**Dynatrace-** End-to-end monitoring (App + DB + infra) with root cause

Dynatrace is a powerful Application Performance Monitoring (APM) and observability tool used to monitor, analyze, and optimize application performance, infrastructure, and user experience in real-time.

**What is Dynatrace?**

Dynatrace is an AI-powered software intelligence platform that provides full-stack monitoring for:

* Applications
* Infrastructure (servers, cloud, containers)CPU, memory,disk, network, process
* Networks
* End-user experiences or Digital Experience.

**Why are we using Dynatrace?**

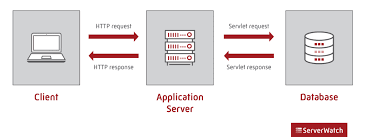
Dynatrace helps teams to:

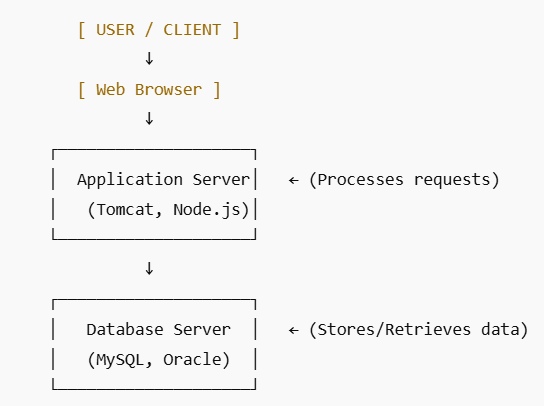
1. **Monitor system health** – real-time visibility into apps, servers, services, and network.
2. **Detect performance bottlenecks** – slow APIs, memory issues, CPU spikes, etc.
3. **Root cause analysis** – AI-powered problem detection (Davis AI).
4. **Reduce MTTR** – quickly identify and resolve issues before users are impacted.
5. **Improve user experience** – track frontend (browser/mobile) performance.
6. **Enable DevOps & SRE** – integrates with CI/CD pipelines, supports automation.
7. **Cloud & microservices visibility** – excellent for Kubernetes, AWS, Azure, etc.

**What are the main components/features in Dynatrace?**

| **Feature** | **Description** |
| --- | --- |
| **OneAgent** | Installed on hosts; collects performance data from OS, apps, containers. |
| **Smartscape** | Visual map showing real-time dependencies between services, processes, and hosts. |
| **PurePath** | Transaction tracing feature showing end-to-end user requests down to method-level code. |
| **Davis AI** | AI engine that automatically detects issues and provides root cause analysis. |
| **Real User Monitoring (RUM)** | Tracks user activity and experience on web/mobile apps. |
| **Synthetic Monitoring** | Simulates user actions to test performance from different locations. |
| **Log Monitoring** | Collects and analyzes logs for errors, trends, and insights. |
| **Dashboarding** | Custom dashboards for performance, usage, alerts, SLAs, etc. |
| **Alerting** | Configurable alerts when metrics go beyond thresholds. |
| **Cloud Native Support** | Deep monitoring for AWS, Azure, GCP, Kubernetes, Docker, etc. |
| **Integration & APIs** | Integrates with tools like JIRA, Slack, Jenkins, Prometheus, etc. |

**How App Server and DB Server Work Together**





### Scenario: A user places an order

* **Frontend**: User selects a product, enters shipping details, and clicks "Place Order".
* **App Server**: Validates order, applies business rules, and calls backend APIs.
* **Database Server**: Saves order details into tables like orders, users, inventory.

🔍 If something is slow:

* It could be the **App Server** taking time (slow Java code or overloaded CPU).
* Or it could be the **DB Server** taking time (slow SQL query or high DB load).

**In Performance Testing (JMeter/Dynatrace/etc.)**

During performance testing, you simulate multiple users to observe how the system behaves under load.

| **Server Type** | **What to Monitor** | **Example Issues** |
| --- | --- | --- |
| **App Server** | CPU, Memory, Threads, Response Time, Errors | High CPU usage due to code bottlenecks |
| **DB Server** | Query Execution Time, Locks, Connections, IOPS | Slow queries, DB connection pool exhaustion |

JMeter + Dynatrace combo gives **end-to-end visibility**: request, app layer, DB call, server metrics, and logs.

#### Scenario:

You run a load test using JMeter and see that order placement API takes **10 seconds**.

