
cisco Live!
Let's go
#CiscoLive

The bridge to possible

Cloud Native Observability

Shannon McFarland- CCIE #5245
Distinguished Engineer, Emerging Technologies & Incubation
@eyepv6

BRKCLD-2158

Cisco Webex App

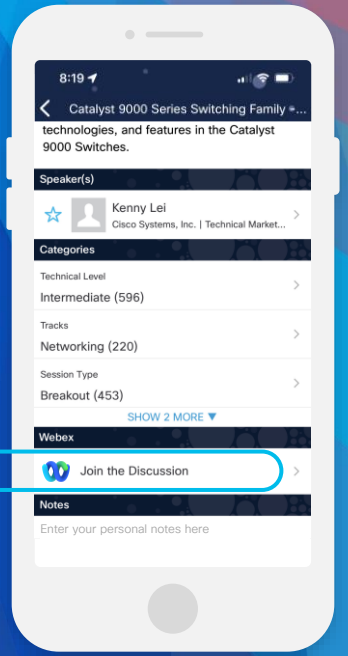
Questions?

Use Cisco Webex App to chat
with the speaker after the session

How

1. Find this session in the Cisco Live Mobile App
2. Click “Join the Discussion”
3. Install the Webex App or go directly to the Webex space
4. Enter messages/questions in the Webex space

Webex spaces will be moderated by the speaker until June 9, 2023.



https://ciscolive.ciscoevents.com/ciscolivebot/#BRKCLD-2158

Agenda

- What is Cloud Native Observability (CNO)?
- What is M.E.L.T?
  - Metrics
  - Events (and Alerts)
  - Logs
  - Traces
- Service Meshes – Built-in CNO
- Cisco Solutions for Observability
- Conclusion

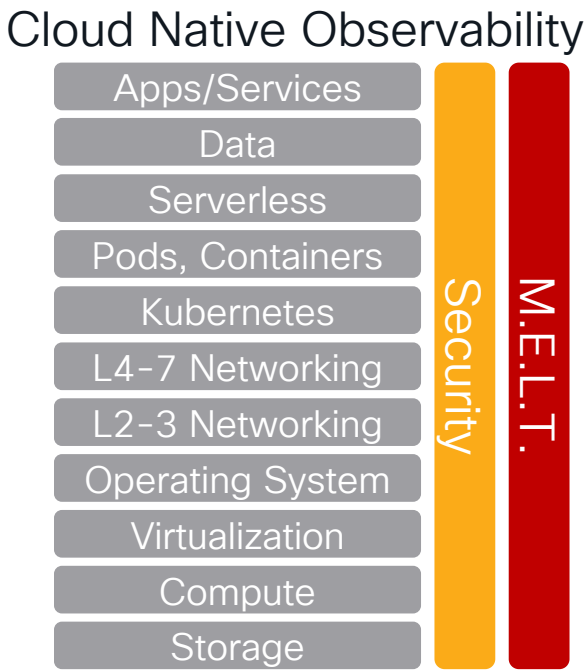
What is Cloud Native Observability?

What is Cloud Native?

- “Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. **Containers**, service meshes, **microservices, immutable** infrastructure, and declarative **APIs** exemplify this approach.
- These techniques enable **loosely coupled** systems that are **resilient, manageable,** and **observable**. Combined with robust **automation**, they allow engineers to make **high-impact changes frequently** and **predictably** with **minimal toil**.” - CNCF
- https://github.com/cncf/toc/blob/main/DEFINITION.md
- Other Cloud Native criteria include:
- Elasticity/Horizontal Scaling of Live Services
- Leveraging Common Frameworks (Application service leverages a Service Mesh)

Persona/Role – Moving from Monitoring to Observability

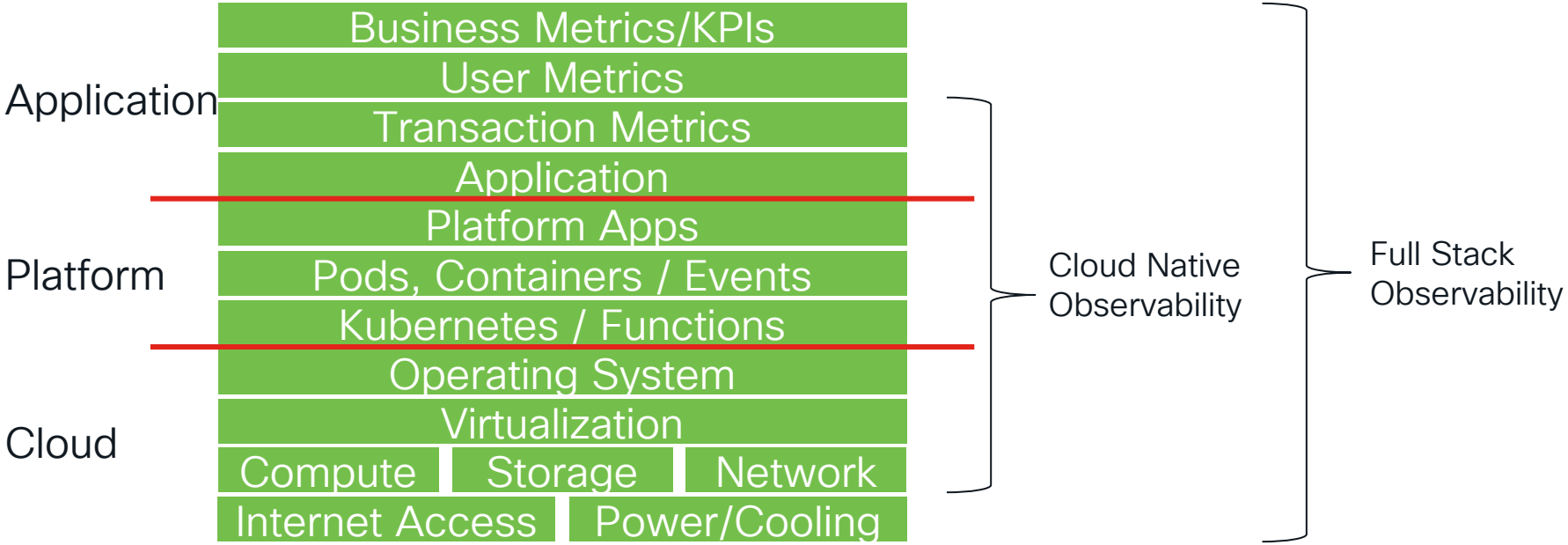
- Platform Operator/Developer
  - They may want to see the application through a Layer 7 (HTTP/gRPC**) lens
  - What is the latency/RPS/memory/CPU for each service component?
  - Where is the bottleneck?
  - Does each component adhere to an SLO?
- Data Scientist/Data Engineer
  - They may want to see very specific parts of the streaming data pipeline that is sub-component of the overall application
- CISO/Security Architect/DevSecOps
  - They may want to see the same application view as the developer, but with a specific focus on CI/CD-centric security (image scanning, code scanning) and internal/external API security



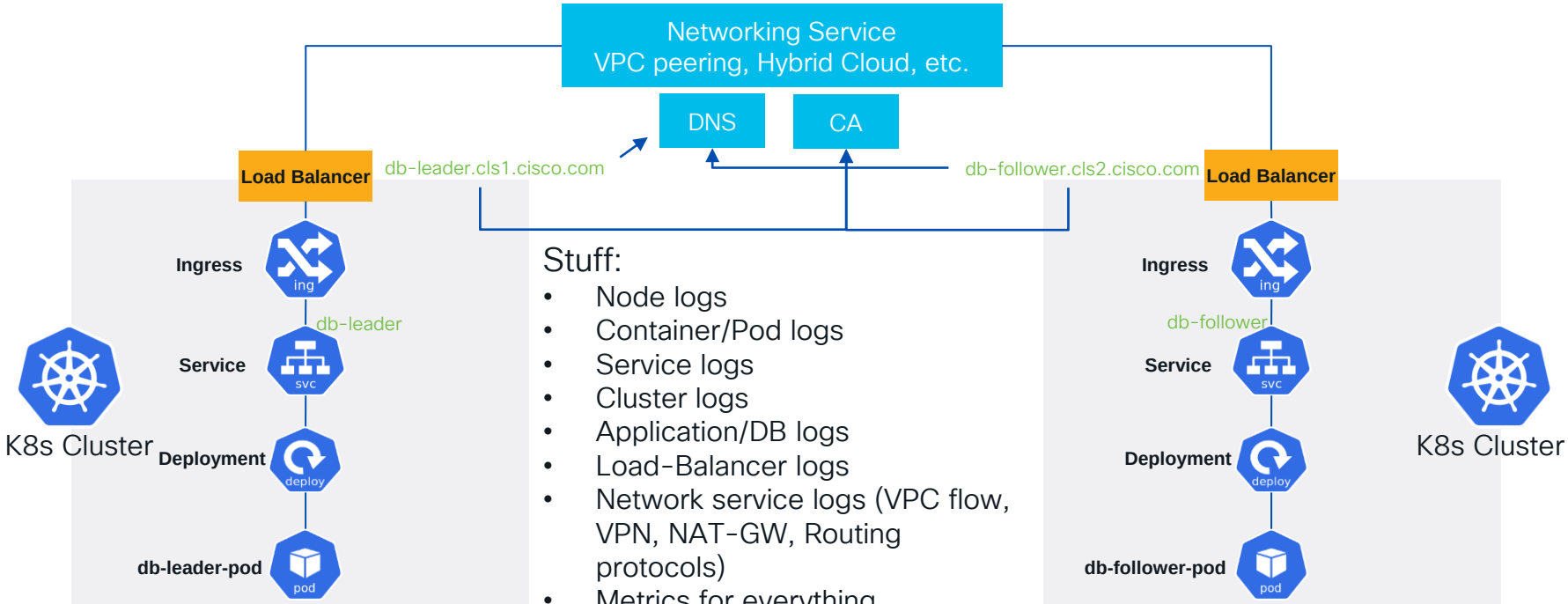
**gRPC = [google]Remote Procedure Calls

*Metrics, Events, Logs, Traces

An Example: Cloud Native Stack



Let's look at a topology



Stuff:

- Node logs
- Container/Pod logs
- Service logs
- Cluster logs
- Application/DB logs
- Load-Balancer logs
- Network service logs (VPC flow, VPN, NAT-GW, Routing protocols)
- Metrics for everything
- Events for some
- Alerts for even less
- Tracing for even less

“Full-Stack Observability” adds to traditional monitoring to support seamless digital experiences for modern architectures and teams

Monitoring

Passive detection of (sub)system “health” issues

Full-Stack Observability

Seamless digital experiences for modern application architectures and teams

- Encompasses full spectrum of **Visibility – Insights – Action** capabilities to actively understand issues and drive remediation
- Broad coverage across **application, infrastructure, networking, and security stack** with rich, real-time correlation across domains
  - Includes all systems impacting the digital experience for users
- Focused on **actively understanding and remediating issues**, enhanced with ML/AI to ultimately predict issues before they occur
- Facilitates **collaboration across modern teams** (e.g., DevOps / SRE) to achieve common objectives (e.g., SLOs)

What is M.E.L.T.?

