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)

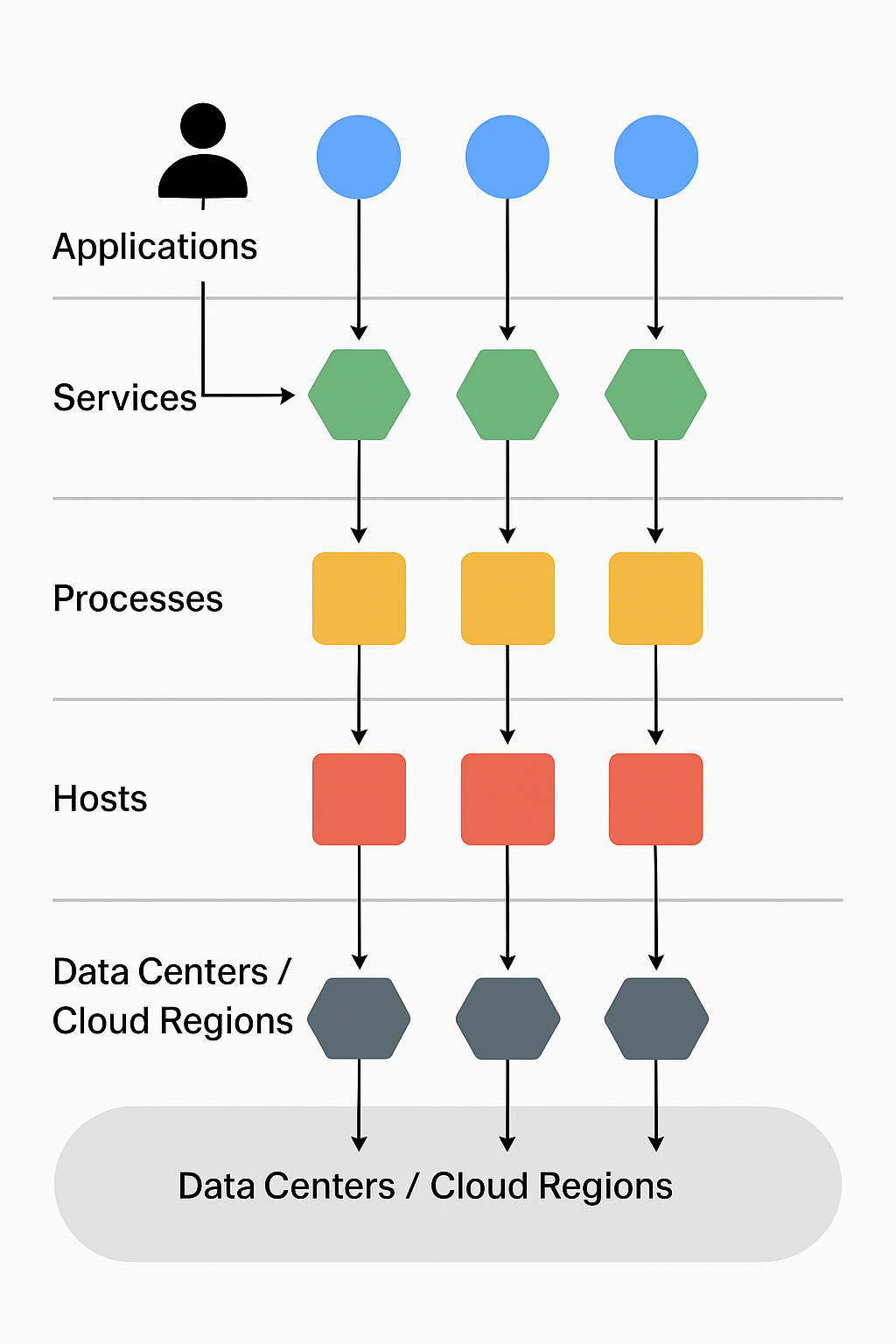
**Key Benefits of Smartscape**

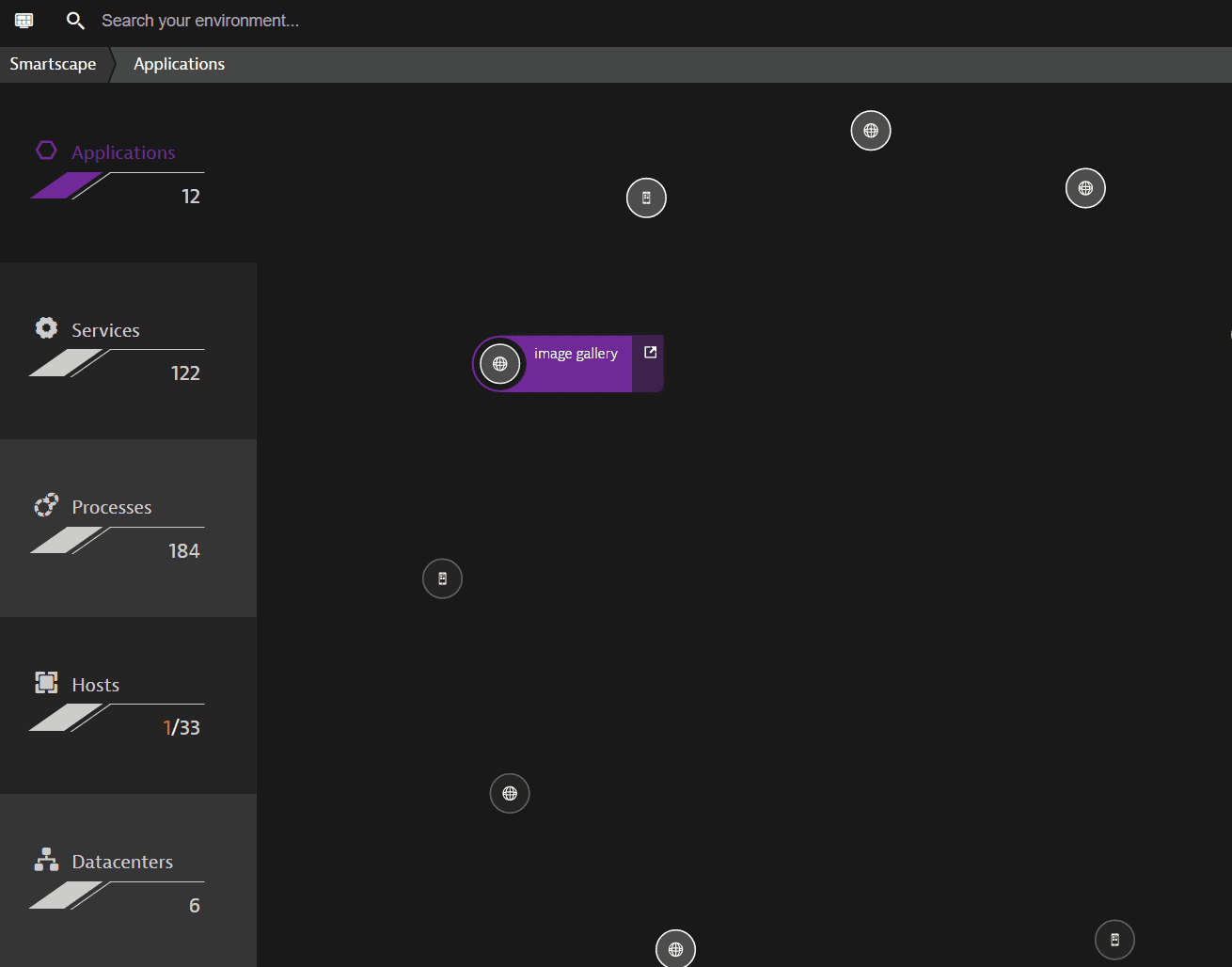
* **End-to-end visibility**: Full topology from users down to infrastructure.
* **Auto-updating**: Topology refreshes in near real-time when entities are added/removed.
* **Dependency mapping**: Shows inbound/outbound service calls.
* **Problem correlation**: Davis AI uses Smartscape to understand root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

