# **01.INTRODUCTION**

## **1.INTRODUCTION TO MONITORING IN SDLC**

### 🔍 What is Monitoring in SDLC?

Before diving into **Datadog**, we must understand what **monitoring** means in the **Software Development Lifecycle (SDLC)** and **why it's crucial**.

* **Monitoring = Keeping a Watch**
  + Just like a security guard watches a building, monitoring in SDLC means **watching over applications and infrastructure**.
  + You observe **critical and non-critical components** to ensure everything is working smoothly.

### 🧠 Why is Monitoring Important?

* You need to **continuously track system performance** to make sure it behaves as expected.
* This can apply to anything:
  + Web apps
  + Desktop software
  + Cloud infrastructure
  + APIs
  + Microservices

### 📊 What Exactly is Measured?

Think of performance monitoring as **checking health indicators**, like a doctor would for a patient. Here are some common ones:

| **Metric** | **Meaning** |
| --- | --- |
| **Latency** | Time taken to respond |
| **Uptime** | How often a system is available |
| **Error rate** | How often things break |
| **Throughput** | Number of requests served |
| **CPU Usage** | How busy the server is |
| **Request rate** | How many requests are received |
| **Data correctness** | Are responses accurate? |

These KPIs vary depending on the application and business goals.

### 🚨 Monitoring with Alerts

* Monitoring is not just observing, it includes **alerting**.
* If something breaks or exceeds a limit (e.g. CPU > 90%, latency > 2s), the system must send **alerts** to developers/DevOps teams.

This allows **fast response** to fix issues before users are impacted.

### ⚡ The Benefits of Monitoring

1. **Proactive Fixes**: Identify and resolve problems before users notice.
2. **Predictive Insights**: Use **machine learning** to forecast failures.
3. **Less Downtime** = Better User Experience = More Revenue.
4. **Improved SLAs**: Meet customer expectations and legal obligations.
5. **Boost Productivity**: Teams can focus on building, not firefighting.

### 🌍 Why Monitoring is More Critical Today?

Modern apps are:

* **Distributed** (e.g., microservices, containers, serverless)
* **Complex** (many layers, tools, environments)
* **Rapidly changing** (CI/CD, frequent deployments)
* **User-facing** (high availability needed 24/7)

This complexity demands **continuous, deep, unified monitoring**.

### 🧩 Example: Then vs Now

| **Then** | **Now** |
| --- | --- |
| One database + one frontend | Multiple APIs, services, UIs |
| Single server | Cloud + containers (Docker, Lambda) |
| Manual testing | Automated pipelines |
| Local logs | Centralized observability tools |

You can’t manage today’s systems **without robust monitoring** tools.

### 💡 Role of Tools like Datadog

* Modern challenges need **modern tools**.
* That’s where **Datadog** comes in – a powerful performance monitoring and observability platform.
* It collects metrics, traces, logs, and helps you **visualize, alert, and respond** across your entire stack.

## **2.WHAT IS DATADOG MONITORING TOOL?**

### 🚀 What is Datadog?

**Datadog** is a **cloud-based monitoring and security platform** built specifically for **cloud-native applications**. It provides full **observability** across your entire tech stack: infrastructure, applications, logs, security, and even user interactions.

#### 💡 Key Definition (from Datadog Docs):

“Datadog is the essential monitoring and security platform for cloud applications.”

### 🧱 Core Purpose of Datadog

Datadog is designed to **collect, analyze, and visualize**:

* **Metrics** (numerical performance data like CPU usage, memory, etc.)
* **Logs** (raw events and error messages)
* **Traces** (APM: Application Performance Monitoring)

It helps you **detect, troubleshoot, and resolve issues** in real time across your infrastructure and services.

### ☁️ SaaS-Based Platform

Datadog is a **SaaS (Software-as-a-Service)** tool:

* **No on-premise setup** required
* You only install a lightweight **Datadog Agent** on your servers/apps
* This agent sends the data to the **Datadog cloud backend**

#### 📌 Why it matters:

You don’t need to maintain complex infrastructure for observability—Datadog handles it.

### 🧰 What Datadog Offers

It’s not just about monitoring one thing. Datadog gives you a **complete end-to-end monitoring solution**.

#### 🧩 Features & Monitoring Types:

| **Monitoring Type** | **What It Watches** |
| --- | --- |
| **Infrastructure Monitoring** | Servers, VMs, networks |
| **Database Monitoring** | Query times, locks, slowdowns |
| **Application Performance Monitoring (APM)** | Traces for APIs, service calls |
| **Log Management** | Centralized logging |
| **Cloud Monitoring** | AWS, Azure, GCP usage and health |
| **Container Monitoring** | Docker, Kubernetes |
| **Real User Monitoring (RUM)** | Frontend performance as seen by real users |
| **Synthetic Monitoring** | Simulated user flows to test performance and availability |
| **Security Monitoring** | Threat detection, configuration checks |

All of these are integrated into **one unified platform**.

### 🧩 Integrations Galore: 500+ and Counting!

Datadog shines with its **massive integration ecosystem**:

* Supports **900+ built-in integrations** including:
  + OS: Windows, Linux, Mac
  + Cloud: AWS, Azure, GCP
  + Containers: Docker, Kubernetes, Helm
  + Messaging: Kafka, ActiveMQ, HiveMQ
  + Security Tools: Okta, HashiCorp Vault
* Comes with **predefined dashboard templates** for these integrations

⚡ You get **quick visibility** without writing a single line of config from scratch.

### 🏗️ Architecture (Simplified)

* Your application → **Datadog Agent** → **Datadog Cloud Backend**
* From the cloud, you access:
  + Dashboards
  + Alerts
  + Metrics
  + Traces
  + Logs
  + Security signals

Details on this architecture are covered later in the course.

### 🌍 Quick Facts & Background

* **Founded:** 2010
* **Founders:** Olivier Pomel (current CEO)
* **Open Source:** Datadog Agent code is open-source (available on GitHub)
* **Customers:** Thousands of companies globally, including:
  + Samsung
  + Shell
  + Sony
  + HashiCorp
  + Nikon
  + Deloitte

### 👥 Community & Growth

* **Active global developer and user community**
* **Continuous updates and new integrations**
* Lots of learning content, hands-on labs, quizzes, etc.

## **3.KEY FUNCTIONALITIES OF MONITORING TOOLS**

### 🔧 Core Functionalities of a Monitoring Tool

Modern monitoring tools—especially those based on **time-series data** (like Datadog, Prometheus, etc.)—are expected to handle a standard set of tasks.

Let’s break them down:

#### 1️⃣ **Data Collection**

**What?** Gathering time-stamped data (aka time-series) from various sources.

| **Examples of Data Sources** |
| --- |
| Servers (CPU, RAM, Disk) |
| Applications (e.g., Java App metrics) |
| Logs (error logs, access logs) |
| Sensors, network devices |

* It should support **multiple data collection protocols** (HTTP, SNMP, agents, APIs).
* **Efficient ingestion** is key to avoid data loss and maintain real-time performance.

#### 2️⃣ **Data Storage**

**What?** Storing large volumes of time-series data reliably.

* Must be **scalable and fault-tolerant**.
* Supports:
  + High write rates (many metrics every second)
  + Long-term retention (historical analysis)
* In tools like **Datadog**, the storage is **cloud-based and highly available**.

#### 3️⃣ **Data Querying and Analysis**

**What?** Retrieving specific data points for analysis.

* Should support:
  + **Time-based filtering** (e.g., last 1 hour, past 7 days)
  + **Aggregation** (average, sum, max, min)
  + **Custom computations** (e.g., error rate = 5xx / total requests)

This allows deep insights and diagnostics for DevOps and engineering teams.

#### 4️⃣ **Visualization & Alerting**

**What?** Making the data visible and actionable.

##### 🖥️ Dashboards:

* Real-time data visualization using:
  + Line charts
  + Heatmaps
  + Gauges, bar charts, pie charts
* Helps **spot trends, spikes, anomalies**.

##### 🔔 Alerting:

* Set **thresholds** (e.g., CPU > 85%)
* Trigger **notifications** (Slack, Email, SMS, PagerDuty)
* Allows **proactive problem detection**.

#### 5️⃣ **Scalability & Performance**

**What?** Handling high data volume without slowing down.

* Should scale to handle:
  + More hosts
  + More users
  + More frequent data (e.g., every second)
* Performance should stay **stable as the environment grows**.

Datadog is designed to scale to **cloud-native** and **enterprise-level workloads**.

#### 6️⃣ **Integration & Extensibility**

**What?** Connecting with other systems & tools.

* A good monitoring tool should integrate with:
  + Cloud providers (AWS, Azure, GCP)
  + CI/CD pipelines (Jenkins, GitHub Actions)
  + Containers (Docker, Kubernetes)
  + Message queues, databases, etc.

🧩 Example: Datadog has **900+ built-in integrations**.

#### 7️⃣ **Security & Access Control**

**What?** Ensuring data privacy and managing user access.

* Features include:
  + **Authentication** (who can log in)
  + **Authorization** (who can see what)
  + **Role-based access control (RBAC)**

🛡️ Sensitive time-series data (e.g., system logs or DB metrics) must be protected.

### 📌 Summary: The 7 Essentials of a Monitoring Tool

| **#** | **Feature** | **Purpose** |
| --- | --- | --- |
| 1 | Data Collection | Collect metrics/logs from systems |
| 2 | Data Storage | Store large volumes of time-series data |
| 3 | Querying & Analysis | Retrieve and compute insights |
| 4 | Visualization & Alerting | View trends, get notified of issues |
| 5 | Scalability & Performance | Handle growth in data and users |
| 6 | Integration | Work with other tools and platforms |
| 7 | Security & Access Control | Protect data and manage user roles |

### 🔁 Other Tools (Optional Exploration)

While this course focuses on **Datadog**, there are other monitoring tools that follow the same core principles:

* **Prometheus**
* **InfluxDB**
* **Nagios**
* **Sensu**
* **Dynatrace**

Each has its **own strengths**, but the **core job is always: Monitoring time-series data effectively**.

## **4.ANNOUNCEMENT**

## **5.MONITORING TERMINOLOGIES**

### 🧠 Monitoring Terminologies – Explained Clearly

#### 🖥️ 1. **Host**

A **host** is any **device or entity being monitored**.

##### ✅ Can be:

* Physical server
* Virtual machine (VM)
* Docker container
* Desktop computer
* IoT device

**In simple terms:**  
If it has an **IP address** and provides computing/network services, it can be considered a host.

#### 📊 2. **Metrics**

Metrics are **time-bound numerical values** that describe the state of a system.

##### 🔁 Captured at regular intervals (e.g., every second, every minute)

##### 📌 Examples:

* Number of logged-in users
* CPU usage %
* Application status (up/down)
* Response time of an API

##### 🧭 Categories of Metrics:

###### 🔹 a. **Work Metrics**

Focus on how well the system is performing from a user perspective.

| **Type** | **Meaning** |
| --- | --- |
| Throughput | Requests/queries per second |
| Success Rate | % of successful operations |
| Error Rate | % of failed operations |
| Performance | Response time, usually in percentiles (e.g., P95 latency) |

**Use:** Quick health check of the system.

###### 🔹 b. **Resource Metrics**

Focus on the **internal state** of system components like CPU, memory, and disk.

| **Type** | **Meaning** |
| --- | --- |
| Utilization | How busy a resource is (% CPU used, % memory used) |
| Saturation | How much workload is waiting (e.g., queue length) |
| Availability | Uptime or response % |
| Errors | Hardware/sof tware faults |

**Use:** Deep diagnosis when something goes wrong.

###### 🔹 c. Other Types:

| **Metric Type** | **Description** |
| --- | --- |
| Network Metrics | Bandwidth usage, packet loss, latency |
| Server Pool Metrics | Aggregated metrics across a group of servers |
| External Dependency | Metrics from 3rd party APIs or services |

#### 📅 3. **Events**

**Events** are **irregular activities or changes** that happen in the system.

##### 🔁 Not time-based like metrics; they occur **as needed**.

##### 📌 Examples:

* Restart of a service
* Deployment of new code
* A new host added or removed
* Suspicious login or security anomaly

Events are often tied to **alerts**—they need immediate user attention.

