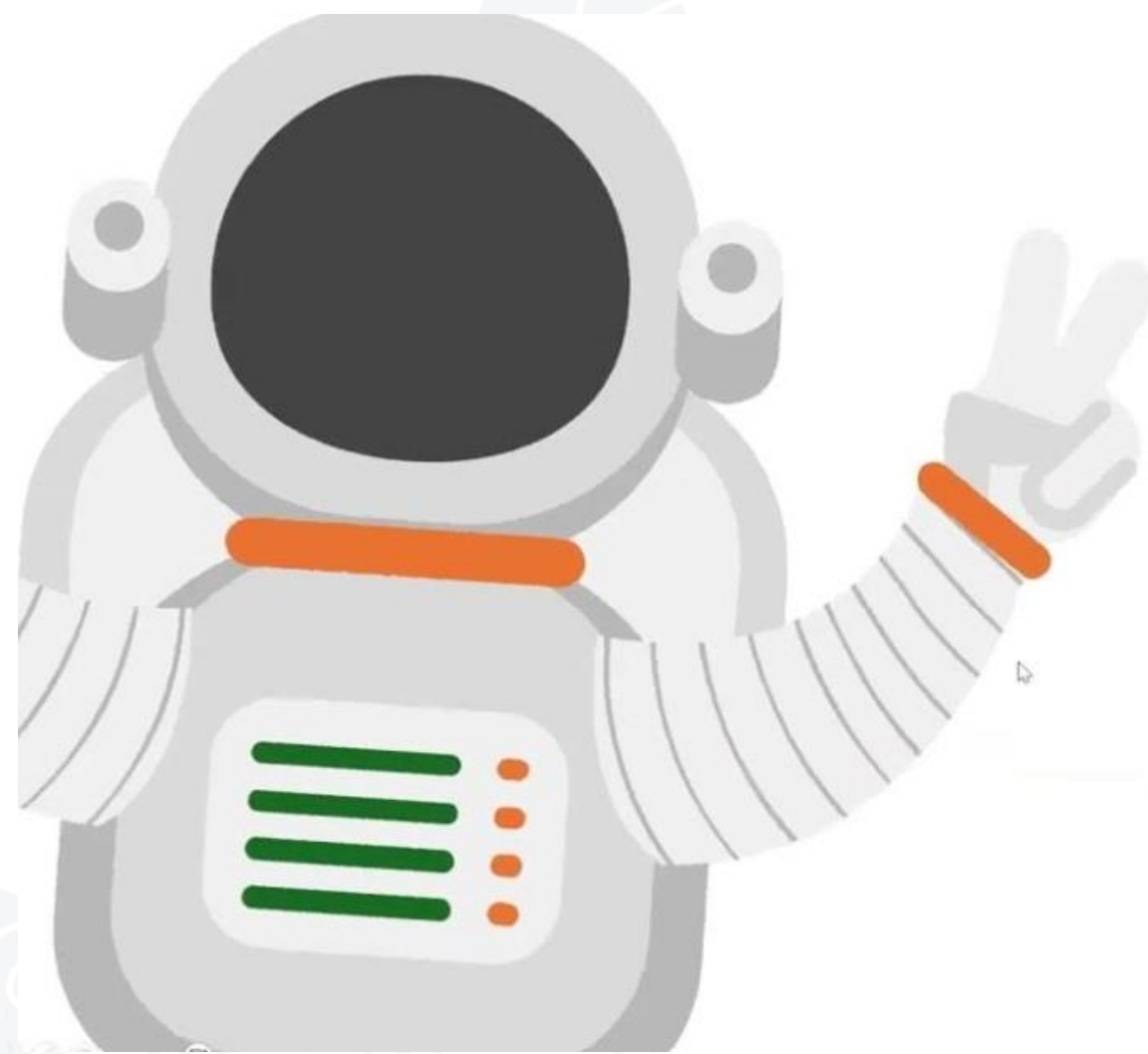


# Discovering and Ingesting Data into Dynatrace



Dynatrace  
OneAgent

Cause I am keeping a close watch  
on everything



# Overview



**Collecting data with OneAgent**

**Demo of OneAgent installation**

**ActiveGate proxy service component**

**Demo of ActiveGate implementation**

**Data collection with logs, APIs, and integrations**

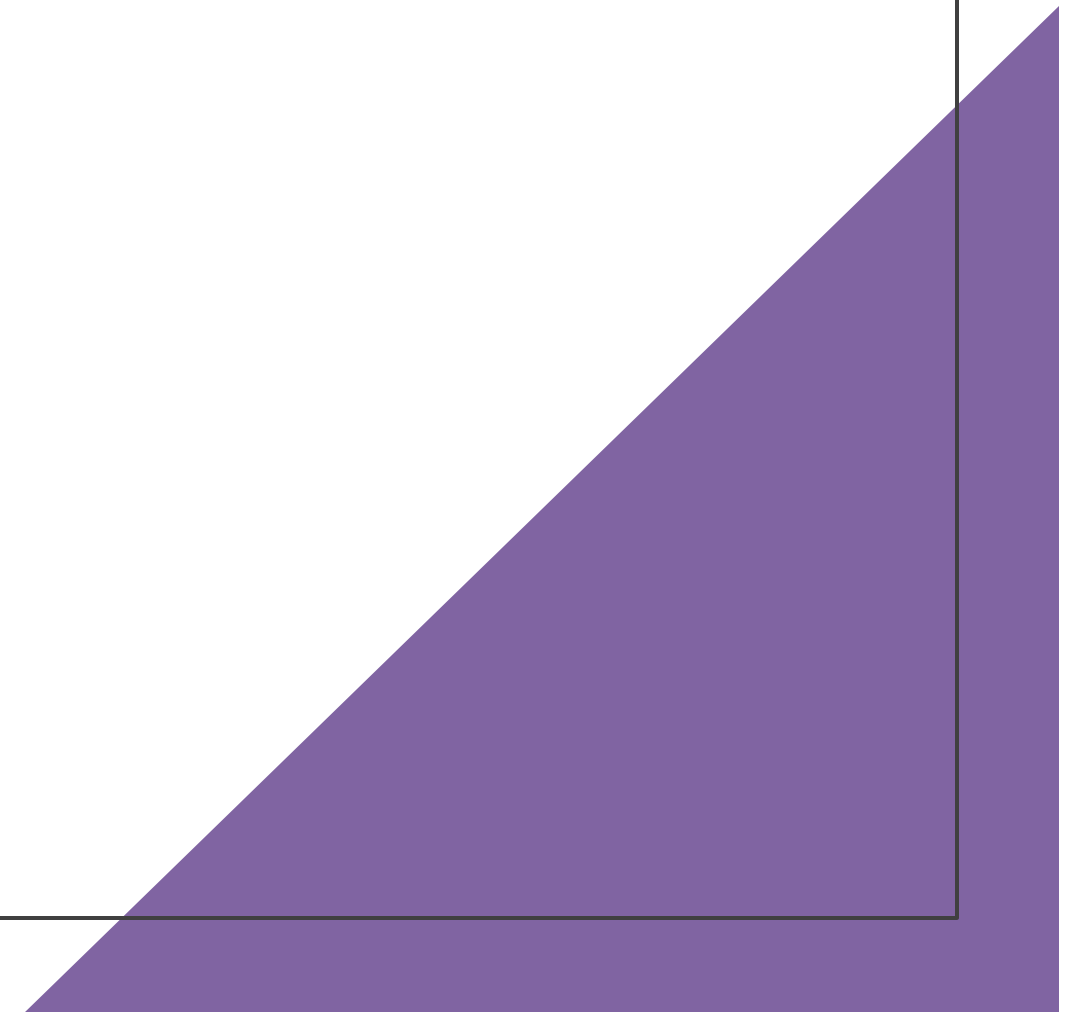
**Demo of other collection method**

**Review agentless data collection with RUM and synthetics**

**Collecting performance data with agents is a staple of performance monitoring.**

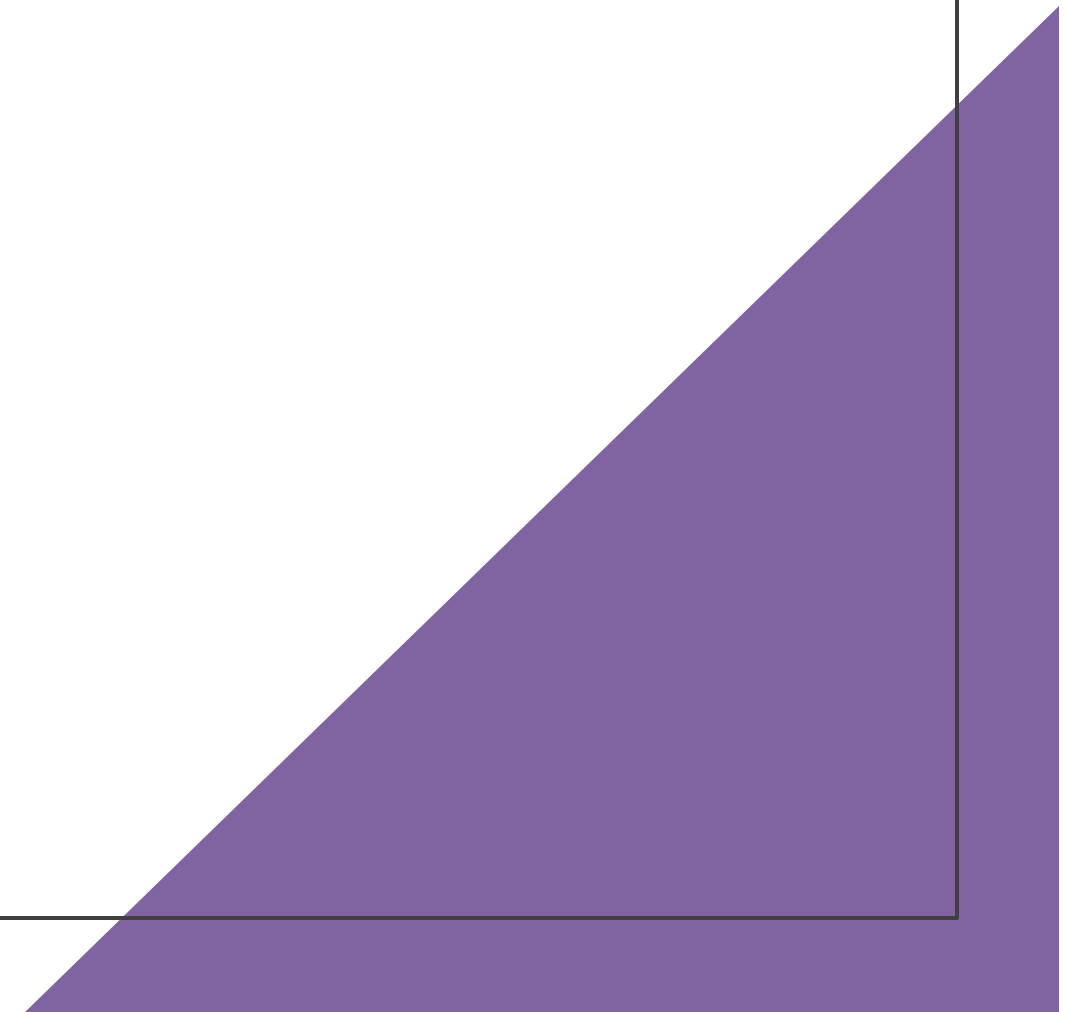
# What is OneAgent?

**Dynatrace OneAgent** is a single, host-level monitoring agent that provides **automatic, code-free observability** for applications, services, infrastructure, and user experience.



# In simple terms:

You install **one agent**, and it automatically instruments **everything running on that host or container**—without changing application code.



# What Dynatrace OneAgent does

OneAgent automatically collects and correlates:

- **Infrastructure metrics** (CPU, memory, disk, network)
- **Application performance** (response times, errors, throughput)
- **Distributed traces** (end-to-end requests)
- **Service dependencies** (who calls whom)
- **Logs & log context**
- **Real User Monitoring (RUM)** via automatic JS injection
- **Security signals** (runtime vulnerabilities, attacks)

All of this feeds into Dynatrace's Smartscape topology and Davis AI.

