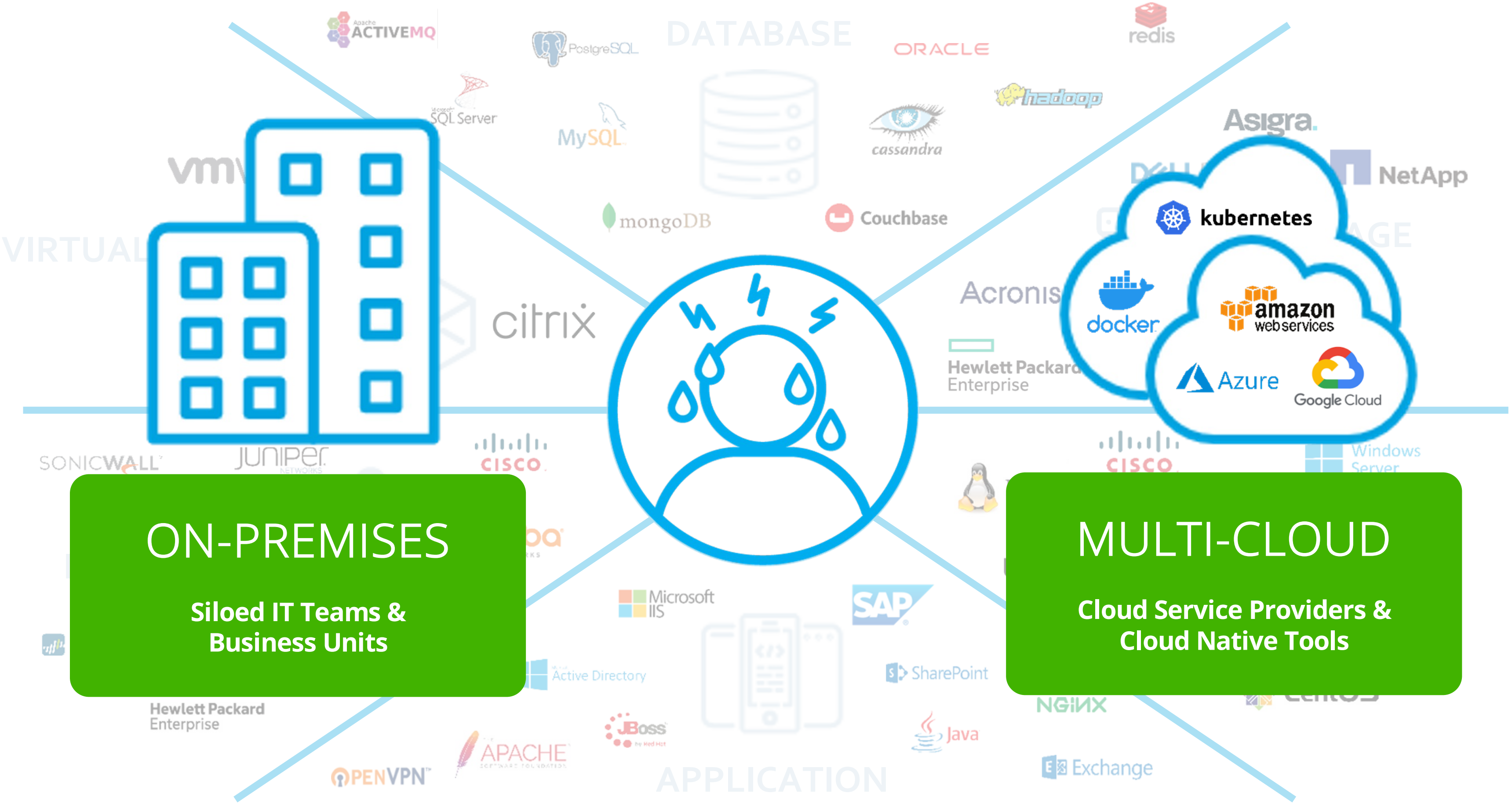
OpsRamp

a Hewlett Packard Enterprise company

Observability in Action

Unify your Log, Tracing, Event, & Performance Telemetry Data
to enrich your IT Operations Monitoring & Event Management

Why OpsRamp?



ON-PREMISES

Siloed IT Teams & Business Units

MULTI-CLOUD

Cloud Service Providers & Cloud Native Tools

- Growing Expectations
- Lack of Visibility
- Disconnected Legacy Tools
- Reactive
- Escalating Costs

ITOM Use Cases

Productive, Efficient, and Proactive

Hybrid Observability

See more

- Discover, observe, and optimize your hybrid IT environment
- Bring server, storage, network, virtualized, cloud, containerized, and application visibility and performance together in a single, unified point of view and control

Intelligent Automation

Do more

- Automate IT processes to make every moment easier
- Help ITOps teams to be more efficient, increase service quality, reduce redundant activities, and ensure audit or compliance policies

AI-driven event & incident management

Know more

- Detect and resolve issues faster
- Leverage the power of machine learning and process automation to manage critical alerts, detect incidents, and resolve them proactively

Part 1:

Journey to Observability

Let’s Take A Closer Look

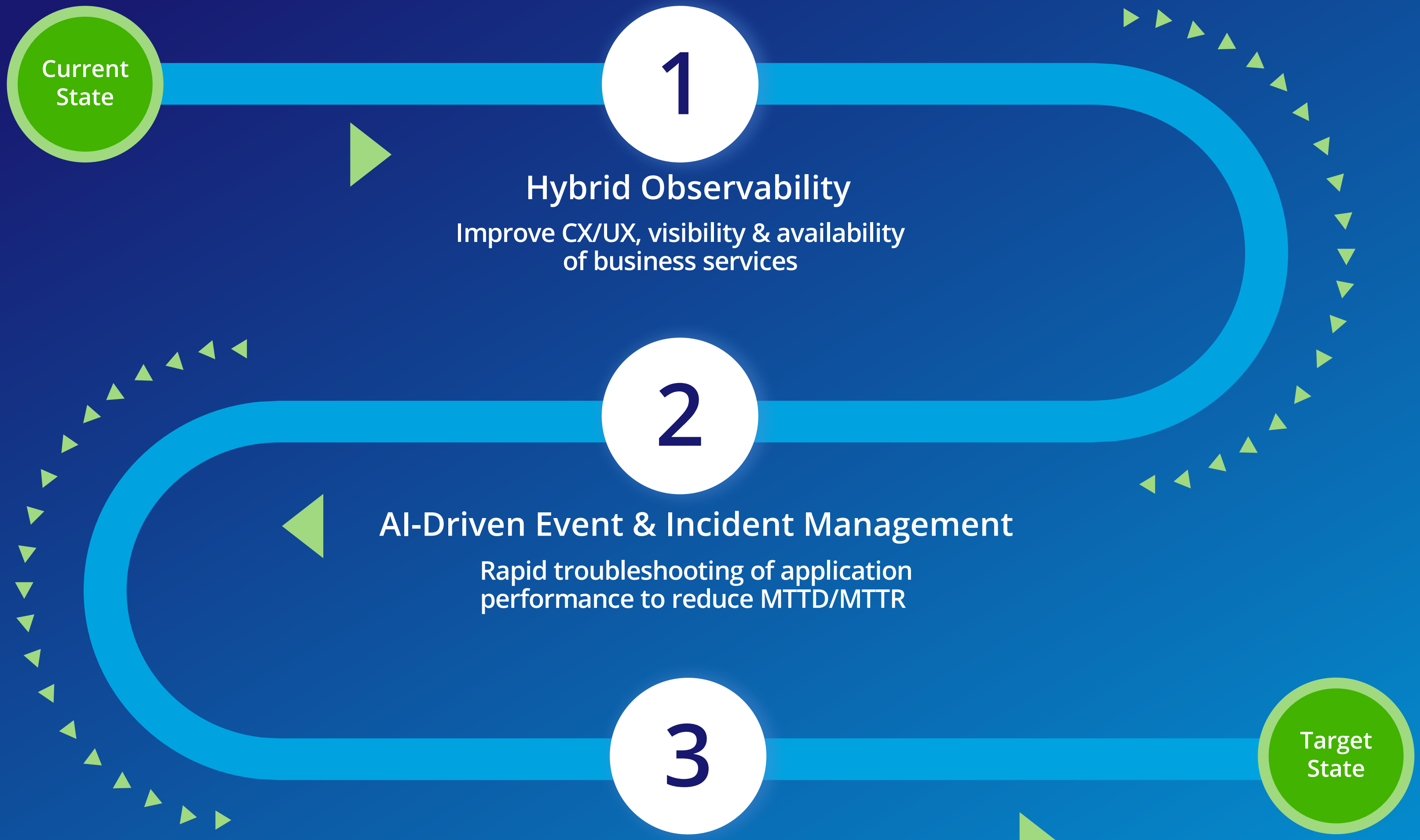
OpsRamp Hybrid Observability

OpsRamp
a Hewlett Packard Enterprise company

Journey to Observability

Key Initiatives

- Private Cloud
- Digital Transformation
- ITSM Transformation



Current State

1

Hybrid Observability

Improve CX/UX, visibility & availability of business services

2

AI-Driven Event & Incident Management

Rapid troubleshooting of application performance to reduce MTTD/MTTR

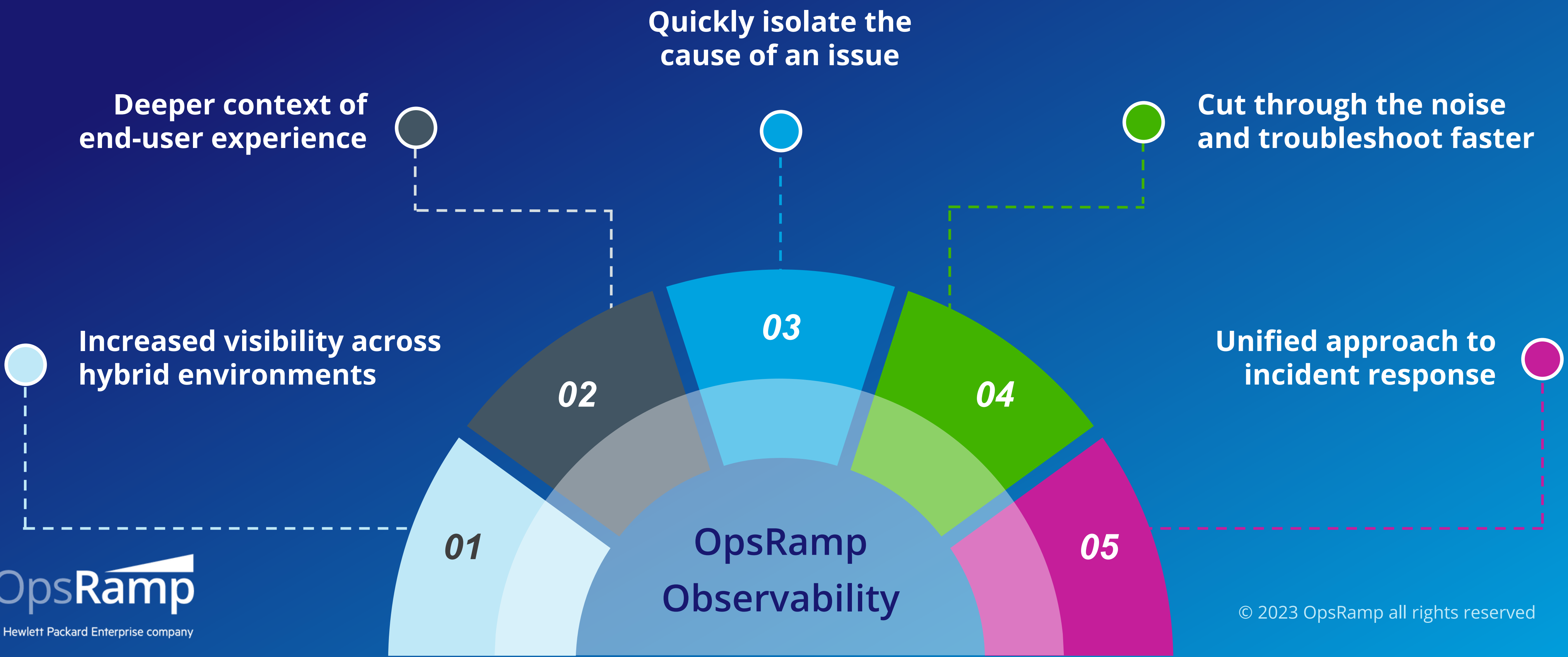
3

Intelligent Automation

Rapid troubleshooting of application performance to reduce MTTD/MTTR

Target State

Improve Observability
with OpsRamp



01 Increased visibility across hybrid environments

02 Deeper context of end-user experience

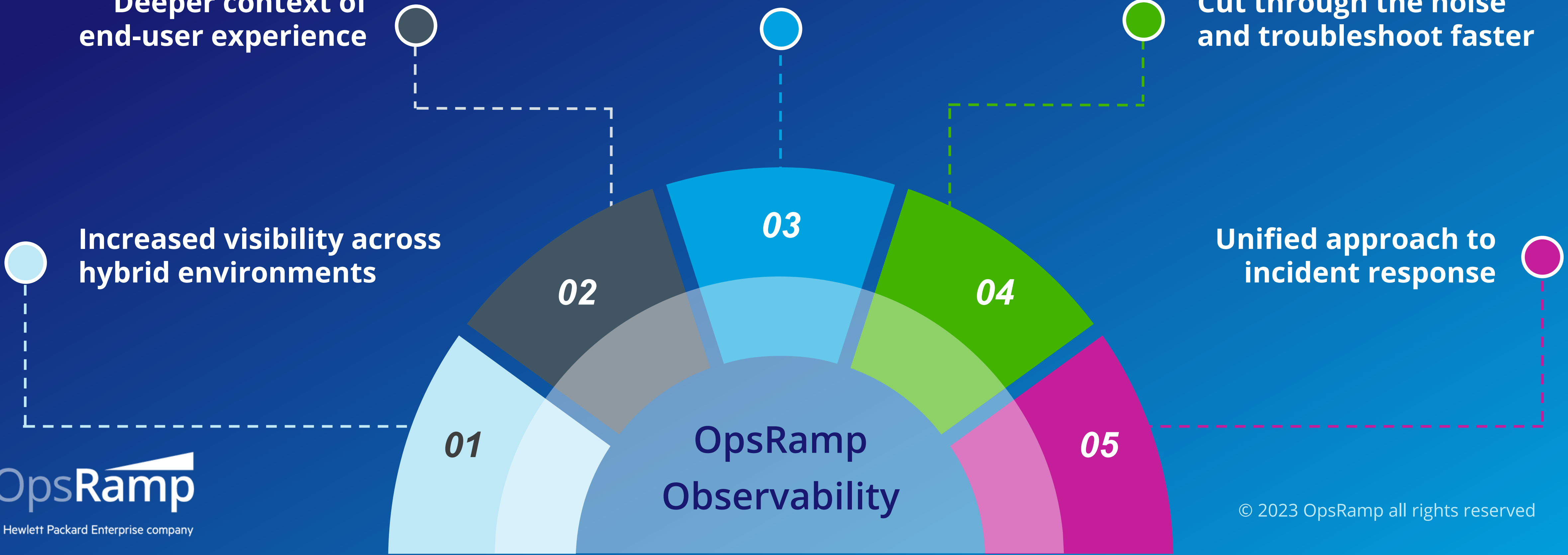
03 Quickly isolate the cause of an issue