Using **Dynatrace**, you trace the request:

* App Server CPU is 30%
* DB query takes 8.5 seconds due to missing index

Note: Without an index: the DB must scan every row in the table (called full table scan) to find a match.

With an index: the DB can jump directly to the matching row, making it much faster.

✅ **Dynatrace AI (Davis)** will automatically highlight:

"The root cause is a slow SQL query in orders table taking 8.5s due to full table scan."

### ****What is JBoss?****

JBoss (now called **WildFly**) is a **Java-based Application Server** developed by Red Hat.

You can install it on any server: physical, virtual (like EC2), or container (Docker).

It is used to **deploy and run Java applications** — especially **Java EE (Jakarta EE)** applications like:

* REST APIs
* Web apps (JSP/Servlets)
* EJBs (Enterprise Java Beans)
* JMS (Messaging)
* and more

**When a user sends a request to a Java-based web app (like an order management system):**

* 🔹 JBoss processes the request.
* 🔹 Executes the Java business logic (e.g., login validation, order creation).
* 🔹 Connects to a Database Server (like MySQL, Oracle) to fetch/update data.

**[ Client (Browser/Postman) ]**

**↓**

**[ Application Server (JBoss) ]**

**↓**

**[ Database Server (MySQL, Oracle, etc.) ]**

**Example Real-Time Log Output (While Test Is Running)**

INFO [org.jboss.as.server] (default task-23) Processing order for user 1001

INFO [org.jboss.as.server] (default task-23) Order saved in 420ms

WARN [org.hibernate.SQL] (default task-12) Query took 5.3s — possible missing index

ERROR [org.jboss.as.controller] (default task-11) java.lang.NullPointerException

During JMeter Test Execution:

* Monitor server.log in real-time to check if the server is responding correctly
* Identify exceptions like NullPointerException, TimeoutException
* Measure how long APIs or services are taking (if you added timing logs)
* Spot performance-related errors immediately

**What is PostgreSQL?**

**PostgreSQL** (also called **Postgres**) is a powerful, open-source **relational database management system (RDBMS)** used to store and manage structured data.

It supports:

* SQL (Structured Query Language)
* Advanced features like **indexes**, **stored procedures**, **triggers**, and **JSON**
* ACID compliance (Atomicity, Consistency, Isolation, Durability)

### PostgreSQL as a Database Server:

When you install PostgreSQL, it runs a **database server process** on your machine (usually on port 5432). This server:

* Accepts SQL queries from clients or app servers
* Processes those queries
* Returns results (data) or performs actions (insert/update/delete)

## **What are AWS Logs?**

**AWS Logs** refer to the logs generated by applications, services, and infrastructure **running on Amazon Web Services (AWS)**.

These logs help you **monitor, troubleshoot, and analyze** everything happening inside your AWS environment — including **application servers, databases, APIs, and more**.

## **What is AWS CloudWatch?**

**Amazon CloudWatch** is a **monitoring and observability service** provided by AWS.

It is **not** the same as a JBoss server. Instead, it is used to **collect, monitor, and analyze logs, metrics, and events** from:

* AWS resources (like EC2, RDS, Lambda, etc.)
* Applications running **on AWS or on-prem**
* Custom metrics from any service or app (including JBoss)

| **Log Type** | **Description** |
| --- | --- |
| **CloudWatch Logs** | Central service to collect, store, and analyze logs from EC2, Lambda, ECS, RDS, custom apps, etc. |
| **EC2 Instance Logs** | System logs (like /var/log/messages, /var/log/httpd/access.log, etc.) from Linux or Windows servers |
| **Elastic Load Balancer (ELB) Logs** | Logs each HTTP/HTTPS request going through the load balancer |
| **RDS Logs (PostgreSQL, MySQL, etc.)** | Database logs: slow query logs, error logs, general query logs |
| **VPC Flow Logs** | Network logs: captures IP-level traffic in and out of VPC/subnets |
| **Lambda Logs** | Function logs automatically pushed to CloudWatch |
| **S3 Access Logs** | Logs access requests to S3 buckets |
| **CloudTrail Logs** | Logs all **API calls** made in your AWS account for auditing and security |

## **Example: EC2 + JBoss + PostgreSQL on AWS**

Let’s say you deploy this architecture:

* EC2 running JBoss App Server
* RDS PostgreSQL database
* JMeter runs load test

| **Component** | **Logs You Can Capture** |
| --- | --- |
| **JBoss on EC2** | Application logs in /opt/jboss/standalone/log/server.log (send to CloudWatch) |
| **EC2 Instance** | OS logs, memory/cpu alerts |
| **RDS PostgreSQL** | Slow queries, DB errors (visible in RDS Logs or CloudWatch) |
| **Load Balancer** | Access logs to analyze traffic |
| **CloudWatch Logs** | Central place to view all logs in real time |

JBoss vs AWS CloudWatch — Key Difference

| **Feature** | **JBoss** | **AWS CloudWatch** |
| --- | --- | --- |
| Type | **Application Server** | **Monitoring/Logging Service** |
| Purpose | Runs Java applications and APIs | Collects logs/metrics, monitors resources |
| Logs | Generates logs (e.g., server.log) | Stores & analyzes logs (e.g., JBoss logs, EC2 metrics) |
| Use case | Host business logic | Watch CPU usage, alert on app errors, analyze performance |

So: **JBoss creates logs**, **CloudWatch stores, analyzes, and alerts** on those logs or metrics.

## **What You Can Do With AWS CloudWatch**

### ✅ ****1. Collect Logs****

* From EC2, JBoss, Apache, Docker, RDS, etc.
* Custom apps like Java, Node.js, Python

📁 Example:

* Send server.log from JBoss to CloudWatch
* Monitor logs in real time

### ✅ ****2. Monitor Metrics****

* CPU, memory, disk usage of EC2 instances
* Response time, error rate of APIs
* DB connection usage, query time (RDS)

📊 Example:

* Track how CPU usage changes during a JMeter test

### ✅ ****3. Set Alarms****

* Create thresholds and get notified when they’re crossed

🔔 Example:

* Alert when:
  + CPU > 80%
  + API response time > 5s
  + Error logs appear in JBoss logs

### ✅ ****4. Create Dashboards****

* Visualize your app/server performance

📈 Example:

* Custom dashboard showing:
  + JBoss memory usage
  + PostgreSQL query time
  + Number of errors per minute

### ✅ ****5. View and Analyze Logs****

* Search logs like grep or filter by message
* Find root cause for errors

🔍 Example:

text

CopyEdit

ERROR org.jboss.as.server - NullPointerException

### ✅ ****6. Integrate with DevOps & APM Tools****

* CloudWatch integrates with:
  + SNS (email/SMS alerts)
  + Lambda (auto-reaction to alarms)
  + Grafana (for dashboards)
  + Dynatrace, Datadog, etc.

## 📘 Summary

| **Feature** | **AWS CloudWatch** |
| --- | --- |
| Type | AWS Monitoring & Logging Service |
| Monitors | EC2, RDS, JBoss, Lambda, etc. |
| Stores | Metrics & Logs |
| Offers | Dashboards, Alarms, Real-time log search |
| Works with | Any app (including JBoss) and AWS services |

**What is EC2?**

**Amazon EC2 (Elastic Compute Cloud)** is a **virtual server** in the cloud, provided by **Amazon Web Services (AWS)**.

Think of EC2 as a **cloud-based machine (computer)** where you can:

* Install software (like JBoss, Tomcat, MySQL, PostgreSQL, etc.)
* Host websites, APIs, or backend apps
* Run performance tests or automation jobs

### ****Why is it called "Elastic"?****

Because you can:

* **Easily scale up/down**: Add more instances when load increases
* **Pay as you go**: No need to buy physical hardware
* **Choose configurations**: RAM, CPU, storage, OS, etc.

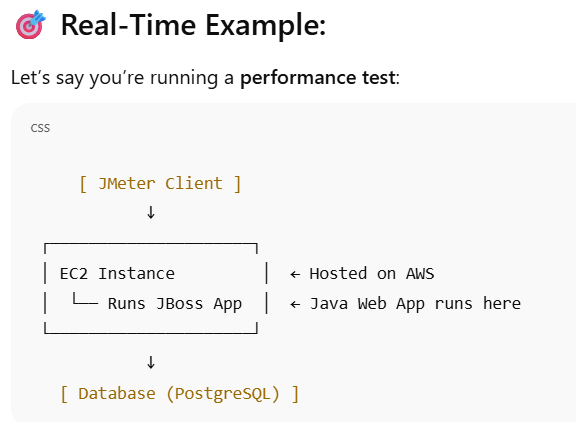
What we can do with E2C

| **Use Case** | **Example** |
| --- | --- |
| 🧪 **Host Application Servers** | Deploy **JBoss** or **Spring Boot** apps |
| 🛢️ **Host Databases** | Install **PostgreSQL** or **MySQL** manually (if not using RDS) |
| 📦 **Run Performance Tests** | Use EC2 to run **JMeter load generators** |
| 🔐 **Secure Systems** | Create **custom firewalls (security groups)** |
| 🧑‍💻 **Remote Login** | Use SSH (Linux) or RDP (Windows) to log into your instance |

