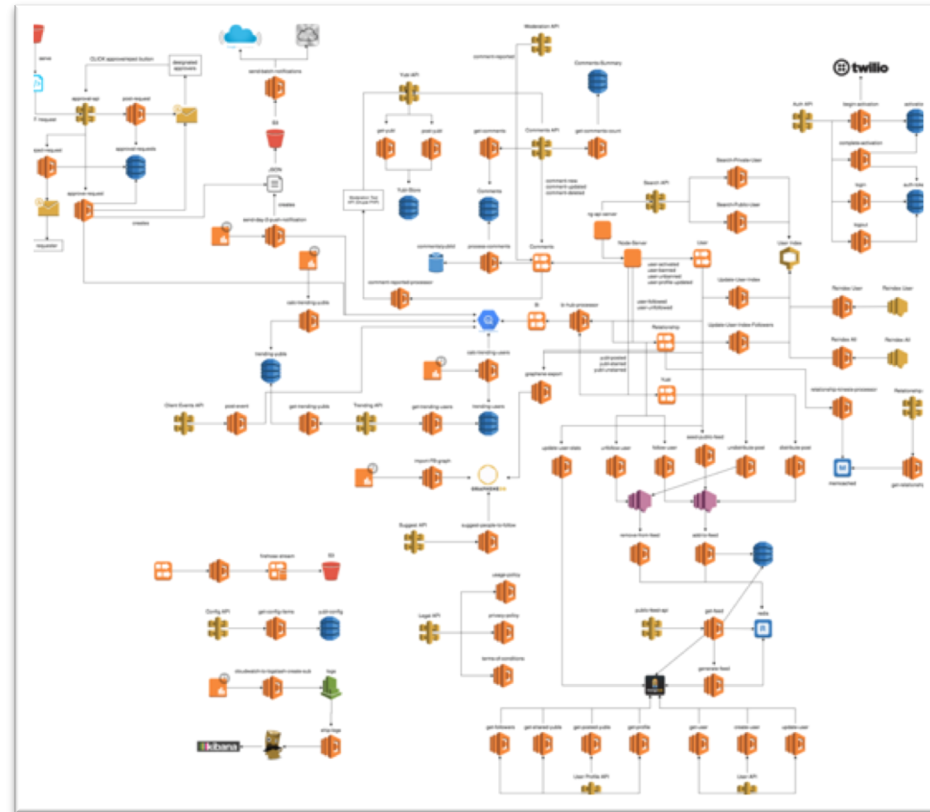
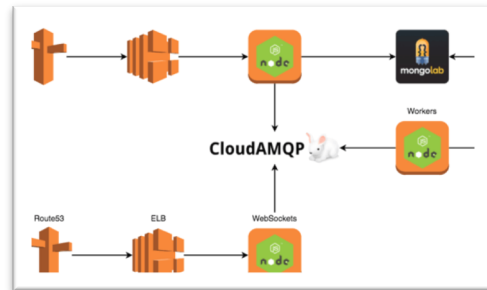


A real example – HSBC



Source: re:Invent 2018

The challenge in serverless



SIMPLE

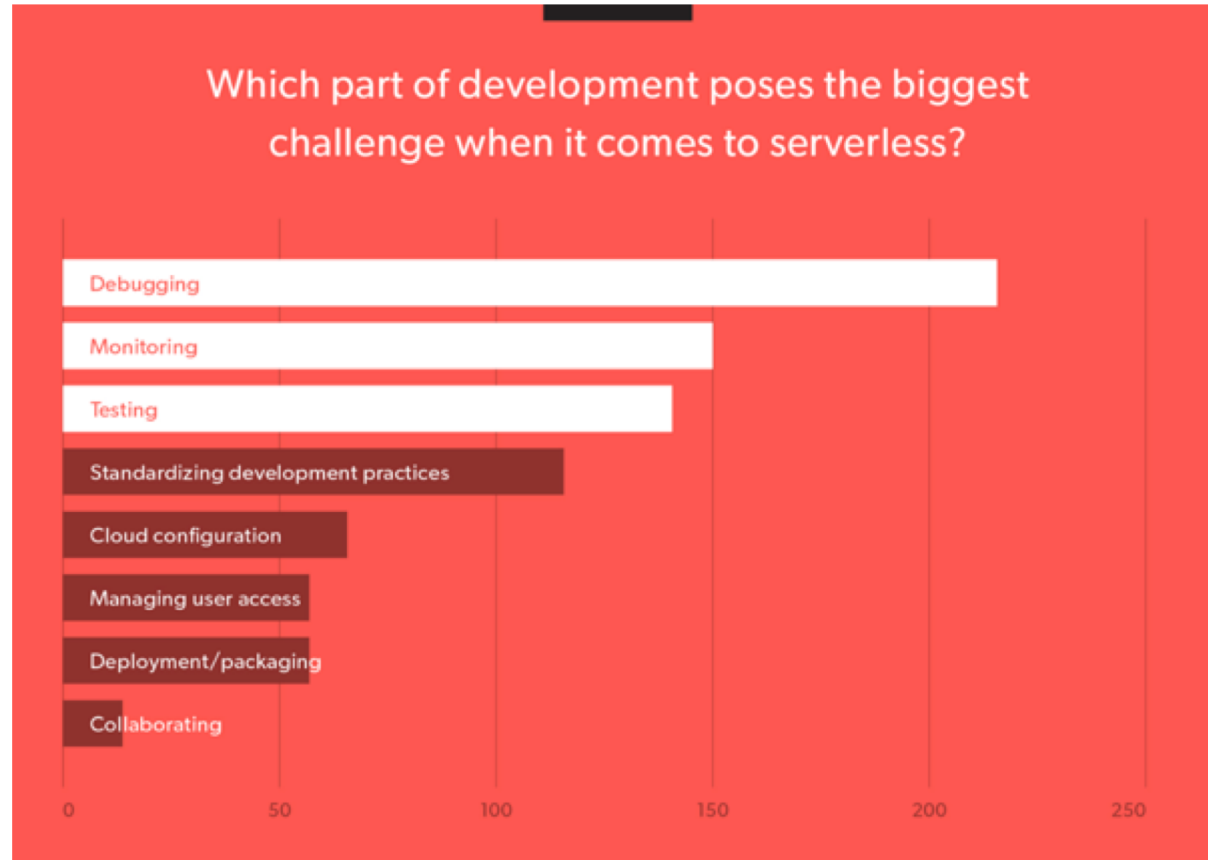
COMPLEX



Yan Cui



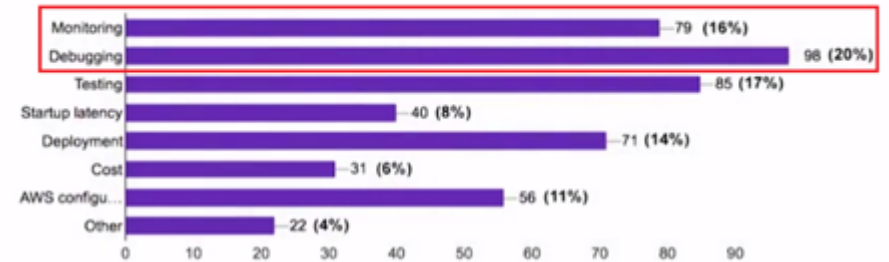
What the community thinks



2018 Serverless Community Survey, serverless.com, July 2018

Which of the following are serious pain points for you in developing serverless architectures?