04 Cut through the noise and troubleshoot faster

05 Unified approach to incident response

OpsRamp Observability

Improve Observability
with OpsRamp


Quickly isolate the cause of an issue
Deeper context of end-user experience
Cut through the noise and troubleshoot faster
Increased visibility across hybrid environments
Unified approach to incident response
01
02
03
04
05
OpsRamp Observability

OpsRamp
a Hewlett Packard Enterprise company

Hybrid Command Center for Digital Ops

Multi-Cloud

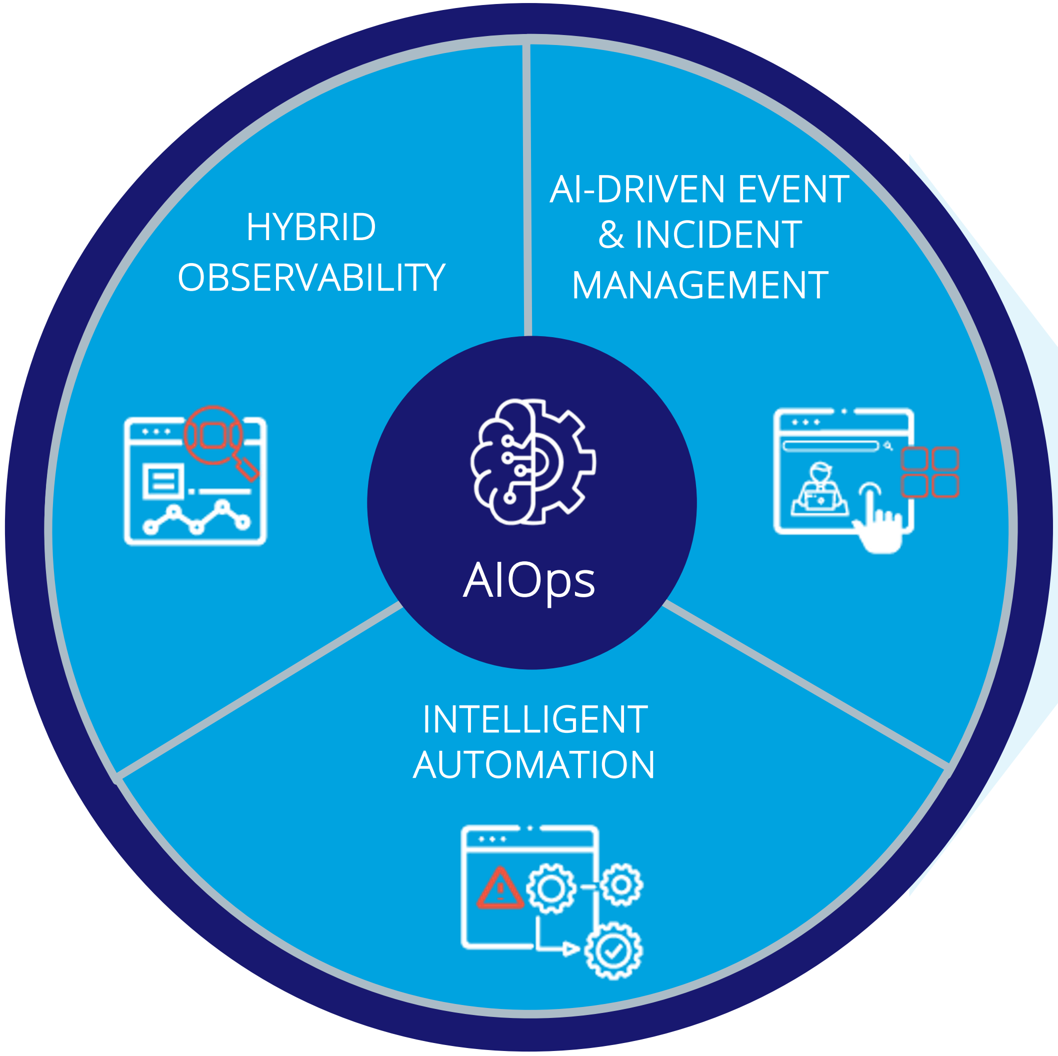
Data Center

Metrics

Events

Traces

Logs



HYBRID OBSERVABILITY

AI-DRIVEN EVENT & INCIDENT MANAGEMENT

AIOps

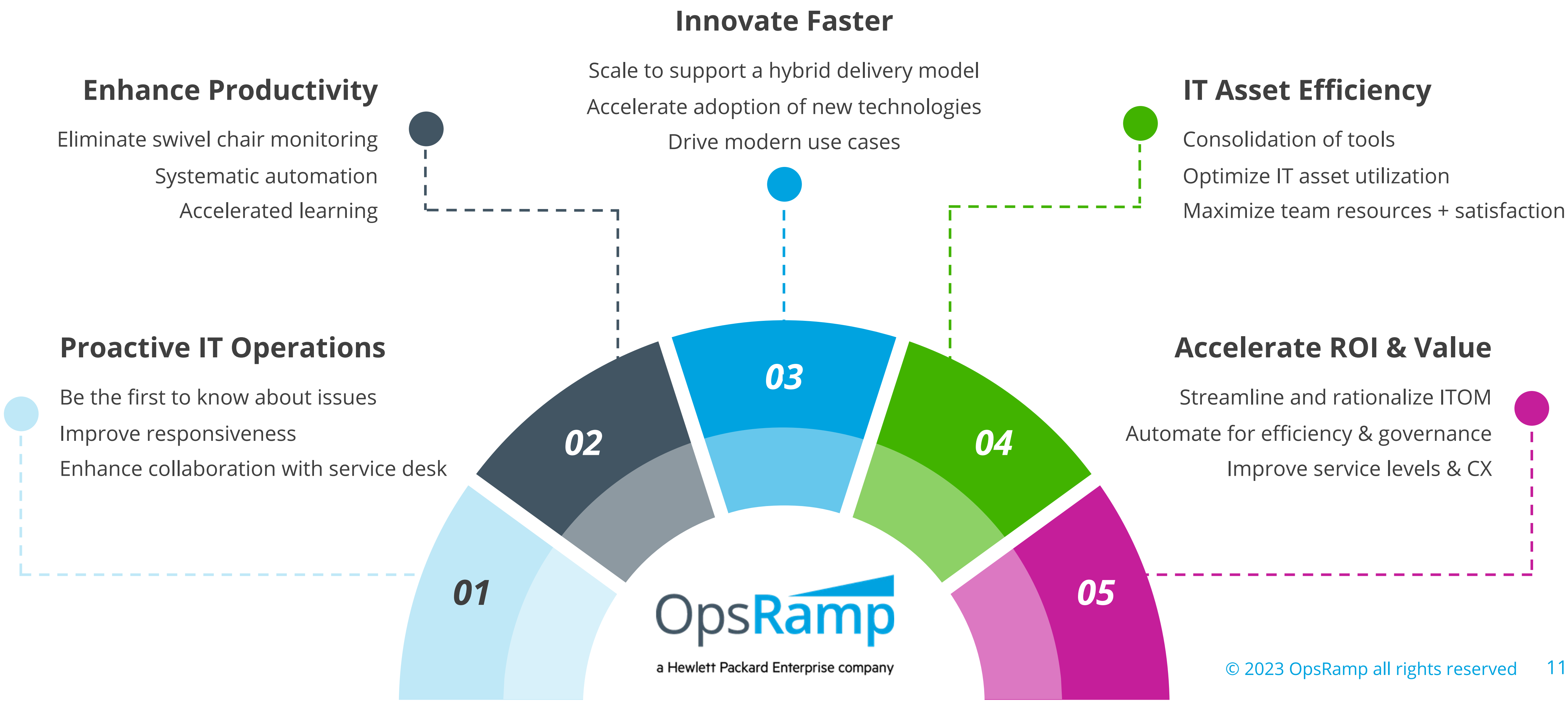
INTELLIGENT AUTOMATION

- Centralize and Simplify Monitoring
- Understand Resource Dependencies
- Detect and Resolve Incidents Faster
- Improve Governance, Uptime & Reliability
- Save Time and Costs
- Improve ITOps and Business Alignment

Discovery → Resolution

OpsRamp
a Hewlett Packard Enterprise company

Improving Observability



Proactive IT Operations

Be the first to know about issues

Improve responsiveness

Enhance collaboration with service desk

Enhance Productivity

Eliminate swivel chair monitoring

Systematic automation

Accelerated learning

Innovate Faster

Scale to support a hybrid delivery model

Accelerate adoption of new technologies

Drive modern use cases

IT Asset Efficiency

Consolidation of tools

Optimize IT asset utilization

Maximize team resources + satisfaction

Accelerate ROI & Value

Streamline and rationalize ITOM

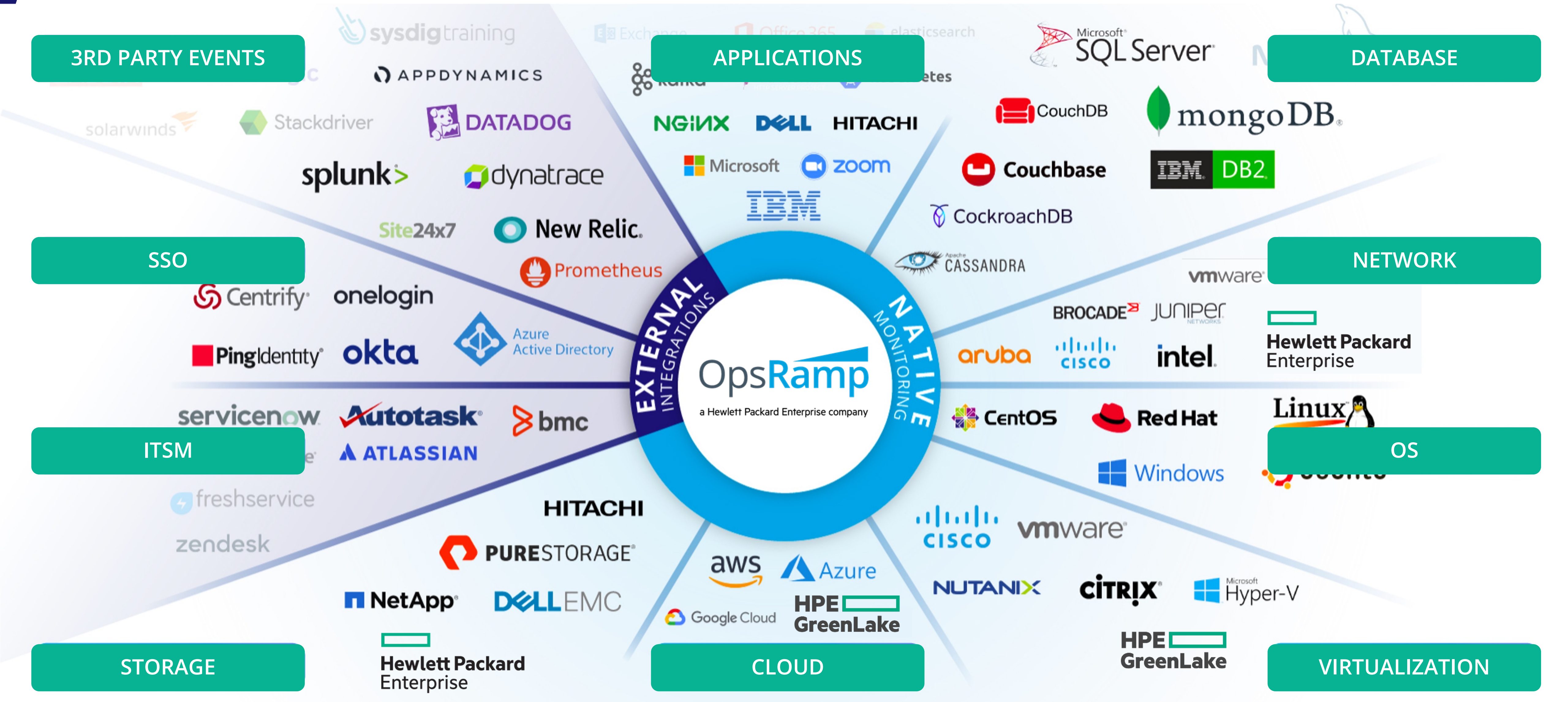
Automate for efficiency & governance

Improve service levels & CX

Comprehensive and Extensible

2500+ Integrations Supported

OpsRamp
a Hewlett Packard Enterprise company



OpsRamp
a Hewlett Packard Enterprise company

EXTERNAL INTEGRATIONS

NATIVE MONITORING

3RD PARTY EVENTS

sysdigtraining
APPDYNAMICS
solarwinds
Stackdriver
DATADOG
splunk>
dynatrace
Site24x7
New Relic.
Prometheus

APPLICATIONS

Exchange
Office 365
elasticsearch
kafka
etes
NGINX
DELL
HITACHI
Microsoft
zoom
IBM

DATABASE

Microsoft SQL Server
CouchDB
mongoDB.
Couchbase
IBM DB2
CockroachDB
Apache CASSANDRA

SSO

Centrify
onelogin
PingIdentity
okta
Azure Active Directory

NETWORK

vmware
BROCADE
JUNIPER NETWORKS
Hewlett Packard Enterprise
aruba
cisco
intel.

ITSM

servicenow
Autotask
bmc
ATLASSIAN
freshservice
zendesk

OS

CentOS
Red Hat
Linux
Windows

STORAGE

HITACHI
PURESTORAGE
NetApp
DELL EMC
Hewlett Packard Enterprise

CLOUD

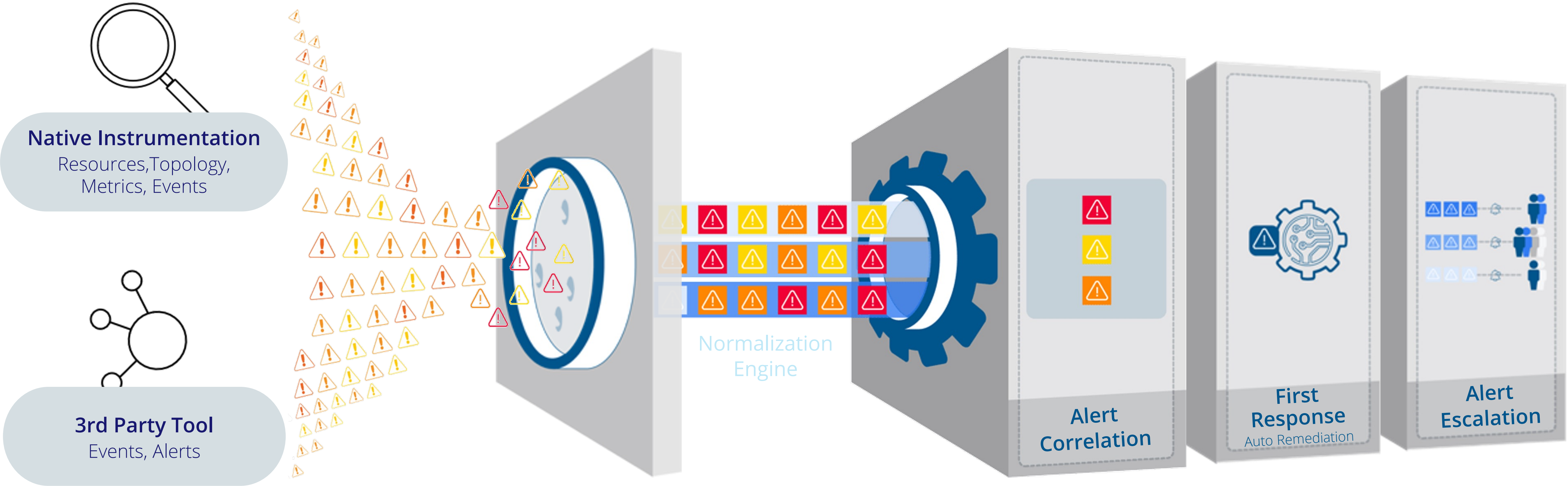
aws
Azure
Google Cloud
HPE GreenLake

VIRTUALIZATION

cisco
vmware
NUTANIX
CITRIX
Microsoft Hyper-V
HPE GreenLake

AIOps

Intelligent Alerting & Event Management



“ OpsRamp has taken the chaos out of our infrastructure.”