#### ⚙️ 4. **Agent**

A **Datadog Agent** is a small software service installed on a host.

##### 🛠️ Its job:

* Collect metrics and events
* Send them securely to the **Datadog cloud**

**Think of it as:**  
The bridge between your infrastructure and Datadog’s monitoring engine.

#### 🚨 5. **Alert**

An **alert** is a **triggered notification** that something has gone wrong.

##### 🔁 Created when a **monitoring rule or threshold is breached**.

##### Example:

* CPU usage > 90% for 5 minutes → Send Slack/Email alert

In Datadog, you can define these rules using **monitors**.

### ✅ Summary Table: Monitoring Terminologies

| **Term** |  | **What it Means** | **Example** |
| --- | --- | --- | --- |
| Host |  | Entity being monitored | A Linux server, Docker container |
| Metric |  | Time-bound value | CPU usage %, API response time |
| Work Metric |  | Output-level health data | Success rate, error rate, response time |
| Resource Metric |  | Physical resource usage | Memory %, disk I/O, CPU saturation |
| Event |  | Irregular system activity | Process restart, service deployment |
| Agent |  | Data collector on host | Datadog agent sending data to cloud |
| Alert |  | Triggered notification | Alert when memory > 90% for 10 mins |

## **6.ARCHITECTURE OF DATADOG**

### 🏗️ Datadog Architecture — Crystal Clear Explanation

#### 🔄 **High-Level View**

Datadog’s architecture can be broken into **two main parts**:

| **Side** | **Description** |
| --- | --- |
| **Client Side (Host Side)** | Where the **Datadog Agent** runs and collects data from your system |
| **Cloud Side (Backend)** | Where the collected data is processed, visualized, and alerted on through dashboards and monitors |

Think of Datadog as a **cloud-based monitoring brain**, and the agent is like its **ears and eyes** on your systems.

#### 🧩 Key Component: **Datadog Agent**

The **Datadog Agent** is a lightweight software installed on each **host** you want to monitor.

It has two primary parts:

| **Component** | **Job Description** |
| --- | --- |
| **Collector** | Gathers all metrics/events/logs from the host every **15 seconds** |
| **Forwarder** | Sends the collected data to Datadog's cloud over **HTTPS (port 443)** |

🧠 It also uses a **memory buffer** to optimize data transmission (sends only when buffer fills or reaches a limit).

#### 📦 Bonus Add-On: **DogStatsD**

**DogStatsD** is an optional **custom metrics aggregator** bundled with the agent.

* Receives **custom application metrics** (like from your Java app) via **UDP (port 8125)**
* Periodically aggregates and forwards them to the Datadog backend
* Uses **StatsD protocol + Datadog extensions**

✅ **Non-blocking** – Since it uses UDP, your app won't hang if DogStatsD is temporarily down.

Example:  
Java app → sends metrics via UDP → DogStatsD → aggregates → forwards to backend

#### ⚙️ Optional Agent Add-ons

There are 3 optional agent processes (enabled via config):

| **Process** | **Function** |
| --- | --- |
| **APM Agent** | Collects **traces** for Application Performance Monitoring |
| **Process Agent** | Collects **live process info** (e.g., running containers) |
| **Agent UI** | Runs a **local web UI** for monitoring the agent itself |

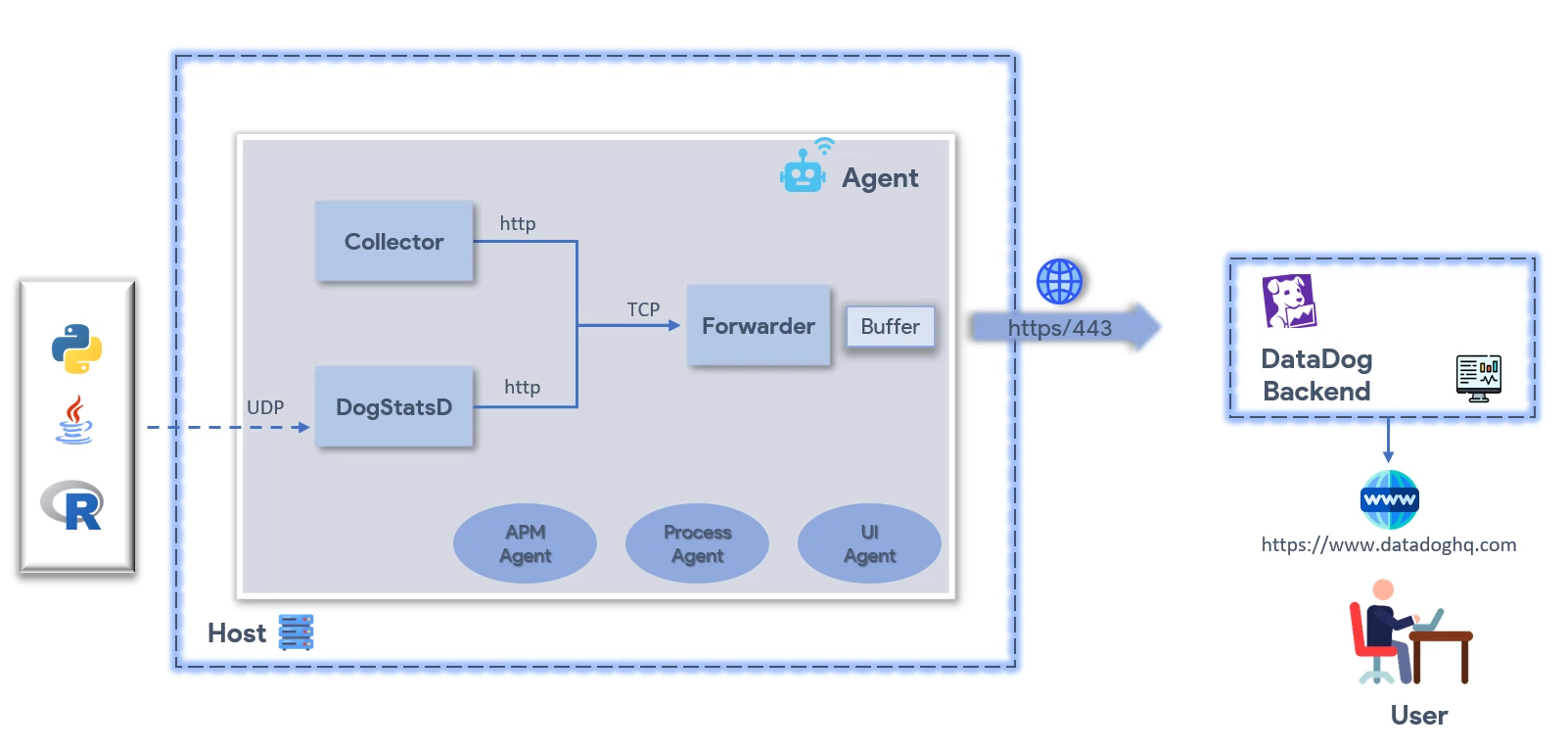
📍 **Agent UI** is often skipped — most users prefer the full dashboard via the **Datadog web portal**.

* Default UI Port: **5002** (enabled by default on **Windows/macOS**)

#### 🔁 **How Data Flows – Step by Step**

1. 🧱 **Install Datadog Agent** on your host (Linux, Windows, container, etc.)
2. 🧠 **Collector** fetches:
   * Metrics (CPU, memory, etc.)
   * Logs
   * Events
3. 🚚 **Forwarder** buffers and sends this data to **Datadog backend via HTTPS**
4. 🌐 **Backend processes** the data in the cloud
5. 📊 Users log into [Datadog website](https://app.datadoghq.com/):
   * View dashboards
   * Set alerts
   * Analyze trends
   * Get real-time monitoring insights

#### 🧭 Diagram



#### 🚀 Quick Highlights

| **Feature** | **Description** |
| --- | --- |
| Agent | Local data collector on your host |
| Collector | Gathers standard host metrics every 15s |
| Forwarder | Sends data via HTTPS with memory buffer |
| DogStatsD | Sends custom metrics from your apps via UDP |
| APM, Process Agent | Add-ons for trace collection and live process monitoring |
| Agent UI | Optional, local browser-based UI for the agent |
| Datadog Cloud | Central brain where users analyze, visualize, and set alerts |

# **02.PRICING AND SETUP**

## **7.DATADOG PRICING**

### 🌐 **Datadog Pricing – Simplified Overview**

Datadog is a **SaaS (Software-as-a-Service)** based monitoring platform. This means:

* No complicated setup.
* You just sign up on [https://www.datadoghq.com](https://www.datadoghq.com/) and start using it.

### 🎁 **Free Trial**

* **14-day full access trial** to all Datadog features.
* Great for **learning, exploring**, or **evaluating** before making a purchase.

### 💡 **Modular Pricing (Pay-Only-For-What-You-Use)**

Instead of bundling everything into a few expensive plans, Datadog offers **individual pricing per service**, like:

* **Infrastructure Monitoring**
* **Log Management**
* **Database Monitoring**
* **Synthetic Monitoring**, etc.

This is useful because:

* You only pay for the **specific services** your project actually needs.
* It avoids unnecessary costs.

### 📊 **Example: Infrastructure Monitoring Plans**

| **Plan Type** | **Cost** | **Features** |
| --- | --- | --- |
| **Free Plan** | $0 | - Up to **5 hosts**- 500+ integrations- 1-day data retention- No alerts, no support |
| **Pro Plan** | Paid | - **15 months** data retention- **Alerting**- 10 containers/host- 100 custom metrics/host |
| **Enterprise** | Higher Paid | - Includes everything in Pro +- **Correlation**, **Anomaly Detection**, **Forecast Monitoring** |

📌 **Note**: Free plan is perfect for **learning**, not for **production** usage.

### 🗂️ **Other Services (Examples)**

| **Service** | **Plan Options** | **Starting Price** |
| --- | --- | --- |
| **Log Management** | Two tiers | Varies – check site |
| **Database Monitoring** | Single plan | Starts at **$70/month** |
| **Synthetic Monitoring** | Two plans: API & Browser | - API: **$5/test/month**- Browser: **$12/test/month** |

📝 **Prices are subject to change**, always check the [pricing page](https://www.datadoghq.com/pricing) for updates.

### 🤔 **Confused by Terms or Plans?**

If you're new to monitoring, some terms may not make sense (e.g., "web recorder", "AI-driven test").

* Go through **FAQs** on the pricing page – they’re helpful.
* As you learn more about Datadog, you’ll understand these features better.
* Still stuck? You can always **contact Datadog support** for help in choosing the right plan.