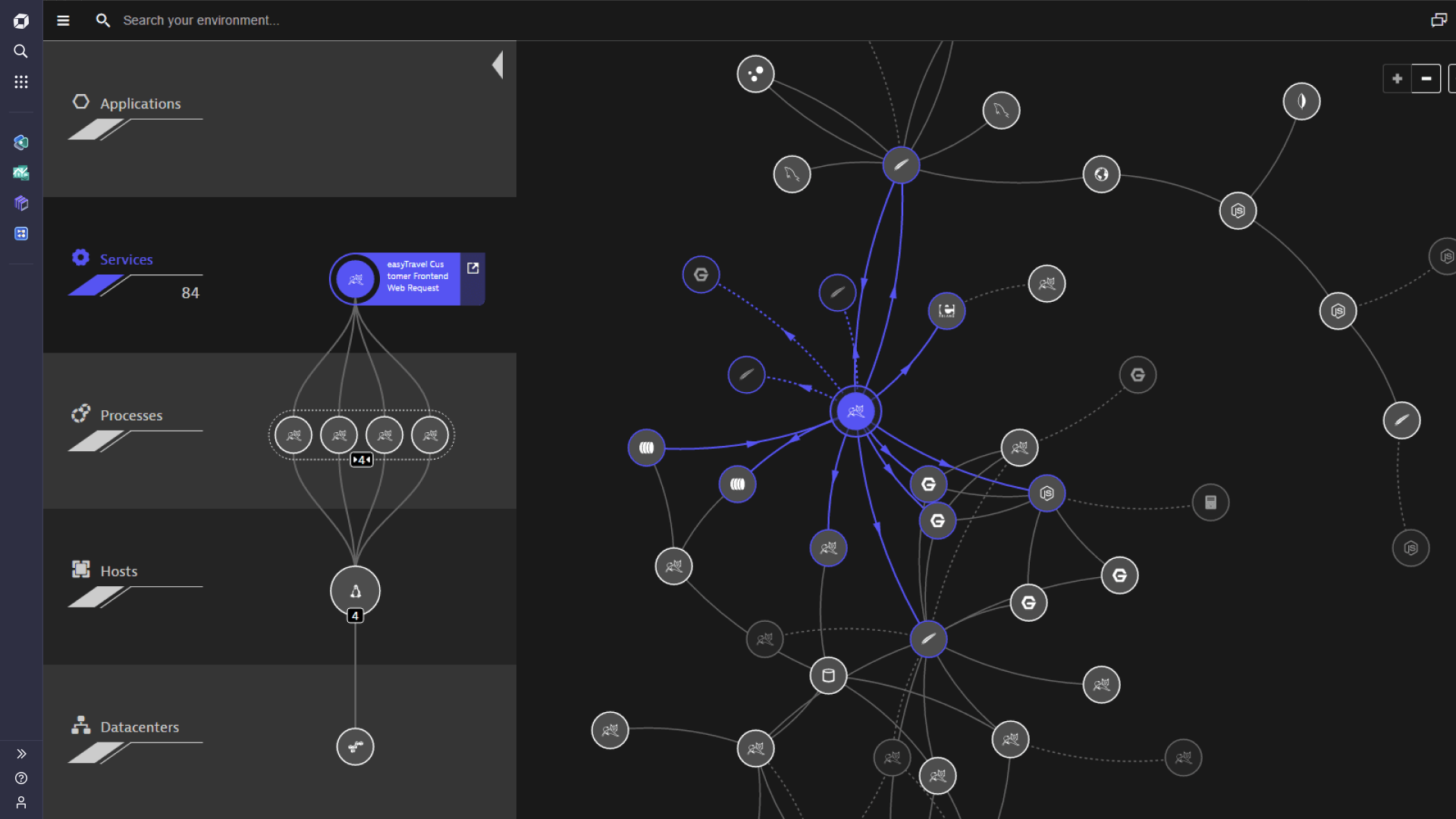
**Visual Representation of Smartscape**

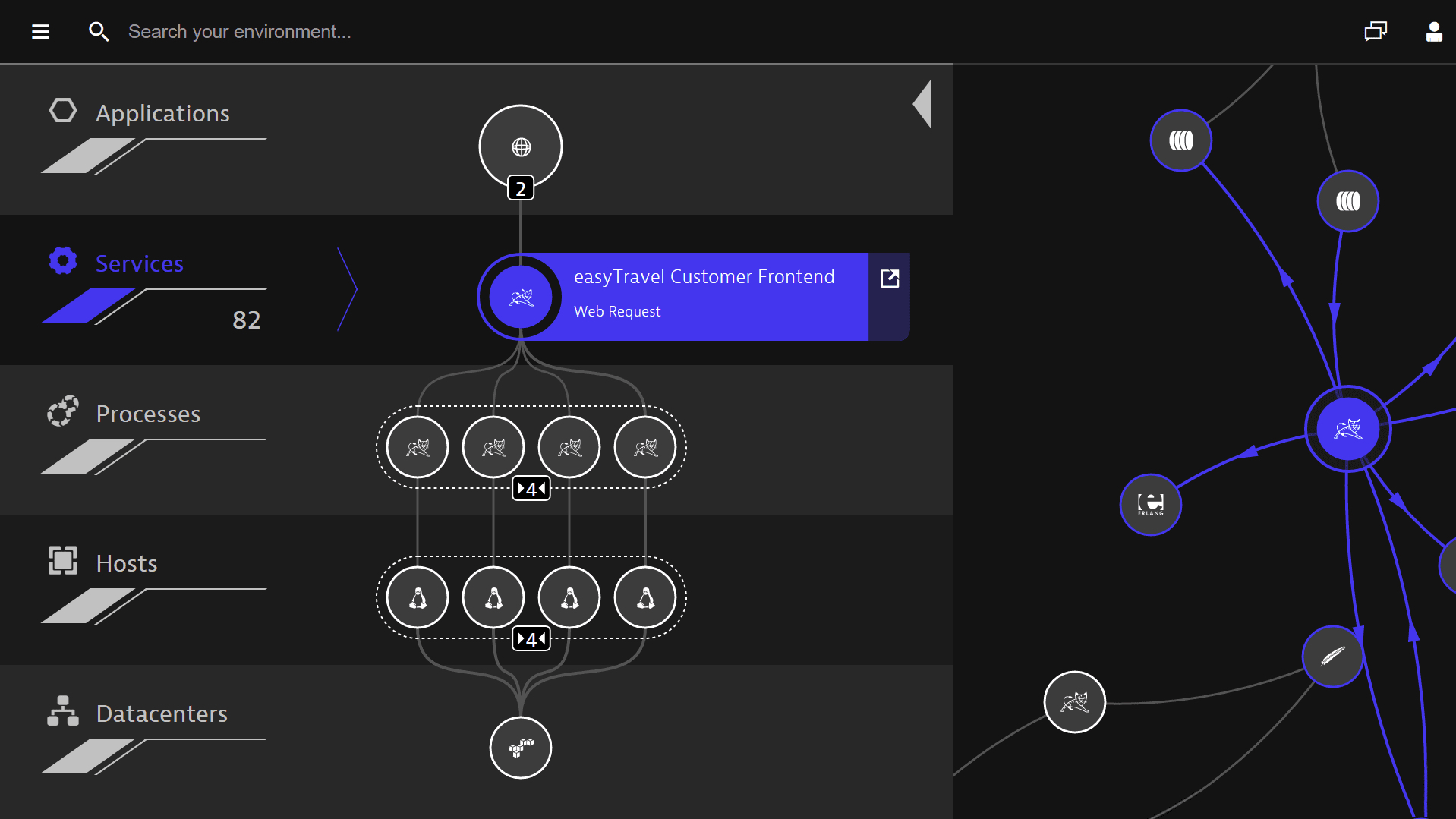
Here are some illustrative diagrams I can create to replicate the **Smartscape views**:

**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+10](https://www.dynatrace.com/platform/service-backtrace/?utm_source=chatgpt.com)[Dynatrace](https://www.dynatrace.com/news/blog/what-is-distributed-tracing/?utm_source=chatgpt.com).
* 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](https://docs.dynatrace.com/docs/license/host-monitoring?utm_source=chatgpt.com).
* **Adaptive Traffic Management** ensures trace sampling remains within your license limits while still giving meaningful coverage [Dynatrace Documentation+1](https://docs.dynatrace.com/docs/ingest-from/dynatrace-oneagent/adaptive-traffic-management?utm_source=chatgpt.com).
* 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](https://docs.dynatrace.com/docs/discover-dynatrace/references/semantic-dictionary/model/trace?utm_source=chatgpt.com).
* 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](https://docs.dynatrace.com/docs/license/host-monitoring?utm_source=chatgpt.com).
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+6](https://www.dynatrace.com/platform/service-backtrace/?utm_source=chatgpt.com)[Stack Overflow+2Reddit+2](https://stackoverflow.com/questions/29704356/how-to-inspect-gwt-rpc-calls-in-dynatrace?utm_source=chatgpt.com)[Dynatrace Community](https://community.dynatrace.com/t5/Alerting/How-to-get-the-stack-traces-for-the-thread-counts-we-see-against/m-p/199693?utm_source=chatgpt.com)[Dynatrace Documentation+2Dynatrace+2](https://docs.dynatrace.com/docs/discover-dynatrace/references/semantic-dictionary/model/trace?utm_source=chatgpt.com).
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+3Dynatrace Documentation+3Dynatrace Documentation+3](https://docs.dynatrace.com/docs/ingest-from/dynatrace-oneagent/adaptive-traffic-management?utm_source=chatgpt.com).

**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](https://www.dynatrace.com/platform/service-backtrace/?utm_source=chatgpt.com)[Dynatrace Documentation](https://docs.dynatrace.com/docs/discover-dynatrace/references/semantic-dictionary/model/trace?utm_source=chatgpt.com).
* This reveals the exact sequence of functions executing along the transaction path.