– VP of Infrastructure Delivery, Epsilon

OpsRamp
a Hewlett Packard Enterprise company

DETECT & RESOLVE POTENTIAL AVAILABILITY, LATENCY, & PERFORMANCE ISSUES – FASTER!

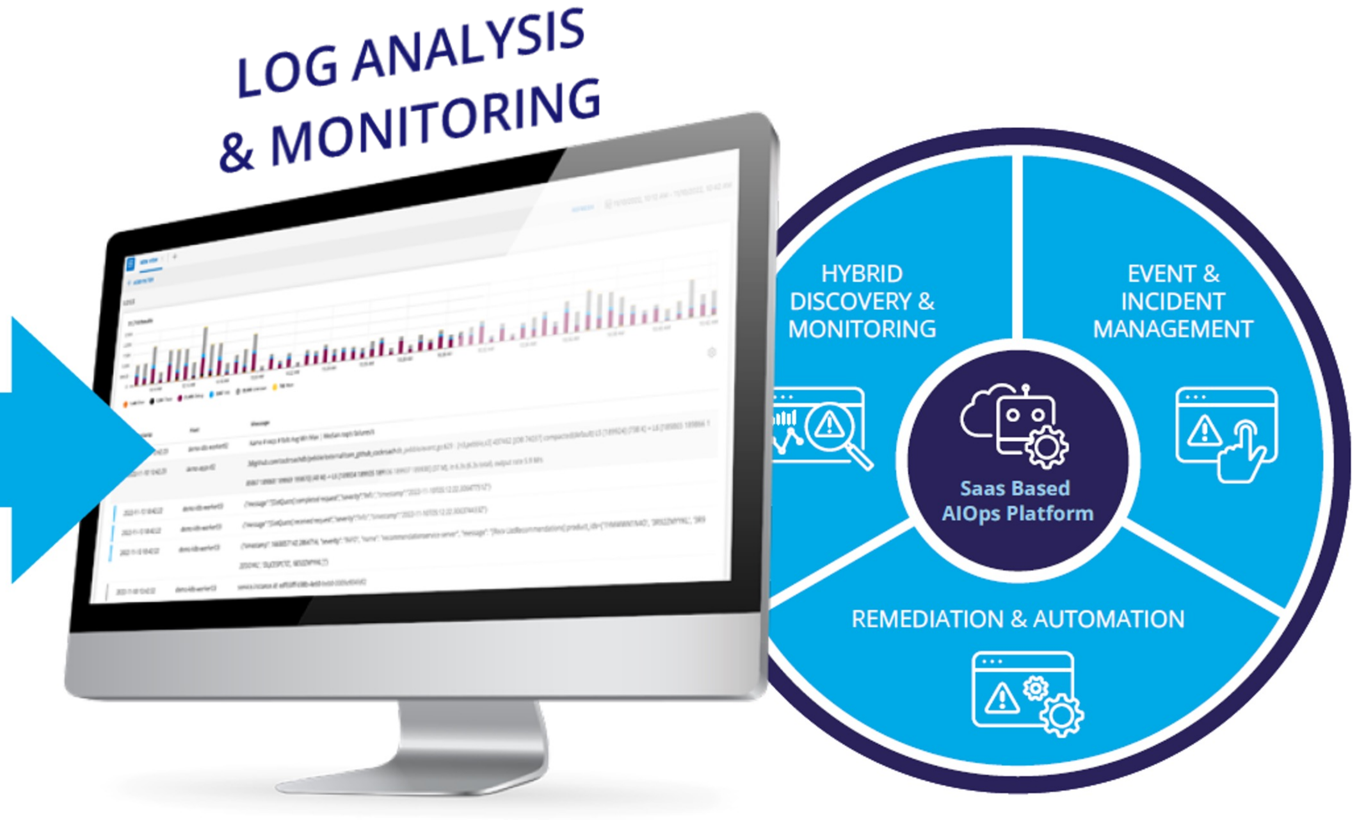
Log Management + Monitoring & Event Mgmt.

INFRASTRUCTURE
Windows host, Linux host, vCenter, storage, network, Kubernetes

APPLICATIONS
Apache, HAProxy, Cassandra, custom apps

CLOUD
AWS, Azure, Google

Logs
Events
Alerts


LOG ANALYSIS
& MONITORING
HYBRID DISCOVERY & MONITORING
EVENT & INCIDENT MANAGEMENT
Saas Based AIOps Platform
REMEDIATION & AUTOMATION

Detect Potential Issues

Identify Probable Root Cause

Take Immediate Action

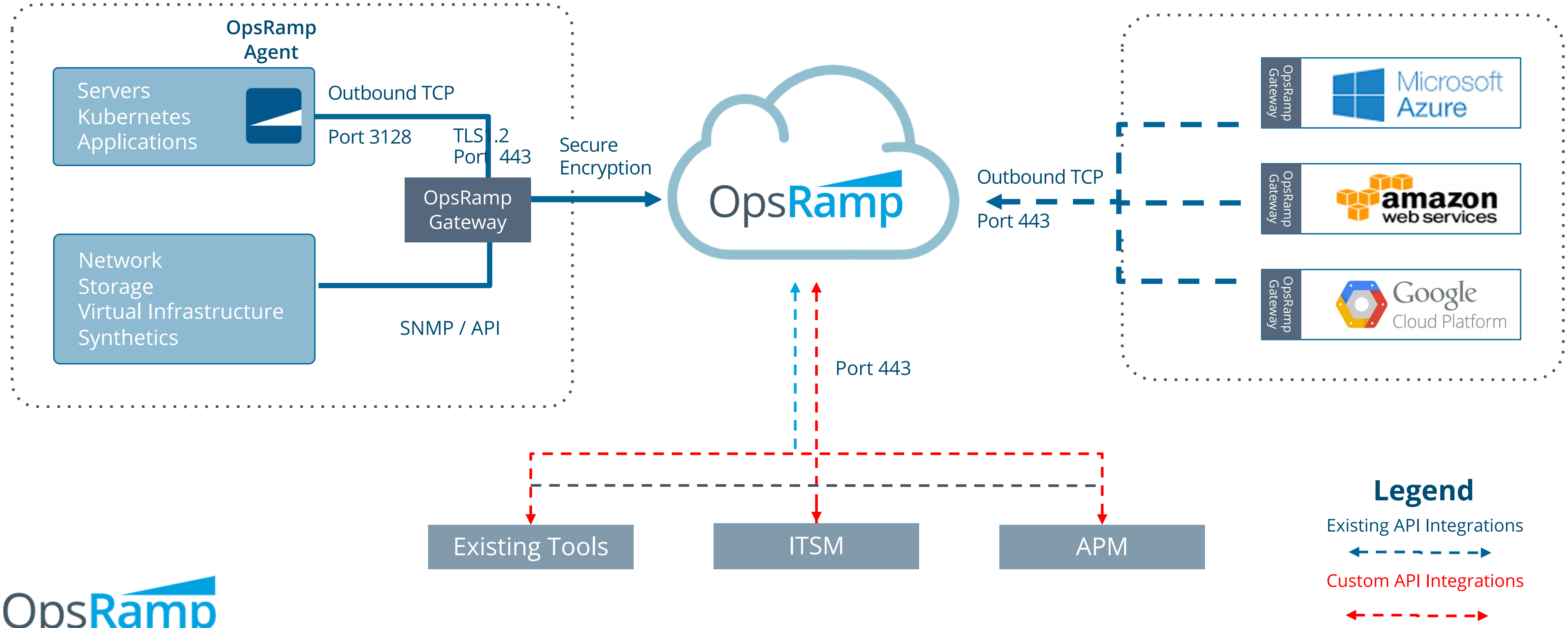
Resolve Incidents Faster

* Traces currently in Beta for Q2 '23 Release

Data Exchange Between:

OpsRamp Cloud and Customer Infrastructure

Customer Environment



OpsRamp Agent

Servers
Kubernetes
Applications

Outbound TCP

Port 3128

TLS 1.2
Port 443

OpsRamp Gateway

Secure Encryption

Network
Storage
Virtual Infrastructure
Synthetics

SNMP / API

OpsRamp

Outbound TCP

Port 443

Port 443

Existing Tools

ITSM

APM

Legend

Existing API Integrations

◂- - - - - - - -▸

Custom API Integrations

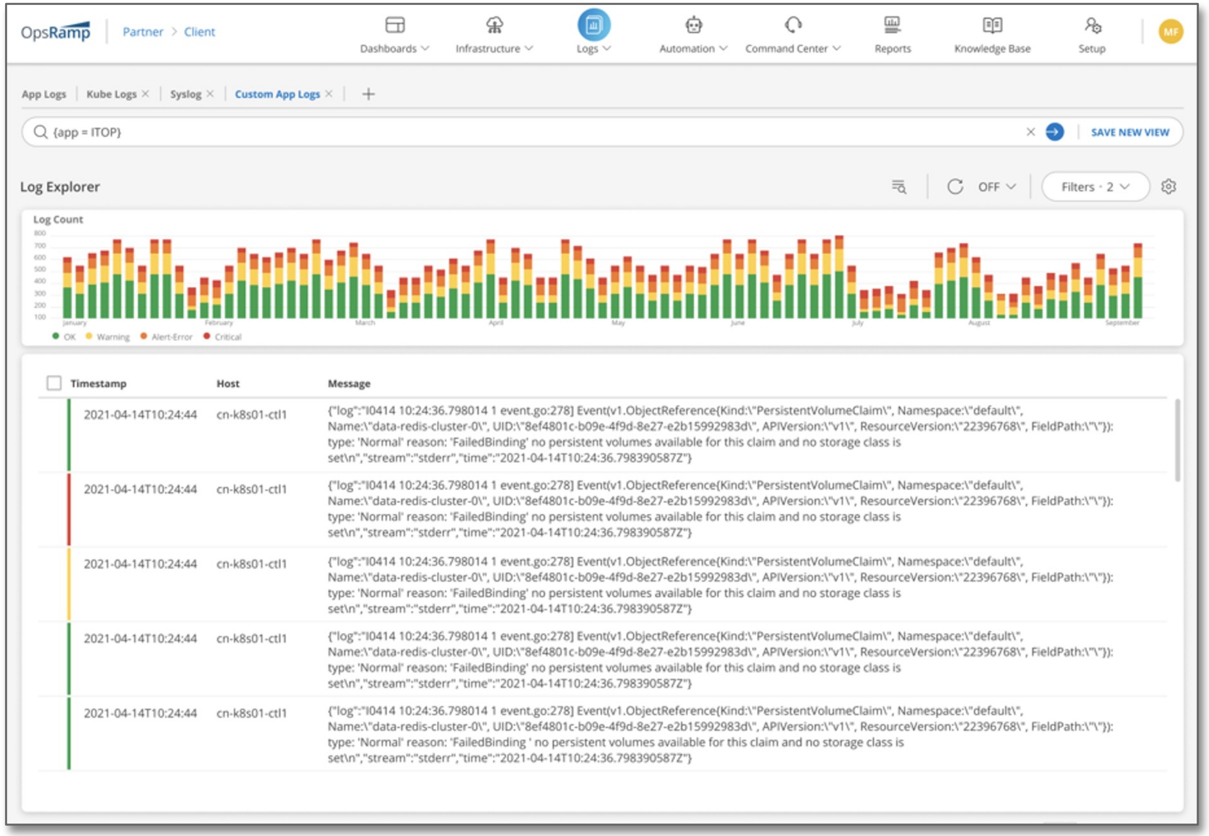
◂- - - - - - - -▸

OpsRamp
a Hewlett Packard Enterprise company

COMPLETE OBSERVABILITY ACROSS YOUR HYBRID ENVIRONMENT

OpsRamp Log Management

- **Complete Visibility** into event, logs, and trace data points to help improve alert correlation and isolate probable root cause.
- **Centralize & Standardize** the collection and maintenance of logs.
- **Reduce Costs** by consolidating tools and more quickly reducing the impact of downtime.
- **Simplify IT Ops & Improve Performance** by centralizing monitoring, log analysis, and automating remediation from your existing OpsRamp command center.