Metrics

Metrics

Collect/Measure Data at Regular Intervals

- Expose
  - Infrastructure:
    - AWS Elastic Compute 2 (EC2) VM hosting Elastic Kubernetes Service (EKS) worker node - CPU, Memory, Storage, Network
  - Application:
    - NGINX, DB
- Collect
  - Scrape from exposed sources
- Store
  - Time-Series Database (TSDB)
- Query
  - PromQL, MQL (monitoring query language), MetricsQL (VictoriaMetrics)

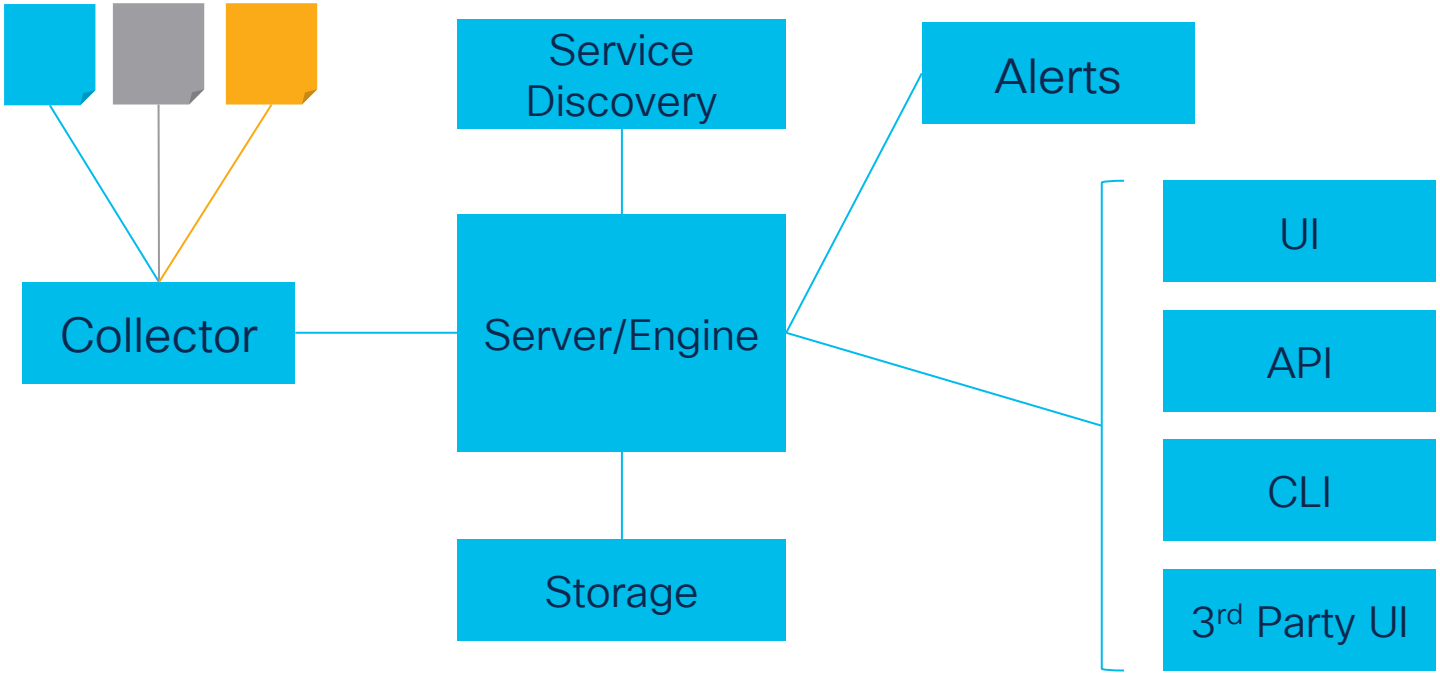
Metrics

Collect/Measure Data at Regular Intervals

- Prometheus (https://prometheus.io/) – Open-source event monitoring and alerting tool
- Thanos (https://thanos.io/) – Adds high availability, long-term storage and global query capabilities for Prometheus
- Cortex metrics (https://cortexmetrics.io/) – Adds high availability, multi-tenant, horizontally scalable and long-term storage capabilities for Prometheus
- Grafana (https://grafana.com/) – Visualization of metrics, logs and events from MANY data sources to include Prometheus
- AWS CloudWatch Metrics
- Google Cloud Metrics
- Microsoft Azure Monitor Metrics

Metrics

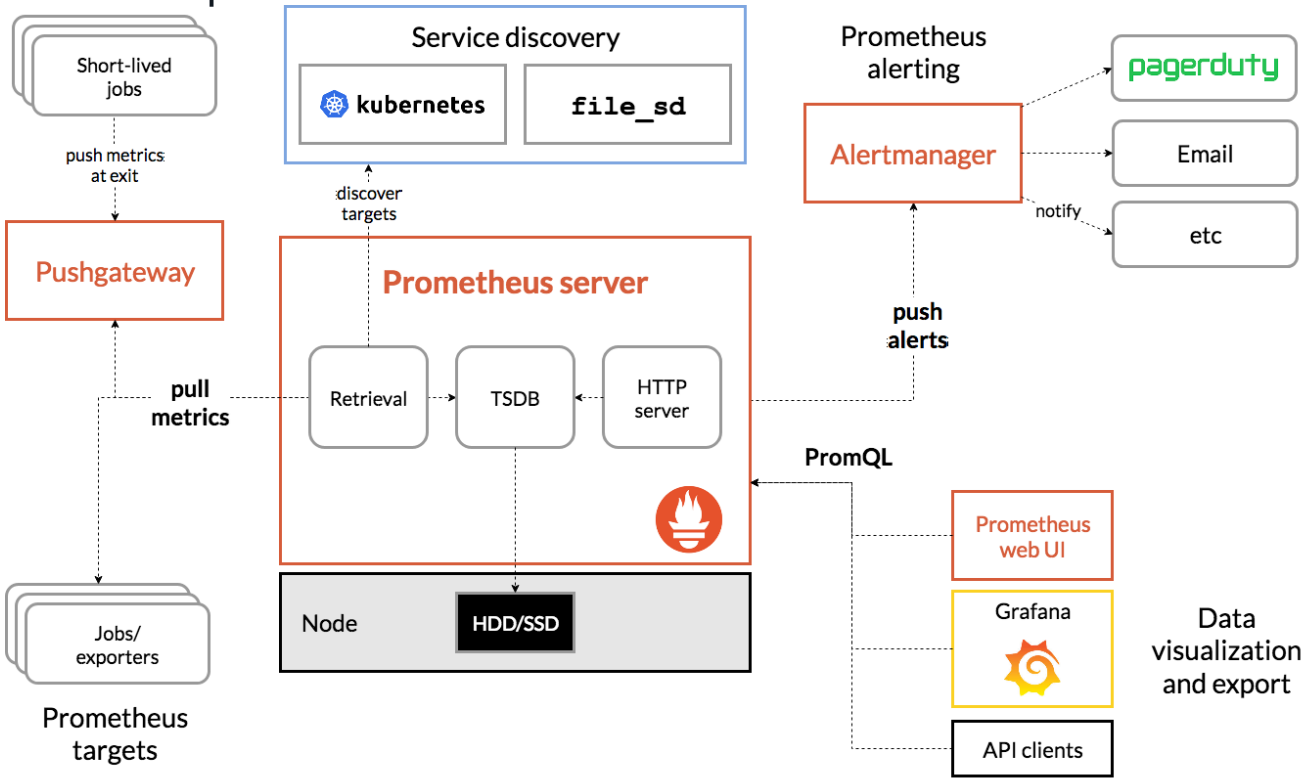
Common Architectural Components



Metrics

https://prometheus.io/docs/introduction/overview/

Prometheus Example



Events
and Alerts

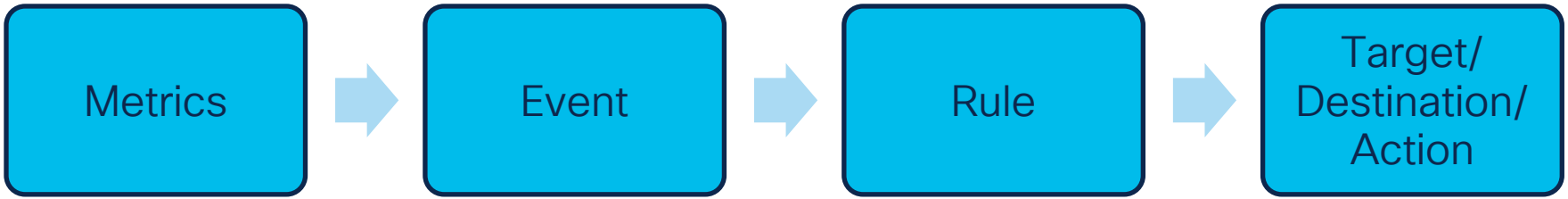
Events

- Metrics and Events are two different data types:
  - Metrics = regular/predictable data
  - Events = irregular/unpredictable data
    - Scheduled or unscheduled state changes
- Kubernetes event example:

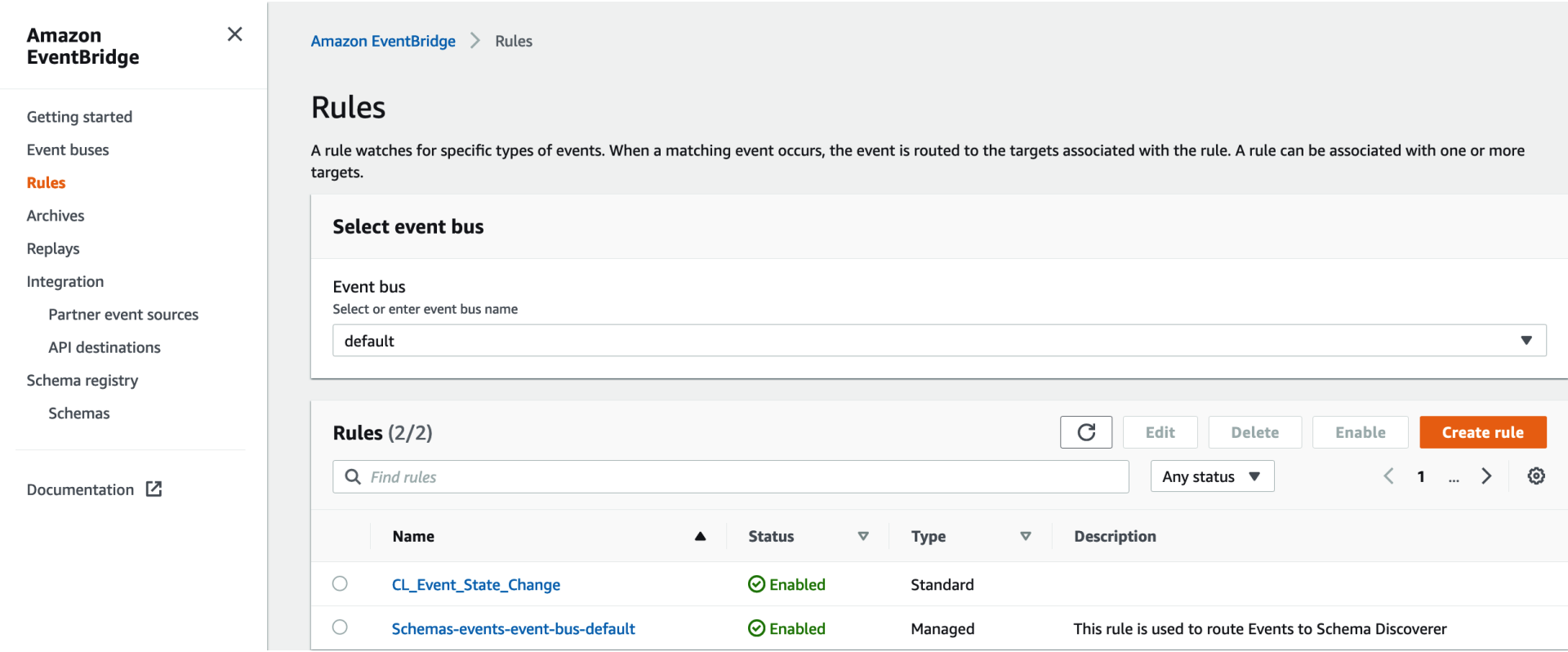
```
# kubectl get events --field-selector reason=NodeHasSufficientMemory
LAST SEEN TYPE   REASON                 OBJECT          MESSAGE
57m       Normal NodeHasSufficientMemory node/<omitted>  Node <omitted> status is now: NodeHasSufficientMemory
```

Events

- Events can be paired with other toolsets to provide a robust Event<>Action framework
  - Metrics, Pub/Sub, AI/ML, DevOps, etc.
- Event-Driven Architecture (EDA):
  - KEDA – Kubernetes-Based Event-Driven Autoscaler: https://keda.sh/
  - AWS EventBridge
  - Many, many more



Events – AWS EventBridge



Events – AWS EventBridge

CloudWatch > Log groups > /aws/events/cl_event_lg > e19753db-94e2-377e-b6e5-ad02f4f4f82f

Log events

You can use the filter bar below to search for and match terms, phrases, or values in your log events. Learn more about filter patterns

View as text | Actions | Create Metric Filter

Filter events | Clear | 1m | 30m | 1h | 12h | Custom

Timestamp	Message
	No older events at this moment. *Retry*
2021-11-23T13:43:11.000-05:00	{"version":"0","id":"5b18f90d-8547-7c50-1d82-8ad503155334","detail-type":"EC2 Instance State-change Notification","so…

Copy


```
{
    "version": "0",
    "id": "5b18f90d-8547-7c50-1d82-8ad503155334",
    "detail-type": "EC2 Instance State-change Notification",
    "source": "aws.ec2",
    "account": "[redacted]",
    "time": "2021-11-23T18:43:11Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:ec2:us-east-1:[redacted]:instance/i-0f8b96fdcd607b54f"
    ],
    "detail": {
        "instance-id": "i-0f8b96fdcd607b54f",
        "state": "terminated"
    }
}
```


No newer events at this moment. *Auto retry paused.* *Resume*

Alerts

- Alerts – A predefined trigger based on a threshold or event
  - Static alert example:
    - HTTP Request Per Second (RPS) of 90% triggers alert to Slack
    - Kubernetes Node isn't ready for 1 minutes (Prometheus Example)

```
- alert: KubernetesNodeReady
    expr: kube_node_status_condition{condition="Ready",status="true"} == 0
    for: 1m
    labels:
      severity: critical
    annotations:
      summary: Kubernetes Node ready (instance {{ $labels.instance }})
      description: "Node {{ $labels.node }} unready \n  VALUE = {{ $value }}\n  LABELS = {{ $labels }}"
```