(175 responses)



2017 results



Observability – why do we need it?



Track system health



Troubleshoot and fix



Optimize performance and cost

Observability in serverless

Let's go one by one

Track system health

System == Functions ?

Functions are important

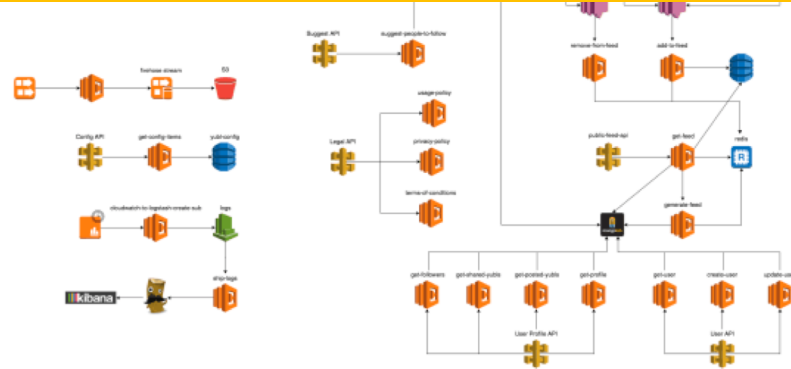


- Errors
- Timeout
- Out-of-memory
- Cold start

Track system health



Serverless != Functions



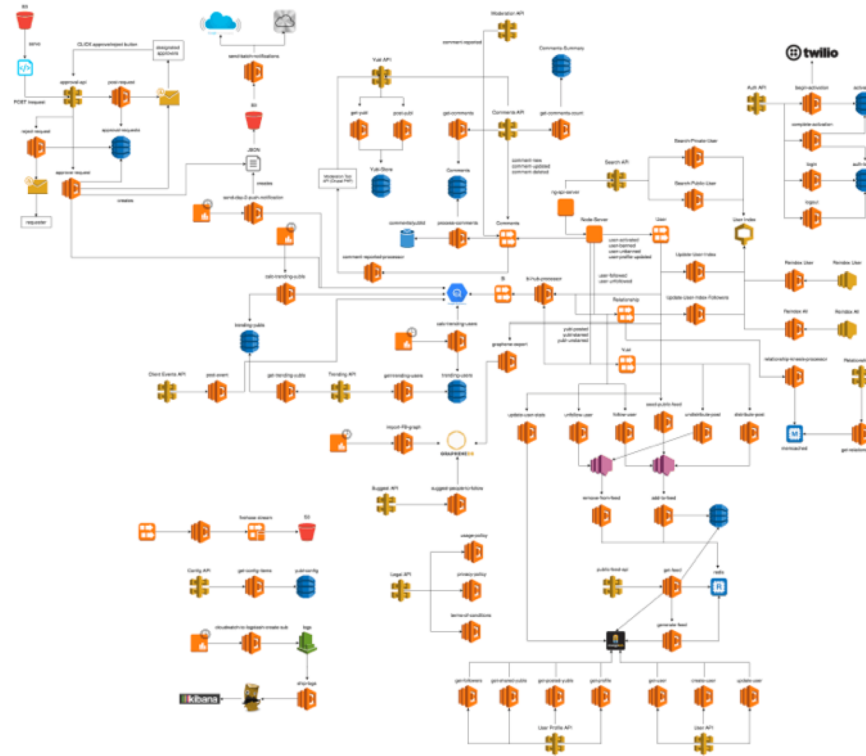
Track system health

System > Functions !

Functions

APIs

Transactions



Troubleshoot and fix

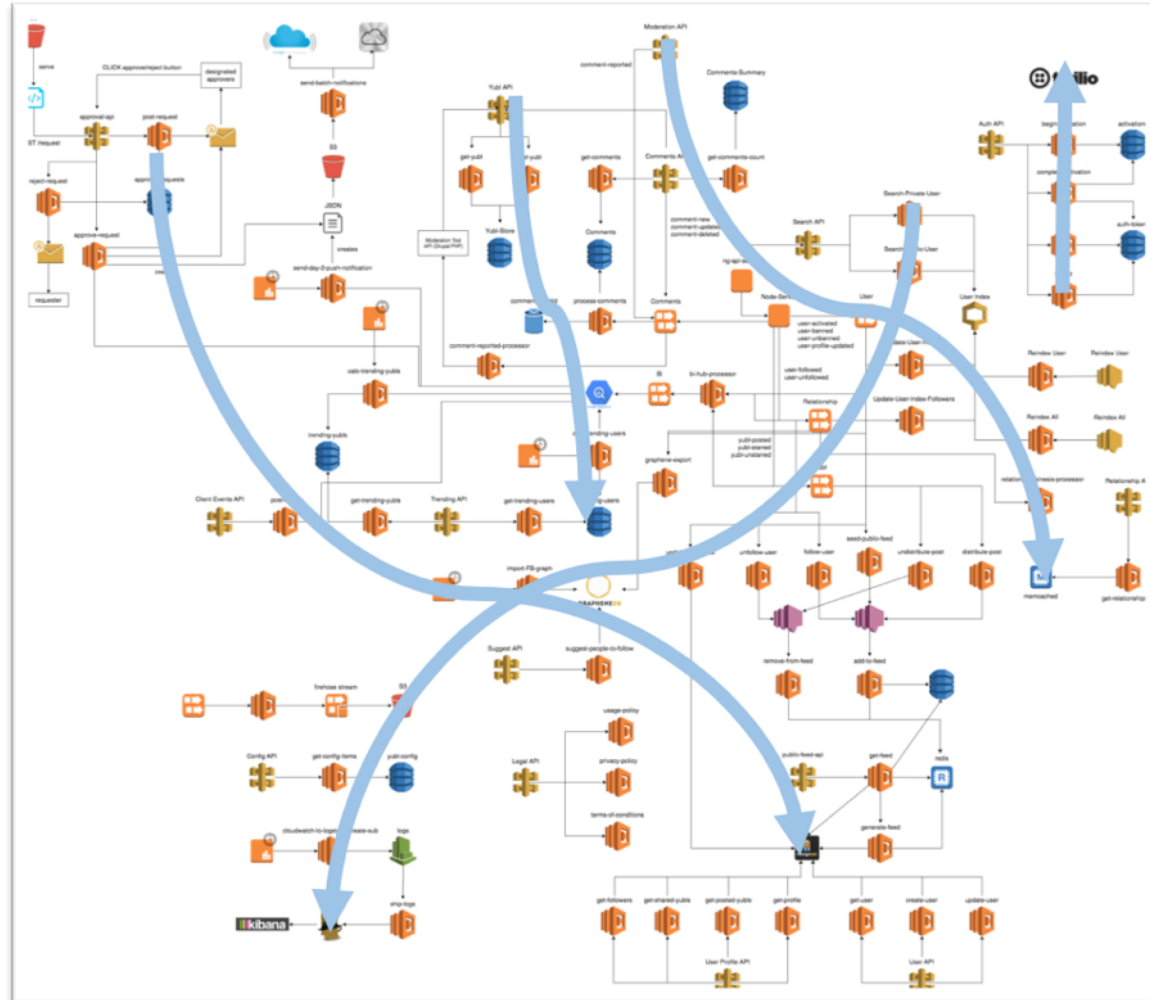
The screenshot displays the AWS CloudWatch console. On the left, there's a navigation pane with 'Log Groups' selected. The main area shows a list of log groups with columns for 'Log Stream Name Prefix' and 'Log Streams'. A 'Log Stream' is selected, and the 'Log Event Viewer' is open, showing a list of log events with columns for 'Time (UTC +00:00)' and 'Message'. The log events include various messages such as 'START RequestId: fb4c77a8-58d9-11e8-a11e-85e5b5f027cc Version: \$LATEST', 'END RequestId: fb4c77a8-58d9-11e8-b2e8-b58d206a6245', and 'REPORT RequestId: fb4c77a8-58d9-11e8-b2e8-b58d206a6245 Duration: 2505.00 ms Billed Duration: 2600 ms Memory...'. Below the log event viewer, there's a 'CloudWatch metrics at a glance' section showing a line graph for 'Invocation count' over time. The graph shows a sharp increase in invocations around 19:00 and then a steady decline. The 'Other monitoring information' section below the graph shows the last modified date as '2015-11-30T16:09:54.794-0000'. In the background, there are diagrams of AWS Lambda functions and their connections to other services like S3 and CloudWatch.