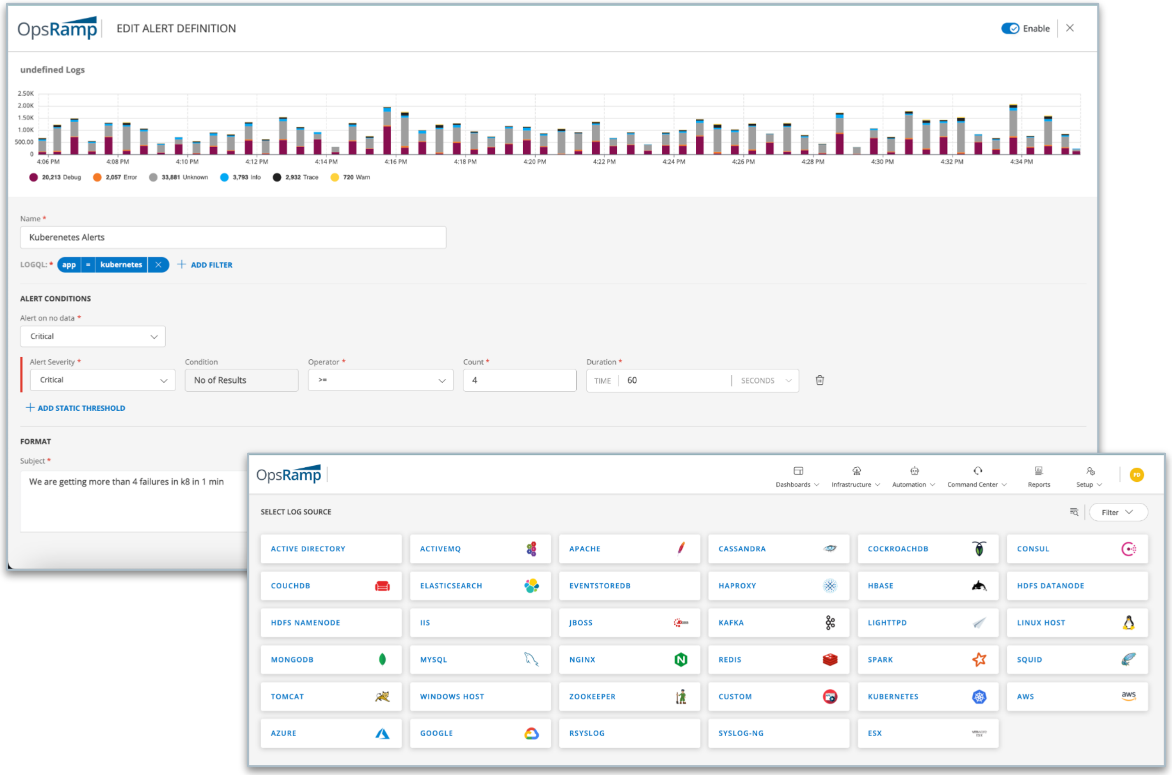


{app = ITOP}

Timestamp	Host	Message
2021-04-14T10:24:44	cn-k8s01-ctl1	{"log":"I0414 10:24:36.798014 1 event.go:278] Event(v1.ObjectReference{Kind:\"PersistentVolumeClaim\", Namespace:\"default\", Name:\"data-redis-cluster-0\", UID:\"8ef4801c-b09e-4f9d-8e27-e2b15992983d\", APIVersion:\"v1\", ResourceVersion:\"22396768\", FieldPath:\"\"}): type: 'Normal' reason: 'FailedBinding' no persistent volumes available for this claim and no storage class is set\n","stream":"stderr","time":"2021-04-14T10:24:36.798390587Z"}
2021-04-14T10:24:44	cn-k8s01-ctl1	{"log":"I0414 10:24:36.798014 1 event.go:278] Event(v1.ObjectReference{Kind:\"PersistentVolumeClaim\", Namespace:\"default\", Name:\"data-redis-cluster-0\", UID:\"8ef4801c-b09e-4f9d-8e27-e2b15992983d\", APIVersion:\"v1\", ResourceVersion:\"22396768\", FieldPath:\"\"}): type: 'Normal' reason: 'FailedBinding' no persistent volumes available for this claim and no storage class is set\n","stream":"stderr","time":"2021-04-14T10:24:36.798390587Z"}
2021-04-14T10:24:44	cn-k8s01-ctl1	{"log":"I0414 10:24:36.798014 1 event.go:278] Event(v1.ObjectReference{Kind:\"PersistentVolumeClaim\", Namespace:\"default\", Name:\"data-redis-cluster-0\", UID:\"8ef4801c-b09e-4f9d-8e27-e2b15992983d\", APIVersion:\"v1\", ResourceVersion:\"22396768\", FieldPath:\"\"}): type: 'Normal' reason: 'FailedBinding' no persistent volumes available for this claim and no storage class is set\n","stream":"stderr","time":"2021-04-14T10:24:36.798390587Z"}
2021-04-14T10:24:44	cn-k8s01-ctl1	{"log":"I0414 10:24:36.798014 1 event.go:278] Event(v1.ObjectReference{Kind:\"PersistentVolumeClaim\", Namespace:\"default\", Name:\"data-redis-cluster-0\", UID:\"8ef4801c-b09e-4f9d-8e27-e2b15992983d\", APIVersion:\"v1\", ResourceVersion:\"22396768\", FieldPath:\"\"}): type: 'Normal' reason: 'FailedBinding' no persistent volumes available for this claim and no storage class is set\n","stream":"stderr","time":"2021-04-14T10:24:36.798390587Z"}
2021-04-14T10:24:44	cn-k8s01-ctl1	{"log":"I0414 10:24:36.798014 1 event.go:278] Event(v1.ObjectReference{Kind:\"PersistentVolumeClaim\", Namespace:\"default\", Name:\"data-redis-cluster-0\", UID:\"8ef4801c-b09e-4f9d-8e27-e2b15992983d\", APIVersion:\"v1\", ResourceVersion:\"22396768\", FieldPath:\"\"}): type: 'Normal' reason: 'FailedBinding ' no persistent volumes available for this claim and no storage class is set\n","stream":"stderr","time":"2021-04-14T10:24:36.798390587Z"}

COMPLETE OBSERVABILITY ACROSS YOUR HYBRID ENVIRONMENT

OpsRamp Log Management



Name * Kuberenetes Alerts

LOGQL: * app = kubernetes ✕ + ADD FILTER

ALERT CONDITIONS

Alert on no data * Critical

Alert Severity * Critical | Condition No of Results | Operator * >= | Count * 4 | Duration * TIME 60 SECONDS

+ ADD STATIC THRESHOLD

FORMAT

Subject * We are getting more than 4 failures in k8 in 1 min

SELECT LOG SOURCE

ACTIVE DIRECTORY	ACTIVEMQ	APACHE	CASSANDRA	COCKROACHDB	CONSUL
COUCHDB	ELASTICSEARCH	EVENTSTOREDB	HAPROXY	HBASE	HDFS DATANODE
HDFS NAMENODE	IIS	JBOSS	KAFKA	LIGHTTPD	LINUX HOST
MONGODB	MYSQL	NGINX	REDIS	SPARK	SQUID
TOMCAT	WINDOWS HOST	ZOOKEEPER	CUSTOM	KUBERNETES	AWS
AZURE	GOOGLE	RSYSLOG	SYSLOG-NG	ESX	

1 **Easily Ingest, Process & Analyze Log Data**
from Virtually Any Source

2 **Log Viewer**
Centralize and simplify log analysis.

- Automatically parse logs
- Search and filter log data
- Save log views

3 **Alert Definitions**
Customize notifications to your business.

- Trigger alerts based on data patterns.
- Create custom log alerts
- View usage consumed

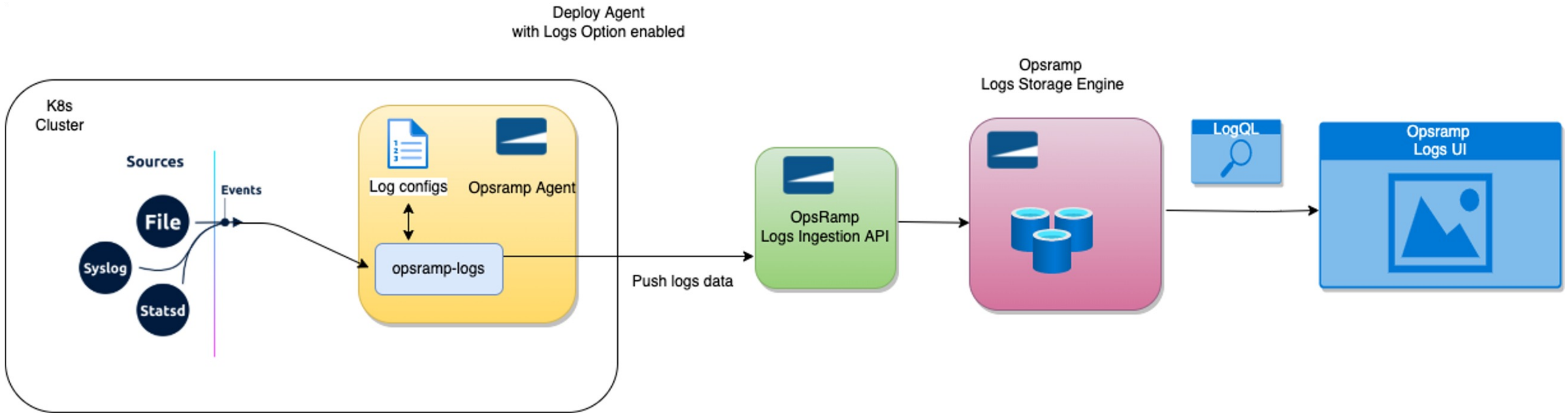
4 **Log Archiving**
Easily store logs for auditing purposes.

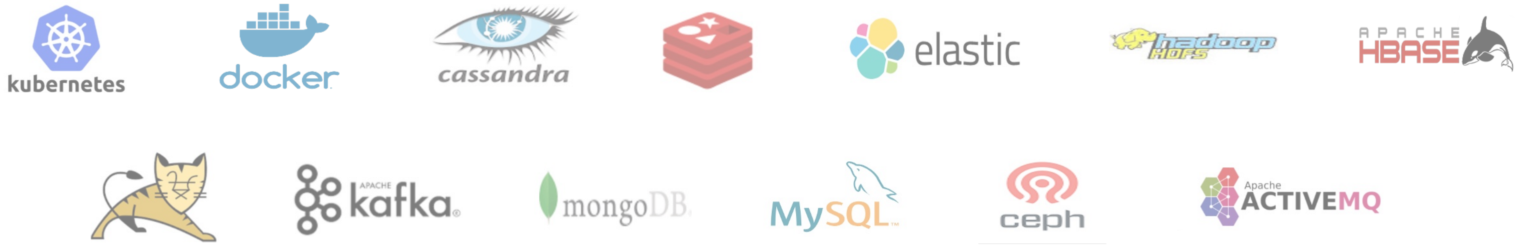
OpsRamp
a Hewlett Packard Enterprise company

INSTRUMENTATION

Log Collection

Unified agent includes discovery, monitoring, automation, patch management, remote consoles sessions & now Log Streaming.





Application Transaction Triggers

Webpage Sign-In / Sign-out Requests

Application Access Audits

IP Address Source & Destination during transaction

Network / System Audit

API Calls – Success / Failure

Cloud Services Audit Logs

Hardware & Software Changes

Configuring Agent Log Collectors

Deploying The Agent

VM Agent:

1. dpkg -i <opsramp-agent-deb-pkg>
2. sudo /opt/opsramp/agent/bin/configure -K <clientKey> -S <clientSecret> -s <apiserver> -M true -L True -T <LogEndpoint>

K8s Agent:

1. Additional Env variables to deployment yaml:
   ENABLE_LOG_MANAGEMENT
   LOG_MANAGEMENT_ENDPOINT
2. Additional mount paths for /var/log for k8s worker agent

Capture Logs from Custom Applications

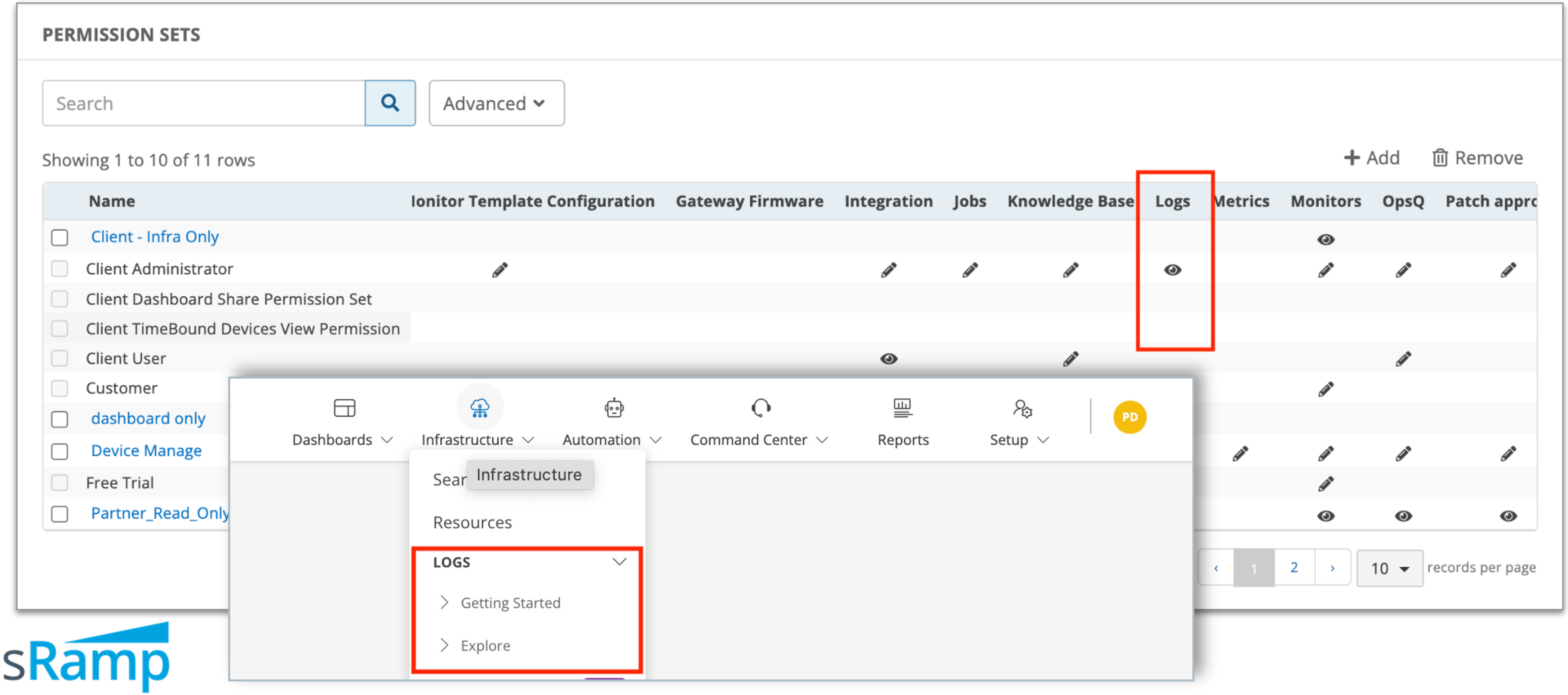
- Configure custom application logs in custom log config file and then restart the agent service /opt/opsramp/agent/conf/log.d/log-config.yaml
- Sample config file so user can easily modify the file log-config.yaml.sample
- If configured, Agent will give priority to the custom config (over default config) to start log collection.

NOTE: Install OpsRamp agent on Windows / Linux servers for Log Forwarding features to be enabled.
Agent Installation documentation provides additional detail for enabling Log Forwarding.

For Cloud Provider Logs no agents are required.

Create Permission Sets

- Create a new permission set with “Log View” enabled
- Current roles are updated with new log enabled permission sets
- “User” to re-login into OpsRamp to visualize log management workspace under infrastructure



PERMISSION SETS

Search | Advanced

Showing 1 to 10 of 11 rows

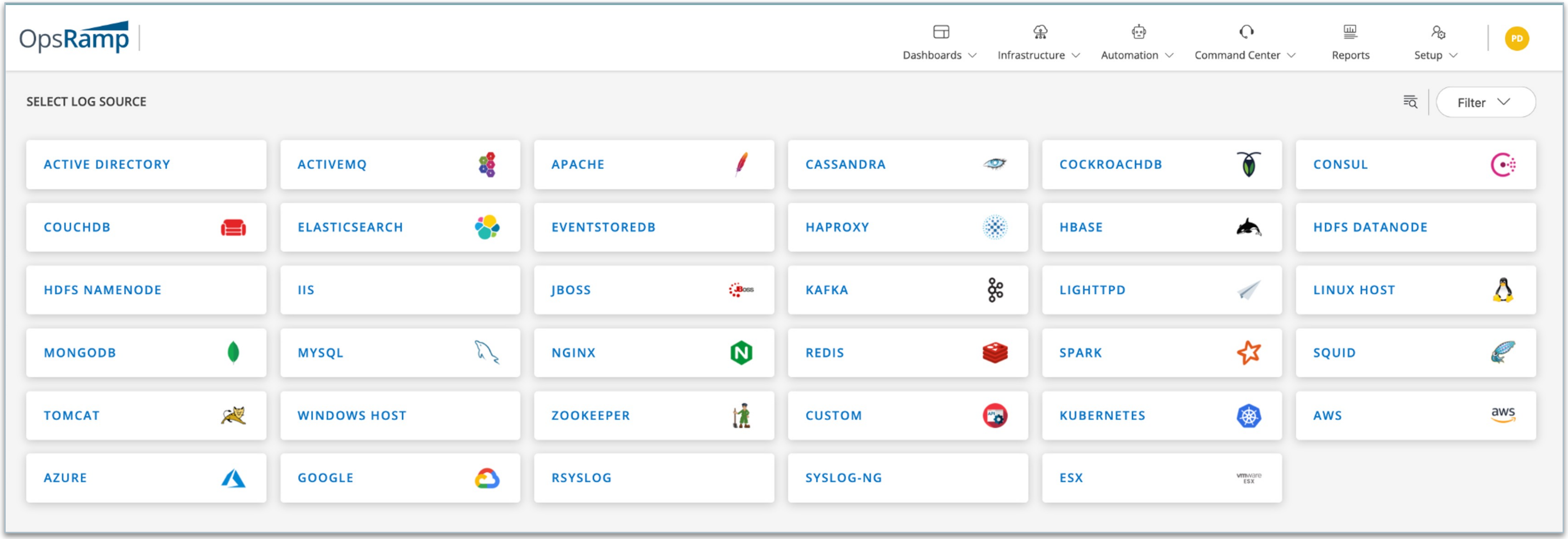
+ Add | Remove

	Name	Ionitor Template Configuration	Gateway Firmware	Integration	Jobs	Knowledge Base	Logs	Metrics	Monitors	OpsQ	Patch apprc
□	Client - Infra Only								👁		
□	Client Administrator	✎		✎	✎	✎	👁		✎	✎	✎
□	Client Dashboard Share Permission Set										
□	Client TimeBound Devices View Permission										
□	Client User			👁		✎				✎	
□	Customer								✎		
□	dashboard only										
□	Device Manage							✎	✎	✎	✎
□	Free Trial								✎		
□	Partner_Read_Only								👁	👁	👁

‹ 1 2 › 10 records per page

Ingest From Virtually Any Source

- To view supported “Log Sources”, select “Infrastructure → Logs → Getting Started”
- To start ingesting Logs, click on your required technology.