# Why it's called "OneAgent"

Traditional setup

Infra agent

APM agent

Tracing agent

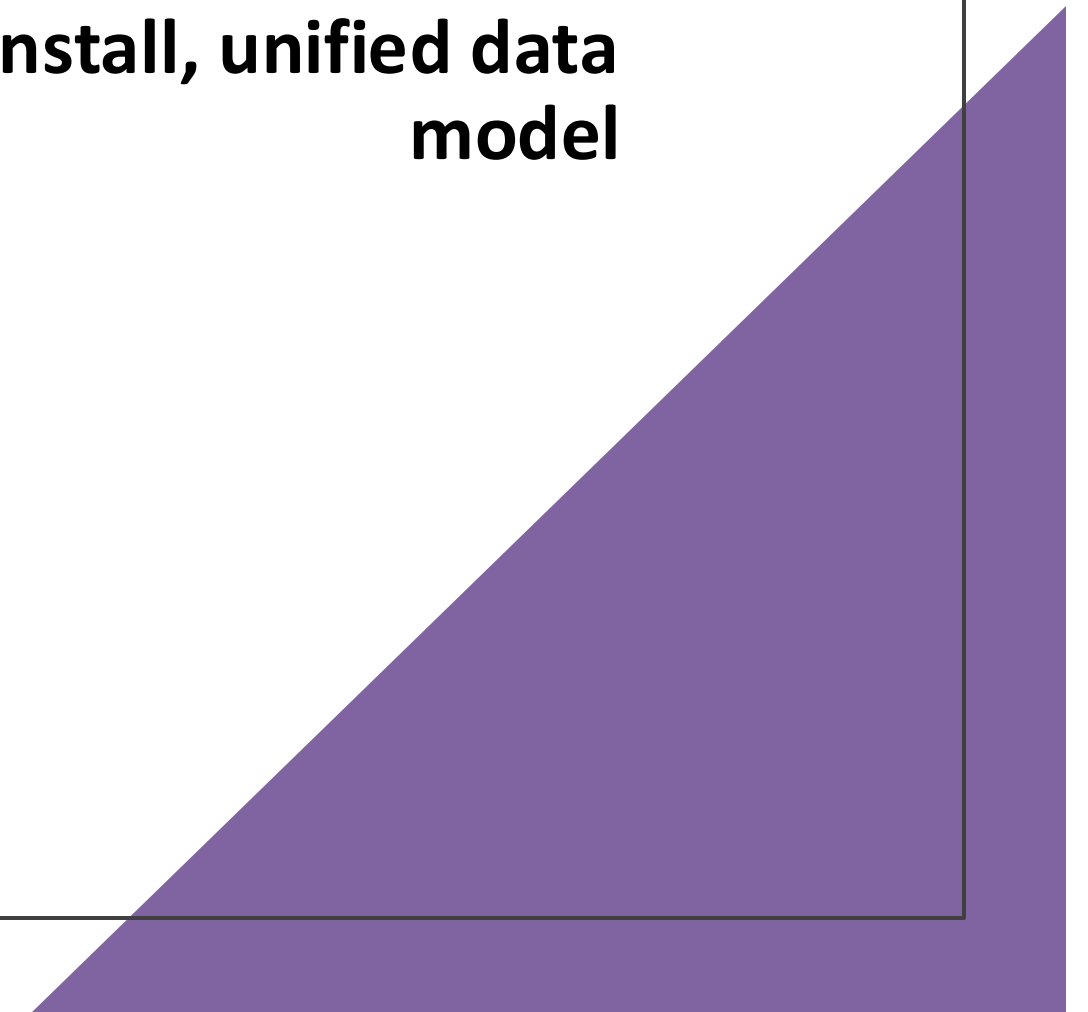
Log agent

RUM agent

Dynatrace



**Single install, unified data model**



# How OneAgent works

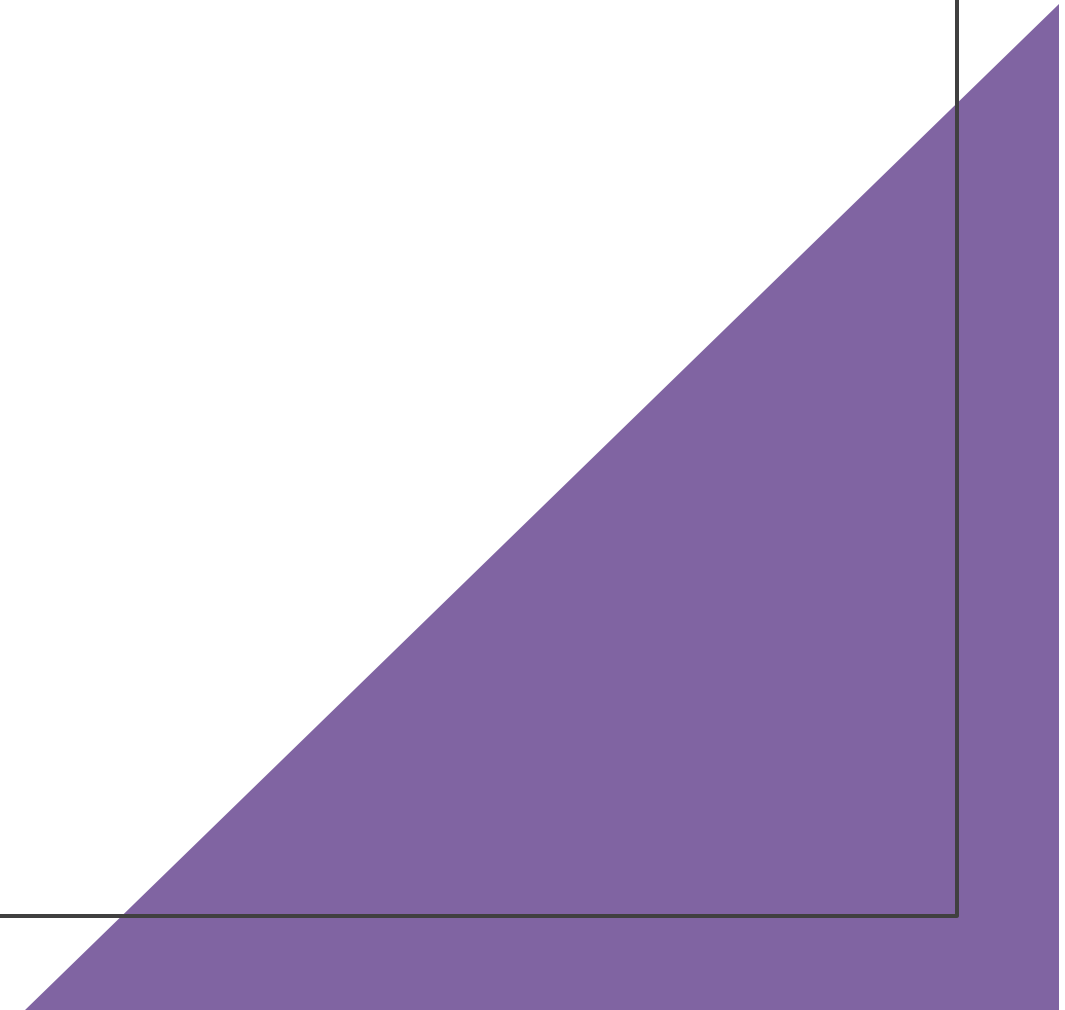
## **1. Installed on host / VM / node**

Linux, Windows

Cloud VMs

Kubernetes nodes (DaemonSet)

Containers (via runtime injection)



# How OneAgent works



## 2. Automatic discovery



OneAgent continuously detects:



Running processes



Technologies (Java, .NET, Node.js, PHP, Go, etc.)



Frameworks (Spring, ASP.NET, Express, DB drivers, message queues)



No manual configuration required.

# How OneAgent works

## 3. Code-free instrumentation (key feature)

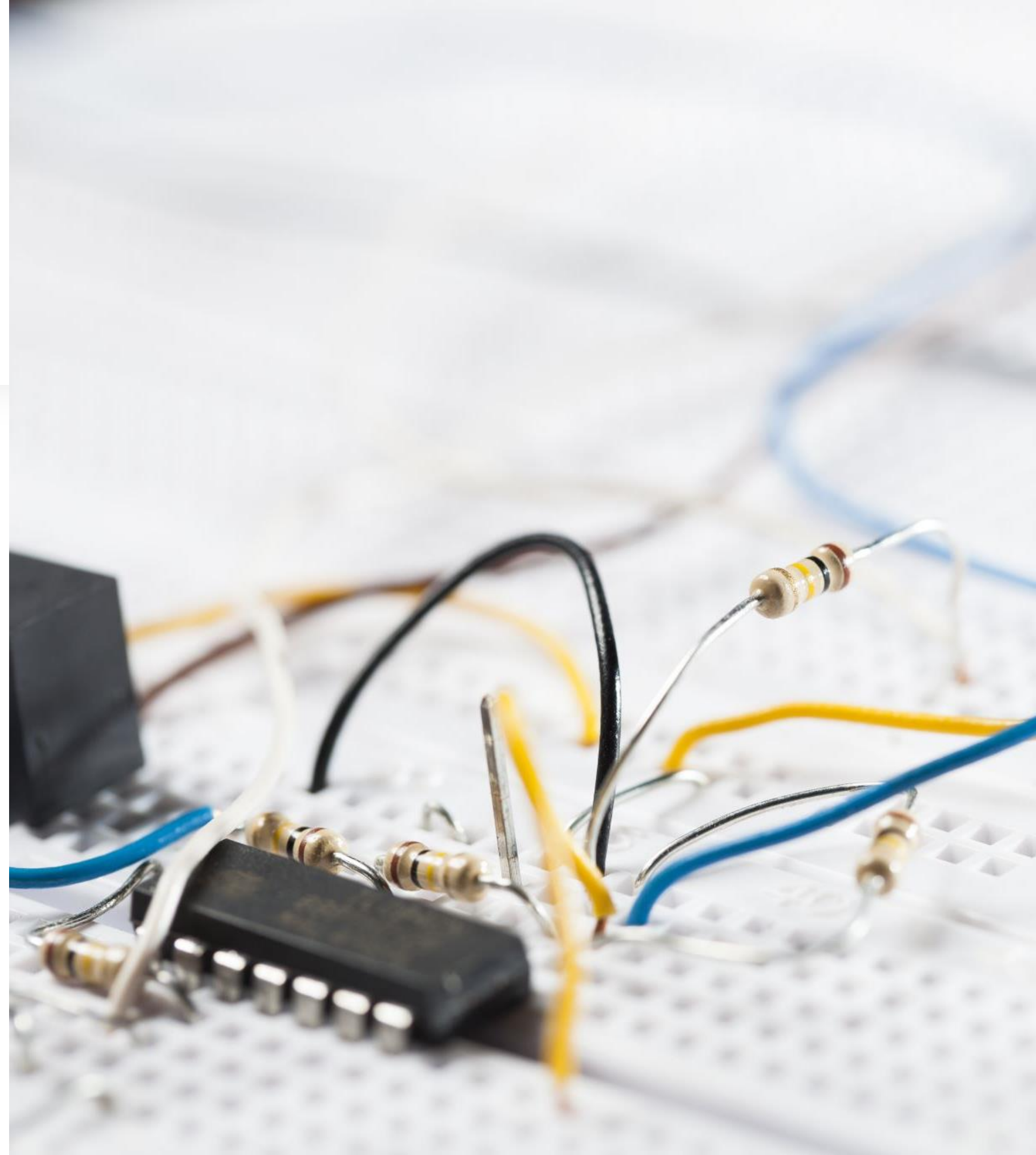
Internally, OneAgent:

Injects technology-specific modules **at runtime**

Hooks into runtimes (JVM, CLR, Node, libc, etc.)