### ✅ Summary

* Start with the **14-day free trial**.
* Learn and explore without paying.
* Later, **choose individual services** based on what your project actually needs.
* Use free tier for practice; opt for Pro/Enterprise for real environments.

## **8.DATADOG SETUP AND ACCOUNT CREATION**

### ✅ **Step-by-Step Guide: Datadog Setup & Agent Installation**

#### 🚀 **1. Start Free Trial (No Credit Card Needed)**

* Go to the official website: [https://www.datadoghq.com](https://www.datadoghq.com/)
* Click **“Free Trial”** or **“Get Started Free”**.
* ✅ **Good News**: **No credit card** required for the 14-day trial.
* You can monitor **unlimited servers** during this period.

#### 🌍 **2. Choose a Region**

You must pick a **region** for your account. Each region is **completely isolated**, and:

* Your **data stays** within the selected region.
* You **cannot** log in to other region sites with the same credentials.

| **Region Name** | **Purpose** | **URL Example** |
| --- | --- | --- |
| **US1** | Default (United States) | https://app.datadoghq.com |
| **US3** | Another US-based region | https://us3.datadoghq.com |
| **EU1** | Europe | https://app.datadoghq.eu |
| **US5** | Additional US-based | (Region-specific) |
| **US Fed** | For US government agencies | (Gov-compliant zone) |

👉 **Choose a nearby region** for better performance or to meet **compliance needs**.

#### 📧 **3. Fill Basic Details**

* Fill in:
  + Email address (required)
  + Phone number (optional)
  + Region (as above)
* Click **Sign Up**

#### 🛠️ **4. (Optional) Tool Stack Info**

* You’ll be asked about:
  + Tools you use (e.g., AWS, Docker, Java, etc.)
  + Number of services you manage
* This step is **optional** – you can **skip it**.

#### 🖥️ **5. Install the Datadog Agent**

The **Datadog Agent** is a lightweight process that:

* Runs on your host machine (server, VM, etc.)
* Collects metrics/events/logs
* Sends data to your Datadog account

##### 🧩 Supported OS:

* Windows
* Linux (Ubuntu, RHEL, CentOS, etc.)
* macOS
* Cloud platforms (AWS, Azure, GCP)
* Docker / Kubernetes

#### 💻 **6. Install Agent on Windows (Example)**

1. Select **Windows** as the platform.
2. Choose UI-based or CLI-based install.
3. Download the **installer** (specific to OS).
4. Run it as **Administrator**.
5. Paste the **API Key** (unique per Datadog account).
6. Select your **region** (must match the one chosen during registration).
7. Click **Install**.
8. ✅ Once it finishes, your agent is successfully installed!

#### 🏁 Done!

You’ve now:

* Created a Datadog account
* Chosen the right region
* Installed the Datadog Agent

Your system is now **ready to send data to Datadog**.

# **03.DATADOG AGENT**

## **9.DATADOG AGENT MANAGER**

### 🔹 What is Datadog Agent Manager?

The **Datadog Agent Manager** is a **GUI (Graphical User Interface)** tool that helps you manage the Datadog Agent installed on your system. It's available on **Windows 64-bit** and **macOS**, but **not available on Ubuntu/Linux** (Linux only supports CLI management).

### 🛠️ After Installing the Datadog Agent

Once you install the agent:

* It **starts automatically** and begins **collecting system metrics**.
* To **verify** the agent is running:
  + Go to **Windows Services Panel** and check for the service named Datadog Agent.
  + Alternatively, open **Task Manager** → you’ll find:
    - Datadog Metrics Agent
    - Datadog Processes Agent
    - Datadog Traces Agent

You can also **stop** the agent temporarily by **killing the service** from the Services panel or Task Manager (no need to uninstall).

### 📋 Opening the Datadog Agent Manager (GUI)

To launch the GUI:

* **Search** for "Datadog Agent Manager" from the Start Menu.
* **Run it as Administrator**.

Once opened, you’ll see:

* ✅ **“Connected to agent”** → This confirms the agent is running.
* 🔢 Agent version (e.g., 7.36.1)
* 🖥️ Hostname → Your computer's name (you can verify it by typing hostname in CMD).
* 📄 Agent Info → Displays:
  + Start timestamp
  + File paths:
    - Configuration file path
    - Confd file path (for integrations)
    - Logs file path

### 🌐 Agent Manager GUI Prerequisites & Limitations

1. **Only supported on Windows 64-bit and macOS**
   * ❌ **Not supported** on Windows 32-bit
2. **Cookies must be enabled** in your browser
   * The UI stores a **token** in your browser for secure communication
3. **Access is limited to localhost (127.0.0.1)**
   * GUI can **only be accessed from the same machine** the agent is installed on
   * ❌ You **cannot access it remotely** (like from your host if agent runs in a VM or Docker)

### 🗂️ File Structure Overview (Preview)

Before diving deeper into the GUI options, the speaker pauses to walk through the file system where Datadog Agent is installed. This usually includes:

* Configuration files
* Log files
* Integration config directories

This helps understand where changes are made manually and how to debug agent issues.

## **10.DATADOG’s AGENT DIRECTORY TOUR**

When Datadog Agent is installed, it creates a directory that contains **important config files, logs, and integrations**. Here’s a detailed walkthrough:

### 📍 **1. Directory Location**

* Path: C:\ProgramData\Datadog
* Note: ProgramData is a **hidden folder**, so:
  + Go to File Explorer → **View** → **Enable "Hidden items"** to see it.

### 📄 **2. Main Configuration Files**

#### ✅ datadog.yaml

* **Main config file** used by Datadog Agent.
* It allows you to control:
  + Agent behavior
  + Logging settings
  + Proxy configs
  + API key and Site
  + Many other options

#### 📎 datadog.yaml.example

* A **reference template**, not used by the agent.
* All settings here are **commented out**.
* Useful for learning what configs are available.

### 🔐 File Permissions

* The actual datadog.yaml file is **locked down**:
  + Only accessible to Datadog Agent and Administrators.
* To edit:
  + Right-click → Properties → Security → Advanced
  + Add your user → Grant **Full Control**

🔴 But this is **not recommended**.  
✅ Preferred way: use the **Datadog Agent Manager (GUI)**, which already has access and handles changes safely.

### ⚙️ Key Parameters in datadog.yaml

#### 🔑 api\_key

* Required: YES
* This is how Datadog knows **which account** your agent belongs to.

#### 🌍 site

* Required: YES
* Specifies the **Datadog region** (like datadoghq.com, us3.datadoghq.com, datadoghq.eu, etc.)
* ❗ Important: Must match the **region your Datadog account** was created in.  
  Otherwise, the agent can't send data.

#### 🌐 proxy

* Optional
* Use if your system accesses the internet via a proxy.

💡 There are **hundreds of optional parameters**, each well-described inline or via URLs in comments.  
But no central place lists them all—you'll need to explore them **as needed**.

### 🔁 Applying Changes

* Any time you change datadog.yaml, you must:
  + **Restart the Agent** for changes to take effect.
  + UI will often show a **Restart button** for this.

### 📁 **3. Subdirectories Overview**

#### 📂 conf.d

* Contains **integration configuration files**.
* Each integration (like Airflow, MySQL, NGINX, etc.) has its own folder.
* Inside each: a \*.yaml.example file.
  + To use it, remove .example and configure it.
* Example: For Airflow, set connection timeouts, proxy, etc., here.

#### 📂 checks.d

* Place for **custom checks** you build.
* Used for extending Datadog with your own logic.

#### 📂 logs

* Stores **log files for different agents**:
  + agent.log → Core agent activity (most important)
  + process-agent.log → Process agent logs
  + trace-agent.log → Trace agent logs

📝 These logs help in **troubleshooting issues**, such as agent not starting, integrations not working, etc.

### ✅ Summary

| **Folder/File** | **Purpose** |
| --- | --- |
| datadog.yaml | Main configuration file (API key, site, proxy, etc.) |
| datadog.yaml.example | Sample config with all settings commented out |
| conf.d/ | Per-integration configuration files |
| checks.d/ | Custom checks built by user |
| logs/ | All log files generated by various Datadog agents |

## **11.DATADOG AGENT MANAGER CONTINUED**

The Agent Manager GUI is a diagnostic and configuration tool used **mainly for troubleshooting** the agent. Let’s walk through each tab/section.

### 1️⃣ **Agent Info**

* Displays general agent details:
  + Agent **version**
  + **Hostname**
  + **Start time**
  + File paths for config/logs
* Already covered in the previous part.

### 2️⃣ **System Info**

* Shows:
  + System **timestamp**
  + **NTP offset** (difference between system clock and time servers)
  + Detected versions of **Go** and **Python**, which are used internally by Datadog Agent

### 3️⃣ **Host Info**

* Details about the **host machine**:
  + OS type (Windows/Linux/macOS)
  + Platform family and version
  + Host uptime (how long it’s been running)
  + Hostname and detected **host tags**
  + List of **enabled integrations** (like Java checks, if Java is installed)

### 4️⃣ **Forwarder**

* Shows status of the **forwarder component**—this is responsible for:
  + Sending collected data to Datadog backend
* You can see:
  + **API key status**: valid/invalid/couldn’t check
  + **Endpoints**: actual URLs the agent sends data to (e.g., https://api.us1.datadoghq.com if you're on US1 site)

### 5️⃣ **DogStatsD**

* Displays internal stats of DogStatsD (used to collect custom metrics).
* Not very important for basic setup or general monitoring.

### 6️⃣ **Collector**

* Key tab that shows data collected from checks:
  + Example: CPU check ran **1731** times and collected **8 metrics** each time → total **13,848** datapoints
* You can check:
  + Which integrations (like memory, disk, network) are actively running
  + Their **statuses** (OK, Warning, Critical, etc.)

### 7️⃣ **Log**

* Displays the content of the agent’s main **log file** in real time.
* Useful to:
  + Diagnose agent startup errors
  + Verify if metrics are being collected and sent
  + Trace issues with integrations

### 8️⃣ **Settings**

* Allows you to **edit the agent configuration** (datadog.yaml) **directly from the GUI**.
* This is the preferred method on Windows.
* Don’t forget:
  + Save your changes
  + **Restart** the agent (button will appear)

### 9️⃣ **Checks**

#### 🔹 My Checks

* Lists all **enabled integrations** (checks) currently running on the agent.
* Example: Docker, NGINX, Windows services, etc.
* Some are pre-enabled by default.

#### 🔹 Add a Check

* Lets you **enable new checks**:
  + Pulls from integration folders inside conf.d
  + If a config file (like airflow.d/airflow.yaml.example) exists, uncomment its contents and rename it to enable
* These are treated as **custom checks**

#### 🔹 Check Summary

* Overview of all running checks:
  + Number of instances
  + Current status (OK, Warning, etc.)
* Helps you quickly verify if any check is misbehaving

### 🔟 **Flare**

* Diagnostic tool for **Datadog Support**:
  + Enter your **support ticket number** and **email**
  + Click **Submit**
  + Sends your config files, logs, and other diagnostic data directly to Datadog support

### 📌 Important Notes

* The Agent Manager is used **mostly for troubleshooting**
* In day-to-day monitoring work, you don’t need to open this unless:
  + Agent is not sending metrics
  + Host is missing from Datadog UI
  + API key/site mismatch issues
* Your actual focus as a **Monitoring Engineer** lies in:
  + Creating dashboards
  + Setting up monitors
  + Analyzing data on the Datadog website (cloud view)

# **04.INFRASTRUCTURE MONITORING – HOST**

## **12.HOST MAP**

### 🚀 Getting Started with the Host Map

1. **Login to Datadog**
   * Visit the **Datadog website**: https://app.datadoghq.<region>.com  
     (Replace <region> with the region you selected while registering, like us, eu, etc.)
   * Enter your **credentials** and login.
   * Once you're in, if the **Datadog Agent** is properly installed and running on your machine, it will begin sending **metrics** to the Datadog backend.
2. **Troubleshooting if Data Doesn't Show Up**
   * Wait a few minutes (sometimes up to **5–10 minutes**) for the data to appear.
   * Try refreshing the page.
   * If it still doesn’t show:
     + **Restart the agent**
     + **Verify the API Key** is correct.
     + **Ensure correct URL** is used for login.
     + **Check agent status** using the Agent Manager.

