Simplifying Multi Cloud Observability

SIRI VARMA VEGIRAJU

Siri Varma Vegiraju

- Software Engineer
- Freelance Contributor
- Book Reviewer
- LinkedIn: /sirivarma
- Email: siri.varma@outlook.com
- Twitter: @siri_vegiraju

Agenda

1. What is Multi Cloud and why it is gaining popularity
2. What is Observability
3. Why the Approach to Observability must change for Single and Multi-Cloud Environments
4. How can we simplify Multi Cloud observability
5. Conclusion

What is Multi Cloud and why it is gaining popularity

Multi-cloud refers to using multiple cloud services from different providers within a single architecture

98% of enterprises use or plan to use multi cloud.

Motivations ?

Data Sovereignty

Cloud Vendor lock-in concerns

What is Observability

The ability to measure the current state of the system.

Three Pillars: Metrics, Logs and Traces.

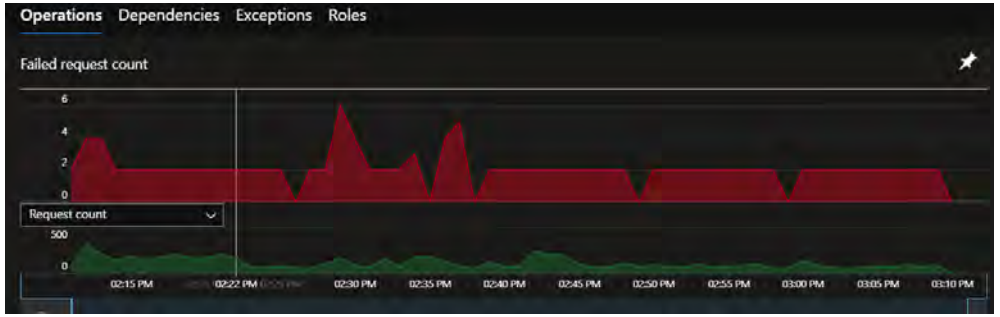
Metrics, Logs and Traces

- **Metrics** are numerical measurements of system performance and behavior, such as CPU usage, response time, or error rate.
- **Logs** are the archival or historical records of system events and errors, which can be plain text, binary, or structured with metadata.
- **Traces** are the representations of individual requests or transactions that flow through a system, which can help identify bottlenecks, dependencies, and root causes of issues.

Why the Approach to Observability must change for Single and Multi-Cloud Environments

Single Cloud World

```java
// Azure Monitor
public static void sendToAzureMonitor(String metricName, double value) {
    // Implement Azure Monitor logic
    MetricsQueryClient client = new MetricsQueryClient(/* authentication details */);
    MetricData metricData = new MetricData()
            .withName(metricName)
            .withValue(value)..
    QueryResponse response = client.query(/* query details */);
}
```



Emit Metric

Cloud Provider dashboard

Multi Cloud World

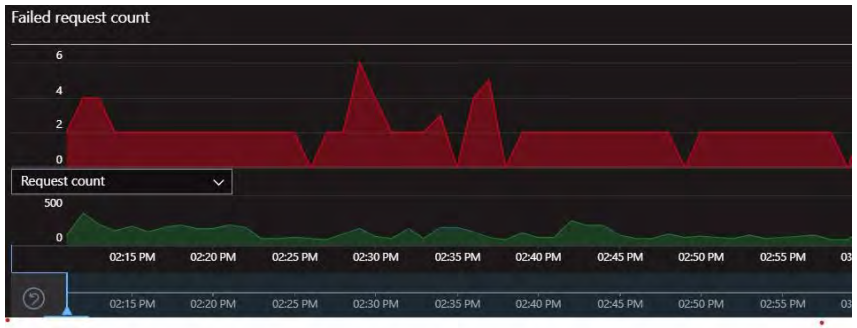
```java
// AWS CloudWatch
public static void sendToAWSCloudWatch(String metricName, double value) {
    AmazonCloudWatch cloudWatch = AmazonCloudWatchClientBuilder.defaultClient();
    PutMetricDataRequest request = new PutMetricDataRequest()
            .withNamespace("MyNamespace")
            .withMetricData(new MetricDatum()..
    PutMetricDataResult result = cloudWatch.putMetricData(request);
}

// Azure Monitor
public static void sendToAzureMonitor(String metricName, double value) {
    // Implement Azure Monitor logic
    MetricsQueryClient client = new MetricsQueryClient(/* authentication details */);
    MetricData metricData = new MetricData()
            .withName(metricName)
            .withValue(value)..
    QueryResponse response = client.query(/* query details */);
}

// Google Cloud Monitoring
public static void sendToGCP(String metricName, double value) {
    try (MetricServiceClient client = MetricServiceClient.create()) {
        MonitoredResource resource = MonitoredResource.newBuilder().build();
        Metric metric = Metric.newBuilder().setType(metricName).build();
        MetricValue metricValue = MetricValue.newBuilder().setValue(value).build();
    }
}

// Oracle Cloud Monitoring
public static void sendToOracle(String metricName, double value) {
    MonitoringClient client = new MonitoringClient(/* authentication details */);
    PutMetricsDataRequest request = PutMetricsDataRequest.builder()
            .metricData(/* build metric data */)
            .namespace("MyNamespace")
            .build();...
    PutMetricsDataResponse response = client.putMetricsData(request);
```