Intercepts key operations (HTTP, DB calls, messaging)

⚠ This happens **without modifying your application code or binaries.**



# How OneAgent works



4. Automatic tracing & topology



From these hooks, OneAgent:



Creates distributed traces



Detects services automatically



Builds real-time dependency maps (Smartscape)

# How OneAgent works

---

## 5. AI-powered analysis

---

All data is analyzed by **Davis AI**, which:

---

Detects anomalies

---

Finds root cause automatically

---

Suppresses noise

---

Explains *why* something is slow or broken

# What makes OneAgent different from other agents

- ✓ **Zero code changes**
- ✓ **No manual service definitions**
- ✓ **Automatic dependency mapping**
- ✓ **Single data model (metrics + traces + logs)**
- ✓ **Built for dynamic environments (cloud, Kubernetes)**

# Simplifying Data Collection with OneAgent

OneAgent collects application and infrastructure data

Configuration is challenging

Improved application performance with OneAgent

# Evolution of Dynatrace Software Agents

**Dynatrace started with multiple software agents**

**Separate agents for applications and infrastructures**

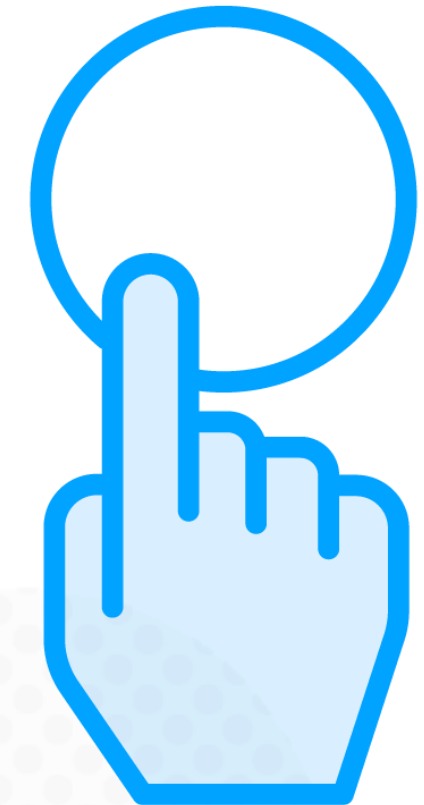
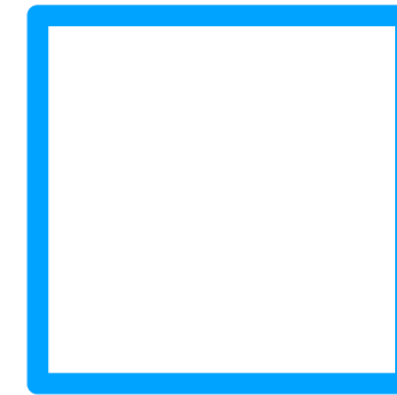
**OneAgent: merged version of all agents**


# OneAgent: All-in-One Monitoring Solution

One agent to monitor all your components

Collects on-premise, in the cloud, and in  
containers

Works with no upfront configuration





# Dynatrace OneAgent — Internal Architecture & Data Flow

---

## Big picture

---

OneAgent is **not a single binary doing everything**. Internally, it is a **host-level runtime instrumentation platform** that:

---

Observes the OS

---

Injects itself into application processes

---

Hooks runtime & framework execution points

---

Correlates everything into one causal graph

---

Streams normalized data into Dynatrace's platform

---

# 1. Host-level foundation (OS & kernel view)

## What runs on the host

### When installed, OneAgent deploys:

- **Core agent process(es)** (always running)
- **Technology-specific code modules** (loaded on demand)

### Data collected at this layer

- CPU usage
- Memory allocation
- Disk I/O
- Network traffic
- Process lifecycle (start/stop/fork)
- Container cgroups & namespaces

### How it works internally

- Uses **OS APIs and kernel interfaces**
- Reads process tables, file descriptors, sockets
- Associates resource usage with **specific processes & containers**

→ This is how OneAgent knows *what is running* before touching application code.

## 2. Continuous process & technology detection

OneAgent constantly scans:

- Process metadata
- Loaded libraries
- Runtime signatures
- Startup parameters

It automatically classifies processes as:

- JVM
- .NET CLR
- Node.js
- PHP
- Go
- Web servers
- Databases
- Message brokers

→ Detection is **dynamic** — if a new service starts, it's discovered and instrumented automatically.

# 3. Runtime injection (core of “code-free” instrumentation)

This is the most important internal mechanism.

## What “injection” means

- OneAgent **loads its monitoring module inside the target process**, so it can observe execution *from the inside*.

## How injection happens (conceptually)

- Depends on runtime and OS, but patterns include:
  - Runtime attach (JVM, CLR)
  - Shared library preload
  - Process startup instrumentation
  - Container runtime injection

- ⚠ **No application code is modified**
- ⚠ **No recompilation**
- ⚠ **No SDKs**

# 4. Deep runtime hooking (inside the process)

Once injected, OneAgent hooks critical execution points:

## Application layer hooks

- HTTP request entry points
- Web framework dispatchers
- Middleware chains
- RPC frameworks

## Dependency hooks

- Database drivers
- HTTP clients
- Message queues
- gRPC / REST clients

## What it captures

- Start/end timestamps
- Errors/exceptions
- Request context
- Call relationships

→ This creates automatic distributed traces without trace annotations.

# 5. Distributed tracing without headers configurati on

## OneAgent:

- Automatically injects correlation headers
- Reads incoming headers
- Propagates context across services

## Even across:

- Different languages
- Different protocols
- Message queues
- Legacy systems



That's why Dynatrace can build end-to-end traces automatically.

# 6. Metrics: how they're generated

## Infrastructure metrics

Collected at OS level:

- CPU, memory, disk, network
- Container-level metrics
- Process-level metrics

## Application metrics

Derived from runtime hooks:

- Response time
- Throughput
- Error rate
- Method-level timings


## Key point

Metrics are **not separate collectors** — they're derived from:

*the same execution events* used for tracing

→ This is why metrics, traces, and topology always match.

# 7. Logs & log enrichment

- OneAgent:
- Detects log files automatically
- Tails logs at OS level
- Enriches log lines with:
  - Trace ID
  - Service name
  - Host
  - Process
  - Request context
-  Logs become **trace-aware** without changing log statements.

# 8. Real User Monitoring (RUM)

## Server-side HTML injection

For web apps, OneAgent:

- Intercepts outgoing HTML responses
- Injects Dynatrace JavaScript automatically

## Browser data collected

- Page load timing
- User actions
- JavaScript errors
- Frontend → backend correlation

→ Frontend clicks are automatically linked to backend traces.

# 9. Smartscape topology (real-time dependency graph)

From all collected signals, OneAgent feeds:

- Service detection
- Process grouping
- Network relationships

Dynatrace builds a **live causal topology**:

- Hosts → processes → services → dependencies

→ No manual service definitions required.