### 🗺️ Host Map Overview

The **Host Map** is a visual interface to monitor **all your hosts on one screen**.

* **Each host is represented by a rhombus (diamond) shape.**
  + If you have only one host → you'll see just **one rhombus**.
  + More hosts → more rhombi on the screen.
* **Color Coding (by default: CPU Usage):**
  + **Green** = low CPU usage (~0%)
  + **Orange** = high CPU usage (up to 100%)
  + You can change this color scheme using the **color palette** (options: warm, cool, etc.)
* **Data Refresh:**
  + The data in the map **refreshes every minute** (unless you're interacting with it continuously).
  + Bottom-left of the screen shows **last update time**.

### 🖱️ Clicking on a Host (Rhombus) – What You See

When you click on a host, Datadog shows detailed information:

#### ✅ Hostname

* **Automatically collected** from different sources.
* Must be **unique** per host in your Datadog account.
* **If not unique**, you’ll see graph conflicts or data mix-ups.

#### ✅ Alias Names

* In **cloud environments**, a host might have:
  + **Instance ID** (e.g., i-12345678)
  + **Hostname** derived from IP (e.g., ip-192-0-0-1)
  + **Custom domain name** from internal DNS
* One of these is chosen as the **main hostname**; the others become **aliases**.

#### ✅ Agent Version & System Info

* Shows:
  + **Datadog Agent version**
  + OS, CPU, Memory, etc.

#### ✅ Metrics Table (Mini-View)

* Displays group-averaged metrics (like **CPU utilization across all cores**).
* **CPU Utilization** = how much your system's processing resources are being used.
* This is a **sample view**, not a complete list of all metrics.

#### ✅ Tags (Skipped for Now)

* Tags help in organizing and filtering your hosts.
* This is covered in the **next lecture**.

#### ✅ Mute Alerts

* You can **mute alerts** from this host (if alerting is already set up).
* Currently, no alerts are active—so no effect here.

### 📊 Metric Graphs (Inside the Host Figure)

* Clicking the rhombus figure opens **metric graphs** collected from that host.
* This is explained further in the **next lecture**.

### 🔁 Summary

| **Feature** | **Purpose** |
| --- | --- |
| Host Map | Visual dashboard of all reporting hosts |
| Color Coding | Visualize CPU usage (default), can be customized |
| Host Details View | Shows hostname, aliases, metrics, agent version |
| Metrics Table | Group-averaged view of key metrics like CPU, Memory |
| Data Refresh Rate | Every 1 minute |
| Mute Alerts Option | Temporarily disable alerting from the selected host |
| Metric Graphs | Click the rhombus to view graphical metric data (covered next) |

## **13. AGENT, SYSTEM, NTP METRICS IN HOST MAP**

### 🧩 **Types of Metrics in Host Map**

When Datadog Agent reports metrics from a host, these are classified into **three main types**:

1. **Agent Metrics**
2. **System Metrics**
3. **NTP (Network Time Protocol) Metrics**

### 1️⃣ **Agent Metrics** (📦 Collected from the Datadog Agent itself)

These include:

* **Dogstatsd** metrics
* **Process Agent** metrics
* **Trace Agent** metrics

🔍 These are:

* Internal metrics about how the agent itself is performing
* Mostly useful for **agent health and troubleshooting**, not so much for application developers

#### Common Agent Metrics:

| **Metric** | **Description** |
| --- | --- |
| datadog.agent.python\_version | Shows 1 if the Agent is properly reporting to Datadog |
| datadog.agent.client\_bytes | Total bytes sent from the client to Datadog |
| datadog.agent.bytes\_dropped | Bytes that failed to be sent from agent to backend |

📌 **Note**: These are less critical for Java devs or app owners. They're more relevant to infrastructure or DevOps engineers.

📎 **Tip**: There are external links in the course resources if you want detailed definitions of each agent metric.

### 2️⃣ **NTP Metrics** (⏱️ Network Time Protocol Sync Info)

* **Metric:** ntp.offset
* **Purpose:** Measures the **difference (in seconds)** between your host's local system clock and the standard NTP clock.
* **Why important?**  
  If system time is out of sync, it may affect:
  + Log timestamps
  + Tracing
  + Metric correlation

📌 Used mainly for **diagnosing time drift issues**.

### 3️⃣ **System Metrics** (🖥️ Core Performance Stats of the Host)

💡 Datadog may **show 60 system metrics**, but only the **first 6** are actual system metrics—the rest are agent-related (repeated due to display design).

#### Key CPU Metrics (Grouped in CPU Graph):

| **Metric Name** | **Meaning** |
| --- | --- |
| cpu.idle | % of time CPU is idle (not doing anything) |
| cpu.system | % of time CPU spends in kernel mode |
| cpu.user | % of time CPU spends executing user-space processes |
| cpu.iowait | % of time CPU waits for I/O operations |
| cpu.stolen, cpu.interrupt, etc. | Other CPU usage breakdowns |

📌 These are better understood by **developers or system engineers** familiar with host internals.

➡️ **Monitoring Engineer Tip:**  
You’re not expected to know the meaning of every system metric. But it’s helpful to understand basic ones like CPU usage, memory, I/O wait, etc.

🎯 A later course section will explain how to **create custom application-level metrics**, which might be more useful for Java devs.

### 📊 Graph Controls

* You can **toggle metrics** on/off in the graph for better visibility.
* Example: You only want to see cpu.user? Uncheck all others.

📈 Default time window: **Past 1 hour**  
You can adjust it to:

* 5 minutes
* 15 minutes
* 30 minutes
* 1 day
* etc.

This helps you **zoom into performance spikes or issues**.

### 🧮 Infrastructure List View

Apart from the **Host Map view**, Datadog offers an **Infrastructure List View**.

#### 🔎 Key Features:

* Displays **all hosts** reporting data
* Default time filter: **Last 2 hours**
* Max range: **1 week**
* Toggle on/off which data columns you want to see
  + Example: Turn off "load15", turn on "instance ID"

### 🧵 One-Stop Host View

Clicking on any host from the **Infrastructure list** gives a **complete detailed view**:

| **Section** | **Info You Get** |
| --- | --- |
| **Host Info** | System info, platform, OS, agent version |
| **Metrics** | CPU, memory, disk, network metrics |
| **Containers** | Lists of running containers (if any) |
| **Processes** | List of running processes |
| **Logs** | Collected logs (if enabled) |
| **Traces** | Application traces (if APM is set up) |

🎯 This is your **control center for each host**. Everything from OS info to logs and traces lives here.

### 🔚 What’s Left in Host Map?

Only one thing pending:

* The **torpedo icon**, which is related to **tagging**.

So before we dive into the torpedo, the next step is to **understand how Datadog tagging works**.

# **05.TAGS IN DATADOG**

## **14.INTRODUCTION TO TAGGING**

### 🔖 **What is Tagging in Datadog?**

Tagging in Datadog is a way to **add metadata (extra information)** to your metrics, logs, and traces. Think of it as **labeling your data** so you can easily **filter, group, and analyze** it in a meaningful way.

### 🎯 **Purpose of Tagging**

* Helps you **group multiple entities** (hosts, containers, services) under a **common label**.
* Enables **filtering and comparison** of data in dashboards and monitors.
* Without tags, it’s difficult to **narrow down issues** or **correlate data** across your infrastructure.

### 🔗 **How Tagging Works**

* Tags are defined in **key:value** format.
  + Example: env:production, region:us-west, team:backend
* These tags can be applied to:
  + **Metrics**
  + **Logs**
  + **Traces**
  + **Hosts**
  + **Containers**

### 🧠 **Why Tags are Useful**

* Imagine you want to monitor CPU usage of all servers in production:
  + Instead of checking each server individually (serverA, serverB), you can tag them all with env:production and filter by that.
* Tags are **especially important** in dynamic environments like **containers and cloud** where hostnames can change frequently.

### 📌 **Tag Format Rules (Important)**

1. **Must start with a letter**.
2. Can contain:
   * Letters, numbers
   * Underscores \_
   * Hyphens -
   * Colons :
   * Periods .
   * Slashes /
3. Special characters are converted to underscores \_.
4. **Cannot end with a colon :**.
5. Maximum tag length: **200 characters**.
6. Tags are automatically **converted to lowercase**.
   * Example: AppName:BackendService becomes app\_name:backend\_service

### ⚠️ **What to Avoid in Tags**

* **Avoid using dynamic values** like:
  + Timestamps
  + User IDs
  + Request IDs
* Why? Because these values are **unique for each event**, and can **explode the number of unique metrics**, leading to:
  + **Higher costs**
  + **Performance issues**

### ✅ **Best Practices**

* Use consistent tag keys and values: env:production, env:staging
* Avoid overly complex or highly variable tags
* Use tagging to represent **logical groupings**: teams, regions, environments, services

### 🧩 Summary

| **Feature** | **Details** |
| --- | --- |
| Format | key:value |
| Purpose | Filter, group, correlate data |
| Applies to | Metrics, logs, traces, hosts, containers |
| Example | env:prod, team:infra |
| Avoid | Dynamic/unique values (e.g. timestamps) |

## **15.UNIFIED SERVICE TAGGING (RESERVED TAGS) IN DATADOG**

### 🏷️ **Ways to Configure Tags in Datadog**

You can configure tags in **multiple ways**:

1. **Through the Datadog UI**
   * Manually assign tags to hosts or services via the web interface.
2. **In the Agent Configuration File**
   * Edit the datadog.yaml file to add global tags.
3. **In Integration Config Files**
   * Each integration (like Apache, Redis) has its own config file under the conf.d directory where tags can be set.
4. **Via the Datadog API**
   * Automate tagging using scripts or CI/CD tools.
5. **Through Code/Docs (Infrastructure-as-Code)**
   * Tags can be declared in tools like Terraform or in Kubernetes YAML manifests.

### 🧠 **Auto-Assigned Tags**

* For **non-containerized** environments, the agent **automatically tags the host** with the key host and the **value as the system name**.
* For **containerized environments** (e.g. Docker, Kubernetes), Datadog **automatically collects tags** like container ID, pod name, service, etc.

### 🛑 **Reserved Tags in Datadog**

Datadog has **reserved certain tag keys** because they are **essential for correlating data** across different telemetry sources (metrics, traces, logs, etc.). These reserved keys have **special meaning and usage**.