Functions are not enough

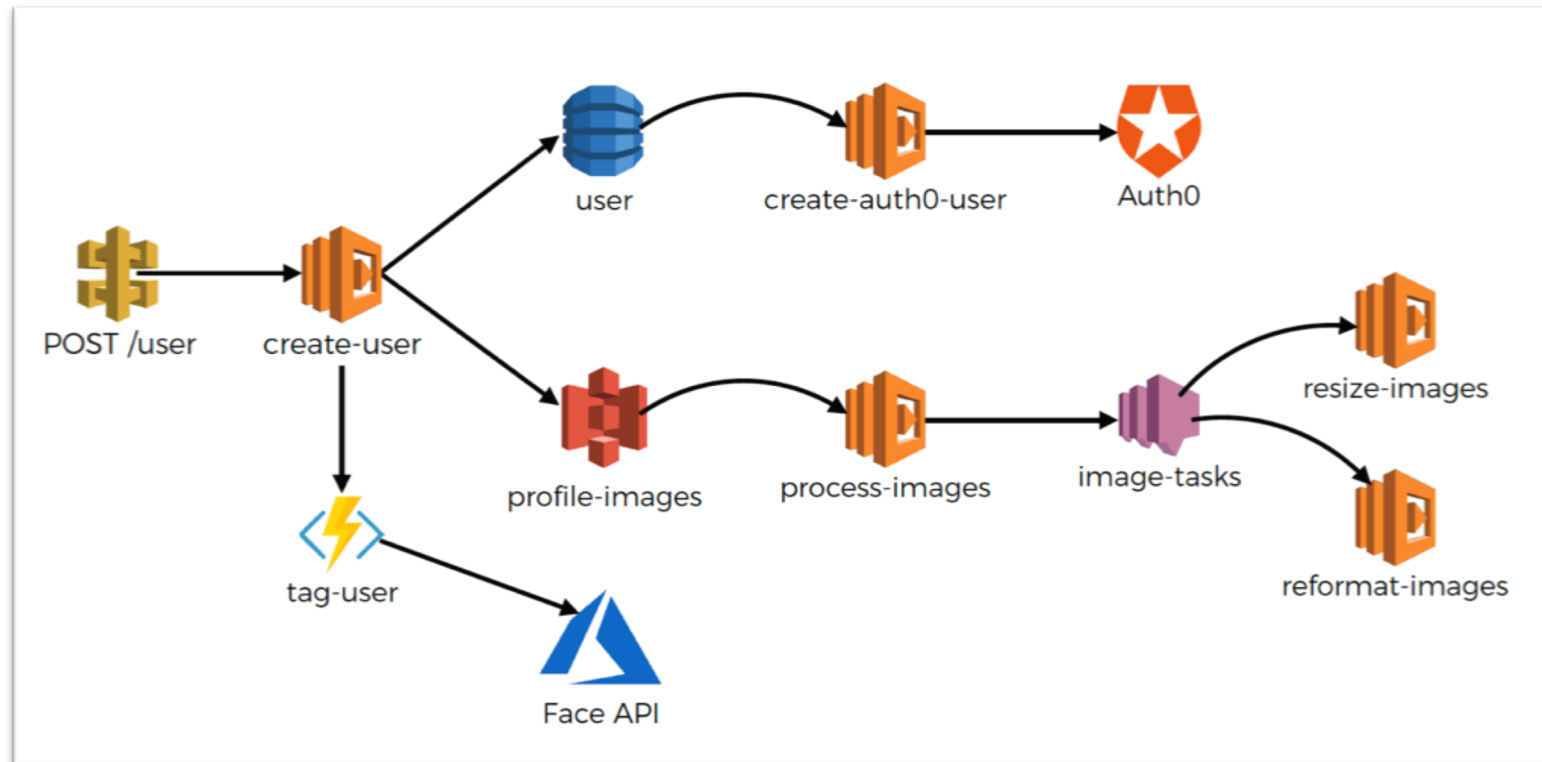
Need: track asynchronous events



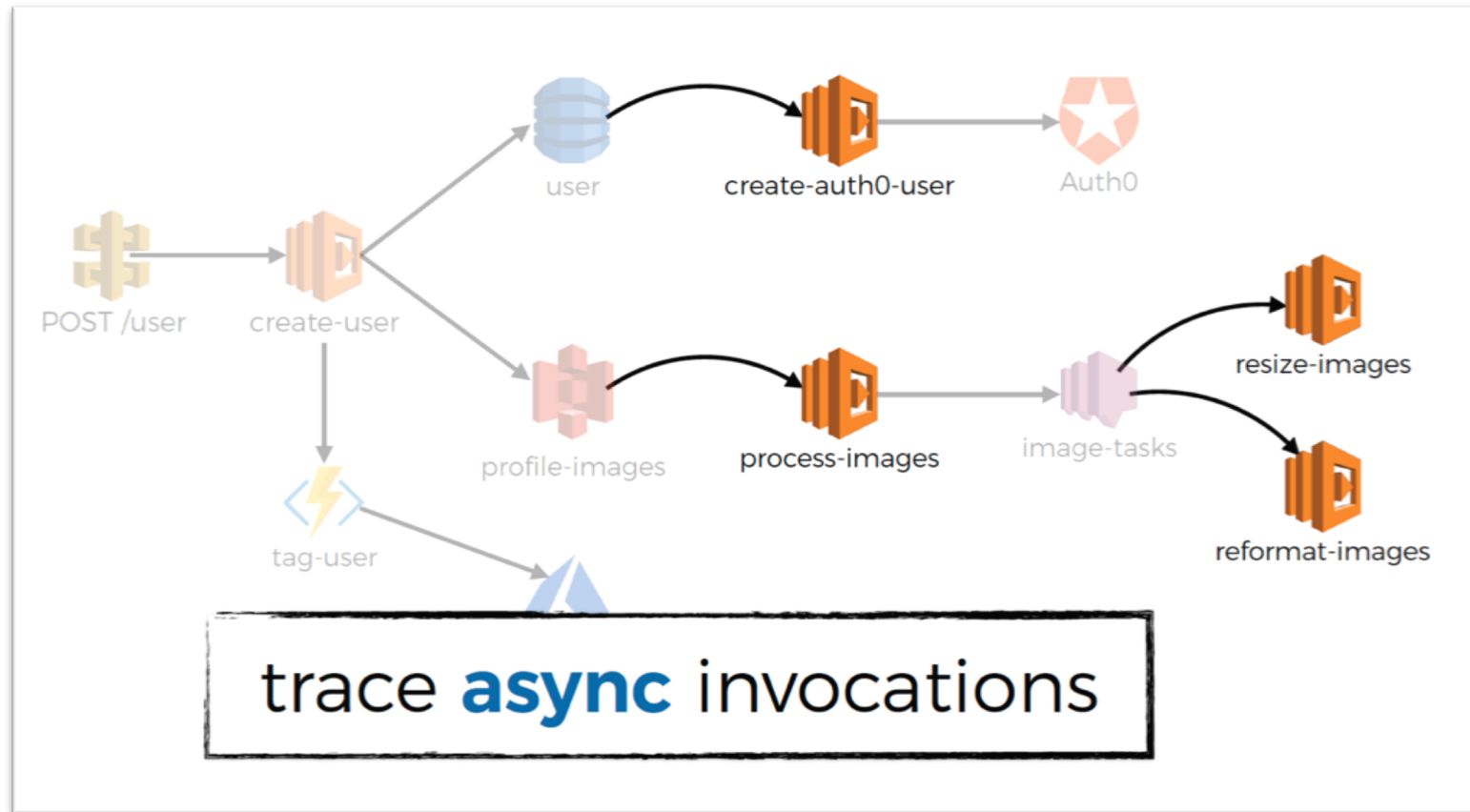
Transactions



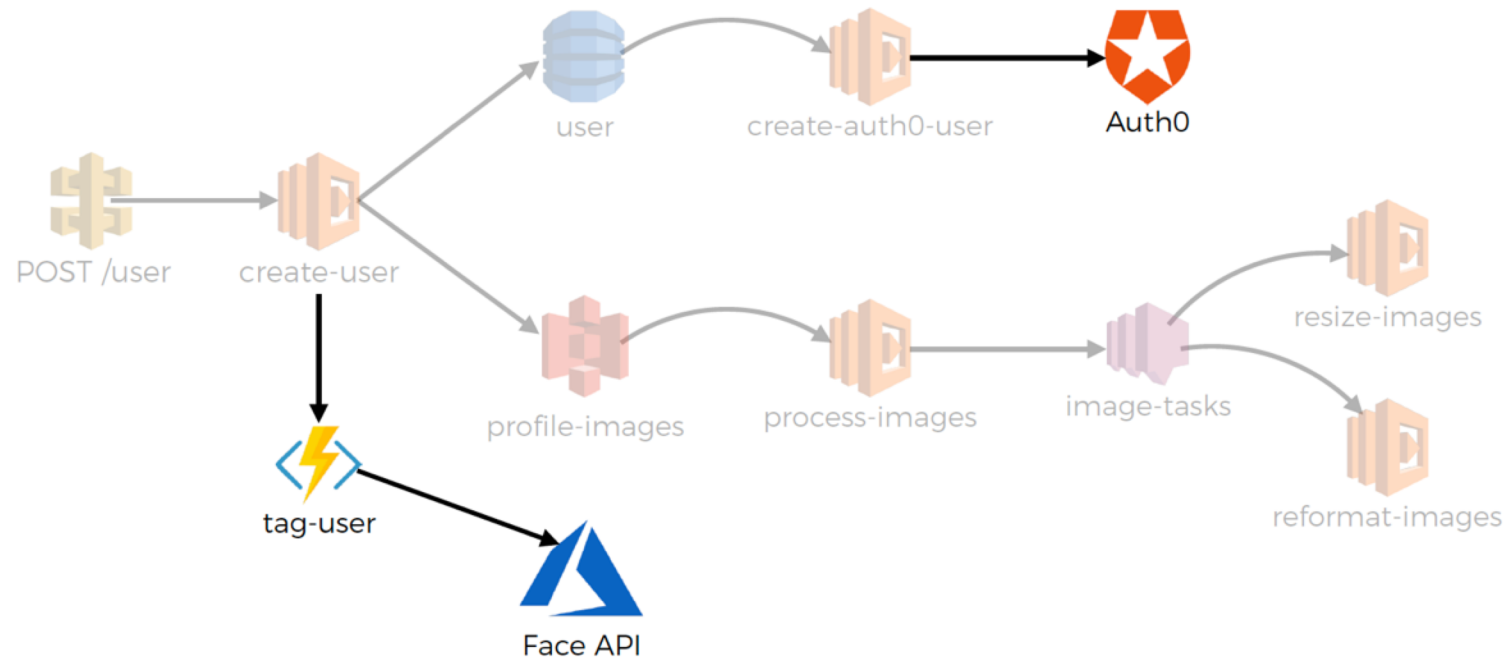
Tracing asynchronous invocations



Tracing asynchronous invocations

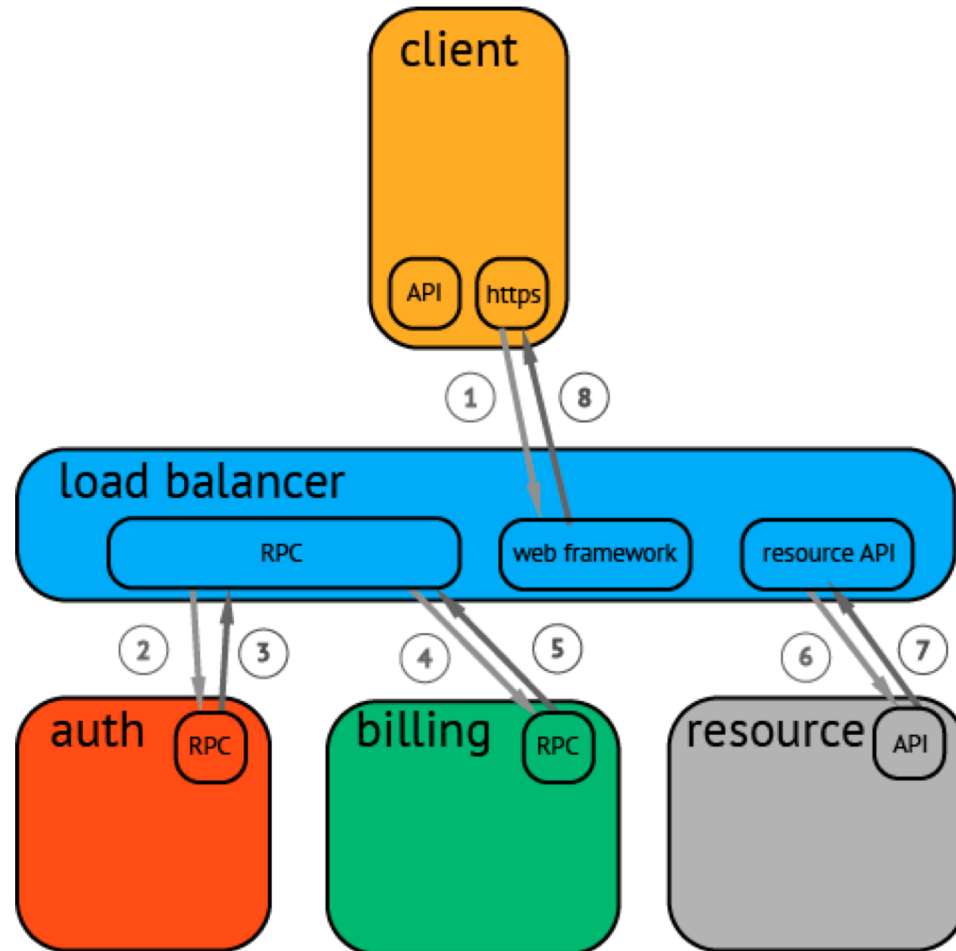


Tracing asynchronous invocations



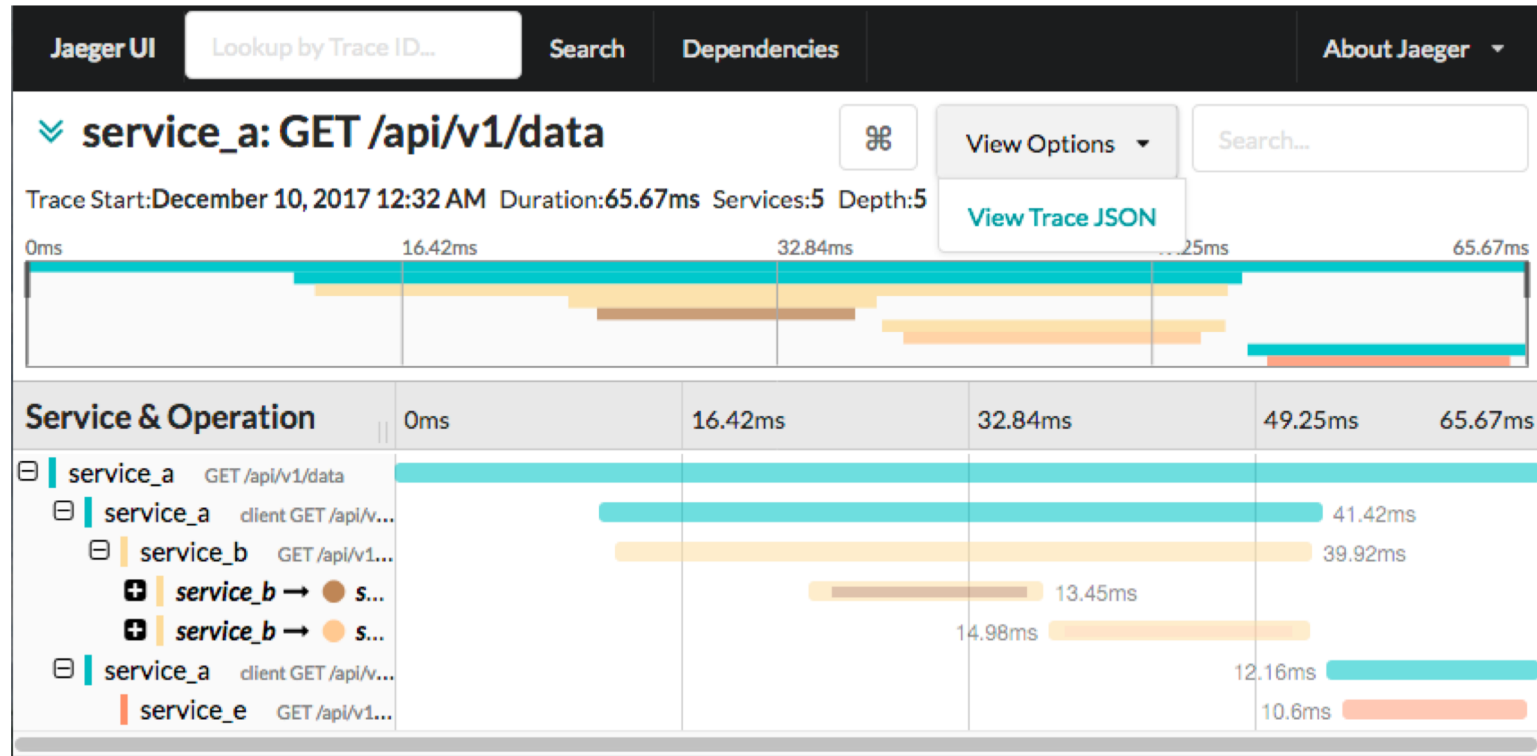
trace **non-AWS** resources

Distributed tracing



...a **trace** tells the story of a transaction or workflow as it propagates through a (potentially distributed) system. Distributed tracing is a method used to profile and monitor applications.

Distributed tracing



Jaeger

Implementing distributed tracing