Smartscape Processes

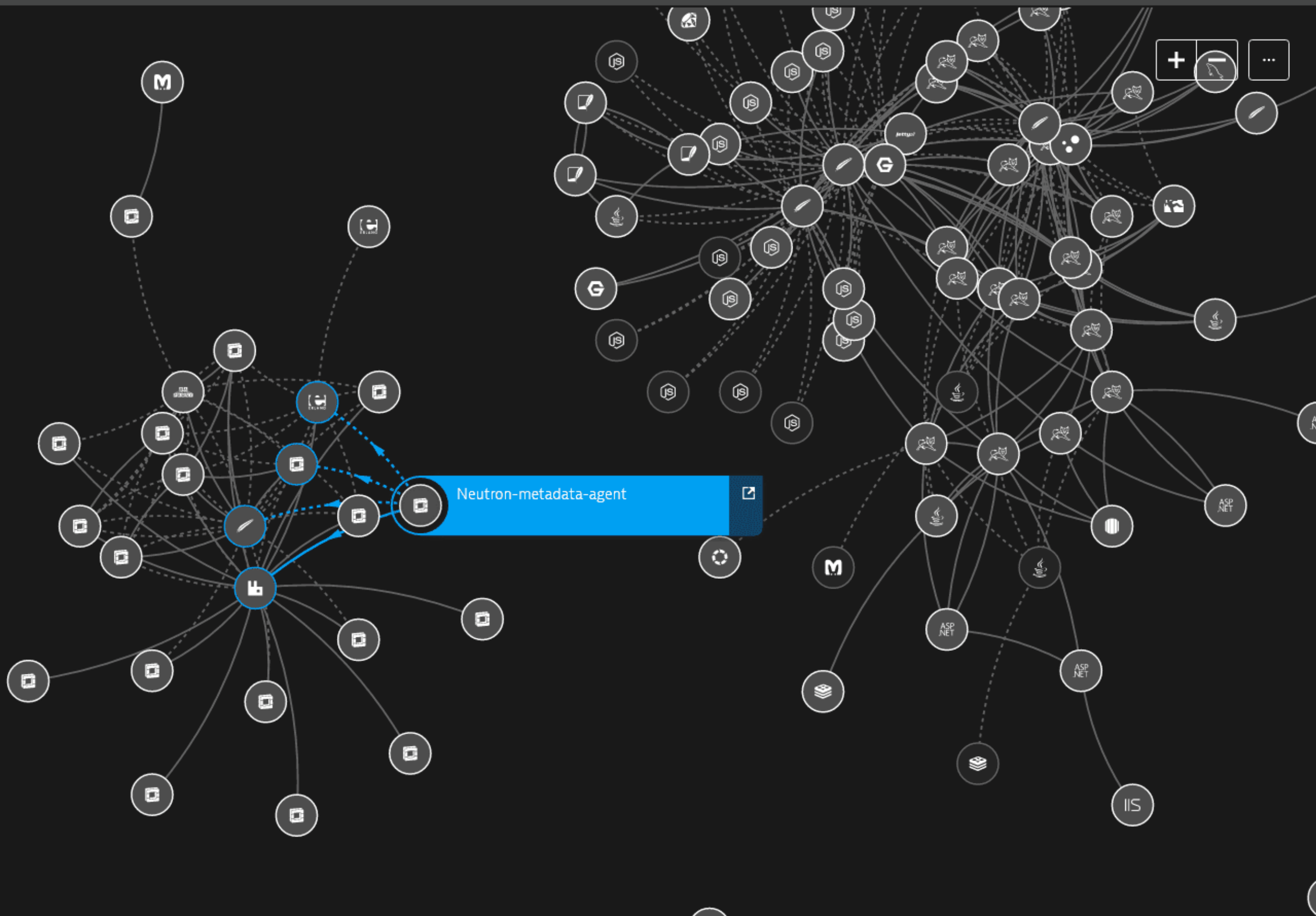
Applications 12

Services 122

Processes 194

Hosts 1/33

Datacenters 6



# 10. Davis AI & causation engine

---

01

All OneAgent data is:

- Time-aligned
- Entity-linked
- Causally connected

02

Davis AI then:

- Detects anomalies
- Finds the *root cause* (not symptoms)
- Suppresses noise
- Explains impact

03

→ This works only because OneAgent collects **deep, correlated data**.

# How OneAgent collects *all* data types (summary table)

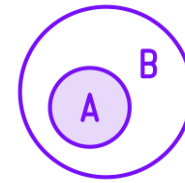
---

Data type	How it's collected
Infrastructure	OS & kernel instrumentation
Metrics	Derived from execution + OS data
Traces	Runtime hooks + context propagation
Logs	OS-level log capture + enrichment
Topology	Dependency inference from traces
RUM	Automatic HTML/JS injection
Security	Runtime behavior & vulnerability context

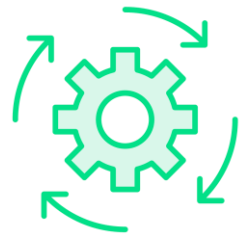
# OneAgent Installation and Modes



**One agent install on hosts**



**Two modes of operation**



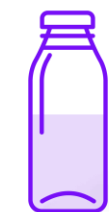
**Automatically identifies running processes**



**Full-stack mode**



**Injects appropriate code modules for supported apps**



**Infrastructure mode**

# Collecting Data with OneAgent

**OneAgent requires process restart after installation**

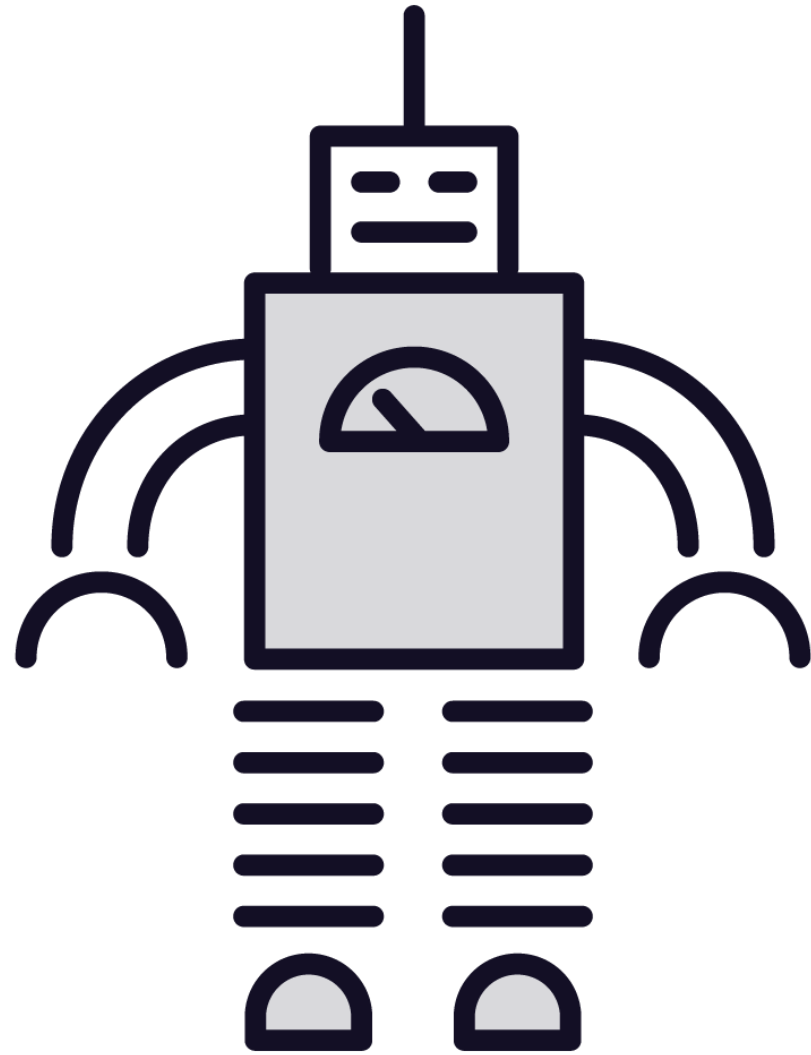
**Data flows back to Dynatrace over port 443**

**One-way communication: OneAgent to Dynatrace**

**Includes system and application processes for monitoring**

**Injects JavaScript tag for RUM**

**Collects log files for troubleshooting**



## **OneAgent supports many different platforms**

- Supports Windows, Linux, and z/OS
- Works in AWS, Azure, and GCP
- Compatible with Kubernetes, OpenShift, and Cloud Foundry

**One installation, many technologies by default**

# Features of Dynatrace OneAgent

- 1. Full-Stack Visibility:** OneAgent provides full-stack monitoring, allowing organizations to gain insights into the performance and behavior of every component in their IT environment, from application code and databases to servers and cloud services.
- 2. Automatic Discovery:** OneAgent automatically discovers and maps application dependencies, making it easier to understand the relationships between different parts of an application or microservices architecture.
- 3. Real-Time Performance Monitoring:** It offers real-time monitoring of application performance, including response times, error rates, and resource utilization. This helps in identifying performance bottlenecks and issues promptly.
- 4. AI-Powered Insights:** Dynatrace employs artificial intelligence (AI) and machine learning to analyze performance data and provide actionable insights, including root cause analysis and anomaly detection.

# Features of Dynatrace OneAgent

- 1. Cloud and Container Support:** OneAgent is compatible with cloud platforms like AWS, Azure, and Google Cloud, as well as containerized environments like Kubernetes and Docker.
- 2. User Experience Monitoring:** It can capture user interactions with web and mobile applications, providing insights into end-user experience and performance.
- 3. Security and Compliance:** OneAgent includes security features for monitoring and detecting vulnerabilities and threats within the monitored environment.
- 4. Automatic Updates:** Dynatrace regularly updates OneAgent to ensure it stays compatible with evolving technologies and maintains the highest level of performance monitoring.

# OneAgent collects

<b>Layer</b>	<b>Data Collected</b>
Infrastructure	CPU, memory, disk, network
Applications	Performance, errors, services, tracing
User Experience	Session replays, clicks, device data
Logs	Custom logs, error logs (optional)
Security (optional)	Vulnerabilities, exploits

# OneAgent collects: Infrastructure Monitoring Data

Tracks the health of your **hosts, VMs, and containers**.