#### 🔑 Common Reserved Tag Keys

| **Reserved Key** | **Purpose** |
| --- | --- |
| host | Identifies the host machine. Used for correlating metrics, logs, traces, and processes. |
| device | Used to isolate metrics/logs/traces for devices or disks. |
| env | Represents the environment (e.g., prod, staging, dev). Helps scope application-level data. |
| service | Identifies the name of a service (e.g., auth-service, payment-api). |
| version | Represents the version of the code or service (e.g., v1.2.3, latest). |

### 🔄 **Unified Service Tagging (UST)**

Unified Service Tagging is a **best practice** recommended by Datadog to **standardize** how you tag services across your infrastructure.

#### 🎯 What is it?

**UST uses three specific reserved tags** consistently across all telemetry data:

* env → Environment
* service → Service Name
* version → Code/Service Version

#### ✅ Why Use It?

* Ensures that **metrics, logs, and traces can be correlated** easily.
* Allows **consistent filtering and grouping** in dashboards and monitors.
* Improves **data organization** across dynamic and distributed systems.

#### 🧪 Example:

| **Key** | **Value** |
| --- | --- |
| env | prod |
| service | payment-api |
| version | v2.1.0 |

Datadog uses these three tags behind the scenes to **link logs, traces, and metrics** — which helps a lot when debugging or analyzing system performance.

### ✏️ **Custom Tags**

While reserved keys are recommended for specific use-cases, you can **freely create your own tags**.

Example:

* Key: system
* Value: my-laptop

These custom tags can be used for any type of grouping or categorization that’s relevant to **your infrastructure or team**.

### ⏳ **Note**

* It might take **up to 5 minutes** for tag changes to propagate and show up in Datadog.
* Once tags are active, you can use them to **filter or group your data**, such as:
  + Filter logs by env:staging
  + Group CPU usage charts by service
  + Create alerts for a specific version

### 📌 Summary

| **Concept** | **Description** |
| --- | --- |
| **Tag Config Locations** | UI, Agent YAML, Integration Configs, API, Code |
| **Reserved Tags** | host, env, service, version, device |
| **Unified Service Tagging** | Standard use of env, service, version across all telemetry |
| **Custom Tags** | Freely defined (e.g., region:us-west) |
| **Use Case** | Filter, group, correlate metrics/logs/traces |

## **16.FILTERING , GROUPING ON TAGS**

### 🧭 What are Filtering and Grouping in Datadog?

* **Filtering**: Narrow down your view to only **specific data points** (hosts, services, logs, etc.) that match certain **tags**.
* **Grouping**: Organize or **cluster similar data points** together based on a tag’s value — super helpful for visualization and troubleshooting.

### 🔍 **Filtering in Datadog**

#### ✅ How to Filter:

* Use the **Filter Box** in dashboards, monitors, host maps, and more.
* As soon as you click the box, Datadog shows **common tag keys** like host, env, service, etc.

#### 🎯 Example 1: Filter by Reserved Tag

* Select key: host
* Select value: my-system-name
* Result: Only that specific host will be shown (e.g., 1 out of 10 hosts).

#### 🎯 Example 2: Filter by Custom Tag

* Key: system
* Value: my-laptop
* Only entities with that exact key-value pair will be displayed.

#### 💡 Tip: Two Types of Filtering

| **Type** | **How it works** |
| --- | --- |
| **Exact Match (Key:Value)** | Filters based on precise tag match (e.g., env:prod) |
| **Text Search (Quoted)** | Filters based on a string anywhere in the tag ("prod" will match env:prod, team:product, etc.) |

### 🧠 **Why Filtering Matters**

* Simplifies **navigation and analysis** in large environments.
* Helps **pinpoint specific resources** during incidents.
* Can be used **across Datadog**: dashboards, notebooks, monitors, host maps, logs, etc.

### 🧩 **Grouping in Datadog**

#### 📌 What is Grouping?

* Instead of just filtering, you can **group** entities (like hosts) by **common tag values** to see patterns, clusters, and anomalies.

#### 🧊 Host Map Example:

Let’s say you have EC2 instances in multiple Availability Zones (AZs):

##### Step 1: Group by availability-zone

* Hosts are split into groups like:
  + us-east-1a
  + us-east-1b

##### Step 2: Add another level — instance-type

* Within each AZ group:
  + You see clusters like c3.xlarge, m3.large, etc.

🔁 **Two-Level Grouping**:

1. First level: availability-zone
2. Second level: instance-type

### 🆓 Default Tags for Grouping

Datadog auto-assigns some useful tags, such as:

| **Tag Key** | **Description** |
| --- | --- |
| availability-zone | AWS zone (e.g., us-east-1a) |
| instance-type | EC2 instance type (e.g., t3.large) |
| region | AWS region |
| image | AMI ID or image used |

### ⚙️ **Why Grouping Is Important**

#### 🚨 Troubleshooting Example

* Suppose CPU spikes are causing high latency.
* You open the **Host Map** and **group by availability-zone**.
* If us-east-1a shows **all red hosts**, you now know the issue is limited to that zone.

#### 🔧 Optimization Example

* Group by instance-type.
* You see that c3.8xlarge is overutilized.
* Now group by role or team to find who's using those hosts the most.

This allows for:

* Smart scaling decisions
* Cost optimization
* Performance tuning

### 📊 Summary Table

| **Feature** | **Purpose** | **Example** |
| --- | --- | --- |
| **Filtering** | Show only items matching a tag | env:prod |
| **Text Filtering** | Search within tags using a string | "my-laptop" |
| **Grouping** | Cluster items based on tag value | Group by instance-type |
| **Nested Grouping** | Use two tags to form sub-clusters | availability-zone + instance-type |

## **17.ASSIGNING TAGS FROM DATADOG.YAML FILE**

### 🧾 What is datadog.yaml?

* It’s the **main configuration file** for the **Datadog Agent**.
* You can define **global settings** like tags, hostname, logging, and more in this file.

### 🎯 Objective of This Lecture

* Show **how to assign tags** directly in the datadog.yaml file (instead of using UI or API).
* Explain the **impact and visibility** of these tags inside Datadog.

### 🛠️ How to Assign Tags in datadog.yaml

#### ✅ Steps:

1. **Open Agent Manager**
   * This is Datadog’s UI for managing the agent locally.
2. **Navigate to the config file**
   * Open datadog.yaml inside the Agent Manager (or directly through file explorer).
3. **Add Tags**
4. tags:
5. - service:windows
6. - user:myuser
   * These are **key:value** pairs.
   * You can define **multiple tags** in this list format.

#### ❌ What You Should Avoid

* Avoid using just **keys without values** (e.g., - staging) — while it is technically valid, it’s **not recommended** because it lacks context.
* # Not recommended:
* tags:
* - staging # No key, no context

### 🔁 Restart Is Required

* After making changes:
  + **Save the file**
  + **Restart the Datadog Agent**

Tags only take effect **after restart**.

### 🌐 Tags Reflection in Datadog UI

* After restarting:
  + Tags become **visible in Datadog Web UI**.
  + You can now **filter and group** based on them.
* ⚠️ UI might take a **few minutes** to update.

### 🖥️ Changing the Hostname (Optional)

#### 🏷️ Default Behavior:

* Datadog auto-assigns the host name using your system’s name and links it via a reserved tag:
* host: my-computer-name

#### 🛠️ To Customize Hostname:

* In datadog.yaml:
* hostname: custom-host-name
* After changing this:
  + You’ll see **two rhombuses (host tiles)** in host maps:
    - One for the **old hostname** (temporary, no new metrics).
    - One for the **new hostname** (receives live metrics).
  + The old one remains visible for **about 2 hours**, as a temporary cache.

### 🔍 Why Assign Tags from datadog.yaml?

| **Advantage** | **Description** |
| --- | --- |
| ✅ Global Scope | Tags apply to **all metrics/traces/logs** from the host |
| ⚡ Agent-Level Control | You don’t need to set tags individually in integrations |
| 📦 Useful for Non-Container Environments | Like Windows machines, on-prem VMs, etc. |

### 📌 Summary

| **Task** | **How** |
| --- | --- |
| Assign tags | tags: section in datadog.yaml |
| Format | List of key:value pairs |
| Restart needed? | ✅ Yes, after editing |
| Show up in UI? | ✅ Yes, after few minutes |
| Customize hostname? | Use hostname: field in the same file |

## **19.HOST MAP OPTIONS**

### 🗺️ What Is the Host Map View?

The **Host Map View** in Datadog visually represents each host (VM, container, EC2 instance, etc.) as a **rhombus (diamond shape)**. This helps in quickly spotting performance issues or anomalies across your infrastructure.

### 🎨 1. **Fill By Selector** – Coloring the Hosts

#### 🔍 What it does:

* Controls the **color** of each host’s rhombus based on a selected **metric**.
* Example: By default, it might be **CPU Utilization (avg)**.

#### 🟢🟠🔴 How it works:

* **Low values** = Green (healthy)
* **High values** = Orange/Red (warning/critical)

#### 📊 Aggregation Options:

* You can change the **aggregator** too:
  + avg: average
  + sum: total
  + max: maximum value
  + etc.

✅ **Use Case**: Spot overloaded or underperforming hosts quickly by color.

### 📏 2. **Size By Selector** – Resizing the Hosts

#### 🔍 What it does:

* Adjusts the **size** of the rhombus figures based on a metric value.

#### 🔁 How it works:

* If no metric is selected: all rhombuses are same size.
* If a metric (e.g., CPU Utilization) is selected: rhombus size reflects its value.

✅ **Use Case**: **Outlier Detection** — extremely large or small rhombuses visually pop out, helping you catch anomalies faster.

### 👻 3. **Hide Hosts with No Metrics**

* **Purpose**: Cleans up the view by hiding hosts that are no longer reporting data (e.g., terminated or renamed).
* Especially useful when testing hostname changes — the old one lingers but sends no data.

### 🧭 4. **Hide Ungrouped Hosts**

#### 🔍 What it does:

* Hides hosts that **don’t belong to any group** (based on your current grouping criteria).

#### 📌 Example:

* If you group by **availability zone** but some hosts don’t have this tag, they’ll disappear (until you uncheck this option).

✅ Use this when you want to focus only on **grouped data** and avoid noise.

### 🏷️ 5. **Label Settings (Settings Icon)**

#### 🔍 What it does:

* Controls what **label** is displayed inside each host rhombus.
* Default: hostname

#### Other options:

* Cloud provider name (AWS, Azure, etc.)
* Custom tags (if applicable)

🧪 Example: If your host doesn’t have a cloud provider, label may show as “unknown”.

### 💾 6. **Save Custom Views**

#### 🔍 What it does:

* Save your current **host map configuration** (filters, grouping, display settings) for future reuse.

#### How to do it:

* Click **Save View**
* Enter a name → Done ✅

#### 🧭 View Examples:

| **View Name** | **Description** |
| --- | --- |
| Default View | Shows default groupings (e.g., by AZ) |
| Demo View | Shows your custom settings |

✅ **Use Case**: Helps create quick-access dashboards tailored to your environment or investigation needs.

### 📌 Summary Table

| **Feature** | **Purpose** | **Example** |
| --- | --- | --- |
| **Fill By** | Colors hosts based on metric | CPU usage shows orange if high |
| **Size By** | Resizes host shapes based on metric | Big rhombus = heavy usage |
| **Hide No Metrics** | Removes inactive hosts | Old hostnames disappear |
| **Hide Ungrouped** | Filters out ungrouped hosts | Hosts without AZ tag hidden |
| **Label Settings** | Changes what label shows in host tile | host, cloud\_provider, etc. |
| **Save View** | Saves current layout and filters | Production-Cluster-Map view |

# **06.INFRASTRUCTURE MONITORING – PROCESSES**

## **20.PROCESS EXPLORER – PART 1**

### 🔍 What is the Process Explorer?

**Process Explorer** in **Datadog** (under the Infrastructure section) is a **real-time dashboard** that provides visibility into all the processes running across your infrastructure—be it **on-premises**, **hybrid**, or **cloud-native**.