**Summary**

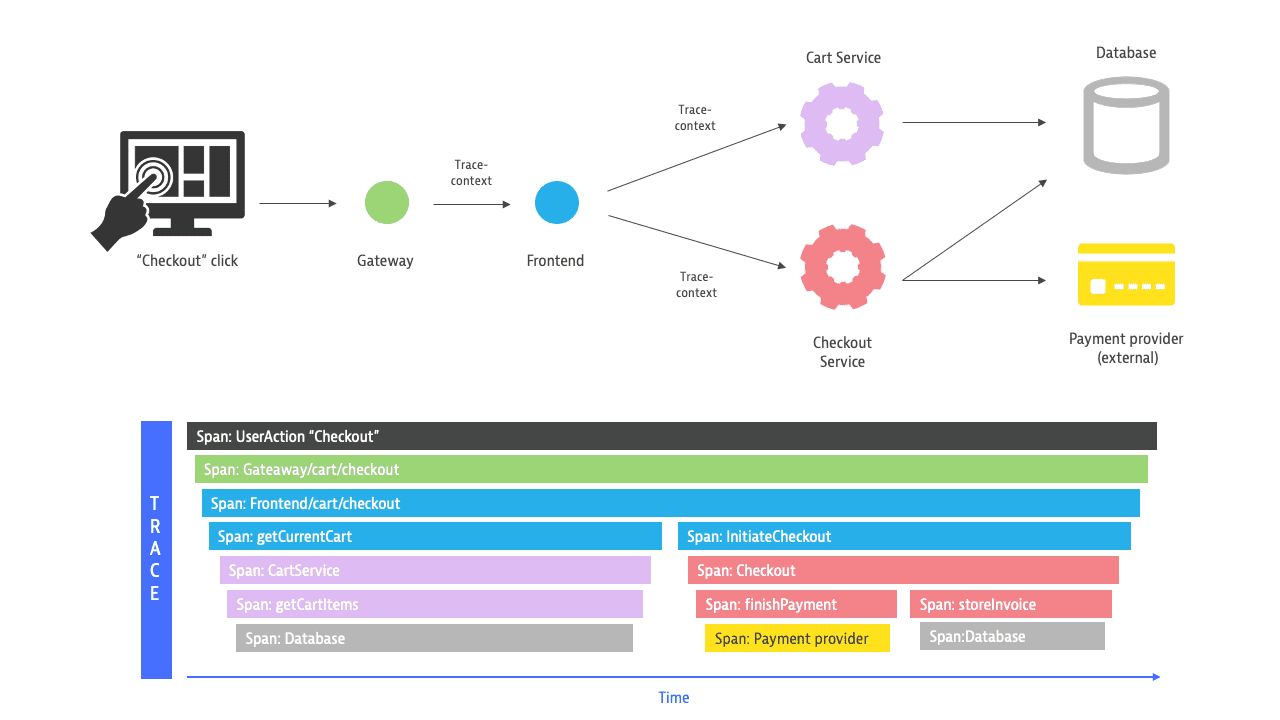
| **Feature** | **Description** |
| --- | --- |
| **PurePath® / Service Backtrace** | Visual “red thread” of call chains from UI click to backend. Drill into method-level details. |
| **Full‑Stack Monitoring** | Auto-instrumentation with OneAgent; captures traces, code-level info, sampling intelligently. |
| **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. |

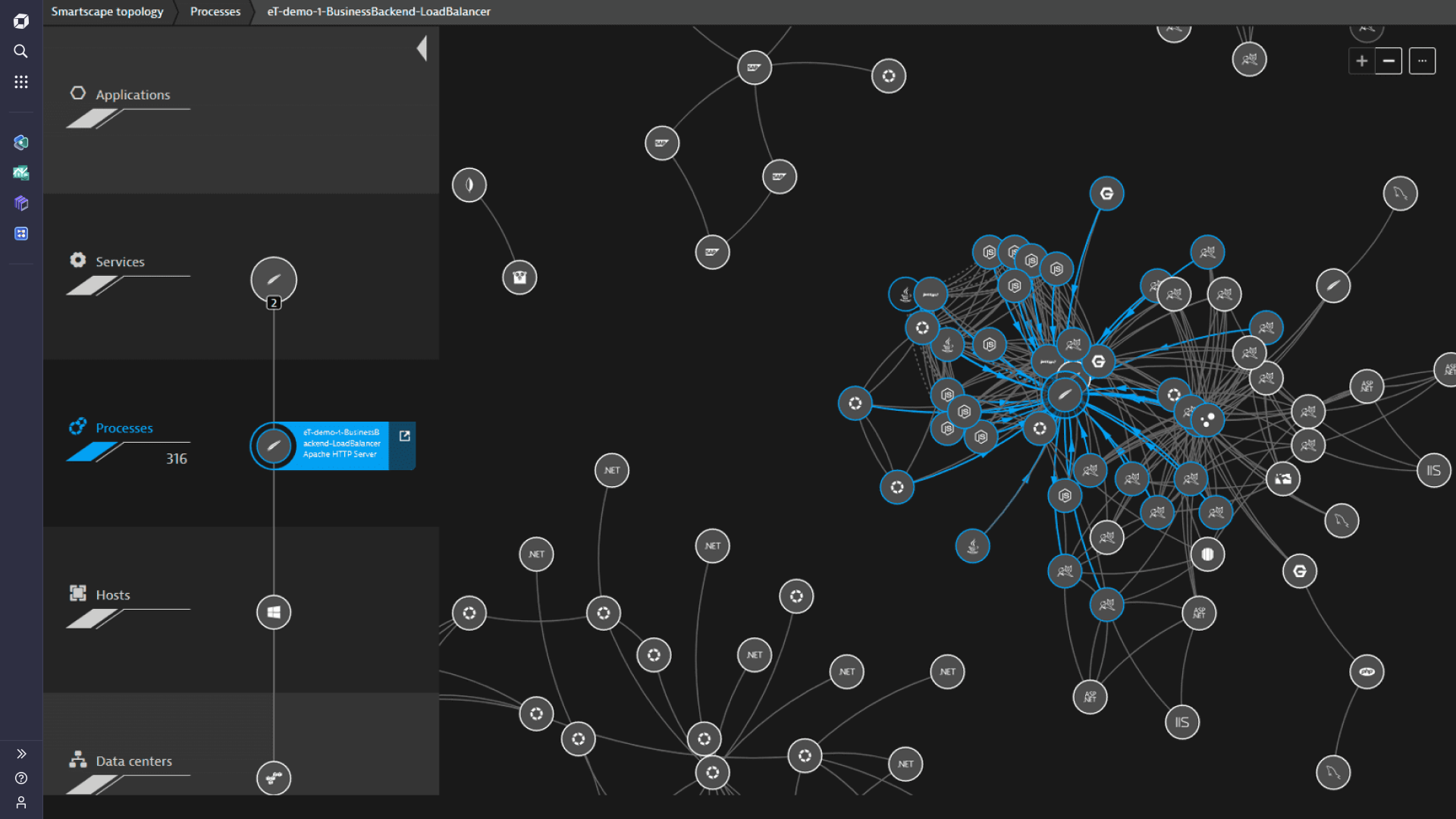
Let me know if you'd like help going through a specific trace in your environment—happy to walk you through the UI or help you interpret call stacks further.

