The Open Source Observability Playbook

Hen Peretz - Head of Solutions Engineering @epsagon

epsagon

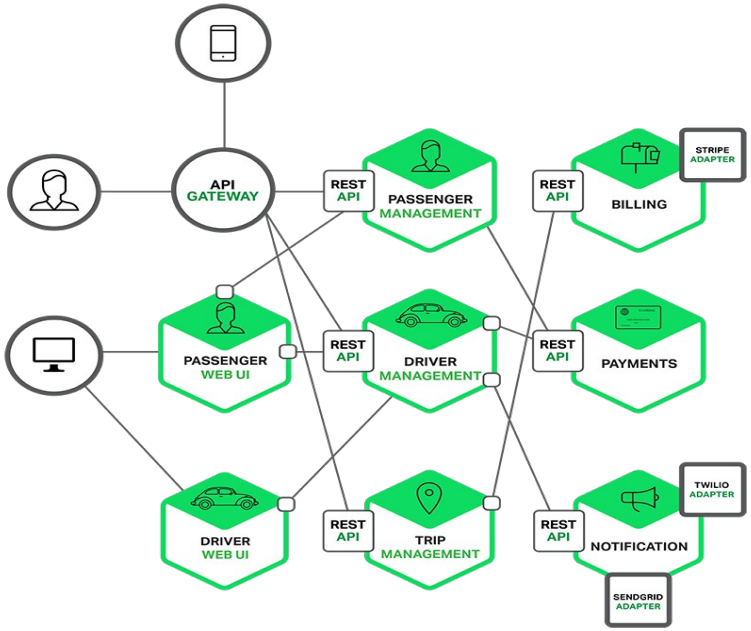
What We'll Discuss Today

Old-School Observability methods

How to achieve observability

Logging, monitoring best practices

Open-source monitoring landscape



epsagon

Why Monitoring?

Make sure our business works

epsagon

What Should We Monitor?

4 golden signals from SRE book *by Google*:

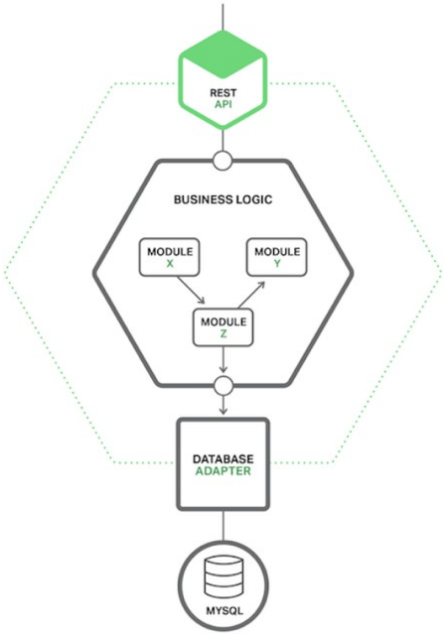
Latency
Traffic
Errors
Saturation

Old-School Monitoring

Agent based

Collects only host data

Collects only metrics



epsagon

Old-School Troubleshooting

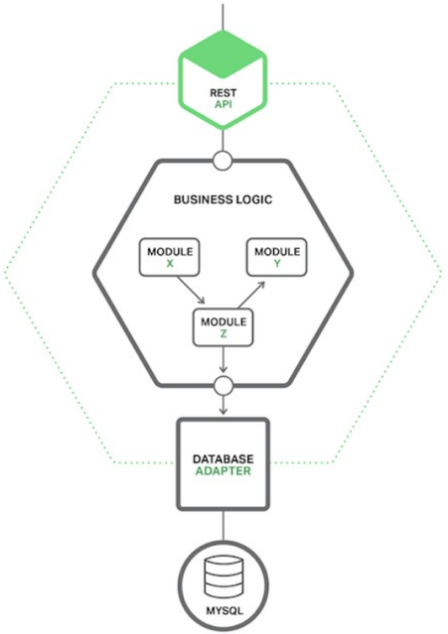
We need more debug data → logs

Old-School Logging

Agent based

Dumps locally or remotely

Collects only logged data

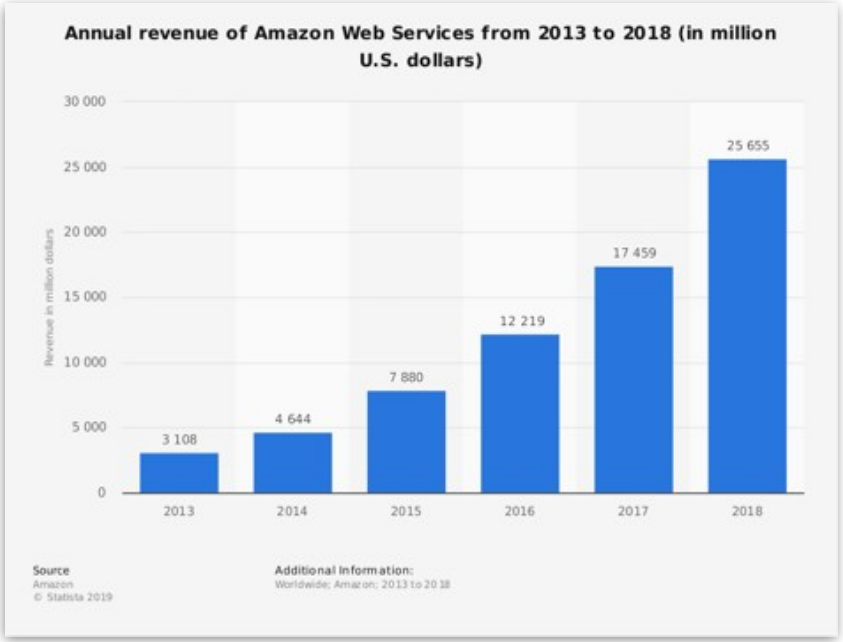


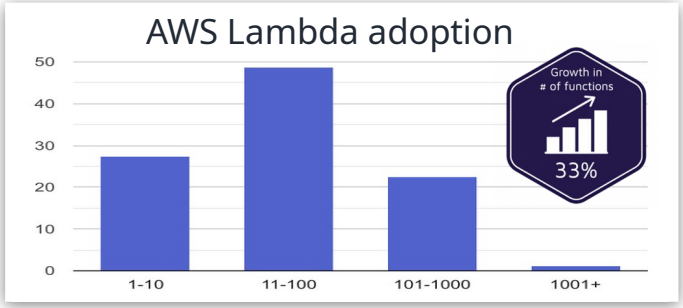
epsagon

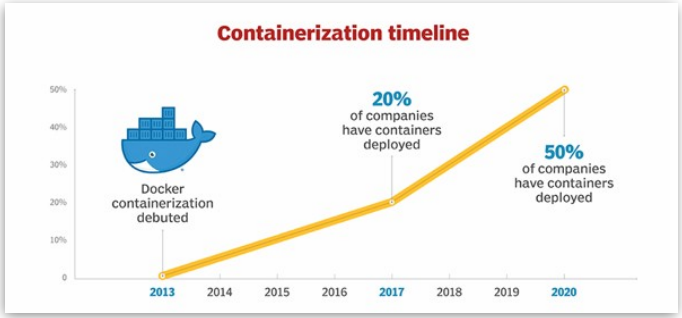
Fast Forward into the Future

epsagon

New Era for Applications: Cloud + Microservices

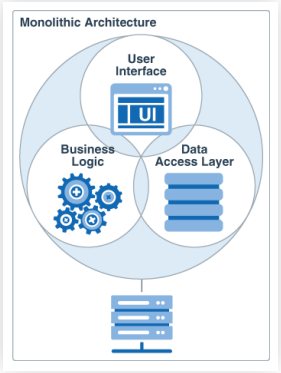


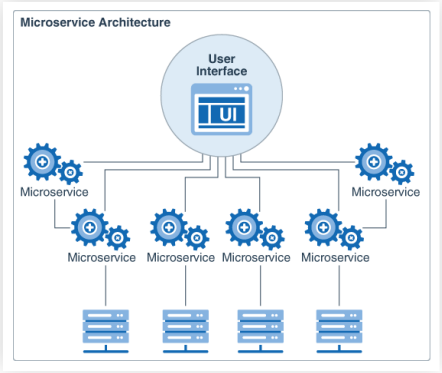


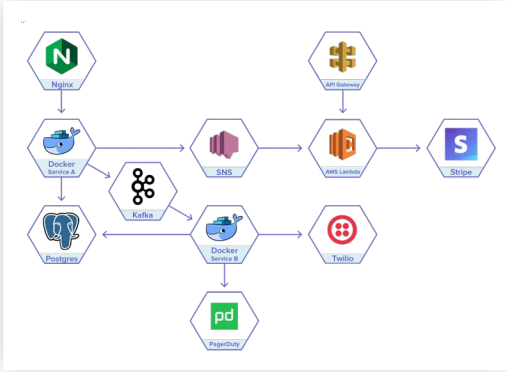


epsagon

The Rise of Microservices in the Cloud







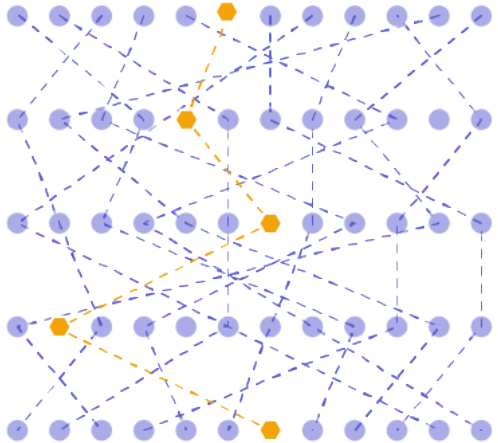
Host-based Monolithic → Host-based Distributed → Abstracted-host Highly Distributed