### So in simple words:

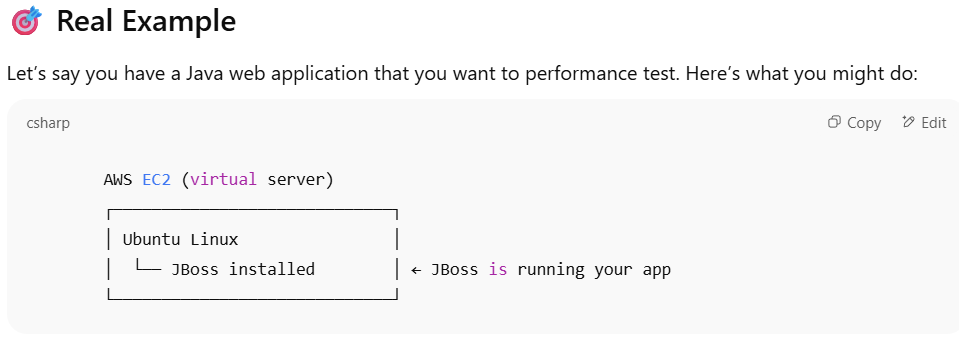
🔹 **JBoss is software.**  
🔹 **EC2 is the cloud server** where you can install and run that software.

| **Analogy** | **EC2** | **JBoss** |
| --- | --- | --- |
| 💻 Computer | Laptop (EC2) | Installed software like Eclipse (JBoss) |
| 🏠 House | Building (EC2) | Air conditioner inside (JBoss) |



Difference between JBoss and EC2

| **Feature** | **JBoss** | **EC2 (Elastic Compute Cloud)** |
| --- | --- | --- |
| 🔧 **What it is** | **Application Server** (software) | **Virtual Machine** (cloud-based hardware) |
| 🧠 **Purpose** | Runs Java applications, APIs, and web services | Hosts applications, databases, or tools like JBoss, JMeter |
| 🏗️ **Layer** | Software layer (middleware) | Infrastructure layer (compute/server) |
| 🖥️ **Installation** | You install JBoss **on** a machine (like EC2 or physical server) | EC2 **is** the machine where apps like JBoss can run |
| 🔌 **Runs on** | Physical/virtual server, Docker, EC2, etc. | AWS cloud (Linux/Windows virtual machines) |
| 📈 **Use case in Performance Testing** | Application to monitor, log, and process requests | Machine used to host load generators (JMeter) or target servers (JBoss app) |
| 🔍 **Log Source** | Generates logs like server.log, access logs | OS-level logs (syslog), app logs (like JBoss), CPU/Memory logs |
| 🌐 **Example Role** | Middleware that handles HTTP/REST requests | Cloud host that runs JBoss, MySQL, or JMeter |



* **JBoss** is not from AWS, but it's **installed on** an AWS **EC2** instance.
* EC2 gives you the **infrastructure**, and you choose what to install on it.

| **Item** | **Type** | **From** | **What it Does** |
| --- | --- | --- | --- |
| **JBoss** | App Server (Software) | Red Hat | Runs Java apps |
| **EC2** | Virtual Machine (Cloud) | AWS | Hosts software like JBoss |
| **JBoss on EC2** | You install it yourself | Combo | Typical real-world setup |