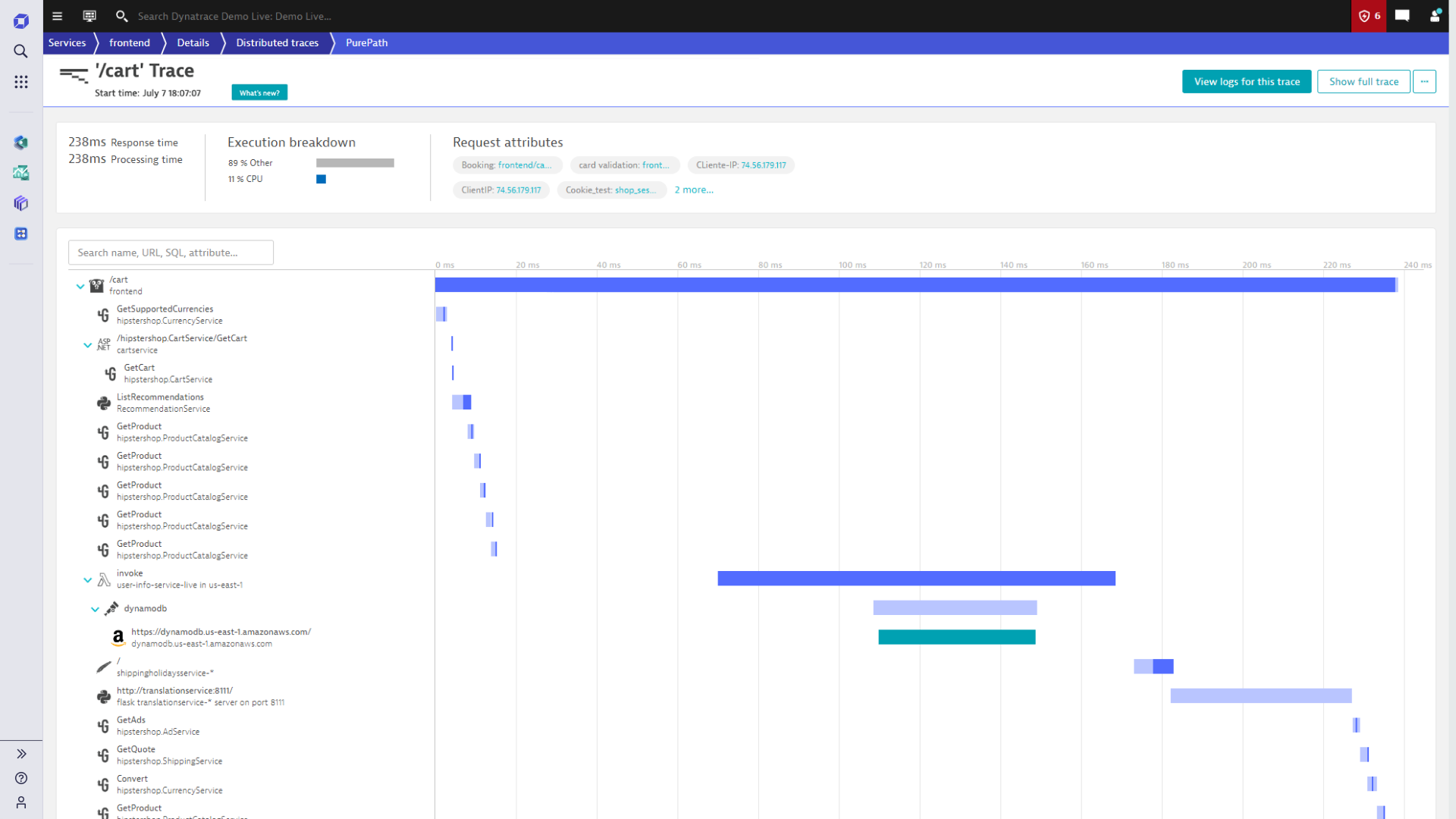
**You said:**

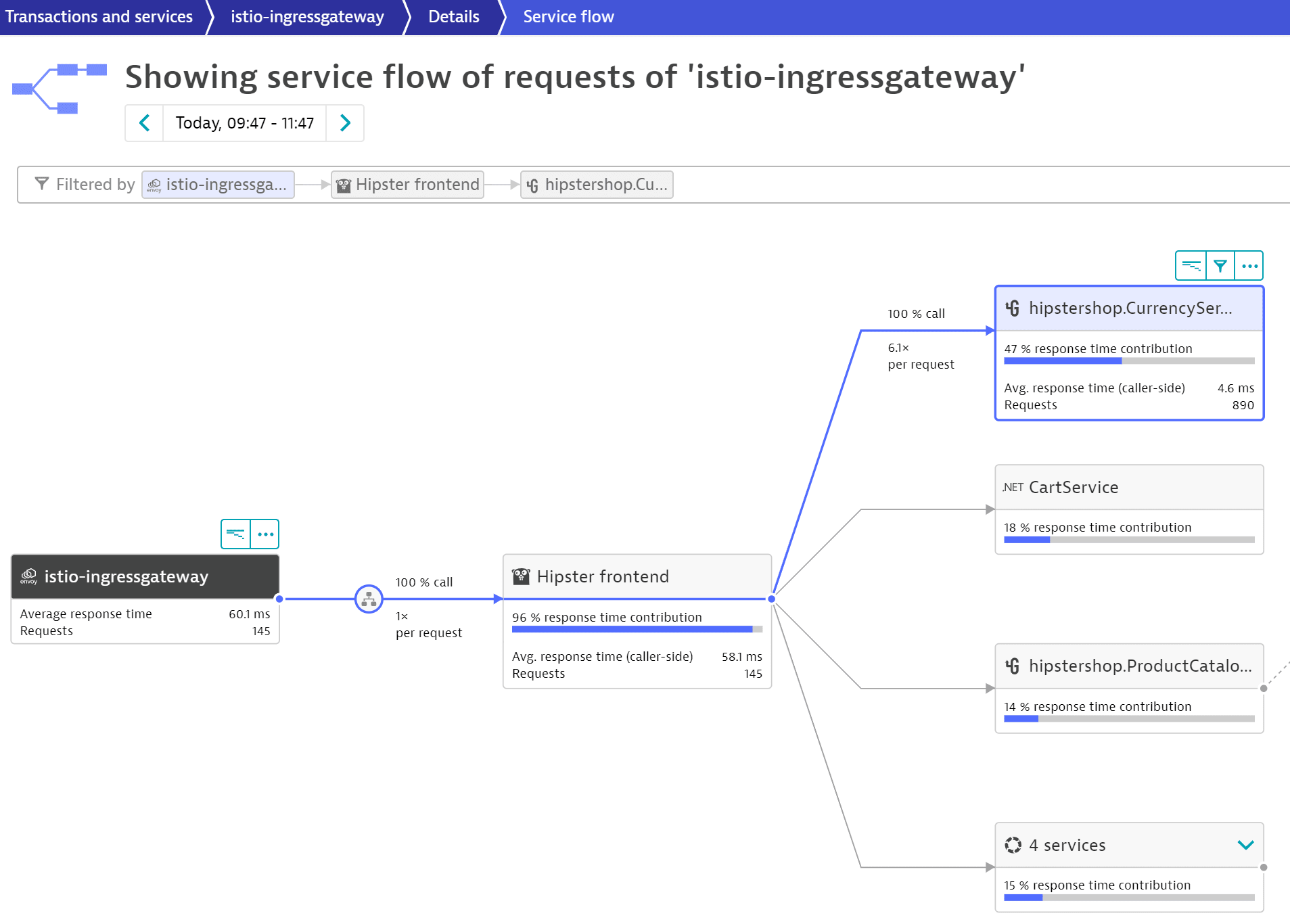
diagram or pictures

**ChatGPT said:**





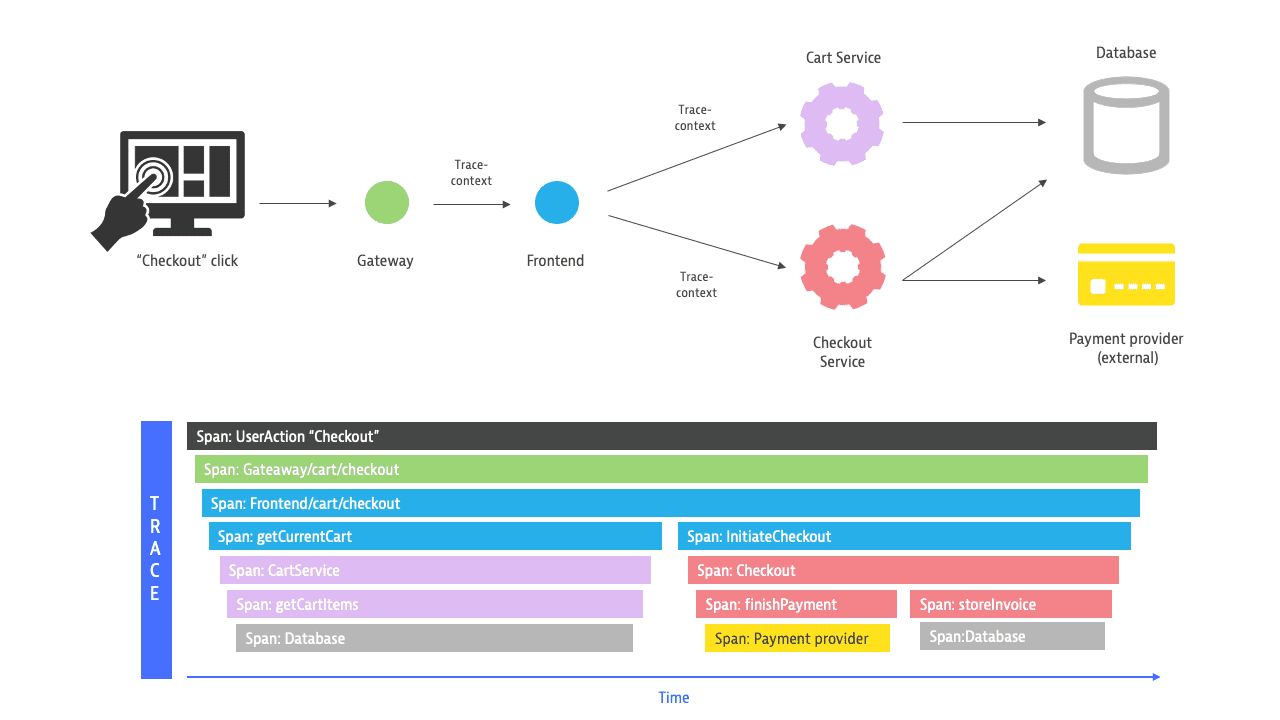