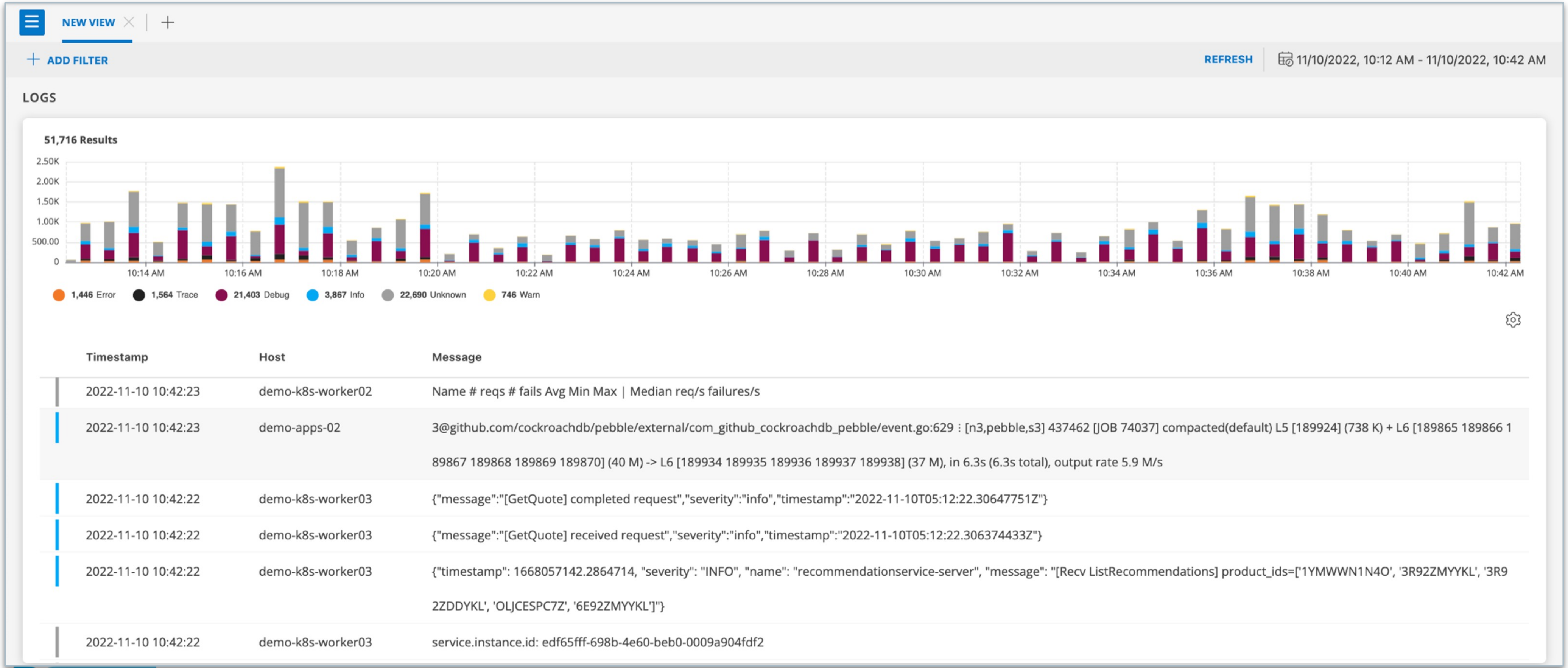


SELECT LOG SOURCE

ACTIVE DIRECTORY	ACTIVEMQ	APACHE	CASSANDRA	COCKROACHDB	CONSUL
COUCHDB	ELASTICSEARCH	EVENTSTOREDB	HAPROXY	HBASE	HDFS DATANODE
HDFS NAMENODE	IIS	JBOSS	KAFKA	LIGHTTPD	LINUX HOST
MONGODB	MYSQL	NGINX	REDIS	SPARK	SQUID
TOMCAT	WINDOWS HOST	ZOOKEEPER	CUSTOM	KUBERNETES	AWS
AZURE	GOOGLE	RSYSLOG	SYSLOG-NG	ESX	

Ingest Logs From Source

- Select technology type, then follow instruction for ingesting logs into OpsRamp.
- To visualize logs collected and indexed by OpsRamp, select “Infrastructure → Logs → Explore”.
- To create custom views of required log conditions, select ”+ ADD FILTER”.



11/10/2022, 10:12 AM - 11/10/2022, 10:42 AM

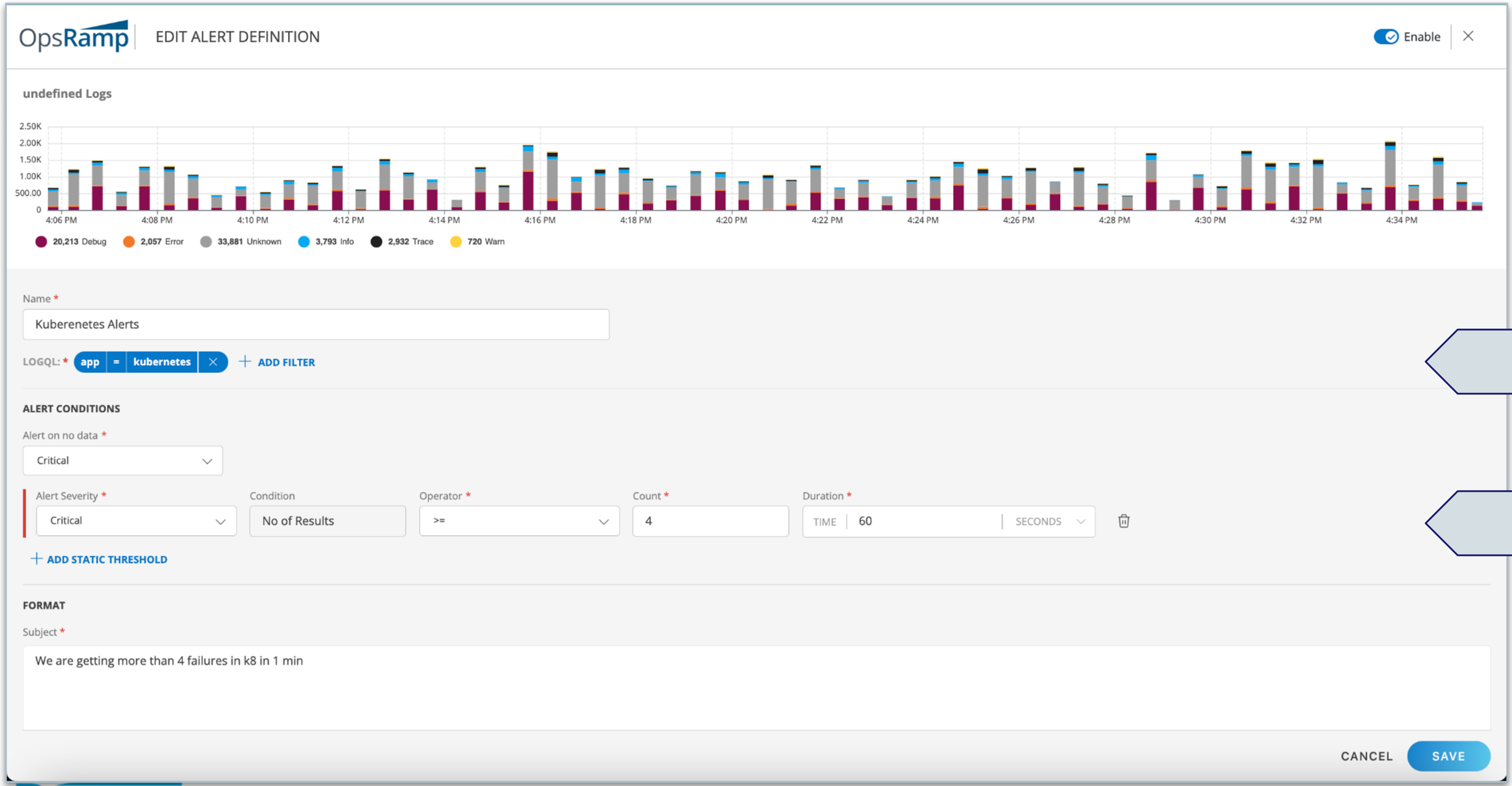
51,716 Results

Timestamp	Host	Message
2022-11-10 10:42:23	demo-k8s-worker02	Name # reqs # fails Avg Min Max | Median req/s failures/s
2022-11-10 10:42:23	demo-apps-02	3@github.com/cockroachdb/pebble/external/com_github_cockroachdb_pebble/event.go:629 ⋮ [n3,pebble,s3] 437462 [JOB 74037] compacted(default) L5 [189924] (738 K) + L6 [189865 189866 189867 189868 189869 189870] (40 M) -> L6 [189934 189935 189936 189937 189938] (37 M), in 6.3s (6.3s total), output rate 5.9 M/s
2022-11-10 10:42:22	demo-k8s-worker03	{"message":"[GetQuote] completed request","severity":"info","timestamp":"2022-11-10T05:12:22.30647751Z"}
2022-11-10 10:42:22	demo-k8s-worker03	{"message":"[GetQuote] received request","severity":"info","timestamp":"2022-11-10T05:12:22.306374433Z"}
2022-11-10 10:42:22	demo-k8s-worker03	{"timestamp": 1668057142.2864714, "severity": "INFO", "name": "recommendationservice-server", "message": "[Recv ListRecommendations] product_ids=['1YMWWN1N4O', '3R92ZMYYKL', '3R92ZDDYKL', 'OLJCESPC7Z', '6E92ZMYYKL']"}
2022-11-10 10:42:22	demo-k8s-worker03	service.instance.id: edf65fff-698b-4e60-beb0-0009a904fdf2

a Hewlett Packard Enterprise company

Ingest Logs From Source

- Enable monitoring of logs with “Log Alert Definitions” within extension to Log Analysis.
- Navigate to “Setup → Monitoring → Log Alert Definitions”
- View monitoring results while creating alert conditions within real-time query builder.



EDIT ALERT DEFINITION

undefined Logs

Name *

Kuberenetes Alerts

LOGQL: * app = kubernetes ✕ + ADD FILTER

ALERT CONDITIONS

Alert on no data *

Critical

Alert Severity *	Condition	Operator *	Count *	Duration *
Critical	No of Results	>=	4	TIME 60 SECONDS

+ ADD STATIC THRESHOLD

FORMAT

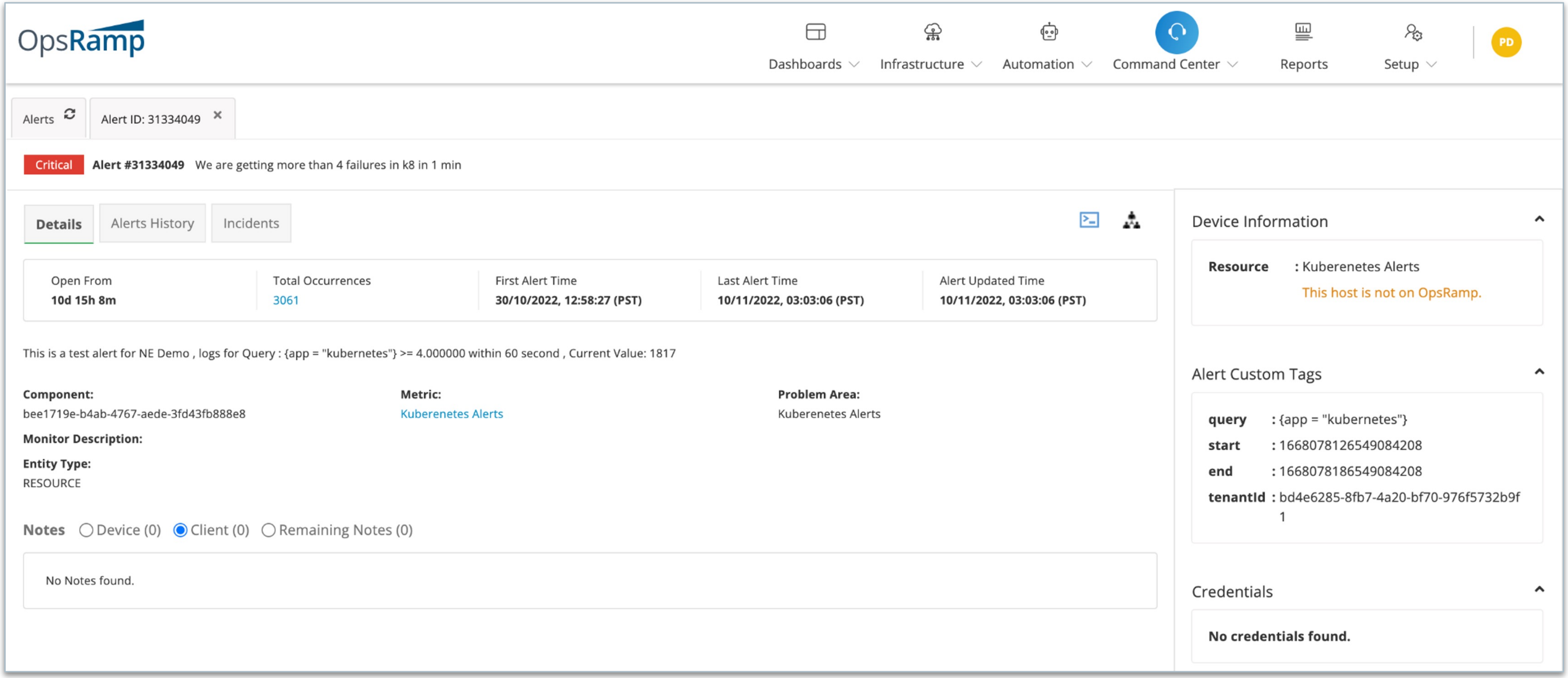
Subject *

We are getting more than 4 failures in k8 in 1 min

Query Builder

Alert Condition

Alert from “Log Alert Definitions” ~ sample



Critical **Alert #31334049** We are getting more than 4 failures in k8 in 1 min

Open From	Total Occurrences	First Alert Time	Last Alert Time	Alert Updated Time
10d 15h 8m	3061	**30/10/2022, 12:58:27 (PST)**	**10/11/2022, 03:03:06 (PST)**	**10/11/2022, 03:03:06 (PST)**

This is a test alert for NE Demo , logs for Query : {app = "kubernetes"} >= 4.000000 within 60 second , Current Value: 1817

Component:
bee1719e-b4ab-4767-aede-3fd43fb888e8

Metric:
Kuberenetes Alerts

Problem Area:
Kuberenetes Alerts

Monitor Description:

Entity Type:
RESOURCE

Notes ○ Device (0) ● Client (0) ○ Remaining Notes (0)

No Notes found.