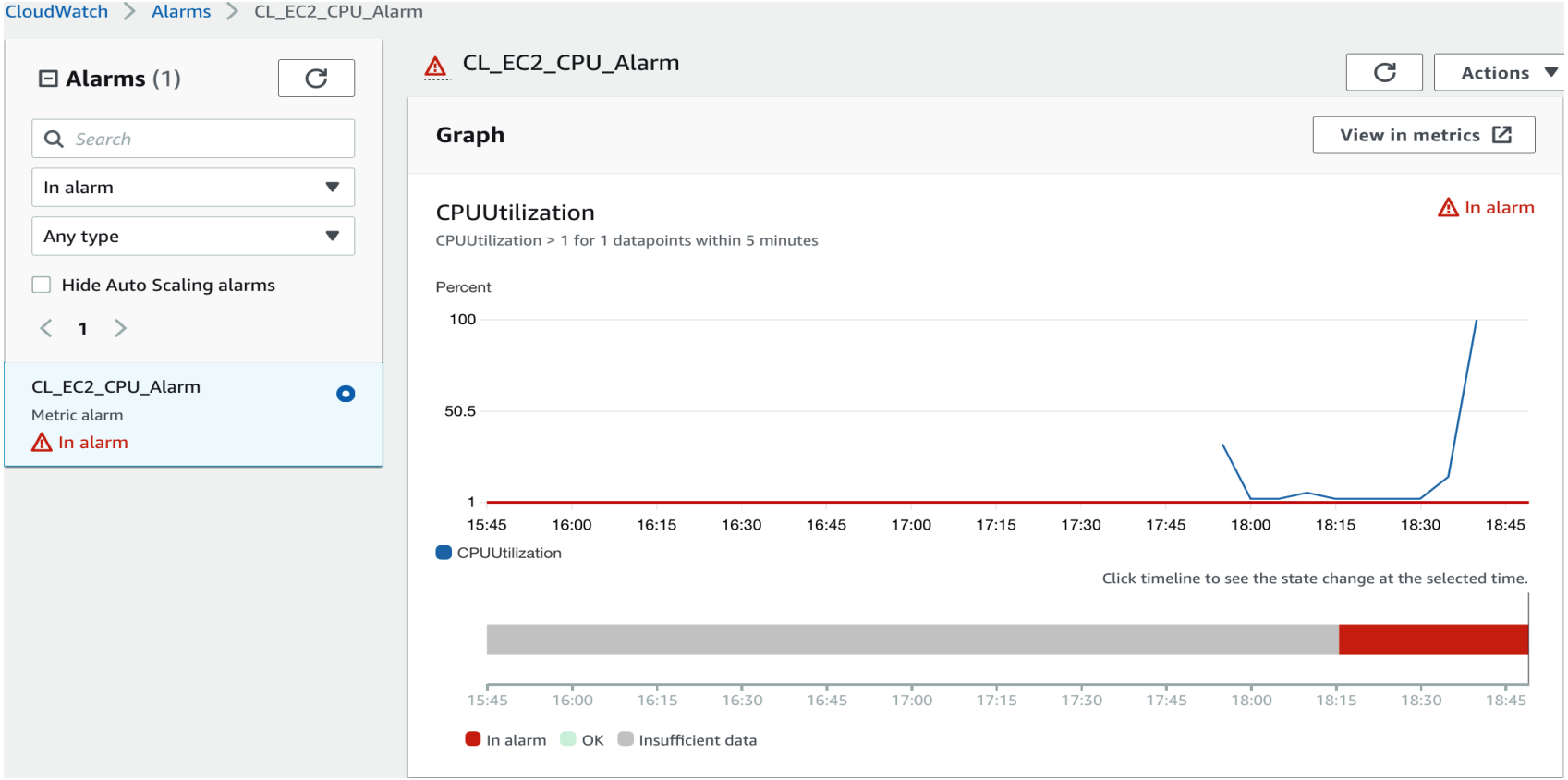
Alerts – AWS CloudWatch Alerts

AWS EC2
-Per-Instance
-CPUUtilization

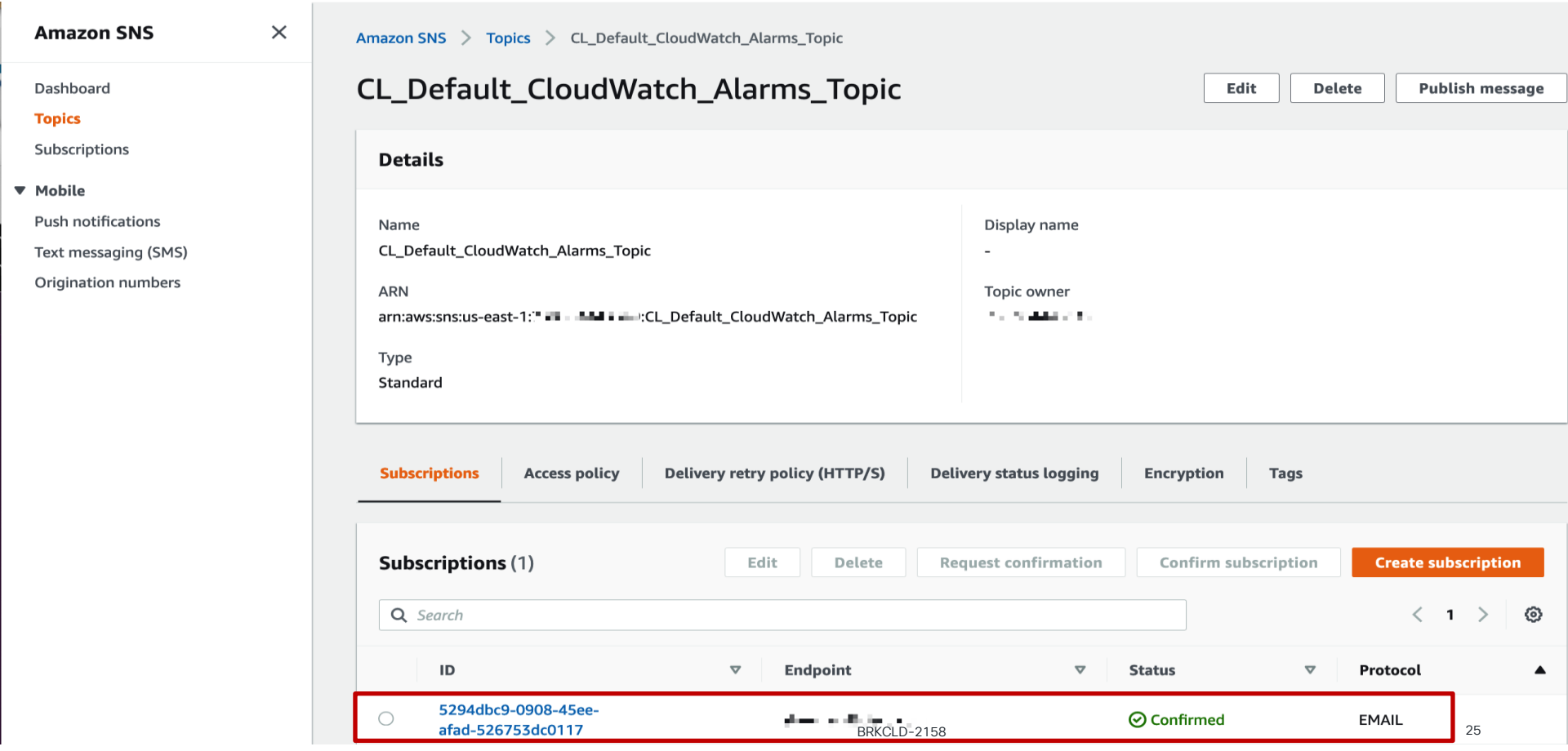
Threshold: Static
-CPU
-Greater than 90%

Notification: In Alarm
-Publish to AWS SNS Topic
-SNS Topic > Email
-(Auto scaling)
-(EC2 action)
-(System Manager action)

Alerts – AWS CloudWatch Example



Alerts – AWS CloudWatch Example



Alerts – AWS CloudWatch Example

You are receiving this email because your Amazon CloudWatch Alarm "CL_EC2_CPU_Alarm" in the US East (N. Virginia) region has entered the ALARM state, because "Threshold Crossed: 1 out of the last 1 datapoints [6.22950819672132 (23/11/21 18:10:00)] was greater than the threshold (1.0) (minimum 1 datapoint for OK -> ALARM transition)." at "Tuesday 23 November, 2021 18:15:36 UTC".

View this alarm in the AWS Management Console:
https://us-east-1.console.aws.amazon.com/cloudwatch/deeplink.js?region=us-east-1#alarmsV2:alarm/CL_EC2_CPU_Alarm

Alarm Details:
- Name: CL_EC2_CPU_Alarm
- Description:
- State Change: INSUFFICIENT_DATA -> ALARM
- Reason for State Change: Threshold Crossed: 1 out of the last 1 datapoints [6.22950819672132 (23/11/21 18:10:00)] was greater than the threshold (1.0) (minimum 1 datapoint for OK -> ALARM transition).
- Timestamp: Tuesday 23 November, 2021 18:15:36 UTC
- AWS Account: [redacted]
- Alarm Arn: arn:aws:cloudwatch:us-east-1:[redacted]:alarm:CL_EC2_CPU_Alarm

Threshold:
- The alarm is in the ALARM state when the metric is GreaterThanThreshold 1.0 for 300 seconds.

Monitored Metric:
- MetricNamespace: AWS/EC2
- MetricName: CPUUtilization
- Dimensions: [InstanceId = i-0f8b96fdcd607b54f]
- Period: 300 seconds
- Statistic: Maximum
- Unit: not specified
- TreatMissingData: missing

State Change Actions:
- OK:
- ALARM: [arn:aws:sns:us-east-1:[redacted]:CL_Default_CloudWatch_Alarms_Topic]
- INSUFFICIENT_DATA:

Logs

Logs

- Nearly everything in a Cloud Native (or other) environment produces logs in some form
- Logging has tremendous potential, but it is very complex to manage all the sources and then derive value out of what the logs say
- Collection and data formatting should be simple, but it isn't:
  - Currently, K8s doesn't enforce uniform structure for log messages*
  - You can't safely assume all log formats are in JSON
  - You may need to transform logs
- There are MANY gotchas on storage, forwarding, rotation – We don't have time for that today

*https://github.com/kubernetes/enhancements/tree/master/keps/sig-instrumentation/1602-structured-logging

Logs

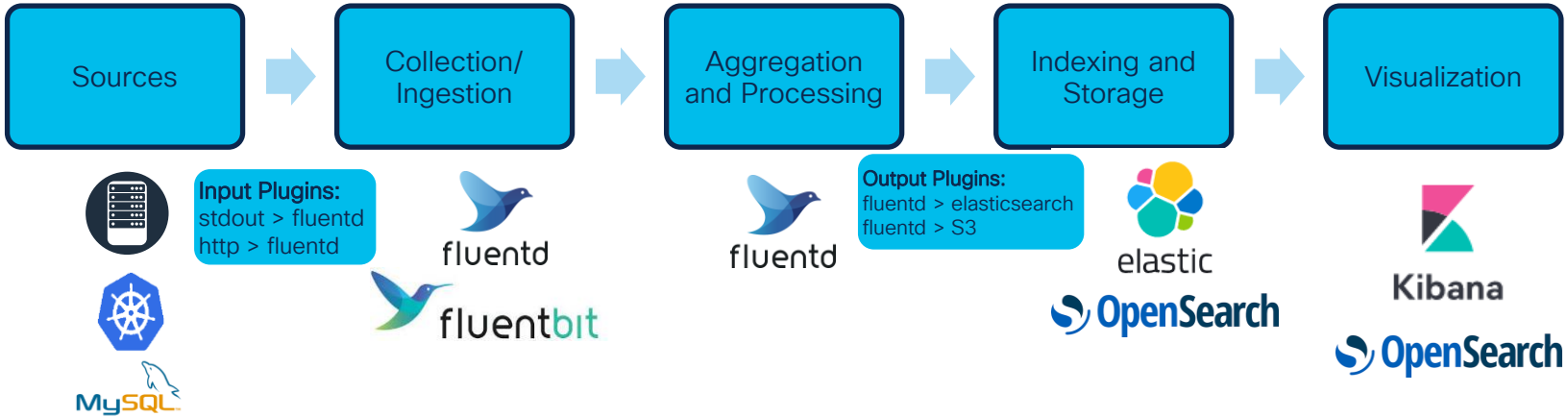
Common Components

- **Fluentd/Fluent Bit** – Open-Source log collection and processing (Bit is for highly resource-constrained environments)
- **Logstash** – Open-Source log collection and processing
- **Elasticsearch, OpenSearch, Grafana Loki**, etc – Aggregation, search and analytics
- **Kibana, OpenSearch Dashboard, Grafana Loki**, etc – Dashboard
- AWS CloudWatch/CloudTrail, GCP Cloud Logging, Microsoft Azure Cloud Monitoring
- Many more…

Common Stacks

- **EFK** – Elasticsearch, Fluentd/bit, Kibana
- **ELK** – Elasticsearch, Logstash, Kibana
- **OFO** – OpenSearch, Fluentd/bit, OpenSearch Dashboard
- ENDLESS combination of tools