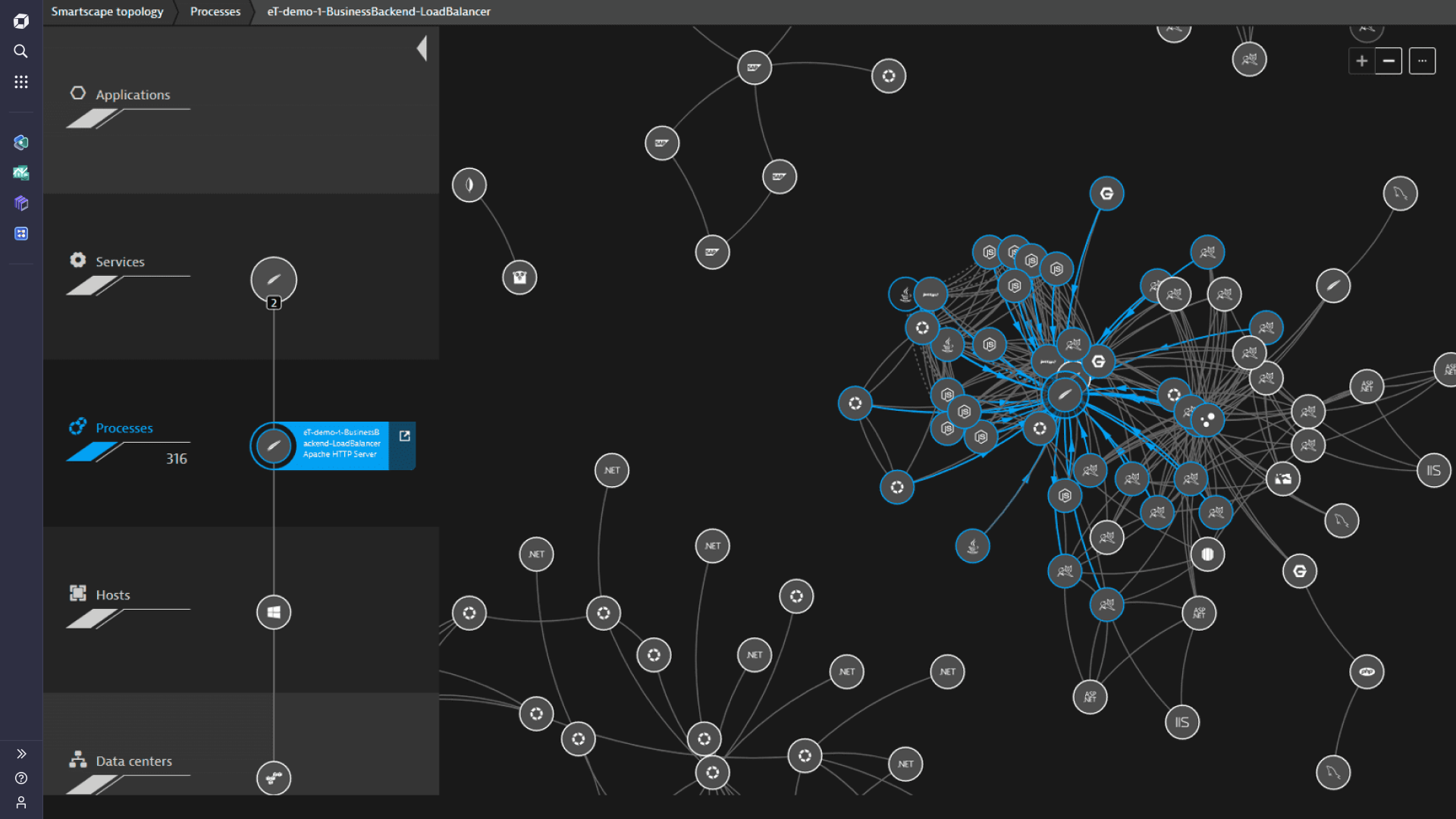
Here are some helpful illustrations from Dynatrace to visually explain how request call stacks and distributed traces (PurePath®) are represented:

**Image Breakdown**

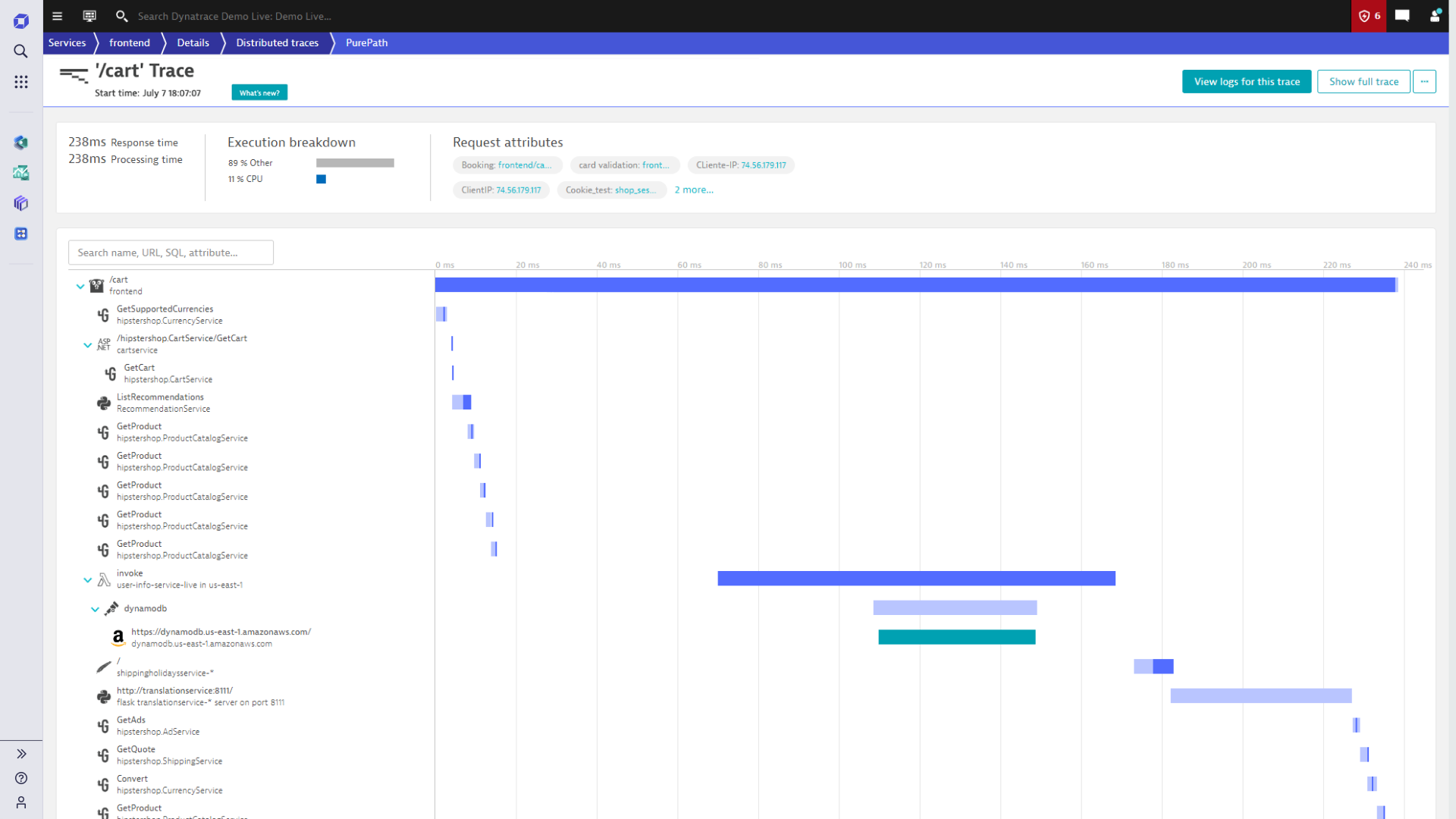
1. **Distributed Trace & Span Anatomy (1st image)**  
   Shows how a single request (trace) spans multiple services, broken down into spans—each capturing service-level timing and context.