Extremely hard to monitor and troubleshoot

epsagon

Challenges for Engineering and DevOps

- **Troubleshooting**
  Are basic logs and metrics the right tool for highly distributed applications?
- **Monitoring**
  “Is my application working properly”?
- **Development**
  I’m not sure what’s currently running in production. How can I build new services?



epsagon

The Three Pillars of Observability

Combining metrics, logs, and traces for observability is the **only** way to understand complex environments

Metrics tell us the “what”

Logs tell us the “why”

Traces tell us the “where”

epsagon

Logging Best Practices

Print out JSONed logs with metadata
(service name, stage, etc.)

Automate the process of logging

Index the fields you're are using

Logging Best Practices - Setup

epsagon

Logging Best Practices

Demo

epsagon

Monitoring Best Practices

Aggregate all metrics into a unified dashboard

Define your critical metrics (thresholds)

Use custom business metrics

Monitoring Best Practices

Monitor application metrics:

Avg. duration of calls to an HTTP API

Minimum number of calls to a message queue

Number of 500/400 errors

Monitoring Best Practices

Demo

epsagon

Something Is Still Missing

How do we correlate between metrics and logs

How do we correlate between data in different services

epsagon

Distributed Tracing DIY

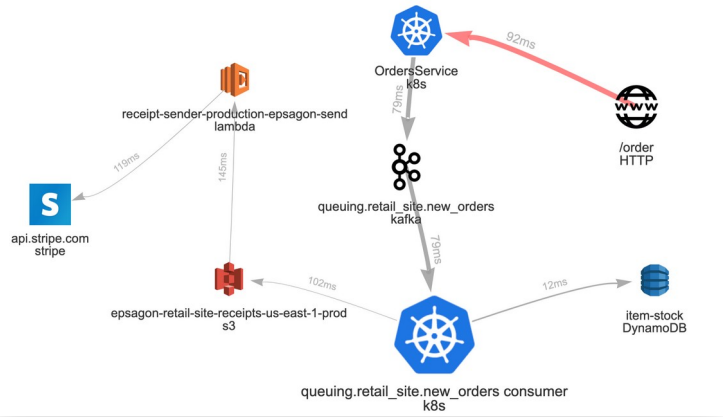
epsagon

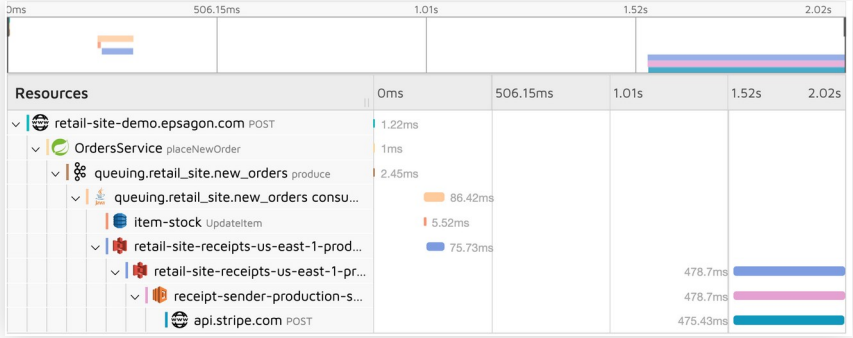
Why Distributed Tracing Is Critical Today

The Only Way to Understand Cloud-Native Workloads

Due to high complexity and the need for manual instrumentation, distributed tracing remained an approach viable only for very **tech savvy** companies