Multiple SDKs



Multiple Provider Dashboards

Ton of Complexity

- Not having a unified experience.
  - Maintaining different SDKs.
  - Understanding various cloud providers Observability implementations.
  - Diagnosing issues across multiple cloud environments can be more challenging.
  - Tracking observability costs across multiple clouds can be complex.
- Training and Skills

How can we simplify Multi cloud Observability ?

What to do we need ?

- Unified Experience
  - Emit Metrics, Logs and Traces
  - Maintaining dashboards
- Vendor agnostic.

Cloud Native Open Telemetry

Provides single open-source standard to export Metrics, Logs and Traces

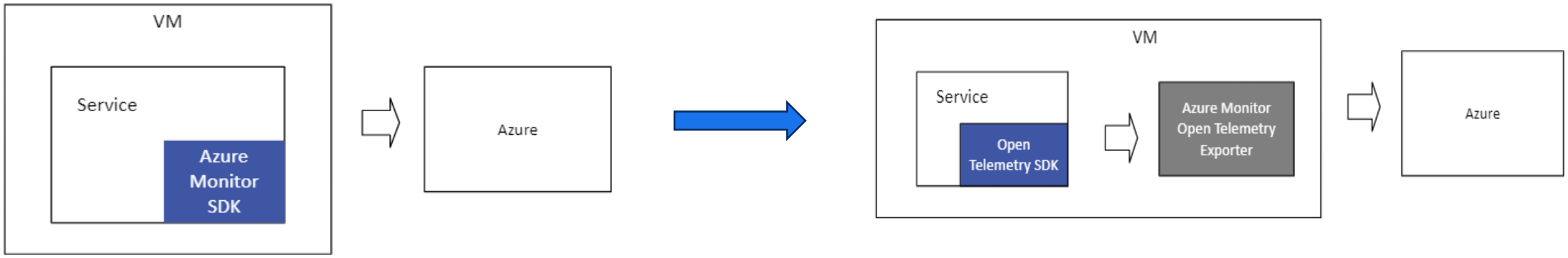
Major cloud providers like AWS, Azure, GCP and Oracle support this CNCF project.

Vendor agnostic.

Components in Open Telemetry

- Specification
- Semantics
- Open Telemetry SDK – Go, C#, Java
- Exporter
  - Azure Monitor, AWS, Google Cloud, Data Dog, Zepkin, Prometheus and many more
- Backend
  - Jaeger, AWS, Azure, Google Cloud and many more.

How does Open Telemetry work ?