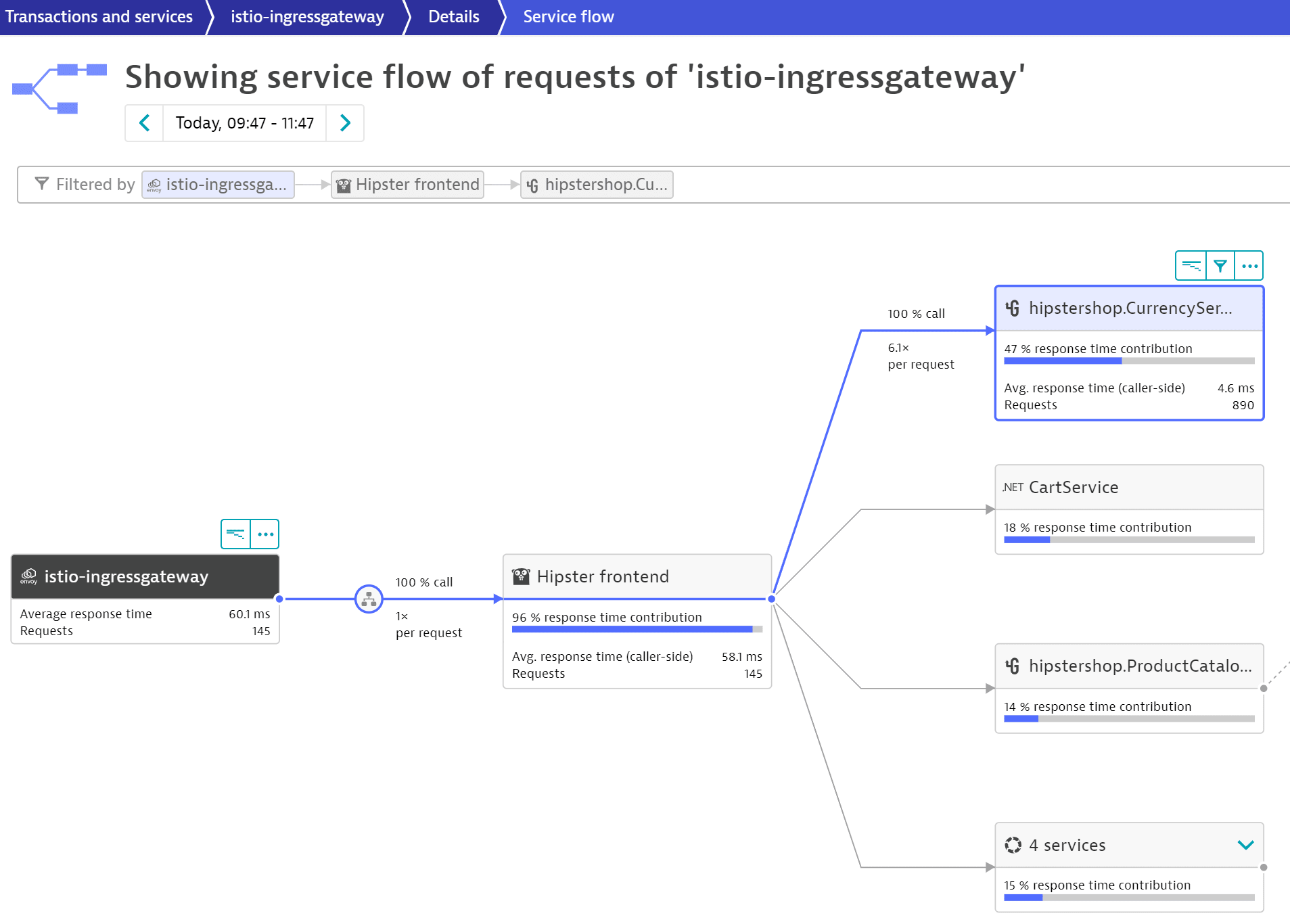
1. **PurePath Trace Visualization (2nd image)**  
   Demonstrates a call flow or “service map” representing how a PurePath traverses through services and microservices, ideal for understanding call patterns.



1. **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.



1. **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

Code-Level Visibility:  
- Dynatrace captures detailed trace data with method-level granularity.  
- Supports frameworks like Spring, ASP.NET, Hibernate, etc.

Method Hotspots & CPU Profiling:  
- Identify methods with the highest execution time or CPU usage.  
- Lightweight profiling can be enabled for live environments.

Web Request Tracing & Service Flow:  
- Visualizes end-to-end request flow across services.  
- Helps identify slow service calls, retry storms, or database delays.

Messaging Queue Monitoring (JMX/Java):  
- Captures queue size, consumer lag, and throughput using JMX.  
- Supports Kafka, RabbitMQ, ActiveMQ, and other brokers.

# 13. Real User Monitoring (RUM) Overview

JavaScript Injection Model:  
- Dynatrace automatically injects a JavaScript tag into monitored web applications.  
- Enables collection of client-side performance data.

RUM vs. Synthetic:  
- RUM shows real user behavior; synthetic is scripted and scheduled.  
- RUM is continuous; synthetic is proactive.

Benefits and Visibility Scope:  
- Measures user experience with KPIs like visually complete and TTI.  
- Tracks device, browser, geographic data.  
- Detects JavaScript errors and slow third-party resources.

# 14. RUM – Web Applications

Page Performance Metrics:  
- Key metrics: Visually Complete, Time to Interactive (TTI), Load Time.  
- Measured per page, per user, and per location.

JavaScript Errors & 3rd-Party Content:  
- Captures JS errors with stack traces.  
- Identifies performance issues from third-party services (ads, CDN).

Geo/User/Device Breakdowns:  
- Filters available for location, browser, device type.  
- Enables segmentation and targeted optimization.

# 15. RUM – User Sessions

Session & Action Analysis:  
- Tracks all user actions: clicks, scrolls, form submissions.  
- Sessions can be viewed chronologically or by impact.

Behavior Metrics & Conversion Tracking:  
- Funnel analysis helps visualize drop-offs.  
- Conversion goals can be defined for performance optimization.

Funnel and Crash Analytics:  
- Shows where users abandon due to slowness or errors.  
- Crash reports include stack trace and session context.

# 16. RUM – Mobile Applications

Mobile Agent SDK Integration (iOS, Android, tvOS):  
- SDKs can be embedded into mobile apps.  
- Supports iOS (Swift/Obj-C), Android (Java/Kotlin), tvOS.

Crash Analysis, Gestures, Custom Actions:  
- Detects crashes with logs and device info.  
- Tracks taps, swipes, and other gestures.  
- Allows custom tagging of events (e.g., purchase complete).