> A trace tells the story of a transaction or workflow as it propagates through a distributed system





epsagon

Distributed Tracing Landscape

+

=

Generating Traces

Ingestion and visualization

Distributed Tracing DIY - Generating Traces

Instrument every call
(AWS-SDK, http, postgres, Flask, ...)
A **span** for every request and response
Add **context** to every span
Inject and **Extract** IDs in relevant calls

```python
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
```

epsagon

Distributed Tracing DIY

Demo

epsagon

Distributed Tracing DIY - Ingestion & Visualization

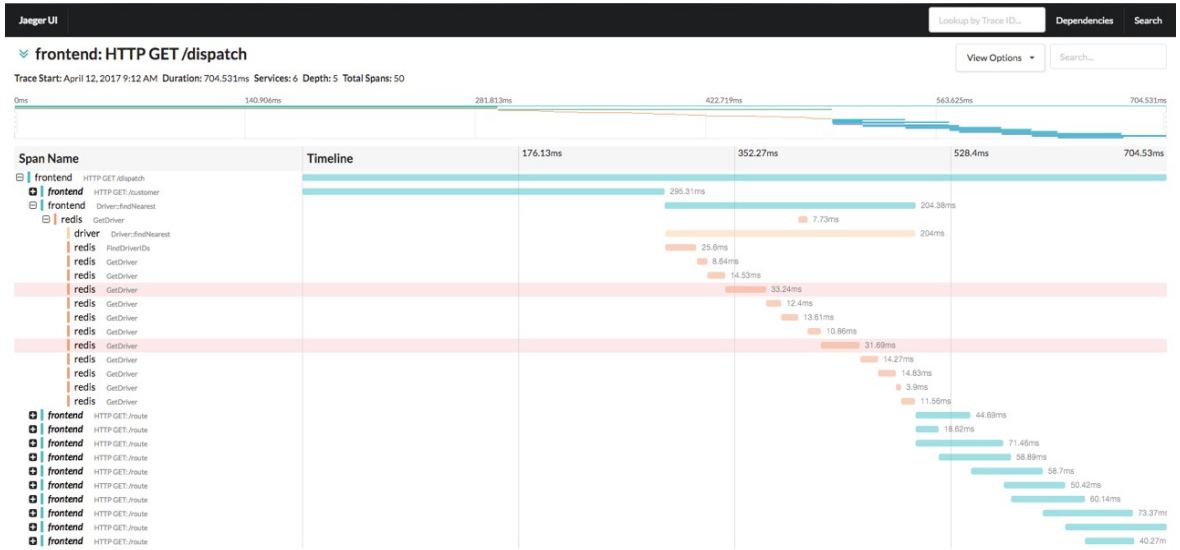
Ingestion according to our scale (millions? billions?)

Index context and tags for easy search

Visualize traces (timeline, graph)

Set alerts

Distributed Tracing DIY - Ingestion & Visualization



epsagon

Tagging Traces

Adding tags for search and aggregations:

Identifiers – user ID, customer ID, device ID
Flow control – event type, business logic
Business metrics – items in cart, minutes watched

Tracing with payload

Search an event according to:

HTTP headers or body - IDs
Key or query in SQL/NoSQL
Response payload from HTTP call

epsagon

Tracing As a Glue

Trace ⟶ Logs

Trace ⟷ Environment

Best Practices for Observability

epsagon

Best Practices for Observability

Automated setup and zero maintenance

Support any environment

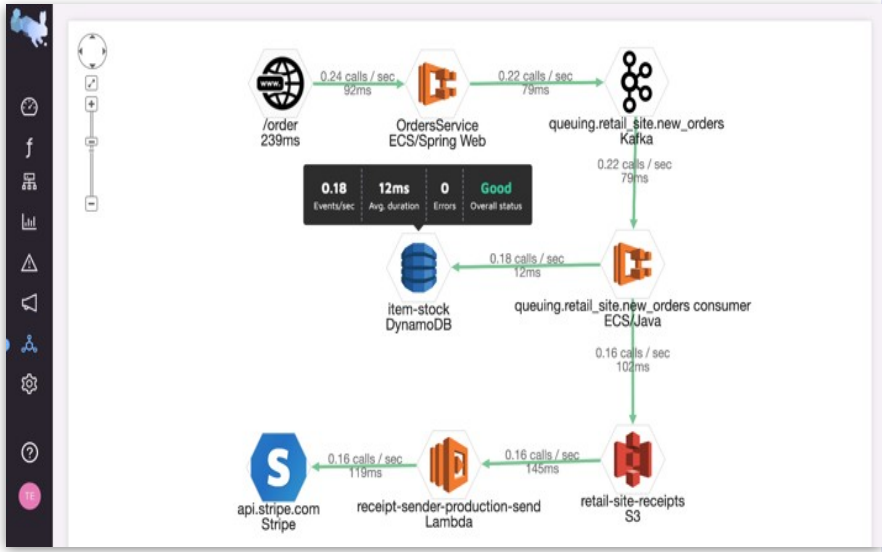
(K8s, Serverless, ECS, AKS, On Prem)

Connects **every request**

in a transaction

Search and analyze your **data**

Helps to quickly pinpoint problems



epsagon

Summary

Modern applications requires more than just monitoring

Distributed tracing becomes a crucial component in such environments

Stop implementing your own solutions unless needed

Q&A

epsagon