Manual tracing/instrumentation

Before/after calls

At the end of each micro-service

High maintenance

High potential of errors



Inbound request

Somewhere in your server's request handler code:

```
def handle_request(request):
    span = before_request(request, opentracing.global_tracer())
    # store span in some request-local storage using Tracer.scope_manager,
    # using the returned `Scope` as Context Manager to ensure
    # `Span` will be cleared and (in this case) `Span.finish()` be called.
    with tracer.scope_manager.activate(span, True) as scope:
        # actual business logic
        handle_request_for_real(request)

def before_request(request, tracer):
    span_context = tracer.extract(
        format=Format.HTTP_HEADERS,
        carrier=request.headers,
    )
    span = tracer.start_span(
        operation_name=request.operation,
        child_of(span_context)
    )
    span.set_tag('http.url', request.full_url)

    remote_ip = request.remote_ip
    if remote_ip:
        span.set_tag(tags.PEER_HOST_IPV4, remote_ip)

    caller_name = request.caller_name
    if caller_name:
        span.set_tag(tags.PEER_SERVICE, caller_name)

    remote_port = request.remote_port
    if remote_port:
        span.set_tag(tags.PEER_PORT, remote_port)

    return span
```



Serverless apps are **very** distributed

Complex systems have **thousands** of functions

What about the **developer velocity**?