### 🚫 Why Is It Empty Initially?

By **default**, the **live process collection is disabled**. So, even though the agent is running, this section might show nothing.

### 🛠️ How to Enable Process Collection?

You need to **edit the agent’s main config file**, datadog.yaml.

In this file, add or update the following block:

process\_config:

enabled: true

* enabled: false (default) → only collects container data, not full process info.
* enabled: true → collects **both processes and containers**.
* enabled: disabled → turns off the process agent completely.

✅ After changing it to true, **save the file** and **restart the Datadog agent**.

Then refresh Datadog – and voilà – you’ll see **all processes like SVC, Chrome, etc.** being listed.

### 🧭 Navigating to Process Explorer

Go to:  
**Infrastructure → Processes → Process Explorer**

Here, you’ll find **two main types of visualizations**:

### 📈 1. **Time Series Graph**

* Shows **CPU and memory usage** of each process **over time**.
* Each line represents a different process (e.g., Chrome, Firefox).
* You can **visually track spikes or trends** in CPU/memory usage for each process.

### 📊 2. **Scatterplot Graph**

* Lets you **compare two metrics simultaneously** across all process types.

#### 🧮 Example Comparison:

* **X-axis**: Average **User CPU Utilization**
* **Y-axis**: Average **System CPU Utilization**
* **Group By**: Command tag (default), which categorizes by process type.

#### 🟢 Visual Details:

* Each **dot** = a group of processes.
* **Position** = metric values.
* **Size of dot** = **number of processes in that group**
  + Bigger circle → more processes.
  + Smaller circle → fewer processes.

For example:

* Chrome group: 0.23% user CPU, 0.048% system CPU.
* SVC host: 76 processes → big dot.
* AudioG: 1 process → small dot.

### ✅ Summary

* **Process Explorer** gives deep insight into what’s running on your systems in real time.
* You get:
  + **List of all processes**
  + **Live metrics (CPU, memory, etc.)**
  + **Powerful graphs (time series + scatterplot)** to compare and analyze
* Just make sure you **enable it** in datadog.yaml.

## **21.PROCESS EXPLORER – PART 2**

### 🔎 Continuation of Process Explorer View

After the graphs (Time Series & Scatterplot), you’ll find a **detailed table view** of all the running processes.

### 📋 **Table of Processes**

This table shows key process details like:

* Process Name
* Username (who started it)
* Host
* Hostname

#### ➕ Add More Columns:

* Click on **“Customize”**
* Toggle additional fields (e.g., **PID** – Process ID)
* Instantly appears in the table

### 🔃 **Sorting the Table**

* Click any column header (e.g., **CPU%**)
  + 🔽 First click: Sort in **descending** order
  + 🔼 Second click: Sort in **ascending** order
* Works for **any column** (CPU%, memory, username, etc.)

### 🔍 **Click on a Process for Details**

You can click on any row to see detailed insights.

#### Inside that detailed panel, you'll get:

1. **Resource Metrics**
   * CPU, memory, etc., shown as graphs
2. **Application Traces**
   * (Empty if not enabled yet)
3. **Logs**
   * (Also empty if not configured)
4. **Network Flow Data**
   * (If network monitoring is enabled)

🧠 This gives you a **single pane of glass** for debugging – everything tied to a specific process is visible in one place.

### 🧩 **Tagging, Filtering, Grouping – Just Like Infrastructure Map**

* You can **filter** and **group** the process list based on **tags** (like host, command, etc.).
* Especially helpful because:
  + Processes have **high cardinality** (lots of different ones even on a single host!)
  + So, use tags to **narrow down your view**.

### 🏷️ **Command Tag & Filtering**

Datadog **auto-generates** a command tag for every process. This allows:

* Filtering third-party software
* Isolating common workloads
* Example: See only Chrome, Audacity, or Firefox by filtering on command:chrome, etc.

### 🔍 **Advanced Search Bar Usage**

The **search bar** lets you create powerful search queries.

#### ✅ Basic Search:

* Just type a text string (min. 2 characters) → shows processes matching that in the **command line or path**.
* Example: typing chr will start showing Chrome-related processes.

#### 🧠 Use **Boolean Operators**:

* **AND** – both strings must appear
  + chrome and utility → shows only utility processes of Chrome
* **OR** – either string must appear
  + chrome or firefox → shows processes for both
* **NOT** – exclude specific results
  + chrome not utility → shows all Chrome processes **except utility**
  + chrome !utility → same effect

### 🧭 **Facet Panel (Left Side) – Quick Filtering & Grouping**

This panel provides a **shortcut for common filters**:

* Based on **default tags** like:
  + host
  + command
  + (In cloud): availability-zone, instance-id, region

Datadog automatically:

* Extracts these tags
* Creates this panel
* Lets you click and **instantly apply filters or groupings** without typing anything

### 💾 **Saving the View**

* You can **save the customized process view** (with columns, filters, sort order) for future use.

### ✅ Summary

* **Table View**: Add/sort columns like PID, CPU%, etc.
* **Click into a process** for full metrics, logs, and traces.
* **Powerful filtering** using tags, Boolean search, and facets.
* **Facet Panel** = smart, auto-built quick filter menu
* Save your custom view for reuse.

## **22.SCRUBBING SENSITIVE DATA**

### 🛡️ What is Process Argument Scrubbing?

In real-world systems, many processes include **sensitive information** in their **command-line arguments** — like:

* Passwords
* API keys
* Access tokens
* Usernames, etc.

If shown directly in Datadog’s **Live Processes page**, this data can pose a **security risk**.

### 🧽 How Does Datadog Handle It?

#### ✅ **Datadog agent scrubs sensitive arguments by default.**

When it detects arguments with **specific keywords** (like password, mysql\_password, etc.), it **automatically hides** (scrubs) their values.

* Matching is **case-insensitive**
* The sensitive part is replaced with \*\*\* or similar masking characters

### ⚙️ Where Is This Controlled?

This is managed via the Datadog agent’s config (datadog.yaml) under the section:

process\_config:

scrub\_args: true

* true (default): enables scrubbing for default keywords
* false: **disables** scrubbing completely — not recommended unless you're testing in a secure, internal environment

### ✍️ Can You Add Custom Sensitive Keywords?

#### ✔️ Yes, you can define **custom sensitive words**!

Under the same process\_config section, use:

custom\_sensitive\_words:

- type

- user\*

* This adds to the **default scrubbing list**
* You can match:
  + Exact strings like type
  + Or **wildcards** like user\* (matches user, username, user\_id, etc.)

### ✅ Rules for Wildcards and Custom Words:

1. **Wildcards** must be part of a word (e.g., user\* is valid, but \* alone is not).
2. Allowed characters: **A-Z, a-z, 0-9**, underscores (\_), and wildcards (\*)
3. Your custom list is **merged with default** scrubbing keywords.

### 🕒 When Does Scrubbing Take Effect?

* Scrubbed values may not appear **instantly**
* Datadog scrubs as **each process is scraped**
* If multiple processes share the same argument, scrubbing may apply **gradually**

So give it a little time after making changes and **restarting the agent**.

### 🧪 Example:

Suppose:

* You want to scrub any argument containing type
* And all arguments starting with user

#### You’d do this:

process\_config:

scrub\_args: true

custom\_sensitive\_words:

- type

- user\*

#### Result:

* In the **Live Processes view**, those arguments will appear as \*\*\*\*\*\*\*\* or masked text
* You’ll notice that arguments like --user=test, --username=admin, --type=token will all be scrubbed

### 💡 Advanced: Scrub **All** Arguments

If you want to **scrub every single argument**, you can also enable:

process\_config:

strip\_proc\_arguments: true

This completely hides all arguments, regardless of keywords.

### ✅ Summary

| **Feature** | **Description** |
| --- | --- |
| scrub\_args: true | Enables default scrubbing (e.g., password) |
| custom\_sensitive\_words | Add your own keywords (type, user\*, etc.) |
| Wildcards | Allowed but must be part of a word |
| scrub\_args: false | Disables scrubbing (not recommended) |
| strip\_proc\_arguments: true | Hides **all** command-line arguments |
| Scrubbing delay | Happens gradually as processes are scraped |

## **23.CREATE CUSTOM PROCESS METRICS**

### ❓ Why Do We Need Custom Process Metrics?

#### 🔴 Limitation:

By default, **Live Processes** data in Datadog is retained for **only 36 hours**.

#### 🟢 Solution:

To **retain process data for up to 15 months**, you can **generate custom metrics** from selected processes.

These metrics allow long-term tracking of:

* CPU and memory usage of critical processes
* Trends over time
* Impact of deployments or stress tests
* Anomaly detection or threshold-based alerts

### 📊 What Are Process-Based Metrics?

These are **time-series metrics** generated from specific running processes. Once created:

* They are retained for **15 months**
* Can be used in dashboards and monitors
* Are sampled every **10 seconds**

### 🛠️ How to Create a Custom Process Metric in Datadog?

#### ✅ Option 1: Using the “New Metric” Button (Recommended UI Flow)

1. **Go to the Live Processes page**
2. Click on **“New Metric”**
3. The interactive table appears listing all current processes

#### 🧪 Example: Create Metric for Zoom Process

##### Step-by-Step:

1. **Filter** the Zoom process:
   * Use the command **tag** filter (e.g., command:zoom)
   * ❌ Note: Free-text (text search) filtering is not supported in the “New Metric” UI
2. **Choose the Measure**:
   * For example: Total CPU % (i.e., total CPU consumption by the process)
3. **Group By (Tags):**
   * Add tags you want to associate with the metric.
   * ✅ Best Practice: Use **bounded/low-cardinality tags** like host
   * ⚠️ Avoid high-cardinality tags (like command or user) as they:
     + Increase costs
     + Impact performance
4. **Name the Metric**:
   * Datadog auto-prefixes with proc.
   * Format: proc.<your\_process\_name>.<measure>
   * Example: proc.zoom.cpu\_total\_percent
5. **Enable Percentile Metrics** (Optional):
   * Check this box to also create:
     + P50, P75, P90, P95, P99 metrics
   * 📌 These **percentiles are billed as custom metrics**
6. **Save the Metric**

#### ⏱️ What Happens After You Create It?

* Metric data starts being reported within **~3 minutes**
* New data points are generated every **10 seconds**
* The metric now behaves like any other in Datadog:
  + Can be viewed in **Metrics Explorer**
  + Can be used in **dashboards and monitors**

#### 📈 Where to View It?

* Navigate to **Metrics Explorer**
* Filter for your metric, e.g., proc.zoom.cpu\_total\_percent
* You’ll see historical trends, and can apply graphs, filters, or time comparisons

### 🧠 Benefits of Custom Process Metrics

| **Use Case** | **Benefit** |
| --- | --- |
| Long-term history | Store CPU/Memory trends for 15 months |
| Deployment analysis | Compare pre/post-deploy performance |
| Issue diagnosis | Look back at past behavior of key processes |
| Monitoring | Create alerts for abnormal CPU/memory usage |
| Capacity planning | Analyze resource patterns across time |

### ✅ Summary

| **Feature** | **Details** |
| --- | --- |
| Default process data retention | 36 hours |
| Metric retention (once created) | 15 months |
| Sample frequency | Every 10 seconds |
| Best filter for process | Tag filters (like command) |
| Avoid high-cardinality tags | command, user, etc. |
| Percentile support | P50, P75, P90, P95, P99 (as custom metrics) |
| Where to use the metric | Metrics Explorer, Dashboards, Monitors |

# **07.INFRASTRUCTURE MONITORING – CONTAINERS**

## **24.INTRODUCTION**

Here’s a **crystal clear breakdown** of the transcript titled **"Introduction to Infrastructure Monitoring - Containers"** in the context of **Datadog**:

### 🔹 **What is Container Monitoring All About?**

Since we're already learning about **infrastructure monitoring**, the next big area is **monitoring containerized environments**, such as **Docker**.