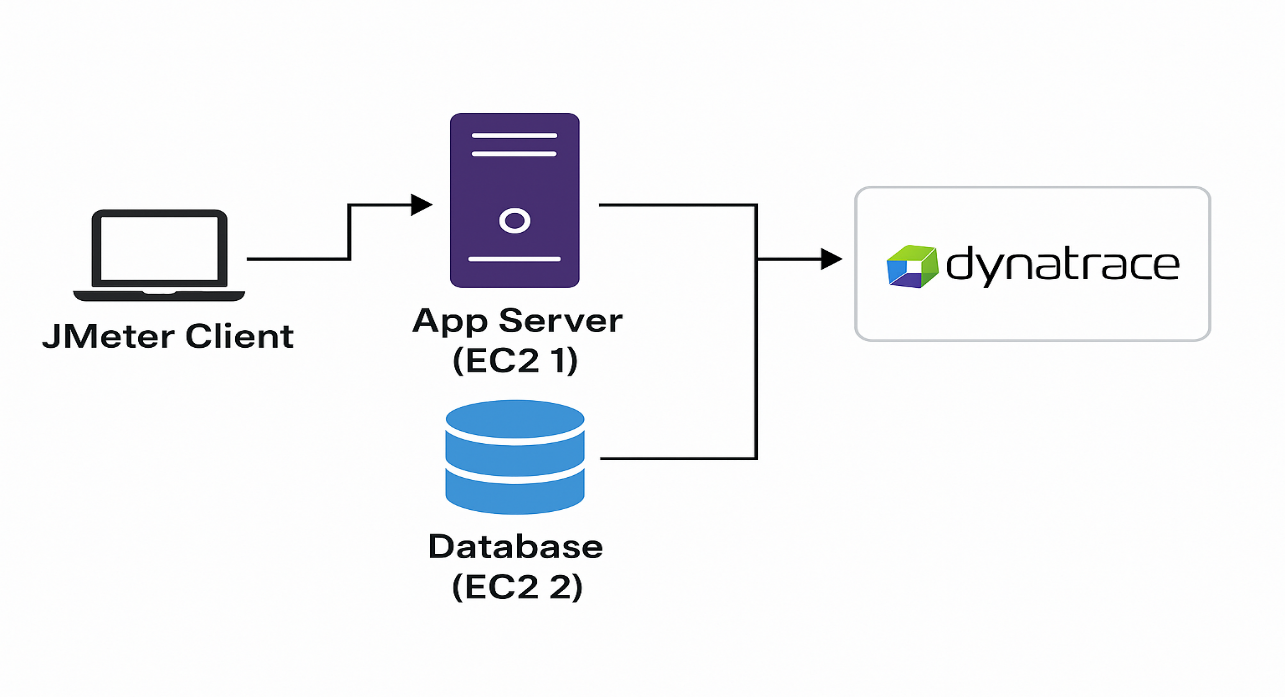
EC2 Alternatives (Virtual Machines)

| **Service** | **Purpose** | **Compared to EC2** |
| --- | --- | --- |
| **Lightsail** | Simplified EC2 for small projects | 🟢 Easier to use, but less flexible |
| **Elastic Beanstalk** | Automatically deploys and manages apps | 🟡 Built on EC2, but fully managed |
| **ECS (Elastic Container Service)** | Runs Docker containers | 🔵 Higher-level than EC2 |
| **EKS (Elastic Kubernetes Service)** | Kubernetes for containers | 🔵 Uses EC2 under the hood |
| **AWS Lambda** | Serverless compute – run code without managing servers | 🟥 NOT like EC2; no server management |