Category	Examples
CPU usage	Per process and system-wide
Memory usage	RAM usage, swap, memory leaks
Disk I/O	Read/write speeds, disk usage
Network	Bandwidth, connections, errors
Host metrics	Hostname, OS, uptime, processes
Container metrics	Docker/Kubernetes container resources

# OneAgent collects: Application and Service Monitoring

Monitors applications automatically, **without code changes**.

Category	Examples
Method-level tracing	Which functions/methods are slow
Response times	For services, APIs, and applications
Error rates	Exceptions, failed requests
Service topology	Which services call which others
Third-party services	External APIs and service calls

# OneAgent collects: Real User Monitoring (RUM)

Captures what **real users** do in your web or mobile apps.

Category	Examples
User sessions	Clicks, page loads, scrolls
Browser metrics	Load time, rendering time, network time
Geo-location	User's country/city
Device & browser	Type, version, screen size
Errors	JavaScript errors, crashes

# OneAgent collects: Log Monitoring (Optional)

Captures what **real users** do in your web or mobile apps.

If enabled, it can **ingest logs** for troubleshooting.

Category	Examples
Application logs	Custom log files, structured/unstructured
Error logs	Error stack traces, warnings
Log patterns	Useful for detecting anomalies

# OneAgent collects: Distributed Tracing (PurePath)

Dynatrace's **PurePath** technology tracks **every transaction end-to-end**.

Category	Examples
Transaction flow	From frontend → backend → database
Bottlenecks	Which microservice slowed down the request
Trace IDs	For identifying a transaction uniquely

# OneAgent collects: Network Monitoring

Monitors **connection and communication** between services.

Category	Examples
Request latency	Between services and databases
Network errors	Timeouts, connection failures
Protocols	HTTP, TCP, gRPC, etc.

# OneAgent collects: Security Metrics (via Application Security module)

If enabled, it can also collect **vulnerability data**.

Category	Examples
Library vulnerabilities	Known CVEs in open-source packages
Exploit detection	Attempts to inject malicious code
Package analysis	Language-specific package versions

# Dynatrace OneAgent Supported Languages

Programming Language	Support Level	Notes
Java	Fully Supported	Supports JVMs like Oracle HotSpot, OpenJDK, IBM Semeru, Eclipse Temurin, etc.
.NET / .NET Core	Fully Supported	Supports both Framework and Core versions on Windows & Linux
Node.js	Fully Supported	Includes libraries like Winston, PostgreSQL (pg). Node.js 14 support <b>ended</b> in 2024
PHP	Fully Supported	Monitors standard PHP-based web applications
Go (Golang)	Fully Supported	Supports OpenTelemetry (v1.0 – v1.7). Go 1.18 support <b>ended</b> in Feb 2024
Python	SDK-based Support	Python 3.7 support <b>ended</b> in Aug 2024. Monitoring via OneAgent SDK
C/C++	SDK-based Support	Instrumentation through OneAgent SDK for C/C++

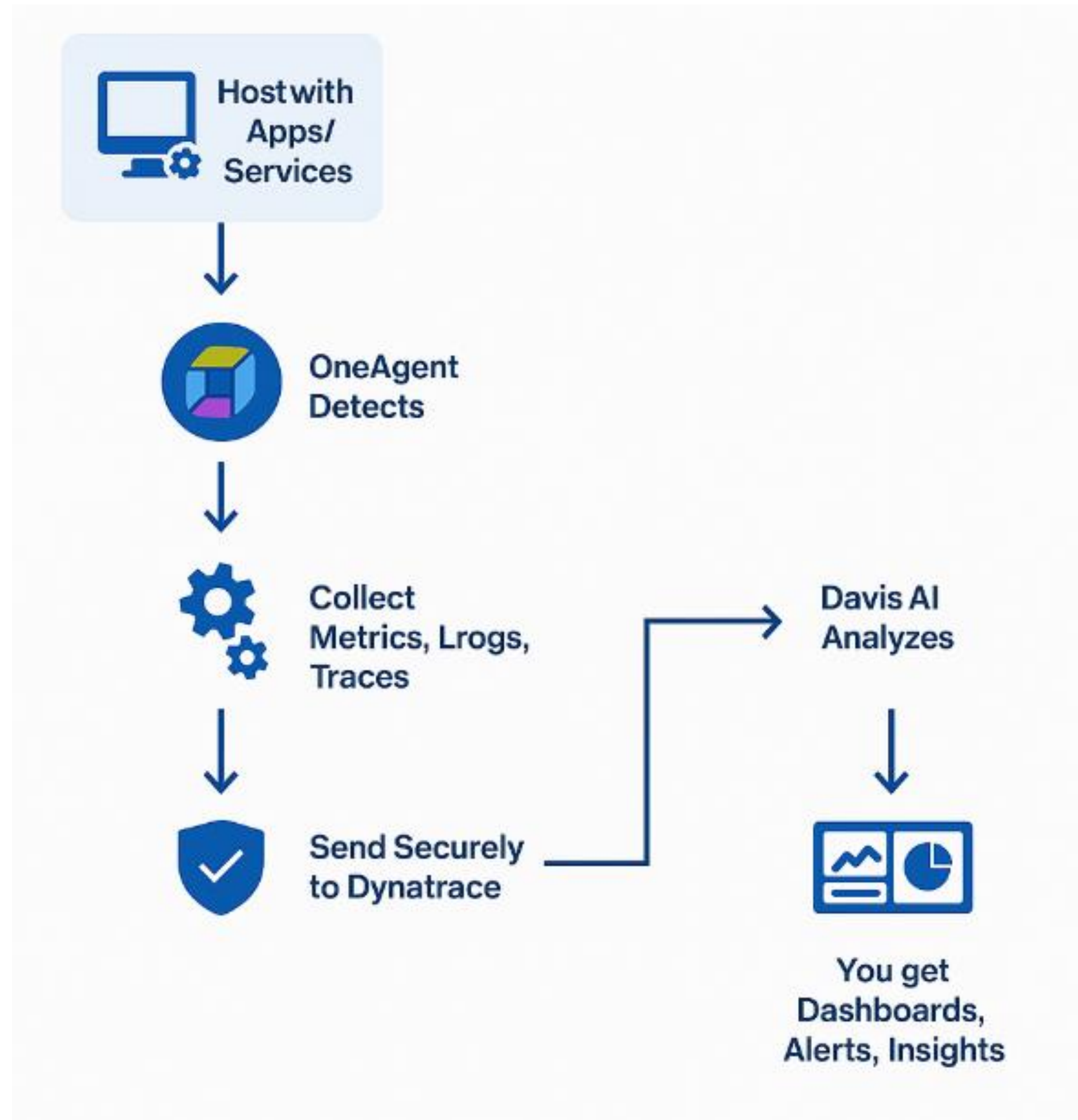
# Languages Not Supported Out-of-the-Box by Dynatrace OneAgent

Programming Language	Support Status	Monitoring Options
Ruby	✘ Not Supported	Use OneAgent SDK for custom instrumentation
Elixir	✘ Not Supported	Use OneAgent SDK for custom instrumentation
Erlang	✘ Not Supported	Use OneAgent SDK for custom instrumentation
Scala	⚠ Partial Support	Monitored when running on JVM; limited visibility into Scala-specific constructs
Kotlin	⚠ Partial Support	Monitored when running on JVM; limited visibility into Kotlin-specific constructs
R	✘ Not Supported	Use OneAgent SDK for custom instrumentation
Perl	✘ Not Supported	Use OneAgent SDK for custom instrumentation
Haskell	✘ Not Supported	Use OneAgent SDK for custom instrumentation
Rust	✘ Not Supported	Use OneAgent SDK for custom instrumentation
Swift	✘ Not Supported	Use OneAgent SDK for custom instrumentation
Objective-C	✘ Not Supported	Use OneAgent SDK for custom instrumentation
COBOL	✘ Not Supported	Use OneAgent SDK for custom instrumentation
Fortran	✘ Not Supported	Use OneAgent SDK for custom instrumentation
VBScript	✘ Not Supported	Use OneAgent SDK for custom instrumentation
PowerShell	✘ Not Supported	Use OneAgent SDK for custom instrumentation
Shell Scripts	✘ Not Supported	Use OneAgent SDK for custom instrumentation