### 🚀 **What are Containers?**

* Containers are **lightweight, standalone software units** that package everything needed to run an application (code, runtime, system tools, libraries).
* They **virtualize the OS** rather than hardware (like VMs), which makes them:
  + Portable: Run anywhere – laptop, data center, or cloud.
  + Fast: No OS boot-up required.
  + Efficient: Share the host OS kernel, so they’re lightweight.

### 📈 **Why Are Containers Important?**

* Their usage is **exploding in modern DevOps and cloud environments**.
* **Gartner Prediction**: By **2023**, **70%+ organizations** will run **multiple containerized apps in production**.
* They are **reshaping how businesses deploy and manage applications**.

### 🐳 **What About Docker?**

* **Docker = Most popular container platform**.
* It’s so dominant that people often use "Docker" and "containers" interchangeably.
* This course/tutorial focuses on **monitoring Docker containers** with **Datadog**.

### 📦 **Docker Monitoring with Datadog**

* Datadog can monitor containers running on **any platform**, but for demo and hands-on, Docker is used.
* To monitor Docker containers:
  + You **must have Docker installed** on your system.

### 💻 **Do I Need to Install Docker Myself?**

* **Yes**, for learning and hands-on purposes, you will install Docker yourself.
* Normally, **installation is a system engineer's job**, not a monitoring engineer’s.
  + But to learn monitoring, we do this ourselves.
* Docker can be installed on **Windows**, **Mac**, or **Linux (Ubuntu)**.

### ⚠️ **Things to Keep in Mind (Especially for Windows Users):**

* Docker install on **Windows can be tricky** depending on your system version.
* The instructor provides:
  + A detailed setup guide for Windows.
  + Troubleshooting tips.
  + Official documentation links to check system requirements.
* If you face too many issues:
  + Consider switching to **Ubuntu**.
  + There are **separate lectures** for installing Docker on Ubuntu or via **VirtualBox**.

### ✅ **Next Steps**

* For Docker setup on Windows, follow the instructions in the **next lecture**.
* For Ubuntu setup, check the **additional learning section**.

### 🔚 **Summary**

* Containers are key in modern infrastructure.
* Docker is the most used container tool.
* You will learn to monitor Docker containers using Datadog.
* First step: **Install Docker** (yourself, for hands-on learning).
* Issues? Switch to Ubuntu – alternative guides are available.

## **25.DOCKER SETUP**

This guide walks you through **Docker installation** on **Windows**, which is a **prerequisite to monitor containers** using Datadog.

### 🧩 **Step 1: Understand Your Windows Version**

There are two paths:

* **Windows Pro**: Has built-in support for Hyper-V and advanced features.
* **Windows Home**: Requires **WSL2 (Windows Subsystem for Linux version 2)** to run Docker.

For this guide, focus is on **Windows Home setup** using WSL2.

### 🔧 **Step 2: Enable WSL2 (Windows Subsystem for Linux)**

**WSL** allows you to run a Linux environment inside Windows.  
This is required because Docker Desktop runs using this Linux layer on Windows Home.

#### 🪄 Here's How to Do It:

1. **Open CMD as Administrator**.
2. Run this command to list available Linux distros:
3. wsl -l -o

→ This will show available distributions like **Ubuntu**, **Debian**, etc.

1. **Install a distribution** (example with Ubuntu 20.04):
2. wsl --install -d Ubuntu-20.04

→ Downloads and installs Ubuntu from Microsoft Store.

1. **After installation**, it’ll prompt you to:
   * Set **username**
   * Set **password**
2. A **Linux terminal (Ubuntu)** will open – this is your full Linux shell on Windows.

### 🔁 **Step 3: Ensure You’re Using WSL2**

Docker needs the **WSL2 version** of your Linux distro.

1. Check the current version:
2. wsl -l -v

→ If it shows version 1, upgrade using:

1. Convert to WSL2:
2. wsl --set-version Ubuntu-20.04 2

### 💾 **Step 4: Install the Linux Kernel Update**

* Download the **Linux Kernel update package** from the official Microsoft link (provided in course).
* Run the installer (.msi) file.
* **Restart your system** after installation.

### 🐳 **Step 5: Install Docker Desktop for Windows**

1. Download Docker Desktop from Docker’s official site.
2. Run the installer:
   * Accept agreements
   * Click through “Next” steps
3. **Restart** your PC if prompted.

### ⚙️ **Step 6: Post-Installation Settings in Docker Desktop**

Make sure Docker is correctly configured:

#### ✅ Enable WSL2 Engine

* Open Docker Desktop → Settings → General
* **Ensure** "Use WSL2 based engine" is **checked**

#### 🔗 Enable WSL Integration

* Go to Settings → Resources → **WSL Integration**
* Enable integration for the installed distro (e.g., Ubuntu)
* Click **Apply & Restart**

### 🧪 **Step 7: Test Docker with a Sample Container**

Let’s test if Docker is running fine.

1. Run a sample container:
   * Example: Run a Redis container or any test container.
   * You can click **Run** from Docker Desktop’s interface or use CLI.
2. If container starts and runs successfully → Docker setup is complete.

### ✅ **At This Point**

* You now have a **working Redis container** (or any test container) running locally.
* It is ready to be **monitored via Datadog**.

## **26.CONTAINERIZED DOCKER AGENT SETUP**

### 🐳💡 **Goal of This Lecture**

To **monitor containers** the **right way** using **Datadog’s Docker Agent**, which is a **containerized version** of the Datadog Agent — designed specifically for environments where containers are running.

### ✅ **Initial Setup Recap**

* Docker is **already installed** on your system.
* A sample **Redis container is running**.
* The **Datadog Agent for Windows** (host agent) was also already running.
  + It **auto-detected the running container** and started collecting data.
  + This is thanks to **Datadog Auto Discovery**, which:
    - Detects when a new container starts.
    - Identifies services inside it.
    - Applies any existing monitoring configs automatically.

### ⚠️ **Why Using Host Agent is NOT Practical in Real Environments**

* In a real-world scenario:
  + You might have **hundreds of hosts**, each with **multiple containers**.
* Relying on **host-based agents** for all those systems is **complex and inefficient**.

#### 🧪 So what’s the solution?

✅ Use **Datadog Docker Agent** – a **containerized agent** built specifically for Docker environments.

### 🔁 **Steps for Switching to Docker Agent**

#### 🔄 Step 1: Stop the Windows Host Agent (Optional for clarity)

* Just to avoid confusion, the **Windows host agent is stopped**.
* Also, the **Redis container is removed** to start fresh.

#### 📦 Step 2: Install the Docker Agent (Containerized)

* This agent is available officially on:
  + **Docker Hub**
  + **Google Container Registry (GCR)**
  + **Azure Container Registry (ACR)**

#### 🧾 One-line Install Command

Here’s what this command does:

docker run -d --name datadog-agent \

-e DD\_API\_KEY=<YOUR\_API\_KEY> \

-e DD\_SITE="datadoghq.com" \

-v /var/run/docker.sock:/var/run/docker.sock:ro \

-v /proc/:/host/proc/:ro \

-v /sys/fs/cgroup:/host/sys/fs/cgroup:ro \

gcr.io/datadoghq/agent:latest

Replace <YOUR\_API\_KEY> with your actual Datadog API key.

#### 🔧 What does the command do?

* **Pulls the Docker agent image**.
* **Mounts Docker socket**: Lets the agent communicate with the Docker daemon to detect containers.
* **Mounts host paths**: Required to collect metrics like CPU, memory, logs.
* **Starts the Datadog agent** as a container.

#### 📊 **After Running the Command**

* You’ll see a new container named datadog-agent in **Docker Desktop**.
* This means the agent is now running **inside a container**, collecting metrics about:
  + The **host system**
  + All **other running containers**

### 🔍 **How to Check Agent Status**

#### 1. **Check if container is running**

docker ps

You should see datadog-agent in the list.

#### 2. **Get Agent Status**

docker exec -it datadog-agent agent status

#### 3. **Get Agent Info**

docker exec -it datadog-agent agent info

### 🛑 **Stopping & Restarting Docker Agent**

#### To Stop:

docker stop datadog-agent

#### To Restart:

* Either **click “Play”** in Docker Desktop
* Or re-run the **original Docker run command**

### ✅ **Summary**

| **What You Did** | **Why It Matters** |
| --- | --- |
| Switched from OS agent to containerized agent | More scalable, ideal for large Docker-based infra |
| Used Docker socket and host mounts | Enables the agent to access container stats |
| Verified the container & agent status | Ensures it's collecting and reporting metrics to Datadog |
| Learned to stop/restart agent | Operational control over the setup |

## **27.CONTAINER MAP AND LIVE CONTAINERS**

### 🚀 **What’s Happening**

Now that:

* Docker is installed ✅
* Docker containers (e.g., Redis) are running ✅
* And the **Datadog Docker Agent** is active ✅

→ You’re now ready to **visualize and monitor** your containerized infrastructure on the **Datadog web interface**.

### 🗺️ **Container Map View**

Just like **host maps** (where you see a visual overview of host metrics), Datadog provides a **Container Map**, which gives you a **real-time visual representation** of all containers in your environment.

#### 👀 What You’ll See:

* Navigate to: **Datadog → Infrastructure → Containers tab**
* You will see all **running containers**, such as:
  + datadog-agent (monitoring agent)
  + redis (your application container)

### 🧠 **How It Works**

As soon as the **Datadog Agent** is installed:

* It **auto-discovers** all containers.
* Starts **collecting metrics**.
* Sends them back to Datadog UI.

Even if you’ve **shut down a host agent**, it may still show old data temporarily until Datadog refreshes the view.

### 📌 **Click on a Container → What Details Do You Get?**

* **Container Name**
* **Host** (e.g., Docker Desktop)
* **Tags**, such as:
  + container.id
  + container.name
  + docker.image
  + image.tag
  + etc.

✅ These tags are **auto-discovered** by the agent.

#### 🏷️ Can You Add Custom Tags?

Yes!

* You can provide **your own tags**.
* You can even **extract Docker labels** as tags.
* This is done using **environment variables** (explained in a later lecture).

### 📊 **Real-Time Resource Graphs**

Each container panel shows:

* **CPU usage**
* **Memory usage**

Useful for spotting performance issues quickly at the container level.

### 🧩 **Live Containers View**

#### 🧾 Definition:

Datadog **Live Containers** is a **real-time dashboard** that provides deep visibility into your **running containers** across all environments.