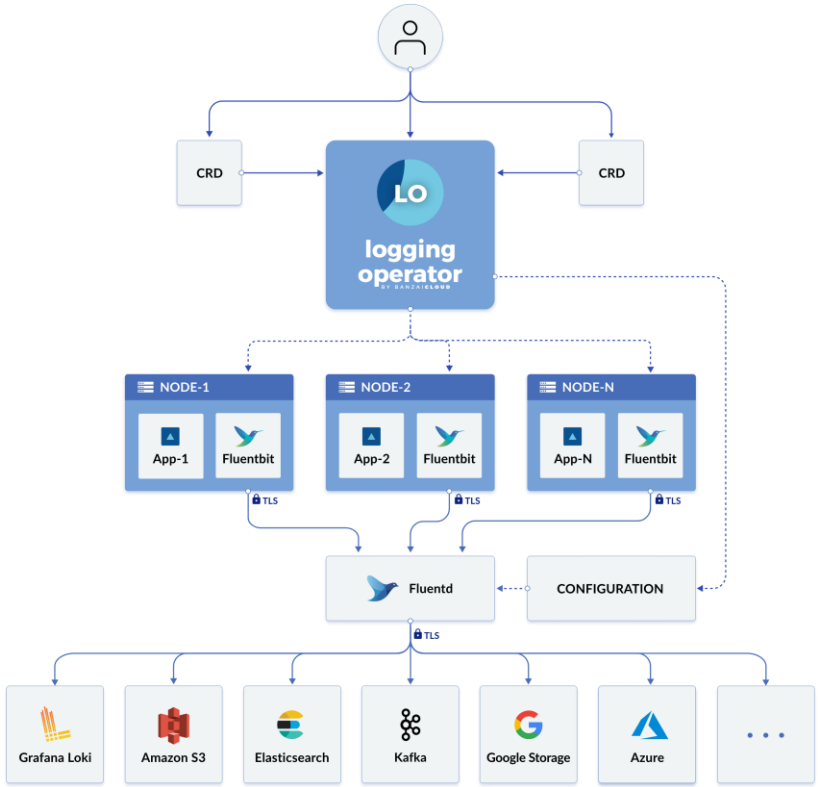
Logs – Common Architecture Components



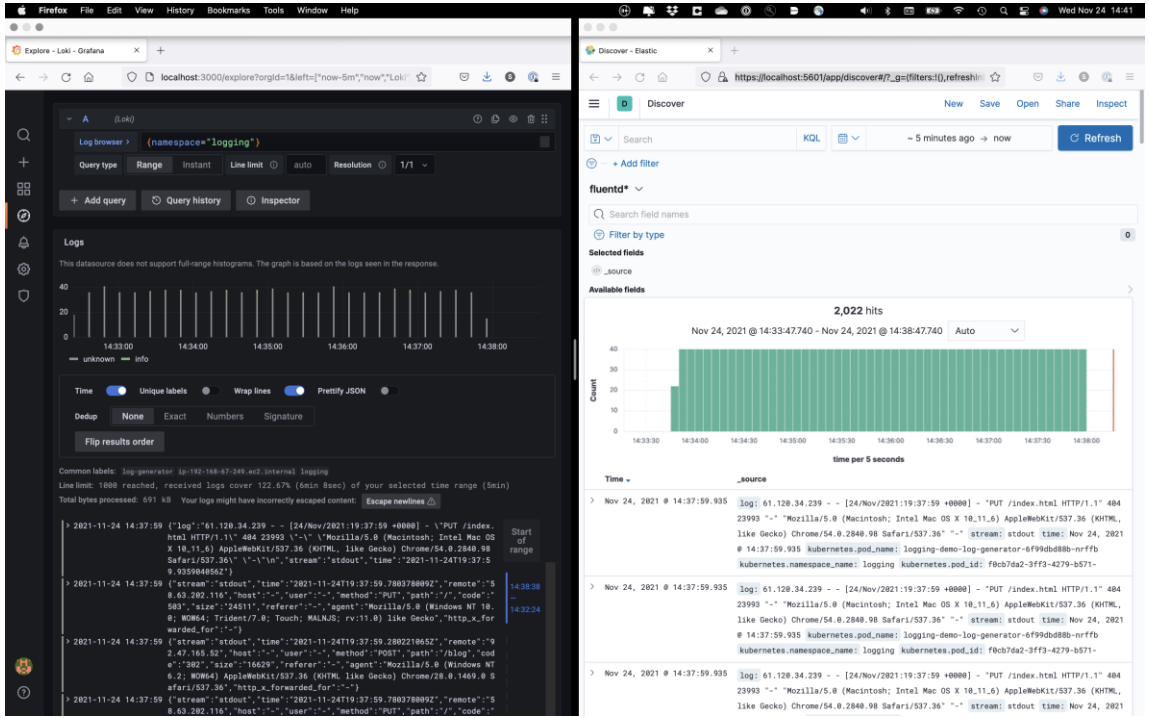
Building this by hand and as independent components is

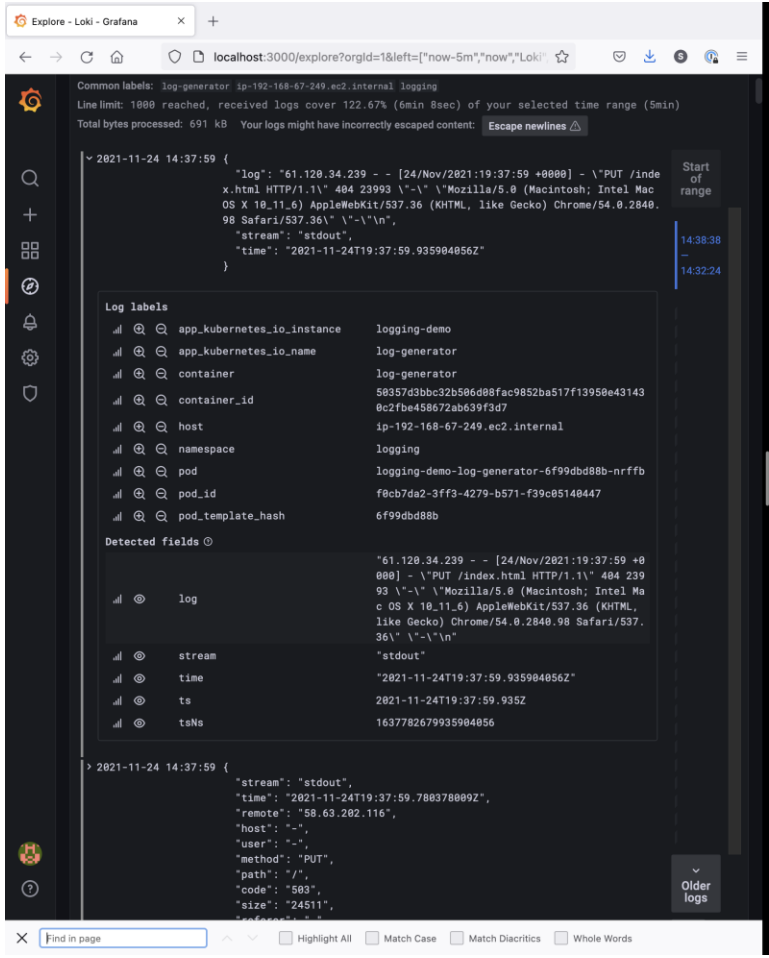
Logs

- Do things the easier way:
  - Kubernetes:
    - Use Operators – Cisco Open-Source Logging Operator
    - https://github.com/kube-logging/logging-operator
  - Fluentd/FluentBit and source configuration
  - Security (TLS, RBAC, etc)
  - Output configuration
    - AWS CloudWatch, S3, Azure Storage, GCP Storage, Elasticsearch, Grafana Loki, Kafka, etc.



Logging Operator – Example – One source, multi-outputs – NGINX to Elasticsearch/Kibana & Grafana Loki





Traces

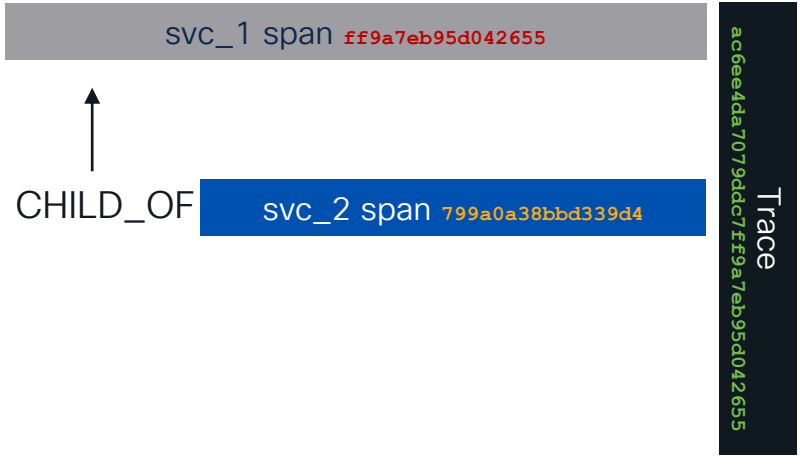
Traces

- Distributed tracing helps with:
  - Service mapping (topology)
  - Bottlenecks/Latency/Drops in a distributed architecture (network, microservice, etc.)
- Example projects/solutions:
  - OpenTelemetry - Combo of OpenCensus + OpenTracing – Library-based collection
  - Service Meshes – Istio, Linkerd, etc. – Sidecar-based collection
  - Jaeger – Visualize traces
  - W3C TraceContext/B3 TraceContext – Bringing some sanity to the format of a trace ID
  - AWS X-ray, GCP Cloud Trace, Azure Monitor (Application Insights) – Tracing libraries and visualization service

Traces

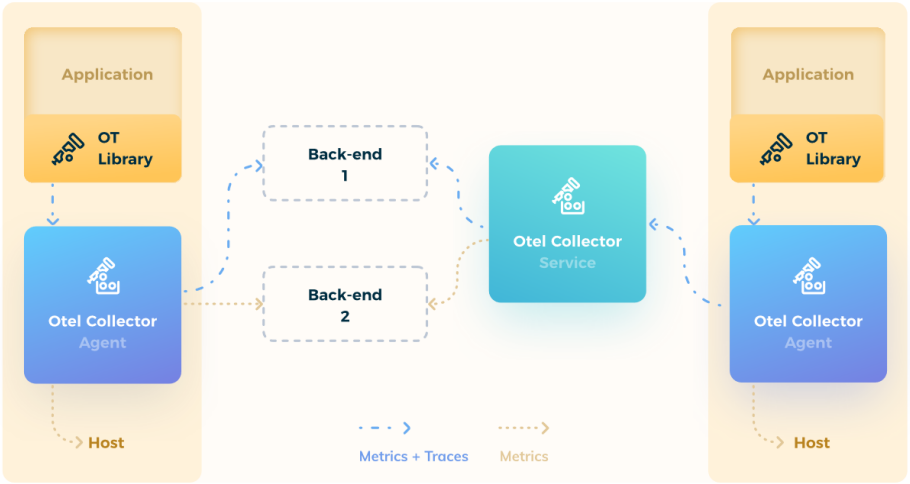
- Primer:
  - A “span” is the foundational element of a distributed trace – it represents an individual unit of work
  - A “span” can reference another span and when assembled, you have a “trace”
  - “Context propagation” – Correlate trace metadata across service boundaries
  - Not using a standard method for trace context propagation can lead to VERY painful deployments and VERY expensive workarounds


```
"traceID": "ac6ee4da7079ddc7ff9a7eb95d042655",
          "spans": [
              {
                  "traceID": "ac6ee4da7079ddc7ff9a7eb95d042655",
                  "spanID": "ff9a7eb95d042655",
                  "operationName": "frontpage.smm-demo.svc.cluster.local:8080/*",
                  "references": [],
                  "startTime": 1638308084933526,
                  "duration": 803461,
              {
                  "traceID": "ac6ee4da7079ddc7ff9a7eb95d042655",
                  "spanID": "799a0a38bbd339d4",
                  "operationName": "frontpage.smm-demo.svc.cluster.local:8080/*",
                  "references": [
                      {
                          "refType": "CHILD_OF",
                          "traceID": "ac6ee4da7079ddc7ff9a7eb95d042655",
                          "spanID": "ff9a7eb95d042655"
```

OpenTelemetry (OTel)