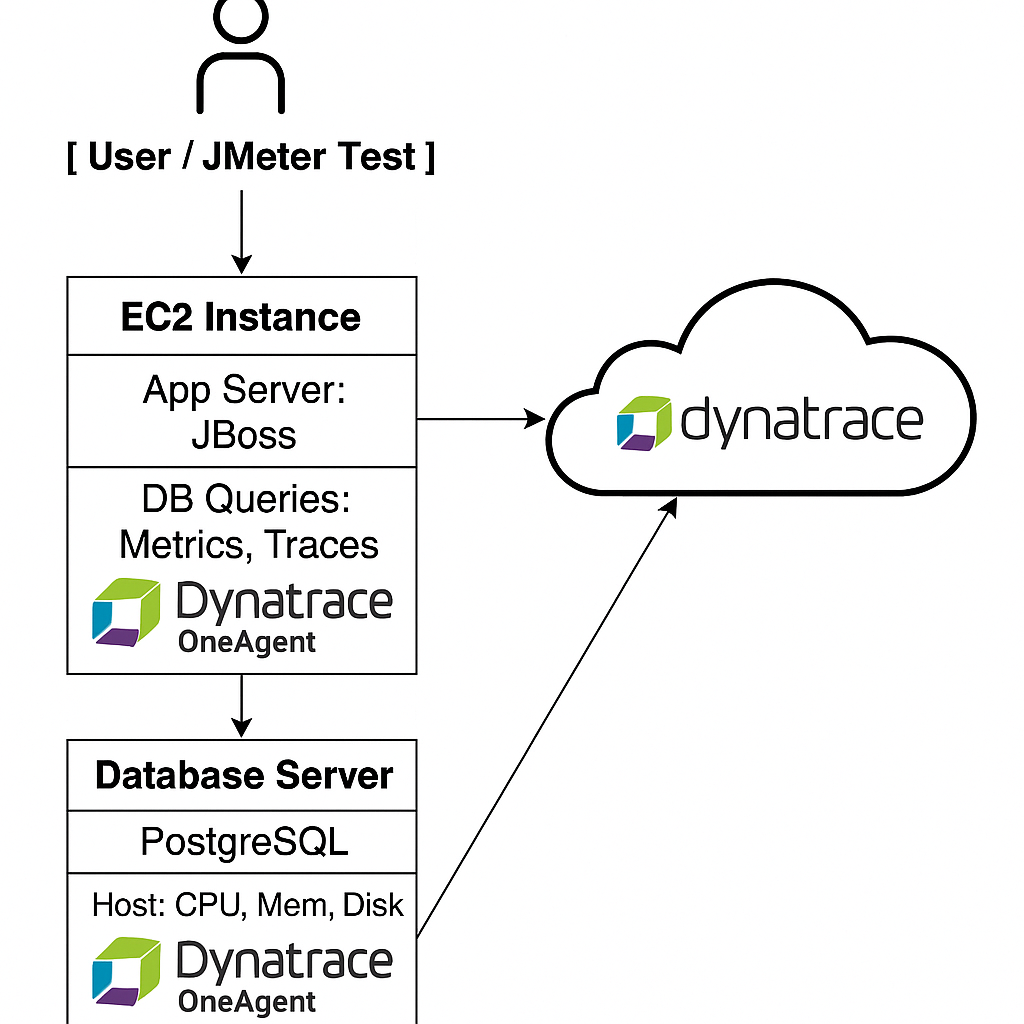
Outside AWS (Other Cloud Providers)

| **Cloud Provider** | **Service Name** | **Similar to EC2** |
| --- | --- | --- |
| ☁️ **Microsoft Azure** | **Azure Virtual Machines** | 🟢 Closest equivalent |
| ☁️ **Google Cloud Platform (GCP)** | **Google Compute Engine (GCE)** | 🟢 Very similar |
| ☁️ **IBM Cloud** | **IBM Virtual Servers** | Similar concept |
| ☁️ **Oracle Cloud** | **Oracle Compute** | Same idea |

**Diagram showing how app server and DB metrics flow into Dynatrace,**



One Agent should be installed in the EC2 VM to see the metrics in the Dynatrace,



How **JBoss logs**, **CloudWatch logs**, **EC2 instances**, and **Dynatrace integration** all work together — especially in the context of **performance testing** and **monitoring**.

| **Component** | **Role** |
| --- | --- |
| **JBoss** | App Server — runs your Java app |
| **EC2** | Virtual Machine — hosting JBoss |
| **CloudWatch** | AWS monitoring/logging tool |
| **Dynatrace** | Full-stack monitoring + observability tool |

If you're running JBoss inside an EC2 VM, you **install Dynatrace OneAgent on that EC2 instance** to collect deep performance metrics.

## How Integration Works

### ✅ 1. ****Install Dynatrace OneAgent on EC2****

* OneAgent is a small software installed on your EC2 instance.
* It **automatically detects**:
  + JBoss app server
  + Java processes
  + OS metrics (CPU, memory, disk)
  + Network activity
  + DB calls made from JBoss
  + Log files (like JBoss logs)

➡️ **This is how JBoss logs and metrics are linked to Dynatrace**.

### ✅ 2. ****Enable Log Monitoring in Dynatrace****

* Dynatrace can **read JBoss log files** directly from the EC2 instance (e.g., /opt/jboss/server.log).
* It **parses** errors, GC activity, slow queries, exceptions.
* You can also set up alerts based on log patterns (e.g., “Timeout”, “500 Error”).