Can it be done differently in serverless?

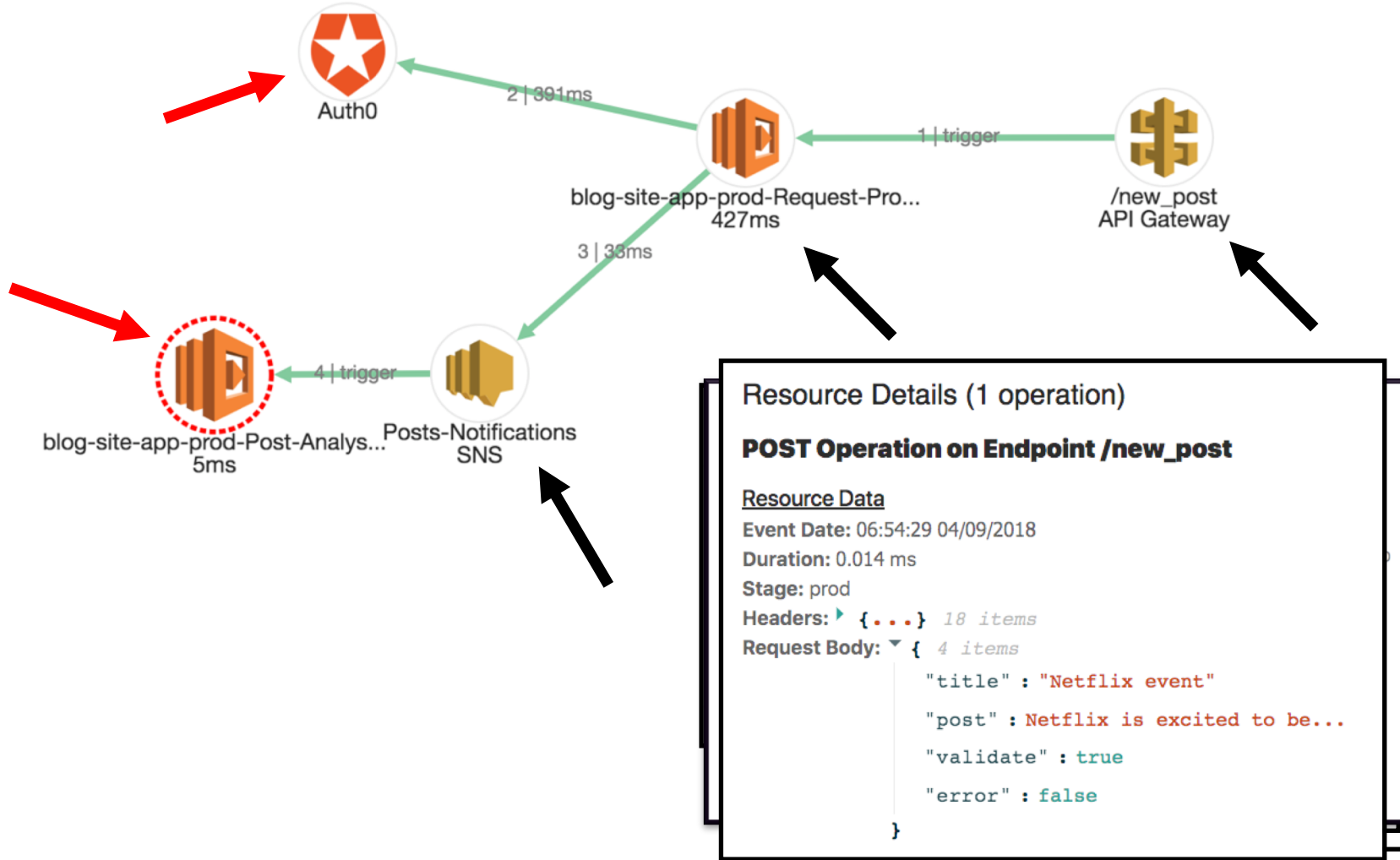


Automation can help to keep up with the development speed of serverless



Example

Example



Monitoring serverless



Limited memory



Limited running time



Stateless



Cold starts

Time is \$\$\$



Where do we spend the most time?