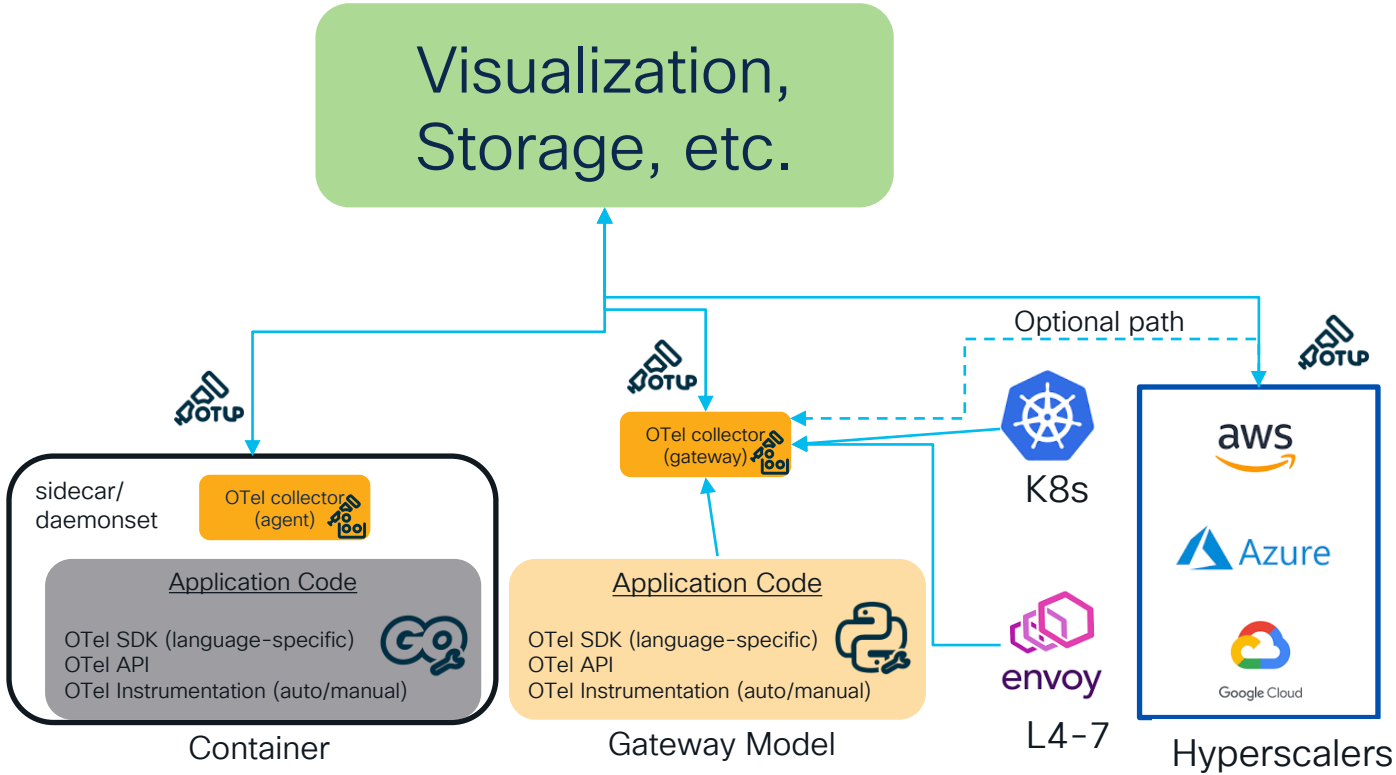
https://opentelemetry.io/



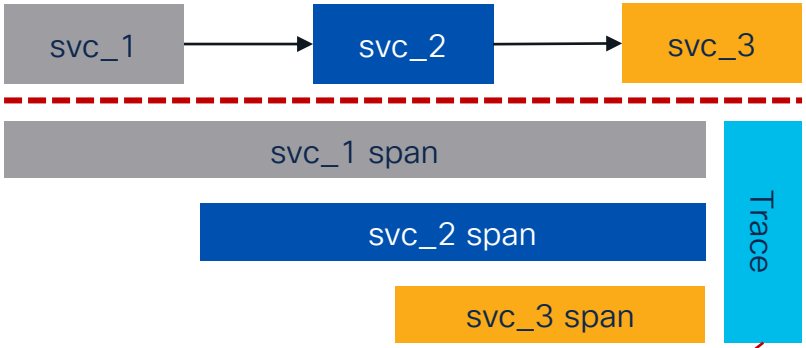
- Language-specific libraries
- Supports: Traces, Metrics, Logs
- The Collector recognizes multiple Trace Context formats
- Different form factors for the Collector
- https://techblog.cisco.com/blog/getting-started-with-opentelemetry

REFERENCE ARCHITECTURE

Example: OpenTelemetry Components in Action



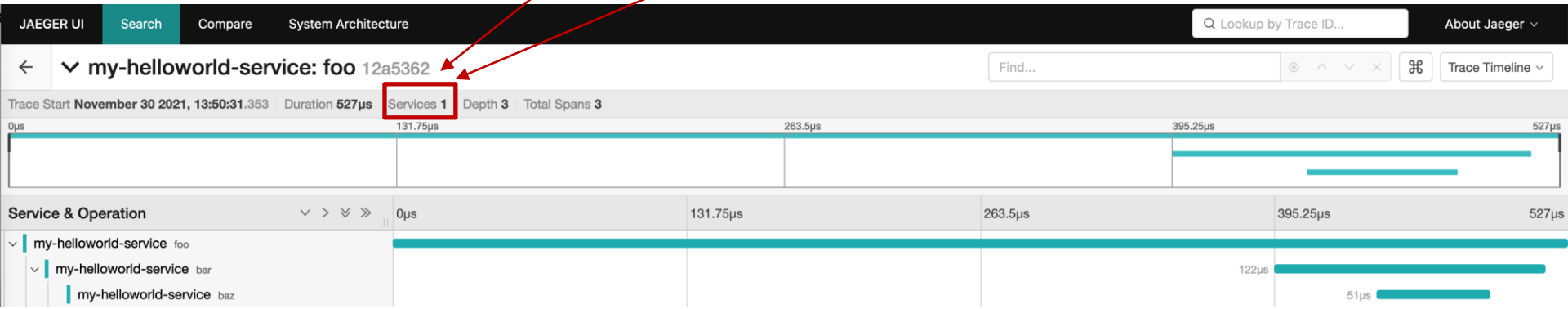
OpenTelemetry + Jaeger



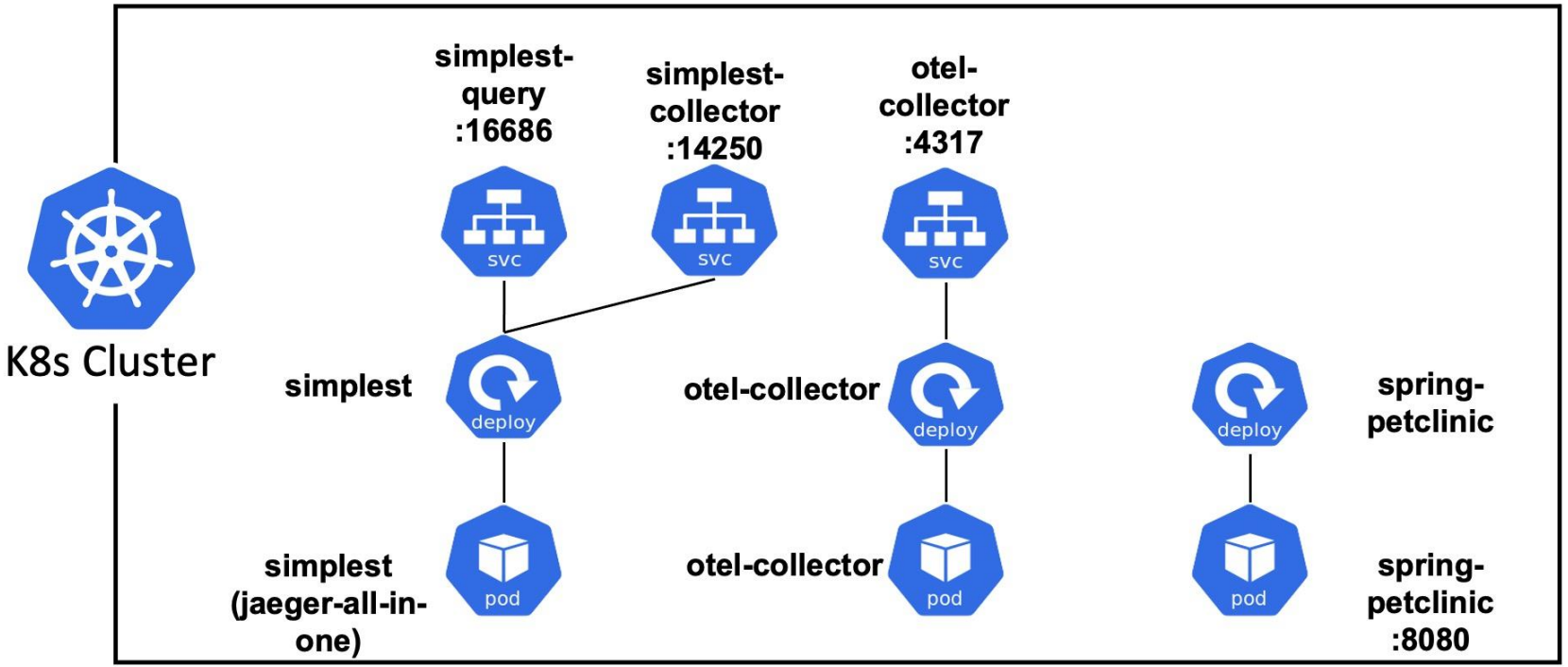
foo

Tags

internal.span.format	proto
otel.library.name	__main__
otel.library.version	
service.name	my-helloworld-service
span.kind	internal
telemetry.sdk.language	python
telemetry.sdk.name	opentelemetry
telemetry.sdk.version	1.7.1



Tracing Deployment Example



Example OpenTelemetry Deploy - 1

Deploy a test KinD cluster

```
# kind create cluster
```

Deploy Cert Manager

```
# kubectl apply -f https://github.com/cert-manager/cert-manager/releases/download/v1.11.0/cert-manager.yaml
```

Deploy the Jaeger Operator

```
# kubectl create namespace observability
# kubectl create -f https://github.com/jaegertracing/jaeger-operator/releases/download/v1.44.0/jaeger-operator.yaml -n
observability
```

Deploy the Jaeger All-in-One Strategy

```
# kubectl apply -f - <<EOF
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simplest
EOF
```

Deploy the OpenTelemetry Operator

```
# kubectl apply -f https://github.com/open-telemetry/opentelemetry-operator/releases/latest/download/opentelemetry-operator.yaml
```