Device Information

Resource : Kuberenetes Alerts
This host is not on OpsRamp.

Alert Custom Tags

query : {app = "kubernetes"}
start : 1668078126549084208
end : 1668078186549084208
tenantId : bd4e6285-8fb7-4a20-bf70-976f5732b9f
1

Credentials

No credentials found.

OpsRamp
a Hewlett Packard Enterprise company

Let’s Take A Closer Look

OpsRamp AI-Driven Event & Incident Management

OpsRamp

a Hewlett Packard Enterprise company

Observability Roadmap Journey for Success

Let’s Take A Closer Look

OpsRamp Intelligent Automation

OpsRamp

Getting Started with OpsRamp Tracing Solution

Complete guide

Internal Documentation - Not to be shared with Customers

Contents

- Introduction to Tracing
- OpsRamp Tracing
  - How Trace Ingestion happens in OpsRamp
  - Trace Proxy - Configurations / Functionalities / Metrics / Insights
- How to Setup Tracing Demo environment ? - Demo
  - Deploying Demo Application (Which is already instrumented) - VM / K8s
  - Deploying / Configuring Trace Proxy
  - Configuring Demo Application to send traces to trace proxy
  - Analyzing Traces / Trace Insights in OpsRamp
- How to Instrument an Application to send Traces - Demo

Observability - Three pillars

Observability is the ability to understand the inner state of your evolving systems by examining and analyzing all available data outputs like **logs, traces & metrics** in real time.

In a nutshell, having observability on your application allows you to ***understand what, how, and why a malfunction has occurred***.

Monitoring Vs Observability : While monitoring tracks the system's health of your application, observability tells you why it's performing a certain way.

The 3 pillars of Observability

Introduction

OpsRamp Tracing solution provides end-to-end visibility into user transactions across services, as well as seamless integration into performance metrics and logs to accelerate issue resolution and root-cause analysis.

OpsRamp Tracing solution supports the **OpenTelemetry** standard and is built to use **OpenTelemetry**, to provide a standardized, vendor-agnostic, and industry-standard solution for distributed tracing.

Note: Solution has some TO-DO items to make it complete. So refer to the last section of doc for upcoming features.

What is Distributed Tracing ?

Distributed Tracing is a technique which allows you to trace and track requests as they flow through different services or components in a distributed architecture.

By capturing and correlating trace data from multiple services, distributed tracing provides insights into the ***performance***, ***latency***, and ***dependencies*** between different parts of a system.

What is Opentelemetry ?

OpenTelemetry is a vendor-agnostic instrumentation library per language to generate, emit, collect, process and export telemetry data.

Our Tracing Product is built to use **OpenTelemetry**, to provide a standardized, vendor-agnostic, and industry-standard solution for distributed tracing.

For more details refer official doc here

What is a Trace ?

A trace represents the path of a single request or transaction as it traverses through various services.

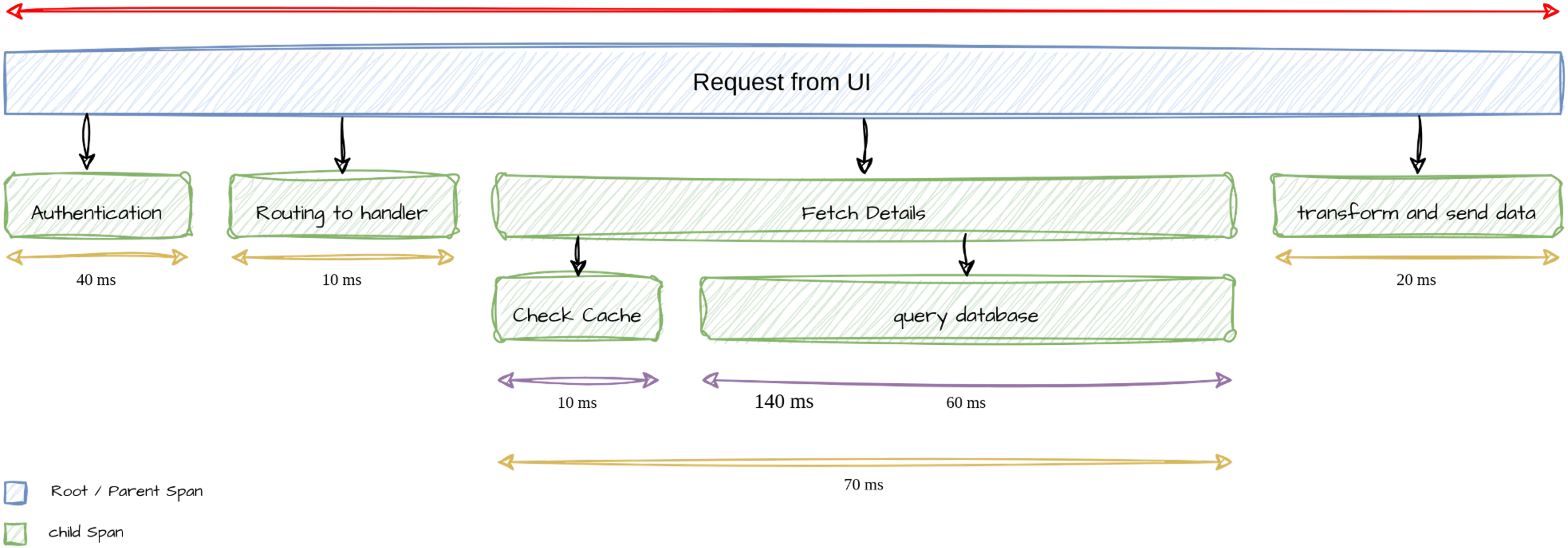
It consists of a collection of spans, where each span represents a specific operation or activity within a service.

A trace record usually consists of additional context like:

- latency of a API request
- Time taken to connect to a database
- Time taken for a function in code
- Time taken for a query to execute

& more depending on the operation performed.

Trace



Request from UI

Authentication

40 ms

Routing to handler

10 ms

Fetch Details

Check Cache

10 ms

query database

140 ms

60 ms

70 ms

transform and send data

20 ms

🟦 Root / Parent Span

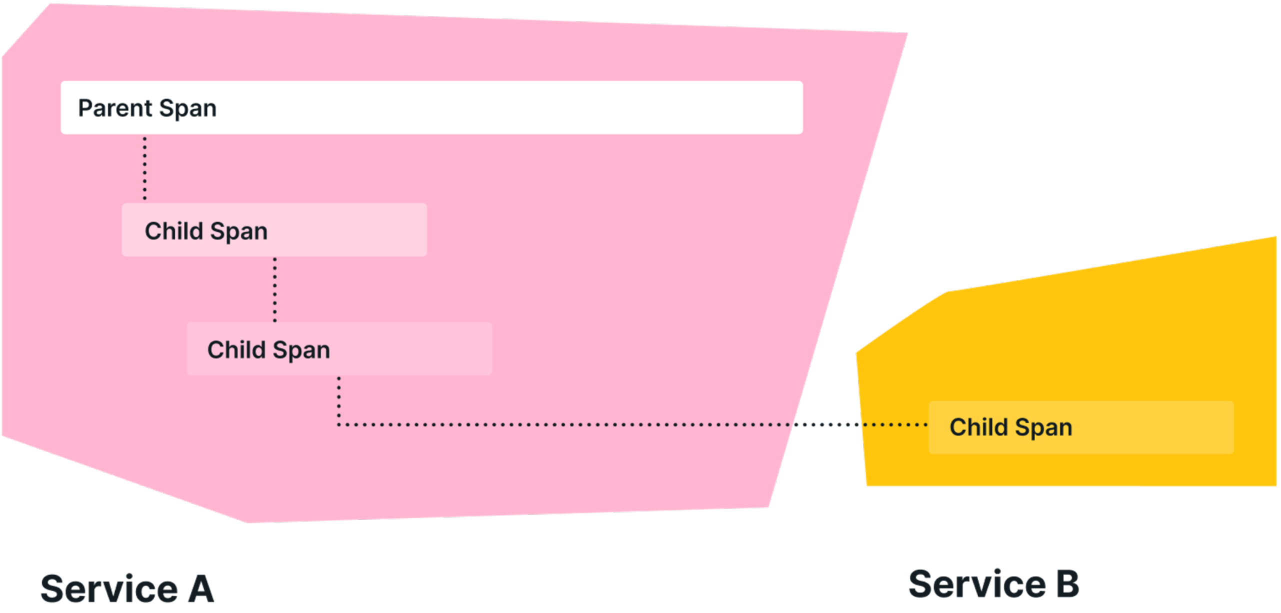
🟩 child Span

Spans

Each span represents a specific operation or activity within a service.

Spans are connected to form a tree-like structure, representing the parent-child relationships between different operations.

Figure 1



Service / Operation

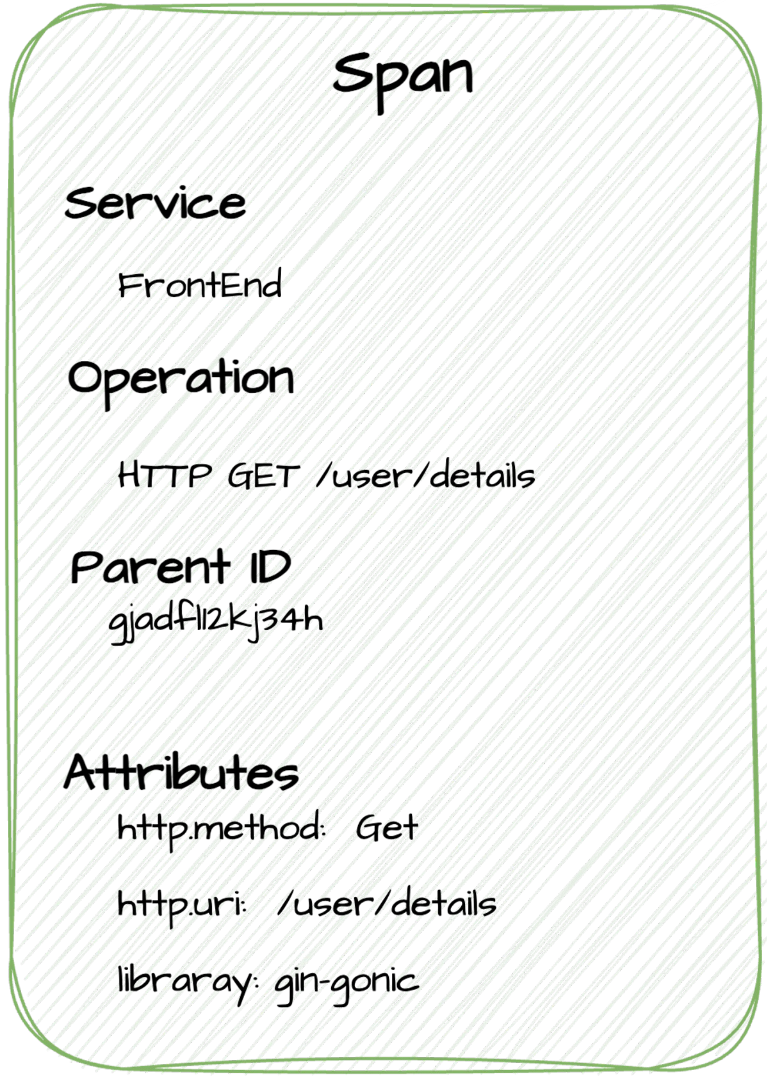
Every span consists mainly of the following components displayed in the illustration

Service
service name is usually the name of the microservice or module the span is representing

Operation
Operation name is the name of the action that the span is representing

Parent ID
It’s the unique identifier of its parent span which encloses the current span. Parent ID is usually empty for root spans and such spans which don’t have any parent id are considered the starting point any trace

Attributes
It comprises of additional metadata



Span

Service

FrontEnd

Operation

HTTP GET /user/details

Parent ID
gjadfll2kj34h

Attributes
http.method: Get

http.uri: /user/details

libraray: gin-gonic

Terminology - At a glance

Request

User action which triggers the generation of a trace.

Example: a user tries to checkout their cart on an e-commerce website.

Span

Smallest unit of work within a distributed system, capturing details such as timing, unique identifiers, and metadata for a specific operation.

Examples: API calls, HTTP calls, cache calls, database calls, etc.

Trace

Lifecycle of a request made by a user. A trace consists of multiple spans. Tracks all the calls that were made and time taken.

Example: A user clicks “Checkout Cart”. This causes the following calls: checkout service → product catalog service → payment service → database service.

Service

A distinct component that performs a specific set of functions.

Examples: Frontend service, payment service, database service, recommendation service, etc.

Operation

Under each service, the actual individual functions.

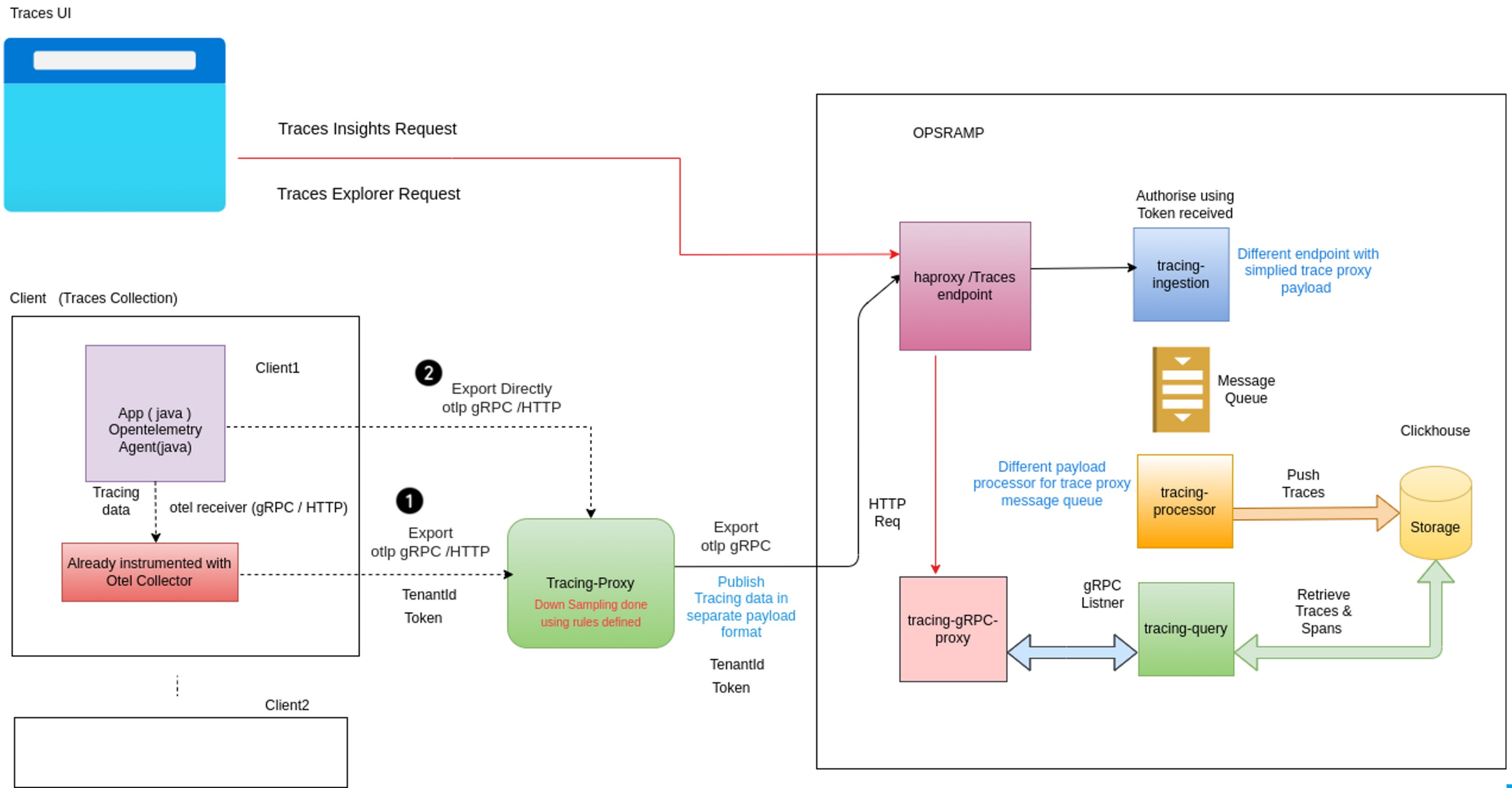
Examples: For a frontend service, we could have “Add to Cart”, “View Cart”, “Checkout”, etc. operations!