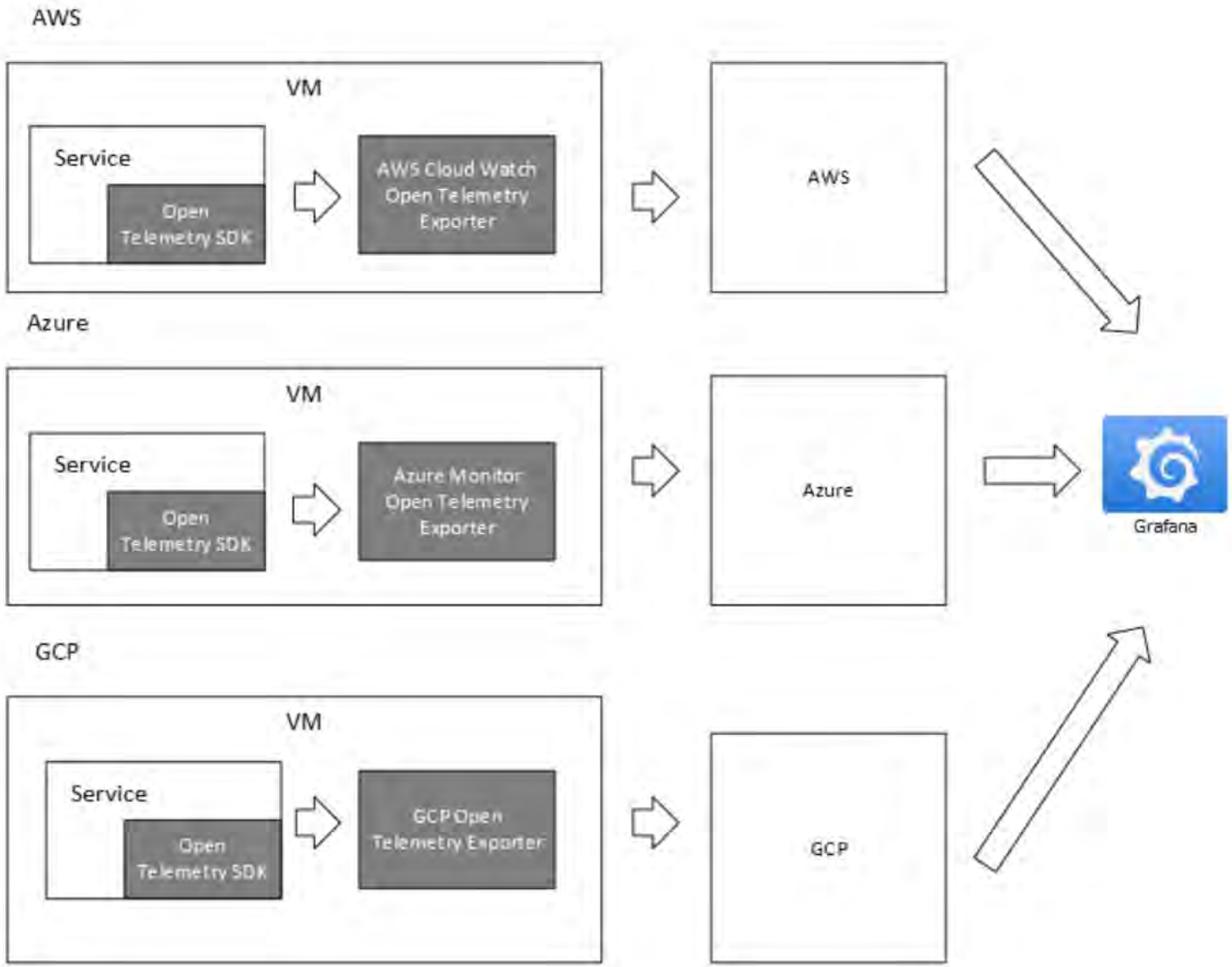
Before Open Telemetry

After Open Telemetry

```
var meter = new Meter("MyCompany.MyProduct.MyLibrary", "1.0");
var requestCounter = meter.CreateCounter<long>("requests_processed");

// Set up OpenTelemetry for metrics
using var meterProvider = Sdk.CreateMeterProviderBuilder()
    .AddMeter("MyCompany.MyProduct.MyLibrary") // Specify which meters to collect metrics from
    .AddConsoleExporter()                      // Export metrics to console
    .AddPrometheusExporter(opt =>
    {
        opt.HttpListenerPrefixes = new string[] { "http://localhost:9464/" }; // Prometheus endpoint
    })
    .AddOtlpExporter(opt =>
    {
        opt.Endpoint = new Uri("http://localhost:4317"); // Set OTLP endpoint
    })
    .Build();

// Emit metrics
requestCounter.Add(1);
```


AWS
VM
Service
Open Telemetry SDK
AWS Cloud Watch Open Telemetry Exporter
AWS
Azure
VM
Service
Open Telemetry SDK
Azure Monitor Open Telemetry Exporter
Azure
Grafana
GCP
VM
Service
Open Telemetry SDK
GCP Open Telemetry Exporter
GCP

Conclusion

- Vendor agnostic
- Other Open-source tooling
  - Prometheus
  - Loki
  - Zepkin

Thank you