#### 📅 Key Features:

* **Auto-refreshing table**, updated every **2 seconds**
* Inspired by terminal tools like:
  + htop, c-top, kubectl top
* Works across:
  + Docker
  + Kubernetes
  + Other container runtimes

### 📐 **Live Containers View - Breakdown**

* You’ll see a **table** listing all **running containers**
* Default columns:
  + **RSS Memory**
  + **CPU usage**
* You can **customize columns** as needed
* Filters and groupings available (like by image, container name, host, etc.)

### 🔍 **Drill-down on a Container**

Click any container row to:

* View detailed metrics
* Logs (if log collection is enabled)
* Network traffic
* Process-level metrics
* All in one pane

### 📚 **Want to See All Docker Metrics?**

* Datadog agent collects **many more metrics** than just CPU/RAM.
* A full list is available in their official documentation (link in the course resources).

Examples of other metrics:

* Disk I/O
* Network usage
* Container uptime
* File descriptor usage

### ✅ **Summary Table**

| **Feature** | **Description** |
| --- | --- |
| **Container Map** | Visual layout of containers, their tags, and health |
| **Auto Discovery** | Detects containers & services automatically |
| **Tags** | Auto-generated + user-defined (via env vars/labels) |
| **Graphs** | CPU + Memory usage per container |
| **Live Containers** | Real-time, table-based monitoring updated every 2s |
| **Click Details** | Deep dive into container’s metrics, logs, network, etc. |
| **More Metrics** | Full list available via official docs |

## **28.REALISTIC APPROACH**

### 🧩 **Context Recap**

Earlier, we manually installed the **Datadog Docker Agent** using a docker run command on a single host (your local machine or laptop).

✅ That works fine for testing or learning.

🚫 But in **real-world production**, this approach is **not scalable** because:

* You’ll have **multiple hosts**
* Containers can **move between hosts**
* And you **can’t SSH into each host manually** to install the agent!

### 🏭 **What’s the Realistic Production Approach?**

Instead of manually running docker run on each machine, a **smarter & scalable** approach is:

#### 👉 Use **Docker Compose** or **Dockerfiles** to define the **Datadog Agent as a service**:

* Just like you define your application containers (e.g., Redis, Java backend)
* You add a **Datadog Agent container** in the same Compose file

So when you bring up your app stack (docker-compose up), the Datadog Agent gets launched **automatically** alongside your application containers.

### 🧱 **Docker Tools Used**

| **Tool** | **Purpose** |
| --- | --- |
| **Dockerfile** | Builds a Docker image |
| **Docker Compose file (docker-compose.yml)** | Describes how to run one or more containers, with settings like: memory, CPU, volumes, networks, environment variables |

### 🔧 **Key Component: Environment Variables**

Now, let’s talk **environment variables** — the heart of this lecture.

When using Docker images of the Datadog Agent:

* You **can’t directly edit datadog.yaml** (the traditional config file used in host installs)
* Instead, you configure everything via **environment variables**

These env vars are defined **inside the Docker Compose file**, under the environment: section for the Datadog Agent service.

### 🧠 **Examples of What You Can Configure Using Env Vars**

* DD\_API\_KEY → Set your Datadog account API key
* DD\_LOGS\_ENABLED=true → Enable log collection
* DD\_TAGS="env:prod team:backend" → Add custom tags
* DD\_PROCESS\_AGENT\_ENABLED=true → Enable process monitoring
* DD\_SITE="datadoghq.eu" → For EU region accounts

These are direct **equivalents** of what you’d configure in datadog.yaml.

### 🔁 **General Rule for Env Vars**

While full rules will be discussed in the **next lecture**, a key point is:

Almost **every setting** you’d configure in datadog.yaml can also be configured using **environment variables** with the prefix DD\_.

### 🔁 **Summary Table**

| **🔍 Area** | **✅ Manual Setup** | **✅ Realistic Setup** |
| --- | --- | --- |
| Scale | Works only for small setups | Scales across all environments |
| Trigger | docker run command | docker-compose.yml or infra-as-code |
| Agent Deployment | Manual per host | Automatically spun up |
| Configuration | datadog.yaml (host agent) | environment: section in Docker Compose |
| Flexibility | Less | High — flexible, reproducible, automated |

### ✅ Final Takeaway

In production, **never manually install the agent per host**.  
Instead, **embed the Datadog Agent** inside your Docker Compose or orchestrated deployment and configure it through **environment variables**.  
This makes container monitoring **automated, scalable, and reliable**.

## **29.ENVIRONMENT VARIABLES TRANSLATION RULES**

### 🎯 **Goal of This Lecture**

To explain **how configuration parameters in datadog.yaml** (used in host-based agent setup) are **translated into environment variables** for the **Docker Agent**.

Why?  
Because when you use the Datadog Agent as a **Docker container**, you can’t use datadog.yaml — instead, you pass configurations using **environment variables**.

### 🔁 **Translation Rules for Environment Variables**

Let’s go step by step with **rules + examples**:

#### ✅ **1. All variable names must be in UPPERCASE**

Environment variables must be written in **capital letters**.

* Example:
  + api\_key → ✅ DD\_API\_KEY

#### ✅ **2. Every variable must start with** DD\_

Prefix all Datadog variables with DD\_.

* Example:
  + hostname → ✅ DD\_HOSTNAME
  + site → ✅ DD\_SITE

#### ✅ **3. List values are space-separated (not comma-separated)**

When passing multiple values like tags, **use spaces** between them.

* tags: env:prod, team:backend in datadog.yaml  
  becomes  
  ✅ DD\_TAGS="env:prod team:backend"

#### ✅ **4. Nesting of predefined keys uses underscores**

If a config parameter has **nested keys** and those keys are **predefined**, use \_ to join them.

* Example:
  + In datadog.yaml:
  + cluster\_agent:
  + cmd\_port: 5005

→ Becomes  
✅ DD\_CLUSTER\_AGENT\_CMD\_PORT=5005

* For proxies:
* proxy:
* http: http://proxy.com
* https: https://proxy.com

→ Becomes  
✅ DD\_PROXY\_HTTP=http://proxy.com  
✅ DD\_PROXY\_HTTPS=https://proxy.com

#### ✅ **5. For user-defined nested configs, use JSON format**

If the nested key is **not predefined by Datadog**, i.e., it's **custom/user-defined**, then you use **JSON inside the environment variable**.

* Example:
* container\_env\_as\_tags:
* ENV\_NAME: env

→ Becomes  
✅ DD\_CONTAINER\_ENV\_AS\_TAGS='{"ENV\_NAME": "env"}'

Note:

* No underscores used in variable name.
* Entire value is JSON.

#### ⚠️ **6. Some settings are exceptions to the rules**

A few environment variables do **not follow the underscore/nesting rule**.

* To enable the **process agent**:
  + Expected: DD\_PROCESS\_CONFIG\_ENABLED ❌
  + Actual: ✅ DD\_PROCESS\_AGENT\_ENABLED=true
* Other such exceptions:
  + DD\_LOGS\_ENABLED
  + DD\_APM\_ENABLED

These are **flat**, simpler forms.

### 🧩 **Final Notes**

* While **most settings** from datadog.yaml have env var equivalents, **not all do**.
* A full list is available in the **official Datadog documentation**, which is linked in the course resource.

### 📌 **Quick Reference Cheat Sheet**

| **datadog.yaml Param** | **Docker Env Var Equivalent** |
| --- | --- |
| api\_key: abc123 | DD\_API\_KEY=abc123 |
| hostname: myhost | DD\_HOSTNAME=myhost |
| tags: env:prod, team:backend | DD\_TAGS="env:prod team:backend" |
| proxy.http: http://proxy | DD\_PROXY\_HTTP=http://proxy |
| cluster\_agent.cmd\_port: 5005 | DD\_CLUSTER\_AGENT\_CMD\_PORT=5005 |
| container\_env\_as\_tags (user-defined keys) | DD\_CONTAINER\_ENV\_AS\_TAGS='{...}' |
| Enable logs | DD\_LOGS\_ENABLED=true |
| Enable APM | DD\_APM\_ENABLED=true |
| Enable process agent | DD\_PROCESS\_AGENT\_ENABLED=true |

## **30.RUN DOCKER AGENT FROM DOCKER COMPOSE FILE**

### 🎯 **Purpose of This Lecture**

To explain how to run the **Datadog Agent as a Docker container** using a **Docker Compose file** in a **realistic and production-ready way** — using **environment variables** for dynamic configuration.

### 🚀 **Key Concepts Covered**

#### ✅ 1. **Avoid Hardcoding in Docker Compose**

Instead of hardcoding values like the API key directly inside the docker-compose.yml, you define **placeholders** (environment variable names), such as:

DD\_API\_KEY: ${DD\_API\_KEY}

This makes your compose file:

* Clean
* Secure
* Reusable across environments (dev/stage/prod)

#### ✅ 2. **How to Supply Variable Values**

There are **three main ways**:

##### 🔹 A. Inline at runtime (e.g., docker run)

You’ve already seen how we can pass env vars like:

docker run -e DD\_API\_KEY=abc123 datadog/agent:latest

##### 🔹 B. Directly inside the docker-compose.yml (not ideal)

environment:

- DD\_API\_KEY=abc123 # Not recommended, hardcoded

##### 🔹 C. From a separate .env file ✅ (best practice)

Create a file named .env with contents:

DD\_API\_KEY=abc123

DD\_SITE=datadoghq.com

DD\_ENV=production

Then in docker-compose.yml:

env\_file:

- .env

This keeps secrets/configs **separate** from the compose logic.

#### ✅ 3. **Benefits of Using Environment Variables**

* **Scalability**: Easily change configs between environments.
* **Security**: Keeps secrets (like API keys) out of your codebase.
* **Flexibility**: You can update values without editing the docker-compose.yml.
* **Service Tagging**: You can include tags like env, version, and service as labels for **Unified Service Tagging** in Datadog.

#### ✅ 4. **Precedence of Environment Variables**

If both are set:

* DD\_PROXY\_HTTP in .env
* proxy.http in datadog.yaml

**Result**: The **environment variable wins**.

Environment variables take **precedence** over config files like datadog.yaml.

#### ✅ 5. **Final Compose File Setup (Realistic)**

Here’s how a realistic docker-compose.yml might look:

version: '3'

services:

datadog-agent:

image: datadog/agent:latest

container\_name: datadog-agent

env\_file:

- .env

environment:

- DD\_DOGSTATSD\_NON\_LOCAL\_TRAFFIC=true

volumes:

- /var/run/docker.sock:/var/run/docker.sock:ro

- /proc:/host/proc:ro

- /sys/fs/cgroup:/host/sys/fs/cgroup:ro

ports:

- "8125:8125/udp"

- "8126:8126"

And .env file:

DD\_API\_KEY=abc123

DD\_SITE=datadoghq.com

DD\_ENV=production

#### ✅ 6. **Running the Setup**

Once everything is ready:

1. **Install Docker Compose** if not already:
2. sudo curl -L "https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
3. sudo chmod +x /usr/local/bin/docker-compose
4. **Run the containers** in detached mode:
5. docker-compose up -d
6. Datadog Agent container + other containers (e.g., Redis) get launched automatically.

### ✅ **End Result**

* Datadog Agent runs as a Docker container.
* Configurations are managed cleanly via .env file.
* Metrics, logs, tags, and labels are captured.
* Scalable and production-ready setup achieved.

<https://chatgpt.com/share/684aa107-d94c-8009-a522-69b67bc6214a>

<https://chatgpt.com/share/684aa133-e59c-8009-85c8-aec108b25395>