Example OpenTelemetry Deploy - 2

Deploy the OTel Collector

```
# kubectl apply -f - <<EOF
apiVersion: opentelemetry.io/v1alpha1
kind: OpenTelemetryCollector
metadata:
  name: otel
spec:
  config: |
    receivers:
      otlp:
        protocols:
          grpc:
          http:
    processors:
    exporters:
      logging:
      jaeger:
          endpoint: "simplest-collector:14250"
          tls:
              insecure: true
    service:
      pipelines:
        traces:
          receivers: [otlp]
          processors: []
          exporters: [jaeger]
EOF
```

Deploy the OTel Java Auto-instrumentation CRD

```
# kubectl apply -f - <<EOF
apiVersion: opentelemetry.io/v1alpha1
kind: Instrumentation
metadata:
  name: my-instrumentation
spec:
  exporter:
    endpoint: http://otel-collector:4317
  propagators:
    - tracecontext
    - baggage
    - b3
  sampler:
    type: parentbased_traceidratio
    argument: "0.25"
  java:
    image: ghcr.io/open-telemetry/opentelemetry-
operator/autoinstrumentation-java:latest
  nodejs:
    image: ghcr.io/open-telemetry/opentelemetry-
operator/autoinstrumentation-nodejs:latest
  python:
    image: ghcr.io/open-telemetry/opentelemetry-
operator/autoinstrumentation-python:latest
EOF
```

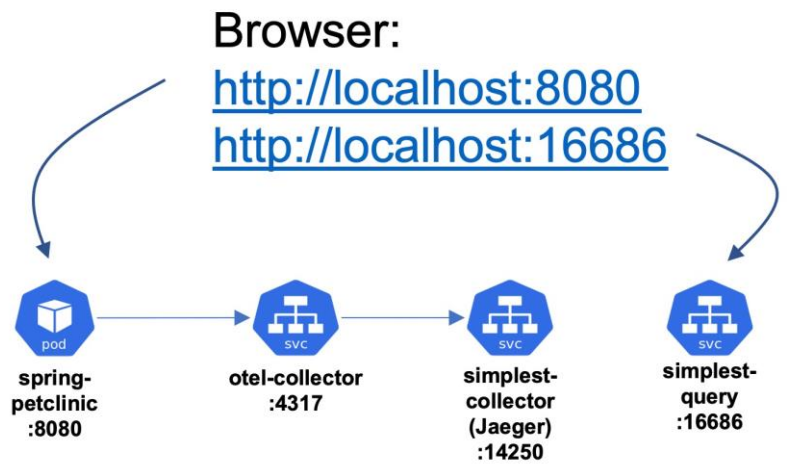
Example OpenTelemetry Deploy - 3

Deploy the Spring Pet Clinic service

```
# kubectl apply -f - <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  name: spring-petclinic
spec:
  selector:
    matchLabels:
      app: spring-petclinic
  replicas: 1
  template:
    metadata:
      labels:
        app: spring-petclinic
      annotations:
        sidecar.opentelemetry.io/inject: "true"
        instrumentation.opentelemetry.io/inject-java: "true"
    spec:
      containers:
      - name: app
        image: ghcr.io/pavolloffay/spring-petclinic:latest
EOF
```

```
# kubectl port-forward deployment.apps/spring-petclinic 8080:8080
```

```
# kubectl port-forward svc/simplest-query 16686:16686
```



Example OpenTelemetry - Validation

```
# kubectl logs deployment.apps/otel-collector
. . . <output summarized>
builder/receivers_builder.go:73 Receiver started. {"kind": "receiver", "name": "otlp"}
. . .
jaegerexporter@v0.41.0/exporter.go:186 State of the connection with the Jaeger Collector backend {"kind": "exporter",
"name": "jaeger", "state": "READY"}
```

Browser-to-Jaeger Query Service (16686) and Otel Collector-to-Jaeger (14250)

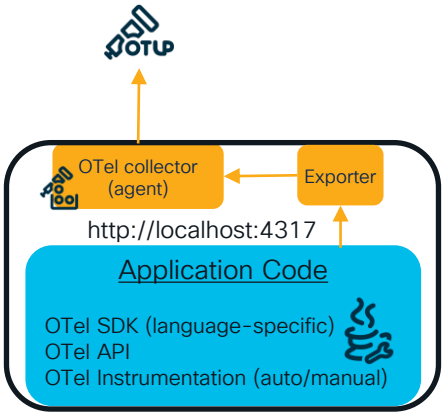
```
# kubectl exec -it simplest-797dd8fc67-6l9q4 -- netstat -at

Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 localhost:47280         localhost:16686         ESTABLISHED
tcp        0      0 simplest-797dd8fc67-6l9q4:14250 10-244-0-11.otel-collector.default.svc.cluster.local:49768 ESTABLISHED
```

Browser-to-Pet Clinic UI (8080) and Java Otel library-to-Otel Collector (4317)

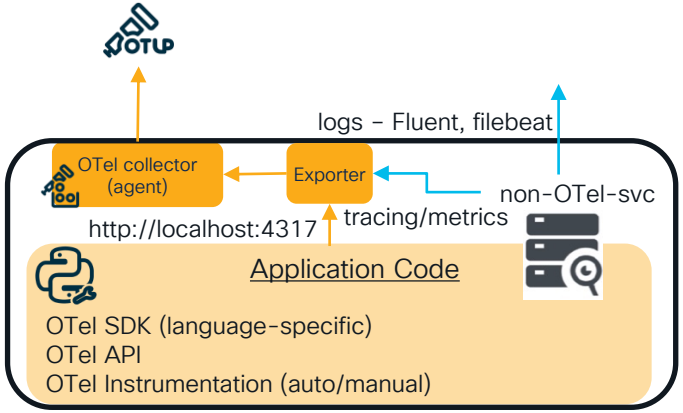
```
root@spring-petclinic-6d569df946-9f26m:/# netstat -at
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 localhost:55028         localhost:8080          ESTABLISHED
tcp        0      0 10.244.0.12:34208       otel-collector.def:4317 ESTABLISHED
```

Hybrid MELT Support



Java Agent – MLT over OTLP

```
java -javaagent:path/to/opentelemetry-
javaagent-all.jar \ -jar myapp.jar
```



Python Agent – Tracing over OTLP

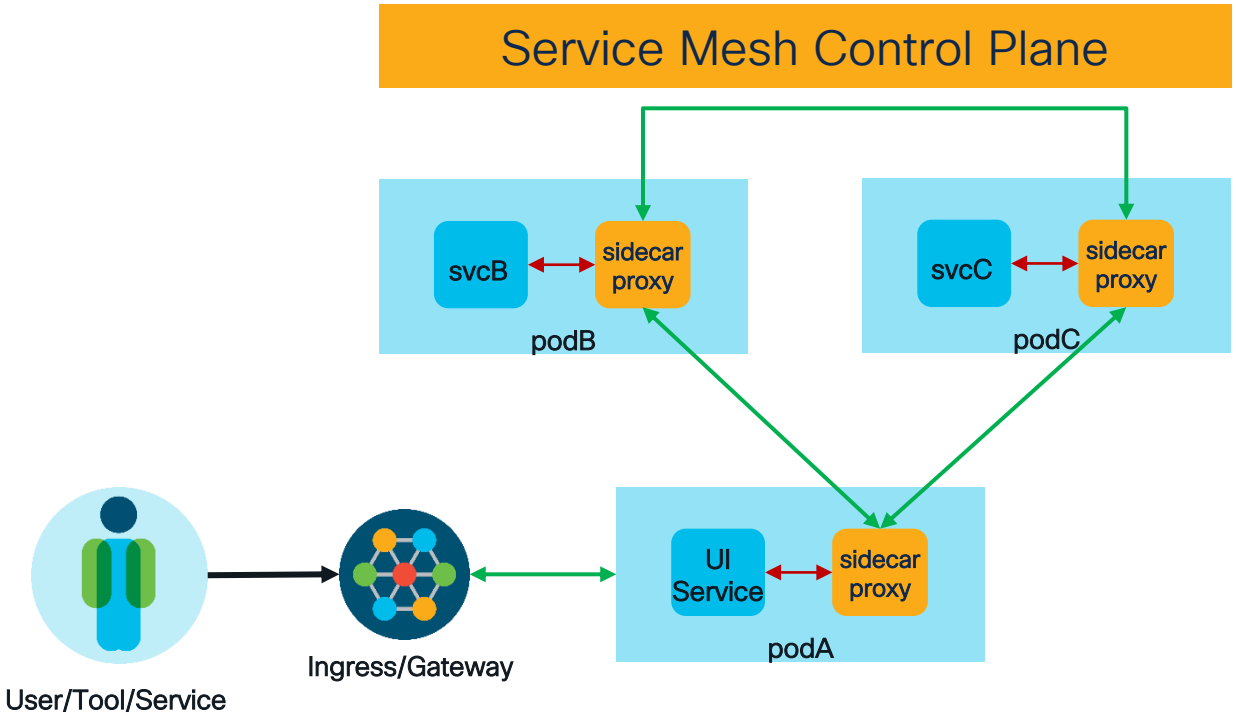
```
pip install opentelemetry-api
pip install opentelemetry-sdk
pip install opentelemetry-exporter-{exporter}
pip install -U aws-xray-sdk
```

Application Instrumentation Options

- Library/SDK-based:
  - OpenTelemetry
  - Cisco/ApplicationDynamics
  - AWS X-ray, GCP Cloud Trace, Azure Monitor (Application Insights)
  - Many others
- Sidecar-based:
  - Service Meshes – Istio, Linkerd, Consul Connect, KongHQ, etc.

Service Mesh

What is a Service Mesh?



- Infrastructure layer for service-to-service communication
- Can use a mesh of sidecar proxies:
  - Can inspect API transactions at Layer 7 and 4 (TCP)
  - Intelligent routing rules can be applied between endpoints
- Allow for tracing and some application instrumentation without the need to add code/libraries/SDK to the application

Service Mesh Observability with Proxies

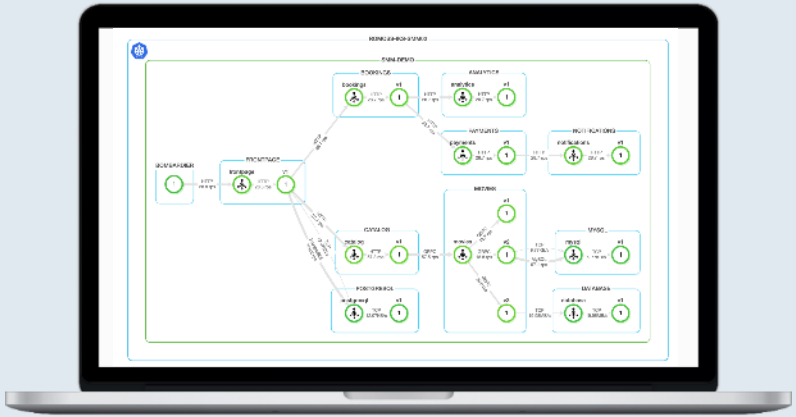
- Service Meshes provide observability via their sidecar proxies
- Observability info:
  - Mesh-specific metrics
  - Application-specific metrics
  - Distributed traces: Layer 4-7: TCP, HTTP, gRPC
  - Access logs (mesh and apps)
- In-mesh dashboards: Istio/Kiali, Linkerd/Viz, Cisco Calisti
- Out-of-mesh dashboards: Prometheus, Grafana, Jaeger, EFK, etc.