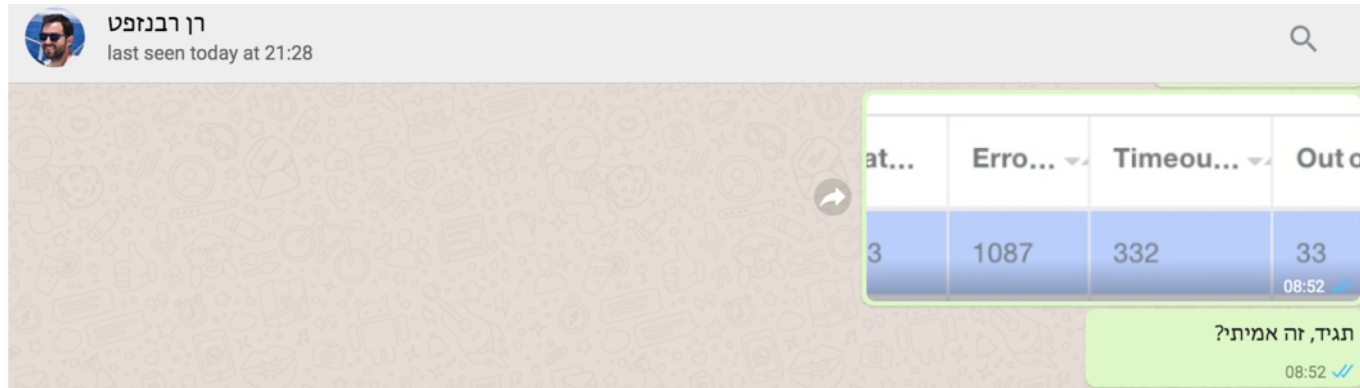
Our own code



API calls

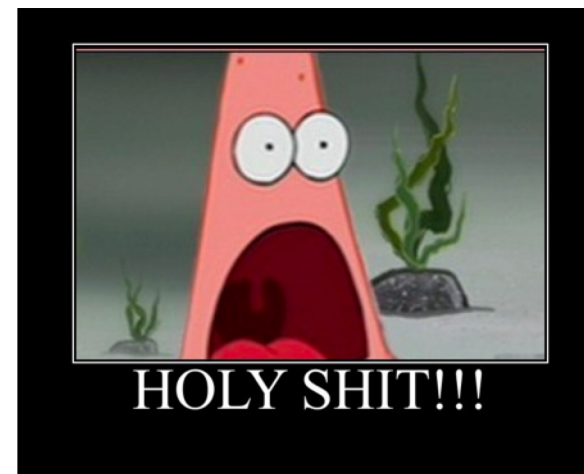
Serverless cost crisis

A real-life example



Last Invocation	p50 Durati...	Monthl...
27/06/2018, 11:40:...	2019.5	12378.41

\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$

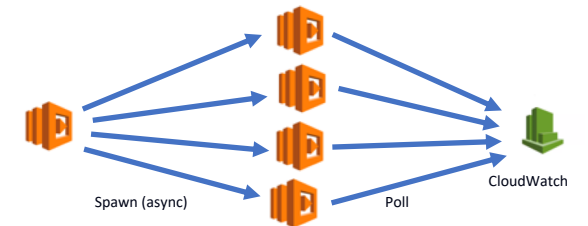


Scanning functions

Scanning CloudWatch using AWS Lambda

Every 5 minutes, save to RDS

A new Lambda is spawned for every customer's function

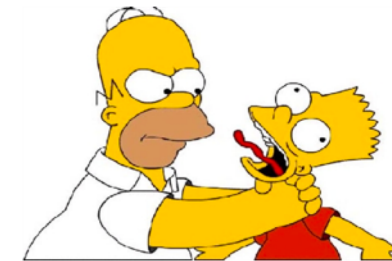


As time flies...

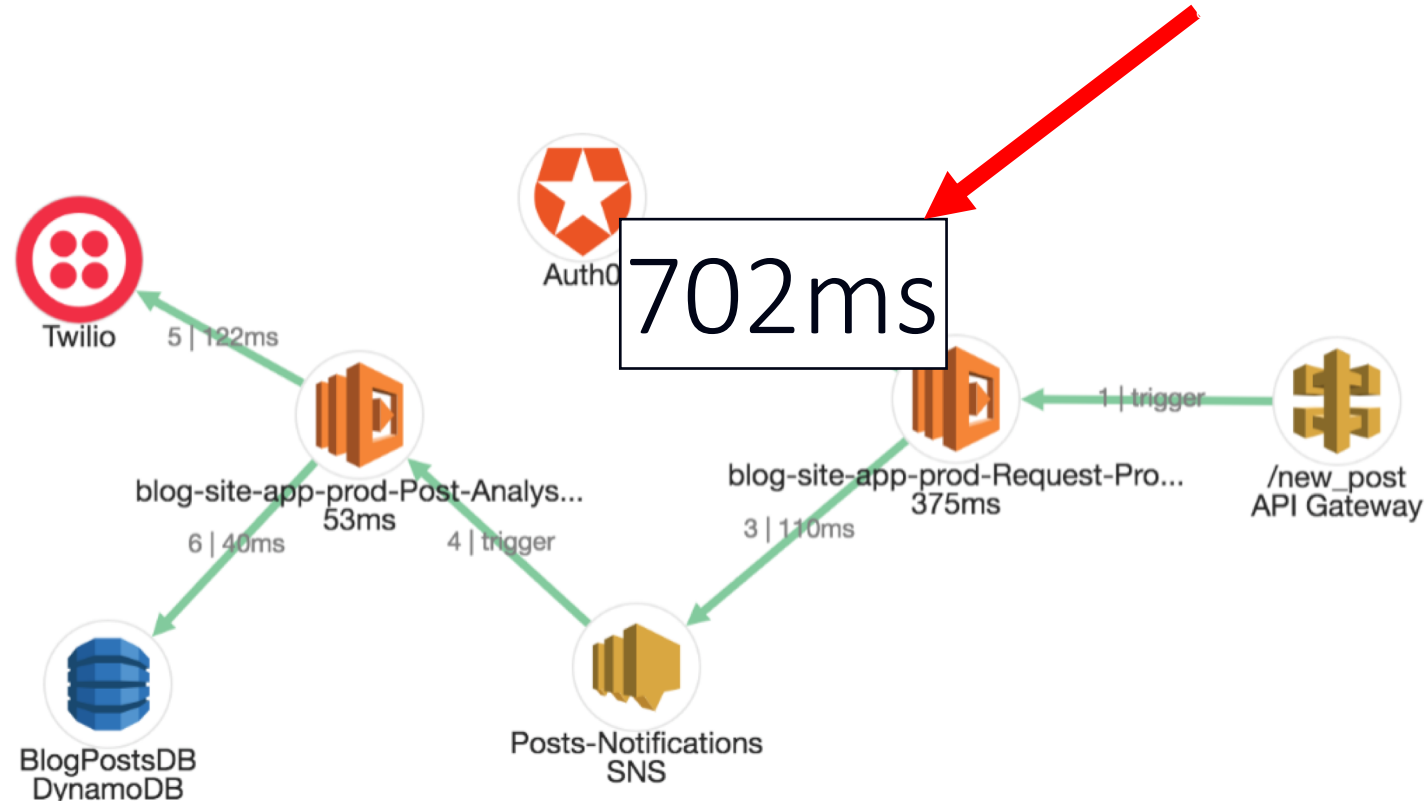
CloudWatch became highly throttled

Requests took too much time

5K concurrent Lambdas, for 5 minutes,
timing out , every 5 minutes



Why you should care about external APIs

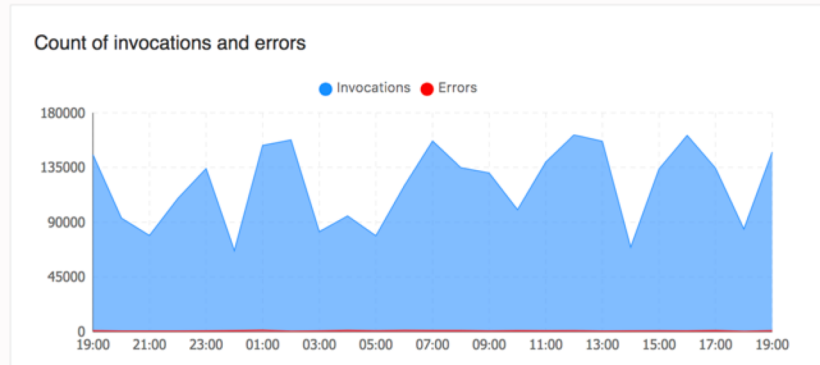


Track service health

Dashboard - Insights

Insights

- ⊗ Application Blog Site Production had 12309 errors
- ⊗ Timeout error for function 'long-calculation' (30 seconds defined)
- ⚠ Function 'long-calculation-2' is close to timeout limit (5 seconds used out of 6 seconds)
- ⚠ Function 'massive-calculation-2' is close to defined memory limit (124MB used out of 128MB)