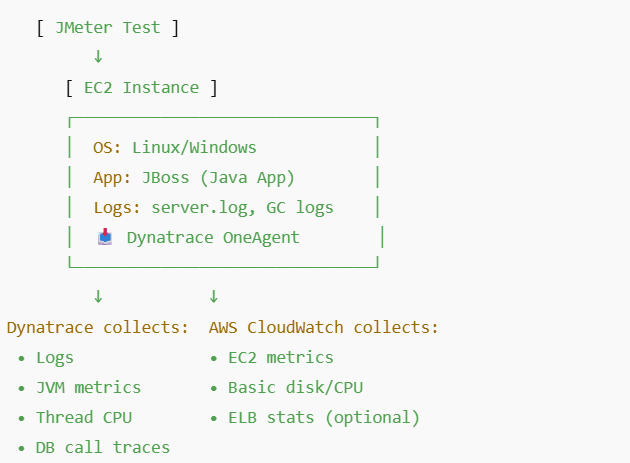
### ✅ 3. ****CloudWatch Logs Integration (Optional)****

You can also integrate **CloudWatch with Dynatrace** to pull logs/metrics if needed:

* Use **AWS integration** in Dynatrace.
* Dynatrace will access:
  + EC2 CPU/memory/network metrics (via CloudWatch)
  + CloudWatch Logs (optional)
  + Auto-scaling and ELB metrics

🔁 This is helpful for **cross-checking with native AWS data**, but **OneAgent is still the deeper option**.

How It All Connects



Why we need Dynatrace Instead of Just CloudWatch?

| **Feature** | **CloudWatch** | **Dynatrace** |
| --- | --- | --- |
| Basic EC2 metrics | ✅ | ✅ |
| App Server metrics (JBoss) | ❌ | ✅ |
| JVM Metrics | ❌ | ✅ |
| Distributed Tracing (PurePath) | ❌ | ✅ |
| Smart AI alerts | ⚠️ (manual setup) | ✅ (Davis AI) |
| Log correlation with requests | ❌ | ✅ |

* You’re right: if your app is in **EC2 + JBoss**, install **Dynatrace OneAgent** on that EC2.
* It automatically reads **JBoss logs**, **CPU**, **memory**, **thread dumps**, **DB timings**, and more.
* Dynatrace can also integrate with **AWS CloudWatch** for extra insights.

## Scenario 1: **DB calls made** from your application (e.g., JBoss app on EC2)

### ➤ ✅ You only need ****OneAgent on the EC2 server (app server)****

* OneAgent captures:
  + SQL query timings
  + Query count
  + Execution time (e.g., 8.5 seconds for one query)
  + Database name, host, port
  + Slowest queries
  + Failed queries
* It gets this info **from the JBoss process itself**, by auto-instrumenting the JDBC driver (or other DB connector).

Example in Dynatrace UI:

Service → Database tab → PostgreSQL (or MySQL, etc.)

→ Shows Top Queries, Response Time, Failed Calls

## Scenario 2: **You want full visibility into the Database Server OS (CPU, disk, memory)**

### ➤ ✅ You also need to install ****OneAgent on the database server****

This gives you:

* DB server CPU, RAM, Disk I/O
* Process resource usage (e.g., postgres, mysqld)
* DB logs (if configured)
* Network usage
* Deep health monitoring of the DB host

| **What You Want** | **Where You Need OneAgent** | **Required?** |
| --- | --- | --- |
| DB query timings, SQL names, slow queries | App Server (e.g., EC2 with JBoss) | ✅ Yes |
| DB host resource metrics (CPU, memory, I/O) | DB Server (EC2, RDS, etc.) | ✅ Yes (if you want host-level metrics) |
| DB logs (Postgres/MySQL logs) | DB Server | ✅ Optional |
| Cloud DB like **Amazon RDS** | Use Dynatrace's **AWS integration**, no agent install on RDS | ✅ Optional |

## Real-World Setup Example

### Let’s say:

* App is running on **JBoss in EC2-1**
* DB (PostgreSQL) is running on **EC2-2**

You would do:

1. ✅ Install OneAgent on EC2-1 → Get app metrics + DB query timings
2. ✅ Install OneAgent on EC2-2 → Get PostgreSQL host CPU, memory, and DB process health

💡 Bonus: If DB is on **AWS RDS**, no OneAgent can be installed there — but you can integrate **CloudWatch to Dynatrace** to get those metrics.

Matrices (infrastructure monitor) + Trace ()

Linux, Windows, Tomcat Code level(APM),Network Level (Syn),User location or session