Cisco Solutions

https://www.cisco.com/c/en/us/solutions/full-stack-observability.html

Cisco Calisti

https://calisti.app/



Operationalize the Service Mesh

Multi-cloud, multi-cluster connectivity and observability
Connect any on-prem and public cloud together

Simplifies service mesh management
Single pane of glass, in depth metrics

Policy-based app networking & security
Policy management for DevOps teams

Apache Kafka on Kubernetes & Service Mesh
Lifecycle management of Apache Kafka and components

Traffic management ensures smooth app updates

Complete application and health **observability**

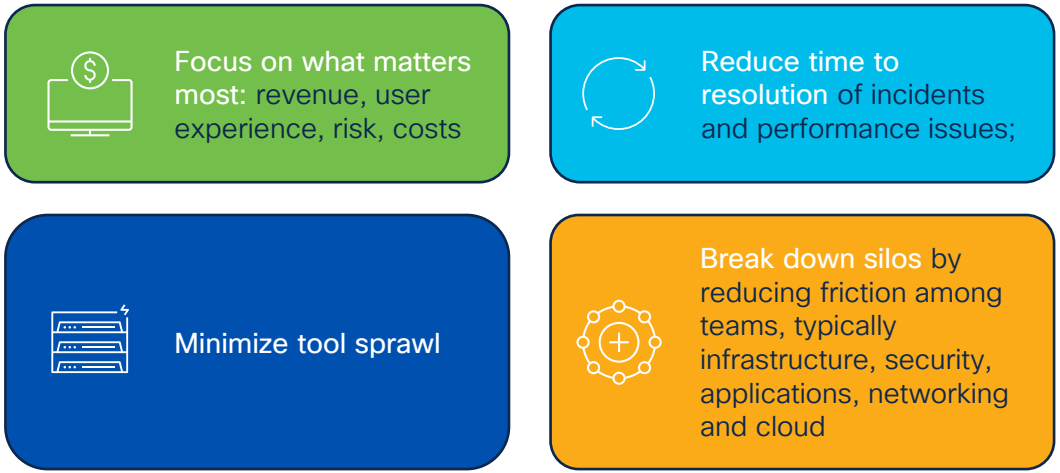
Security at all layers between clusters and clouds

Demo

Cisco Full-Stack Observability

Full Stack Observability (FSO) is a requirement for business to deliver the most **optimal and secure experience** to users and applications.

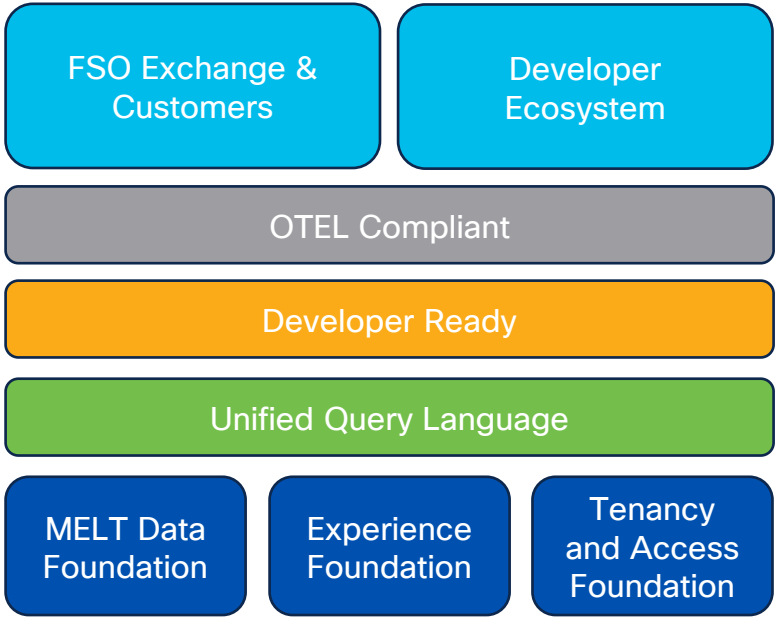
Cisco Full-Stack Observability brings together data from multiple operations domains to provide **unified visibility**, derive **real-time insights** and recommend **actions** helping to:



Cisco FSO Platform

Empowers an extensible observability ecosystem

- Fully extensible entity-based data modeling
- Natively supports OpenTelemetry
- Anchored on MELT
- Provides Unified Query Language (UQL)



Summary

- There are a lot of things to keep track of and a lot of tools to help you do so:
  - Metrics, Events, Logs, Traces
  - Proprietary solutions, open-source solutions
  - Most solutions (vendor and OSS) do a handful of things well - most of the time up to you to 'integrate' them
- Next-gen solutions such as Cisco Full Stack Observability will reduce/remove the burden of you having to stitch together various tools to gain visibility to – derive value from and take action on your data
- Check out Cisco FSO: https://www.cisco.com/c/en/us/solutions/full-stack-observability.html
- Start working with Cisco Calisti!
  - https://calisti.app/

Fill out your session surveys!

Attendees who fill out a minimum of four session surveys and the overall event survey will get **Cisco Live-branded socks** (while supplies last)!

Attendees will also earn 100 points in the **Cisco Live Challenge** for every survey completed.

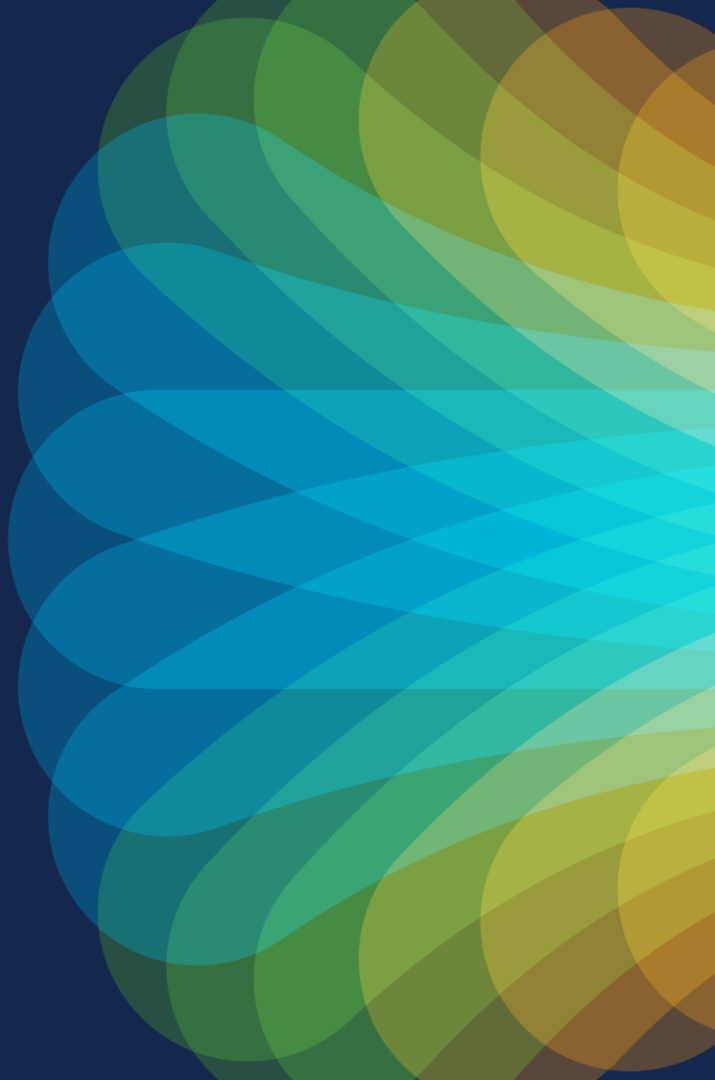
These points help you get on the leaderboard and increase your chances of winning daily and grand prizes

Continue your education

- Visit the Cisco Showcase for related demos
- Book your one-on-one Meet the Engineer meeting
- Attend the interactive education with DevNet, Capture the Flag, and Walk-in Labs
- Visit the On-Demand Library for more sessions at www.CiscoLive.com/on-demand

The bridge to possible

Thank you



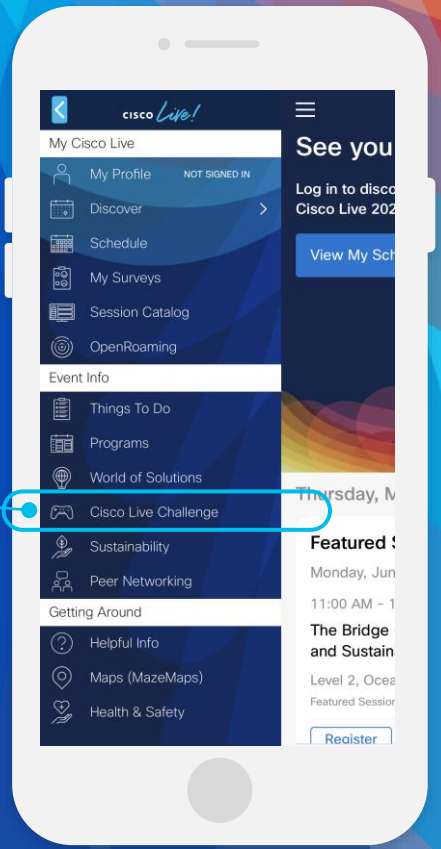
Cisco Live
Challenge

Gamify your Cisco Live experience!

Get points for attending this session!

How:

1. Open the Cisco Events App.
2. Click on 'Cisco Live Challenge' in the side menu.
3. Click on View Your Badges at the top.
4. Click the + at the bottom of the screen and scan the QR code:



cisco Live!


cisco Live!
Let's go
#CiscoLive