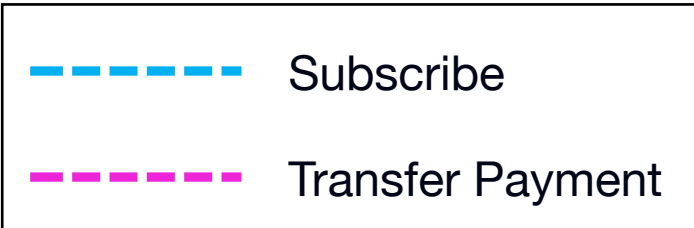
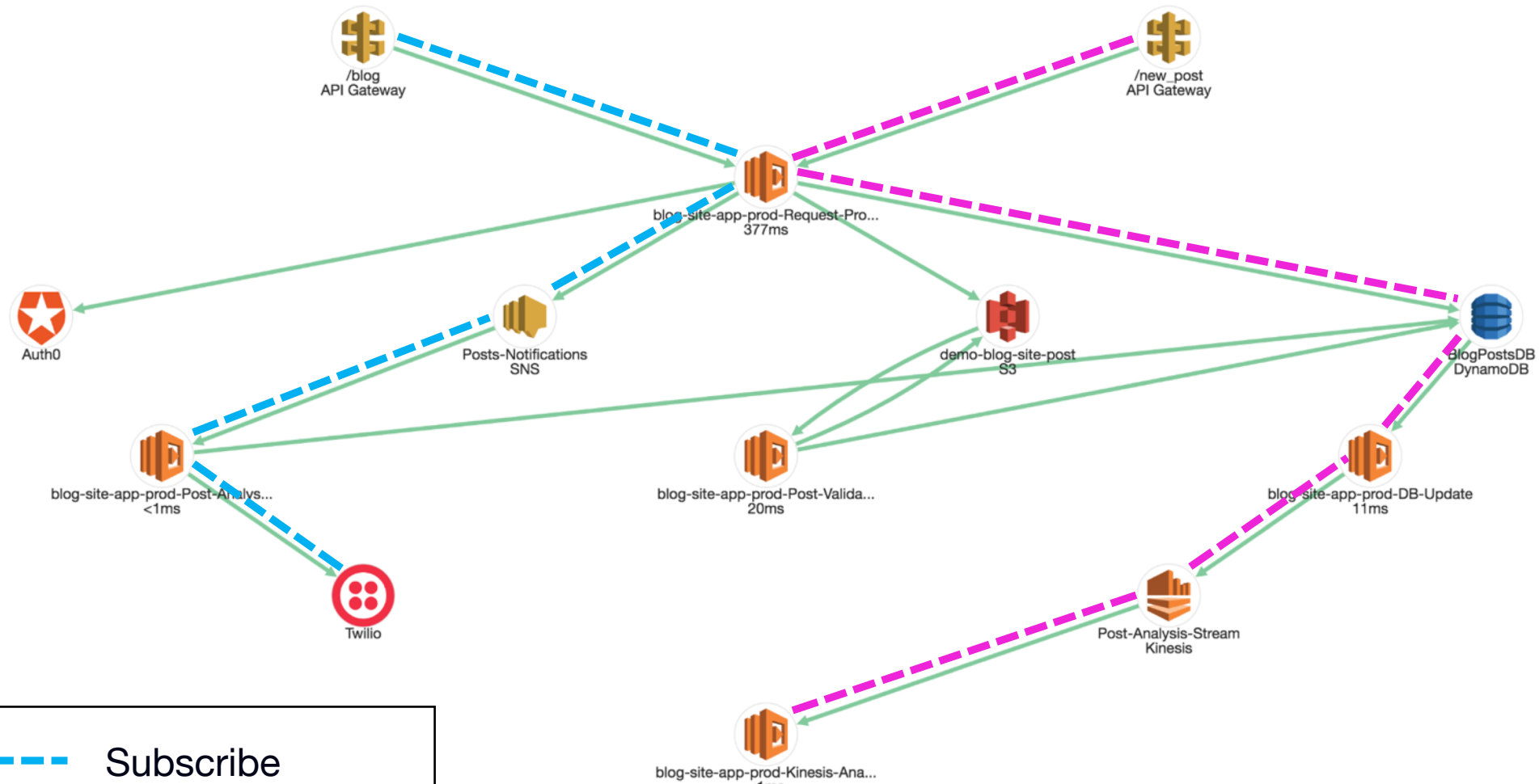


Statistics

System Health [?]	Cold Starts [?]
98.3%	136
▲2%	▼-12%
Total Cost [?]	Monthly Cost [?]
\$24.86	\$745.95
▲32%	▲32%
Invocations [?]	Errors [?]
1,498,273	9,049
▲12%	▲13%



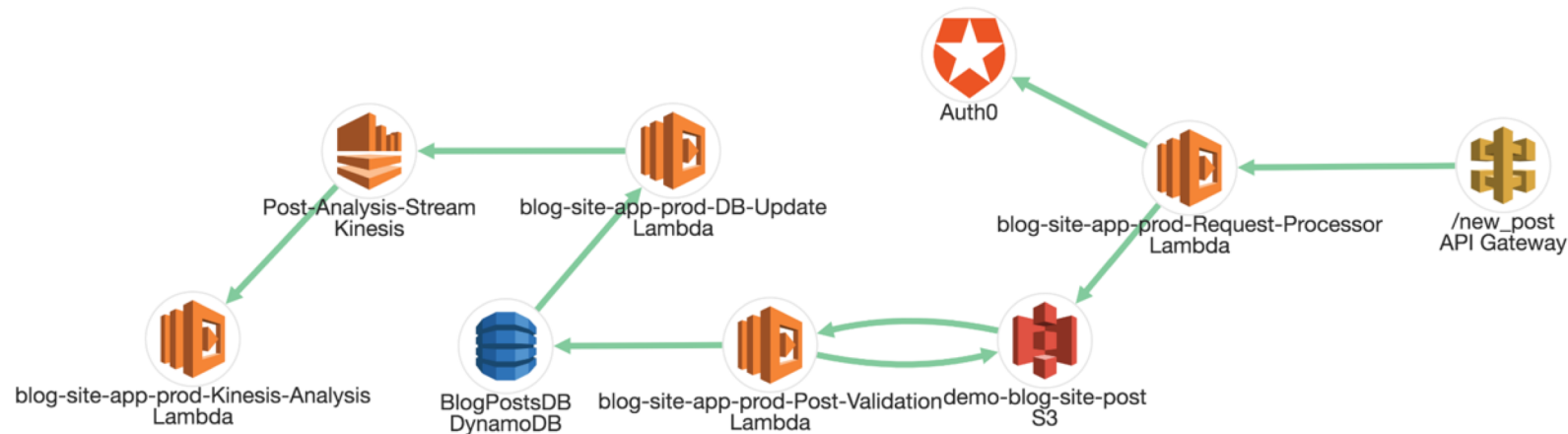
Business flows



What should I optimize first?

Dataflows 24 hours 7 days

Dataflow Name	Invocation Count	p50 Duration (ms)	p99 Duration (ms)	
Posts Analysis	6,464	2229.96	3240.58	Transactions
Posts Upload	7,171	515.70	67722.40	Transactions
Read Blog Posts	5,252	403.39	462.81	Transactions



Remember...

Serverless + Distributed Tracing

=

Perfect marriage
(but only if you automate)





Thank you!

nitzan@epsagon.com

@nitzanshapira

www.epsagon.com