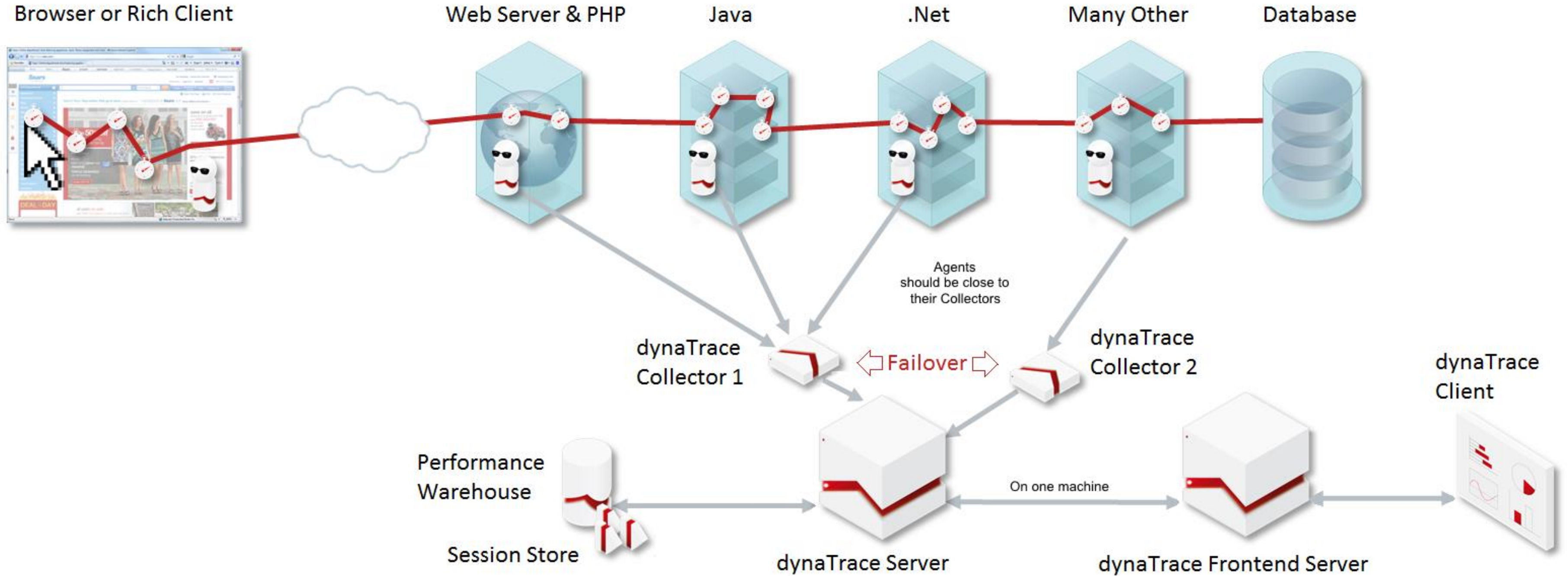
# Solutions for Unsupported Languages in Dynatrace

Method	Description	Use Cases
<b>1. OneAgent SDK</b>	Manual instrumentation using SDKs for custom metrics, events, and tracing.	C, C++, Python, Go, and others
<b>2. OpenTelemetry Integration</b>	Use language-specific OpenTelemetry SDKs to generate traces, metrics, and logs and export to Dynatrace	Ruby, Rust, Swift, Scala, Kotlin, etc.
<b>3. Log Monitoring</b>	Use log files to monitor behaviors, errors, and patterns via Dynatrace log ingestion.	Shell scripts, Perl, R, etc.
<b>4. External API Monitoring</b>	Monitor app behavior using API endpoints with Dynatrace Synthetic Monitoring.	Any RESTful-based or CLI-interfacing applications
<b>5. Custom Metrics via API</b>	Send metrics manually using Dynatrace's Metrics Ingestion API (via HTTP POST).	Any system that can make HTTP calls
<b>6. Infrastructure Monitoring</b>	Even if code-level support is lacking, you can monitor host performance, network, CPU, memory, etc.	Legacy COBOL apps, Fortran, etc.

# How Dynatrace OneAgent works?



# How Dynatrace OneAgent works?



# Key Features That Enable Automatic Data Collection

<b>Feature</b>	<b>What It Does</b>
Auto-discovery	Detects new processes/services automatically
Runtime injection	Injects monitoring code dynamically
Real user monitoring	Captures frontend user interactions
Distributed tracing	Tracks requests across services end-to-end (PurePath)
Smart sampling	Minimizes overhead with intelligent data collection
Lightweight agent	One binary, low resource consumption

# Dynatrace Oneagent Monitoring Mode

## **Full-Stack Monitoring**

Enables both infrastructure + deep application.level monitoring (PurePath, RUM, etc.). Auto-instruments apps, services, containers, etc.

## **Infrastructure Monitoring**

Collects only host-level metrics (CPU, RAM, disk, network, processes), no code-level insight

## **Discovery Mode**

Minimal mode used for initial setup or troubleshooting. Only detects the host and some metadata, no metrics or application monitoring

# Dynatrace Oneagent Monitoring Mode

<b>Mode</b>	<b>Description</b>
<b>Full-Stack Monitoring</b>	Enables both infrastructure + deep application-level monitoring (PurePath, RUM, etc.). Auto-instruments apps, services, containers, etc.
<b>Infrastructure Monitoring</b>	Collects only host-level metrics (CPU, RAM, disk, network, processes), no code-level insight.
<b>Discovery Mode</b>	Minimal mode used for initial setup or troubleshooting. Only detects the host and some metadata, no metrics or application monitoring.

# Dynatrace Oneagent Monitoring Mode

The screenshot shows the Dynatrace OneAgent Monitoring Mode configuration page. The breadcrumb navigation is: Hosts > ip-172-31-45-54.ap-south-1.compute.internal > Settings > Host monitoring > Monitoring Mode. The left sidebar contains navigation options: Dynatrace, Search (Ctrl K), Apps, Notebooks, Dashboards, Workflows, Hub, Hosts Classic, Collapse, Support, and user profile (Rajesh K rgg53291). The main content area has a note: "Note that for this schema only, the GET objects api will usually not return any objects as these settings are stored on the agents - please use the GET effective values api instead." Below the note is a "Monitoring mode" dropdown menu with an information icon. The dropdown is open, showing four options: "Full-stack" (selected), "Discovery", "Infrastructure", and "Full-stack". The selected "Full-stack" option is highlighted in light blue and includes the text: "also includes application performance, user experience data, code-level visibility and PurePath insights". Below the dropdown is a "Last 30 days" filter button. At the bottom of the page, there is a legend for monitoring modes: Full stack (dark blue), Infrastructure (medium blue), Discovery (light blue), and Unmonitored (yellow). The date "13. Apr" is visible at the bottom right.

Hosts > ip-172-31-45-54.ap-south-1.compute.internal > Settings > Host monitoring > Monitoring Mode

Note that for this schema only, the [GET objects](#) api will usually not return any objects as these settings are stored on the agents - please use the [GET effective values](#) api instead.

Monitoring mode ⓘ

- Full-stack
- Discovery  
includes topology discovery and basic health monitoring of your host
- Infrastructure  
also includes detailed performance monitoring of your host and enables Network monitoring, Log monitoring, and Extensions
- Full-stack**  
also includes application performance, user experience data, code-level visibility and PurePath insights

Last 30 days

13. Apr

■ Full stack ■ Infrastructure ■ Discovery ■ Unmonitored

# How to change Dynatrace Oneagent Monitoring Mode?

Method	Command / Steps
<b>Command Line</b>	oneagentctl --set-monitoring-mode=FULL_STACK etc.
<b>UI</b>	Go to <b>Host</b> → <b>Edit</b> → <b>Monitoring Mode</b>
<b>Config File</b>	Add MONITORING_MODE=... in config and restart agent

<https://www.devopsschool.com/blog/dynatrace-oneagent-monitoring-mode/>

One Agent Command

## Dynatrace OneAgent Command-Line Interface (`oneagentctl`)

The `oneagentctl` tool allows you to configure and manage OneAgent settings post-installation. It's available on all supported platforms: Linux, Windows, and AIX.