OpsRamp Tracing - Different Components

- **Trace Proxy:** It is OpsRamp trace collector where all the traces from a customer's application are aggregated, down-sampled and exported to OpsRamp.

- **Traces UI**: Traces Explorer UI to visualize the traces and Trace Insights computed from the metrics collected by Trace Proxy.

How Trace Ingestion in OpsRamp works ?



Traces UI

Traces Insights Request

Traces Explorer Request

Client (Traces Collection)

Client1

App (java)
Opentelemetry
Agent(java)

Tracing
data

otel receiver (gRPC / HTTP)

Already instrumented with
Otel Collector

2 Export Directly
otlp gRPC /HTTP

1 Export
otlp gRPC /HTTP

TenantId

Token

Tracing-Proxy
Down Sampling done
using rules defined

Export
otlp gRPC

Publish
Tracing data in
separate payload
format

TenantId

Token

Client2

OPSRAMP

HTTP
Req

haproxy /Traces
endpoint

Authorise using
Token received

tracing-
ingestion

Different endpoint with
simplied trace proxy
payload

Message
Queue

Different payload
processor for trace proxy
message queue

tracing-
processor

Push
Traces

Clickhouse

Storage

tracing-gRPC-
proxy

gRPC
Listner

tracing-query

Retrieve
Traces &
Spans

Application - Trace Instrumentation

- ➢ In order to make a system observable, it must be instrumented: That is, the code must emit traces, metrics, and logs.
- ➢ Without being required to modify the source code you can collect telemetry from an application using Automatic Instrumentation.
- ➢ To facilitate the instrumentation of applications even more, you can manually instrument your applications by coding against the OpenTelemetry APIs.

Application - Trace Instrumentation Contd..

Note, that for most languages it is possible to use both manual and automatic instrumentation at the same time: Automatic Instrumentation will allow you to gain insights into your application quickly and manual instrumentation will enable you to embed granular observability into your code.

Next, you can deep dive into the documentations for the language you are using:

- C++
- .NET
- Erlang / Elixir
- Go
- Java
- JavaScript / TypeScript
- PHP
- Python
- Ruby
- Rust
- Swift

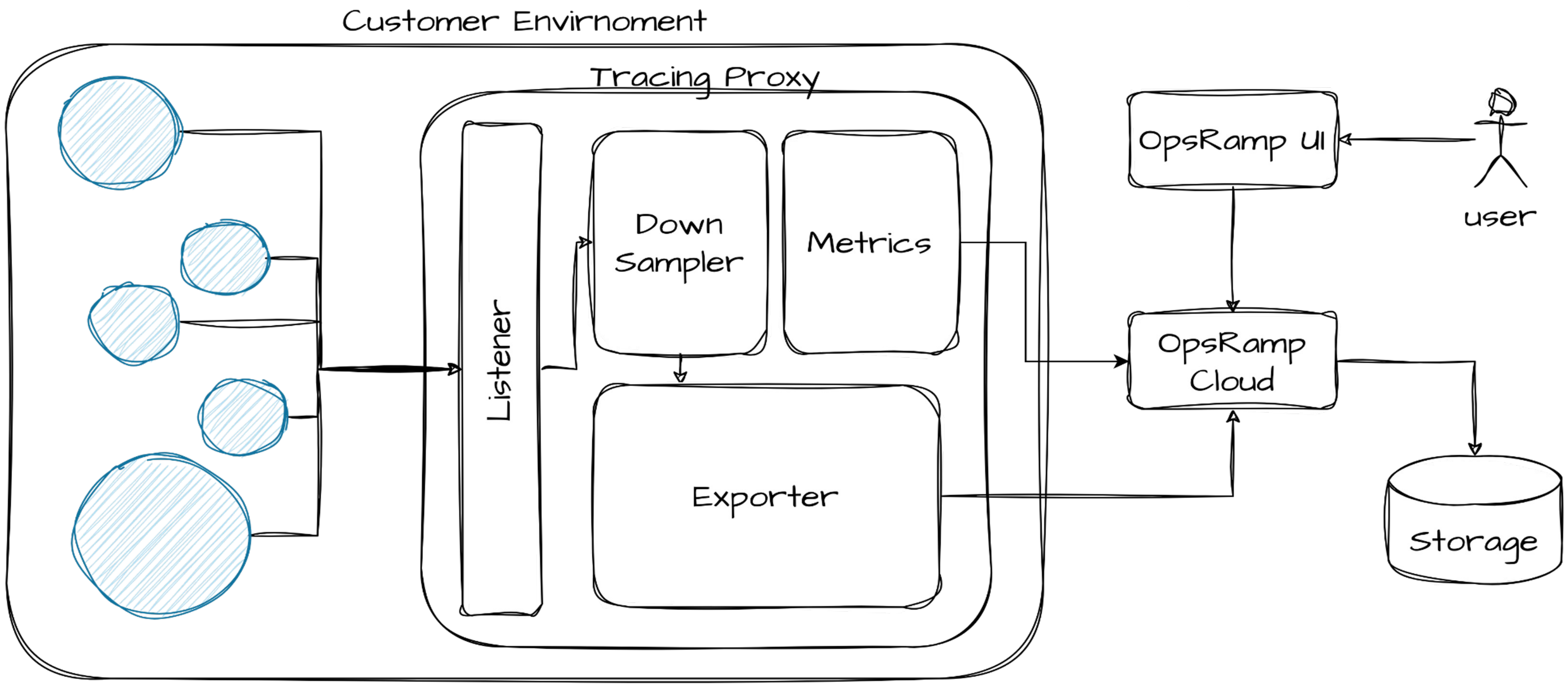
For detailed examples for each programming language with sample instrumented code, please refer doc https://github.com/opsramp/tracing-docs

Trace Proxy

Trace Proxy is an application that collects spans emitted by your application, downsamples them based on sampling rules, and generates several useful trace metrics.

It is designed to sit within your infrastructure as a single deployment or a cluster of multiple trace proxy services. In the case of multiple trace proxy services, the proxy containers/processes must be able to communicate with each other to consolidated traces.

Trace Proxy - Design



▨ applications instrumented for tracing

Trace Proxy - Functionalities

- Trace Metrics
  - Golden signals: Latency, Errors, Operations/sec
  - Other metrics: Operations failed/succeeded, span counts, duration, etc.
  - Include/Exclude Metrics
- Downsampling
- Deployment Methods: Kubernetes or Host Based Deployment
- Clustering & Peer Management

Trace Proxy - Document links

- Trace Proxy Configuration
- Supported Sampling Methods
- Metrics Collected by Trace Proxy : List
- Trace Insights : Queries
- Github

Trace Proxy - Sampling Methods

- **Deterministic Sampler** is a static sample rate, choosing traces randomly to either keep or send.
- **Dynamic Sampler** will adjust the sample rate of traces and events based on their frequency.
- **Exponential Moving Average (EMA) Dynamic Sampler** maintains an Exponential Moving Average of counts seen per key, and adjusts this average at regular intervals.
- **Rule-Based Sampling** allows you to define sampling rates explicitly based on the contents of your traces.
- **Throughput-Based Sampling** meet a goal throughput rate of a fixed number of spans, not traces, per second per trace-proxy node.

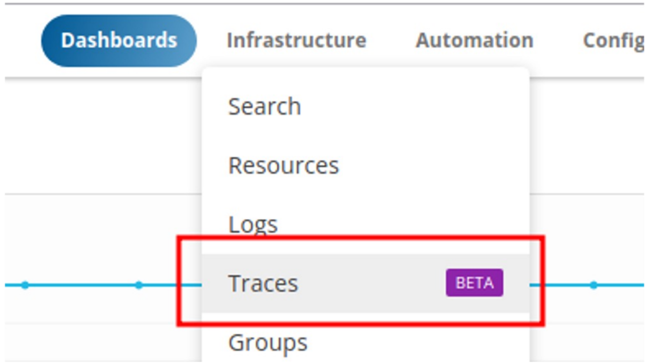
Tracing Demo Setup - Guide

- Deploying Demo Application (Which is already instrumented) - VM / K8s
- Deploying / Configuring Trace Proxy
- Configuring Demo Application to send traces to trace proxy
- Analyzing Traces / Trace Insights in OpsRamp

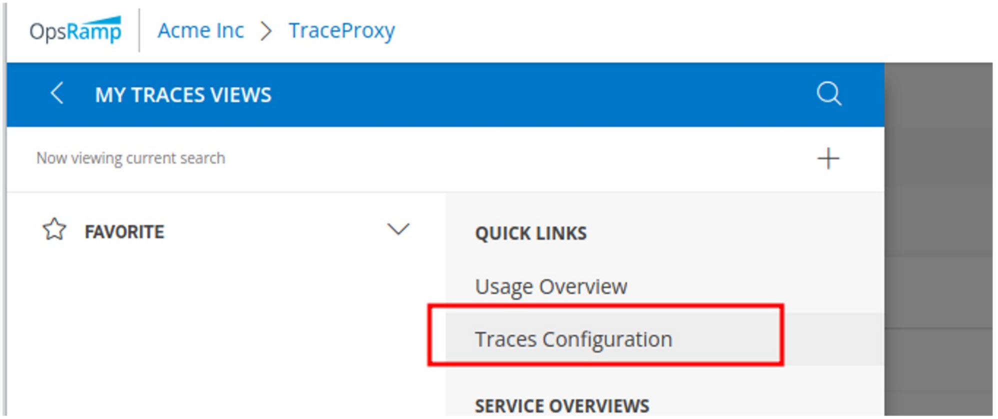
Deploying OpsRamp Trace Proxy

Navigate to Trace Getting Started Page in Portal

1. Select Traces from infrastructure tab

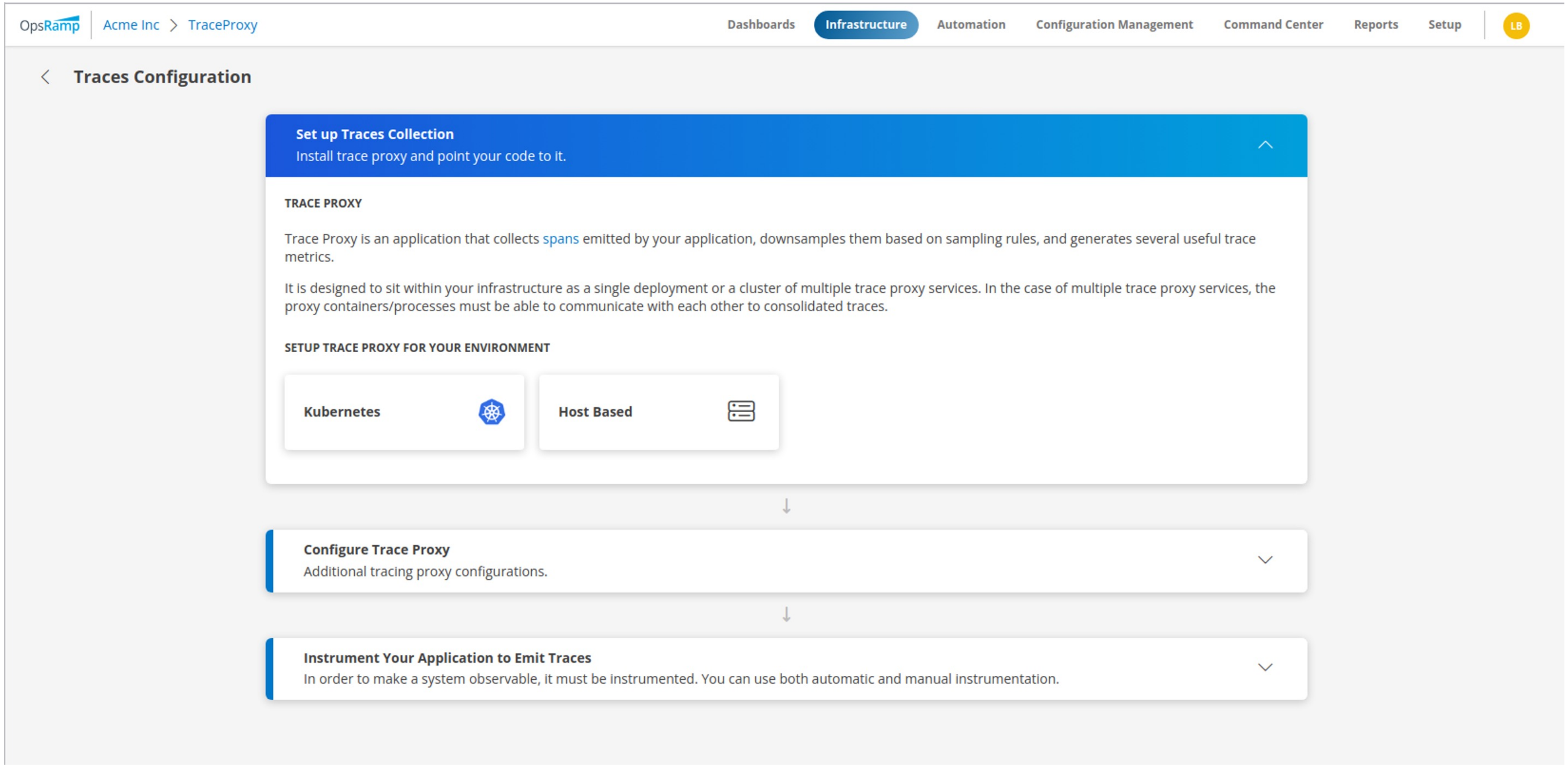


1. Click on the hamburger menu on the top left side of the page and select “Traces Configuration”



Deploying OpsRamp Trace Proxy

3. Follow the onscreen instructions to deploy the trace proxy



Traces Configuration

Set up Traces Collection
Install trace proxy and point your code to it.

TRACE PROXY

Trace Proxy is an application that collects spans emitted by your application, downsamples them based on sampling rules, and generates several useful trace metrics.

It is designed to sit within your infrastructure as a single deployment or a cluster of multiple trace proxy services. In the case of multiple trace proxy services, the proxy containers/processes must be able to communicate with each other to consolidated traces.

SETUP TRACE PROXY FOR YOUR ENVIRONMENT

Kubernetes

Host Based

Configure Trace Proxy
Additional tracing proxy configurations.

Instrument Your Application to Emit Traces
In order to make a system observable, it must be instrumented. You can use both automatic and manual instrumentation.

Setting Up Demo App In VM

We will be using a sample spring application with java auto instrumentation for traces for this example

Prerequisites

- Have Java 17 or higher installed

Setup

```
# Clone the repository for petclinic project
git clone https://github.com/spring-projects/spring-petclinic.git

# move into that directory & run the mvnw command
cd spring-petclinic
./mvnw package

# Download the opentelemetry java agent for auto-instrumentation
wget https://github.com/open-telemetry/opentelemetry-java-instrumentation/releases/latest/download/opentelemetry-javaagent.jar
```

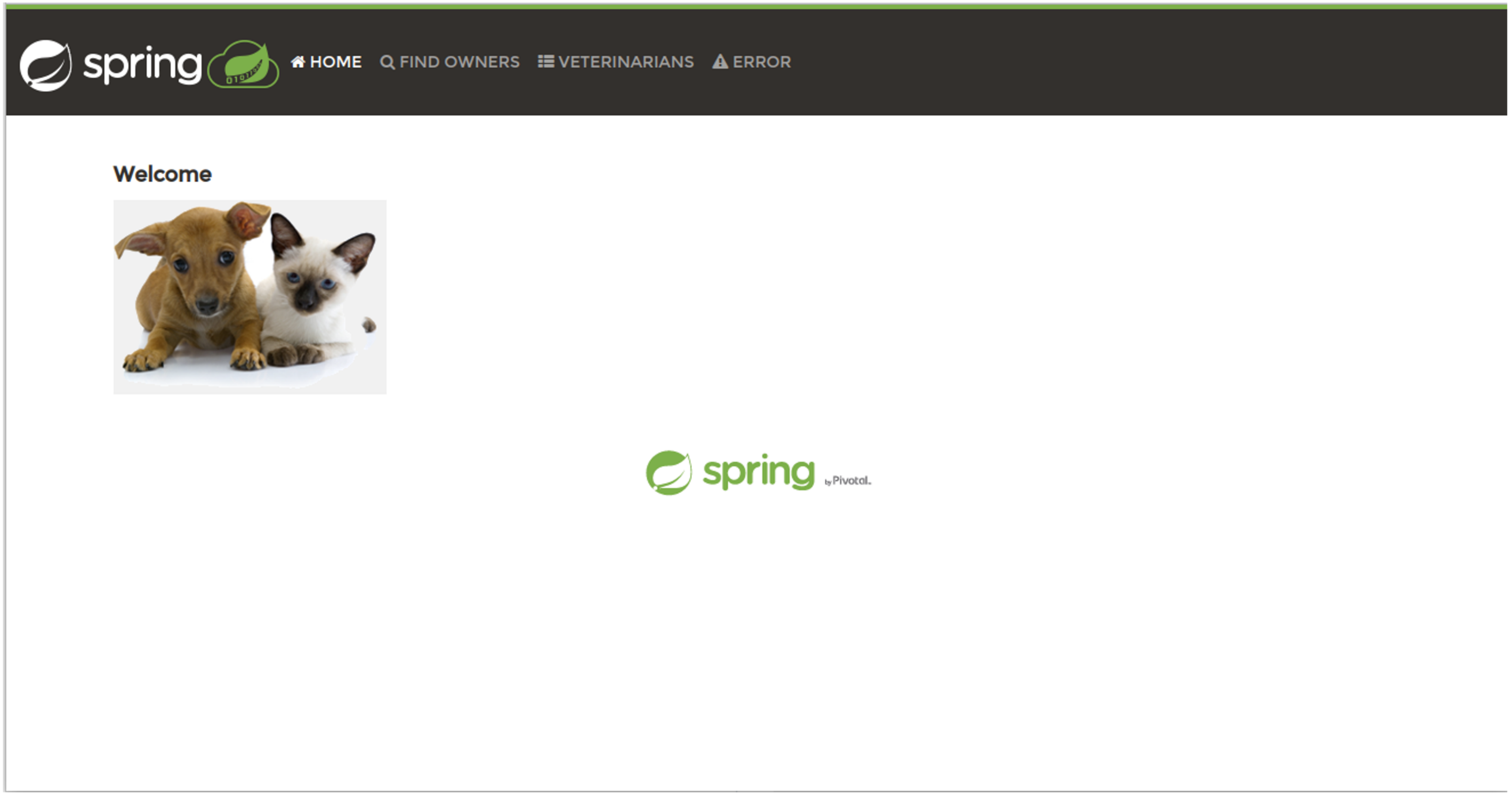
Running

```
# For exporting traces via GRPC
java -javaagent:opentelemetry-javaagent.jar \
-Dotel.exporter.otlp.endpoint=http://localhost:9090 \
-Dotel.resource.attributes=service.name=PetClinicSampleApp \
-jar target/*.jar

# For exporting traces via HTTP
java -javaagent:opentelemetry-javaagent.jar \
-Dotel.exporter.otlp.traces.protocol=http/protobuf \
-Dotel.exporter.otlp.endpoint=http://0.0.0.0:8082 \
-Dotel.resource.attributes=service.name=PetClinicSampleApp \
-jar target/*.jar
```

Setting Up Demo App In VM ....Contd

Petclinic application web ui is accessible at localhost:8080 once the we run the command and each operation performed here results in a new trace



Setting Up Demo App in Kubernetes (YAML)

We will be using Open Telemetry Demo application for Kubernetes

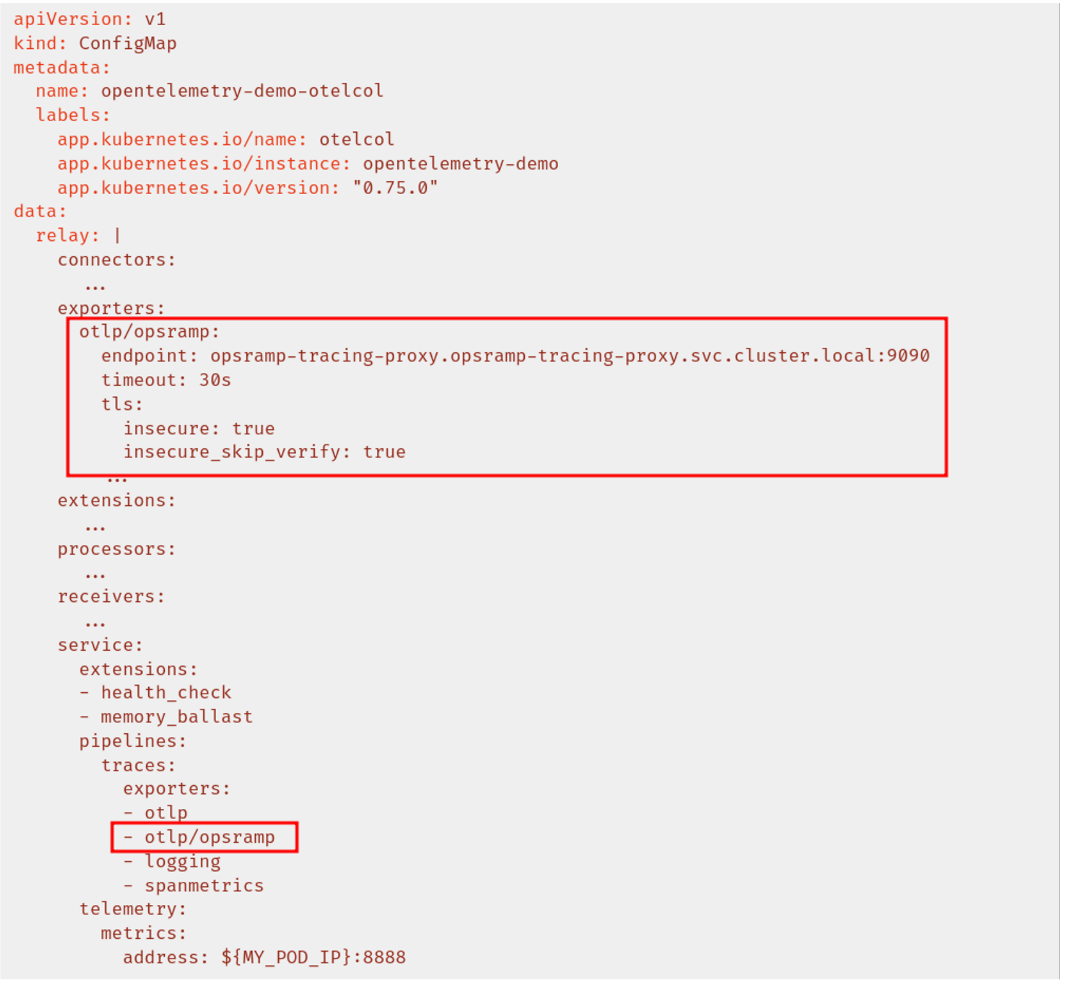
It can be installed using normal kubernetes deployment files or using Helm

Deployment Using Normal YAML

To export Traces to OpsRamp Tracing Proxy the following configuration needs to be added to ConfigMap

1. Download the Open Telemetry Deployment Yaml opetelemetry-demo.yaml
1. Adding opsramp exporter in the exporter section of the configuration as shown towards to the right
2. apply the yaml in kubernetes

```
# Creating the Namespace
kubectl create namespace otel-demo
# Deploying the application
kubectl apply --namespace otel-demo -f opentelemetry-demo.yaml
# Port forwarding to view the UI
kubectl port-forward svc/my-otel-demo-frontendproxy 8080:8080
```

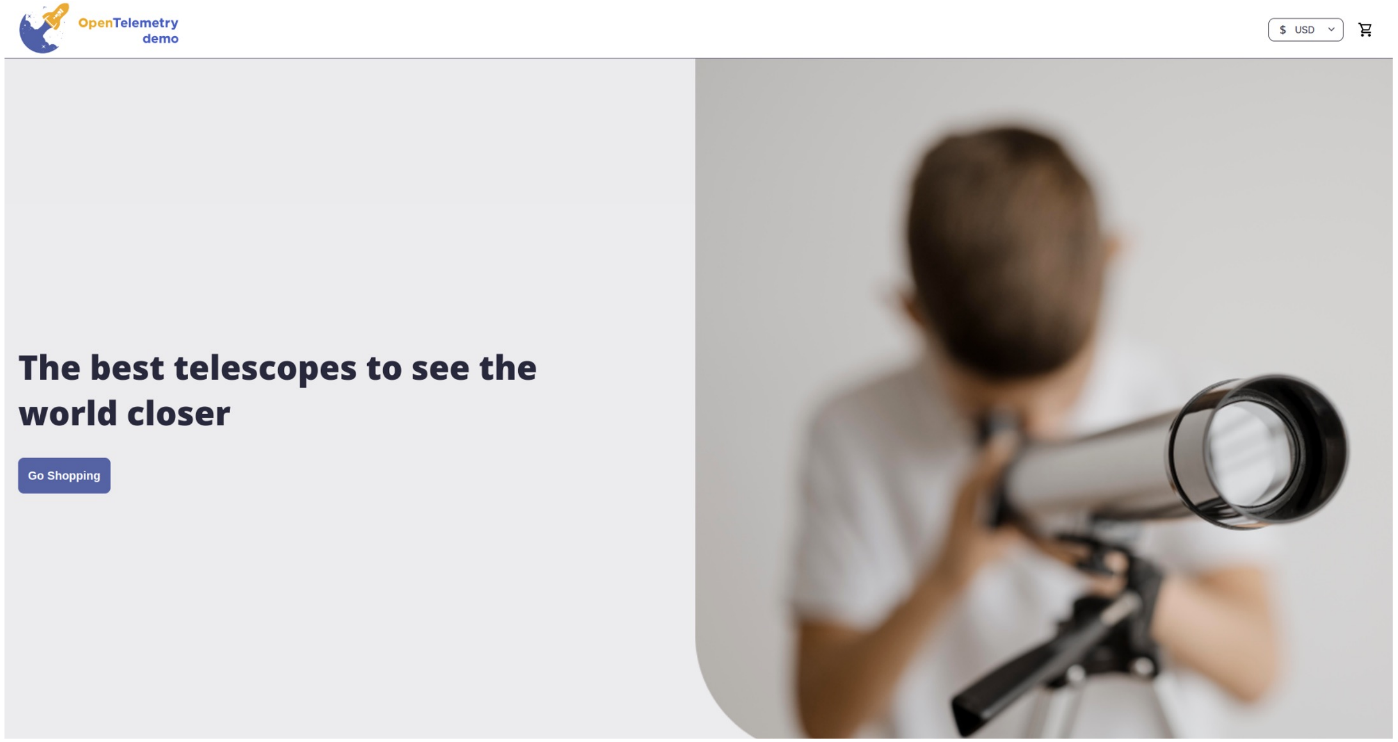


```
apiVersion: v1
kind: ConfigMap
metadata:
  name: opentelemetry-demo-otelcol
  labels:
    app.kubernetes.io/name: otelcol
    app.kubernetes.io/instance: opentelemetry-demo
    app.kubernetes.io/version: "0.75.0"
data:
  relay: |
    connectors:
      …
    exporters:
      otlp/opsramp:
        endpoint: opsramp-tracing-proxy.opsramp-tracing-proxy.svc.cluster.local:9090
        timeout: 30s
        tls:
          insecure: true
          insecure_skip_verify: true
      …
    extensions:
      …
    processors:
      …
    receivers:
      …
    service:
      extensions:
      - health_check
      - memory_ballast
      pipelines:
        traces:
          exporters:
          - otlp
          - otlp/opsramp
          - logging
          - spanmetrics
      telemetry:
        metrics:
          address: ${MY_POD_IP}:8888
```

Setting Up Demo App in Kubernetes

With the frontendproxy port-forward set up, you can access:

- Webstore: http://localhost:8080/
- Grafana: http://localhost:8080/grafana/
- Feature Flags UI: http://localhost:8080/feature/
- Load Generator UI: http://localhost:8080/loadgen/
- Jaeger UI: http://localhost:8080/jaeger/ui/



Setting Up Demo App in Kubernetes (helm)

Deployment Using Helm

```
# Installing Helm (https://helm.sh/docs/intro/install/)
curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash

# Adding helm repository
helm repo add open-telemetry https://open-telemetry.github.io/opentelemetry-helm-charts

# Download the values file and modify to add exporter config similar to what we added in YAML Deployment
wget https://raw.githubusercontent.com/open-telemetry/opentelemetry-helm-charts/main/charts/opentelemetry-demo/values.yaml

# Install the chart with modified values file
helm install my-otel-demo open-telemetry/opentelemetry-demo --values my-values-file.yaml
```

Application - Trace Instrumentation Demo

High Level Steps to follow while instrumenting an application

- Define Resource & Exporter
- Create a Tracer from defined resource & exporter
- Define Span with context & name where ever required in application logic
- Propagate context of parent span in all child spans
- Set attributes, events & errors where ever necessary

For detailed examples for each programming language with sample instrumented code, please refer doc https://github.com/opsramp/tracing-docs

OpsRamp

a Hewlett Packard Enterprise company

Part 2:

Observability in Action

OpsRamp

a Hewlett Packard Enterprise company

Part 3:

Q & A

Thank you

Neil Pearson
neil.pearson@hpe.com

OpsRamp

a Hewlett Packard Enterprise company

OpsRamp

a Hewlett Packard Enterprise company