### Location

- **Linux/AIX:** `/opt/dynatrace/oneagent/agent/tools/oneagentctl`
- **Windows:** `%PROGRAMFILES%\dynatrace\oneagent\agent\tools\oneagentctl.exe`

**Note:** Administrator/root privileges are required to execute `oneagentctl` commands.

## General Commands

<b>Command</b>	<b>Description</b>	<b>Example</b>
<code>--help</code>	Displays all supported parameters and their usage.	<code>./oneagentctl --help</code>
<code>--version</code>	Shows the installed OneAgent version.	<code>./oneagentctl --version</code>
<code>--restart-service</code>	Restarts the OneAgent service to apply configuration changes.	<code>./oneagentctl --restart-service</code>

## Configuration Commands

### Communication Settings

Command	Description	Example
<code>--set-server</code>	Sets the communication endpoint (e.g., ActiveGate or Dynatrace Cluster).	<code>./oneagentctl --set-server=https://my-server.com:9999/communication</code>
<code>--get-server</code>	Displays the current communication endpoint(s).	<code>./oneagentctl --get-server</code>
<code>--set-tenant</code>	Sets the environment ID for the OneAgent.	<code>./oneagentctl --set-tenant=abc123456</code>
<code>--get-tenant</code>	Displays the current environment ID.	<code>./oneagentctl --get-tenant</code>
<code>--set-tenant-token</code>	Sets the tenant token used for authentication.	<code>./oneagentctl --set-tenant-token=abcdefg123456790</code>
<code>--get-tenant-token</code>	Displays the current tenant token.	<code>./oneagentctl --get-tenant-token</code>

# Proxy Configuration

Command	Description	Example
<code>--set-proxy</code>	Configures the proxy server for OneAgent communication.	<code>./oneagentctl --set-proxy=my-proxy.com:8080</code>
<code>--get-proxy</code>	Displays the current proxy configuration.	<code>./oneagentctl --get-proxy</code>
<code>--set-no-proxy</code>	Specifies domains/IPs to bypass the proxy.	<code>./oneagentctl --set-no-proxy=localhost,127.0.0.1</code>
<code>--get-no-proxy</code>	Displays the current no-proxy settings.	<code>./oneagentctl --get-no-proxy</code>

## Monitoring Configuration

Command	Description	Example
<code>--set-monitoring-mode</code>	Sets the monitoring mode: fullstack, infra-only, or discovery.	<code>./oneagentctl --set-monitoring-mode=infra-only</code>
<code>--set-network-zone</code>	Assigns the OneAgent to a specific network zone.	<code>./oneagentctl --set-network-zone=zone1</code>
<code>--get-network-zone</code>	Displays the current network zone assignment.	<code>./oneagentctl --get-network-zone</code>
<code>--set-host-group</code>	Assigns the host to a specific host group.	<code>./oneagentctl --set-host-group=MyHostGroup</code>
<code>--set-host-name</code>	Overrides the automatically detected host name.	<code>./oneagentctl --set-host-name=custom-hostname</code>
<code>--set-host-property</code>	Adds custom metadata properties to the host.	<code>./oneagentctl --set-host-property=Environment=Dev</code>
<code>--set-host-tag</code>	Adds custom tags to the host.	<code>./oneagentctl --set-host-tag=role=webserver</code>

## Log Monitoring

### Command

`--set-app-log-content-access`

`--get-app-log-content-access`

### Description

Enables or disables access to application log content.

Displays the current setting for application log content access.

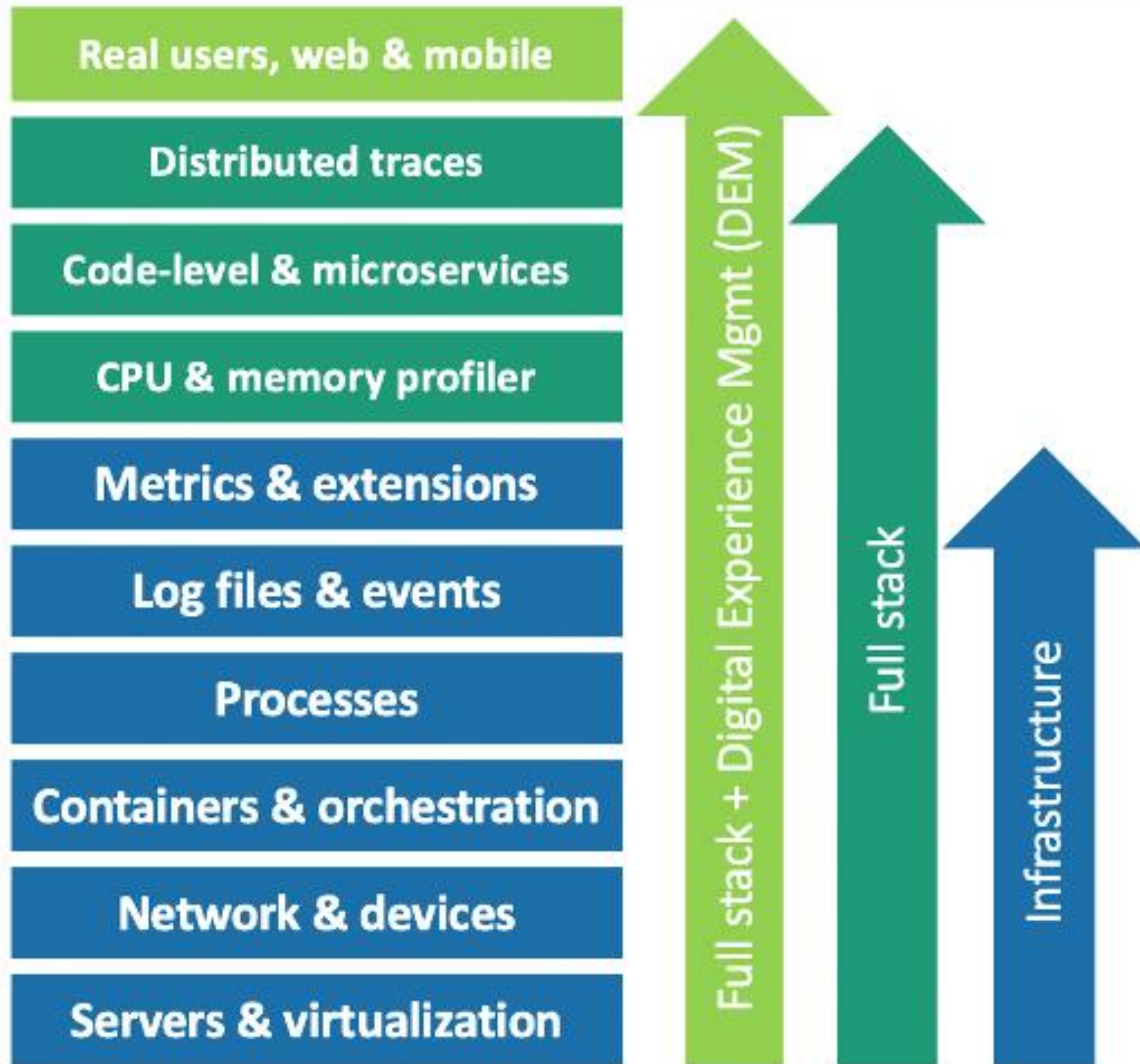
### Example

```
./oneagentctl --set-app-log-content-access=true
```

```
./oneagentctl --get-app-log-content-access
```

# One Agent Configuration

<https://docs.dynatrace.com/docs/ingest-from/dynatrace-oneagent/oneagent-configuration-via-command-line-interface>



## OneAgent – Capabilities

- Application Monitoring – Java, .NET, PHP, Node.js
- Web Server Monitoring – Apache, IIS, Nginx
- Database Monitoring
- Third Party and Cloud Content
- Real and Synthetic User Monitoring
- AWS, VMware, Docker
- Host Monitoring – CPU, Memory, Disk Utilization
- Process Monitoring
- Network Monitoring
- Log File Monitoring

**Demo**



**Step-by-step guide to install OneAgent**

**Tips for getting more application visibility**