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)

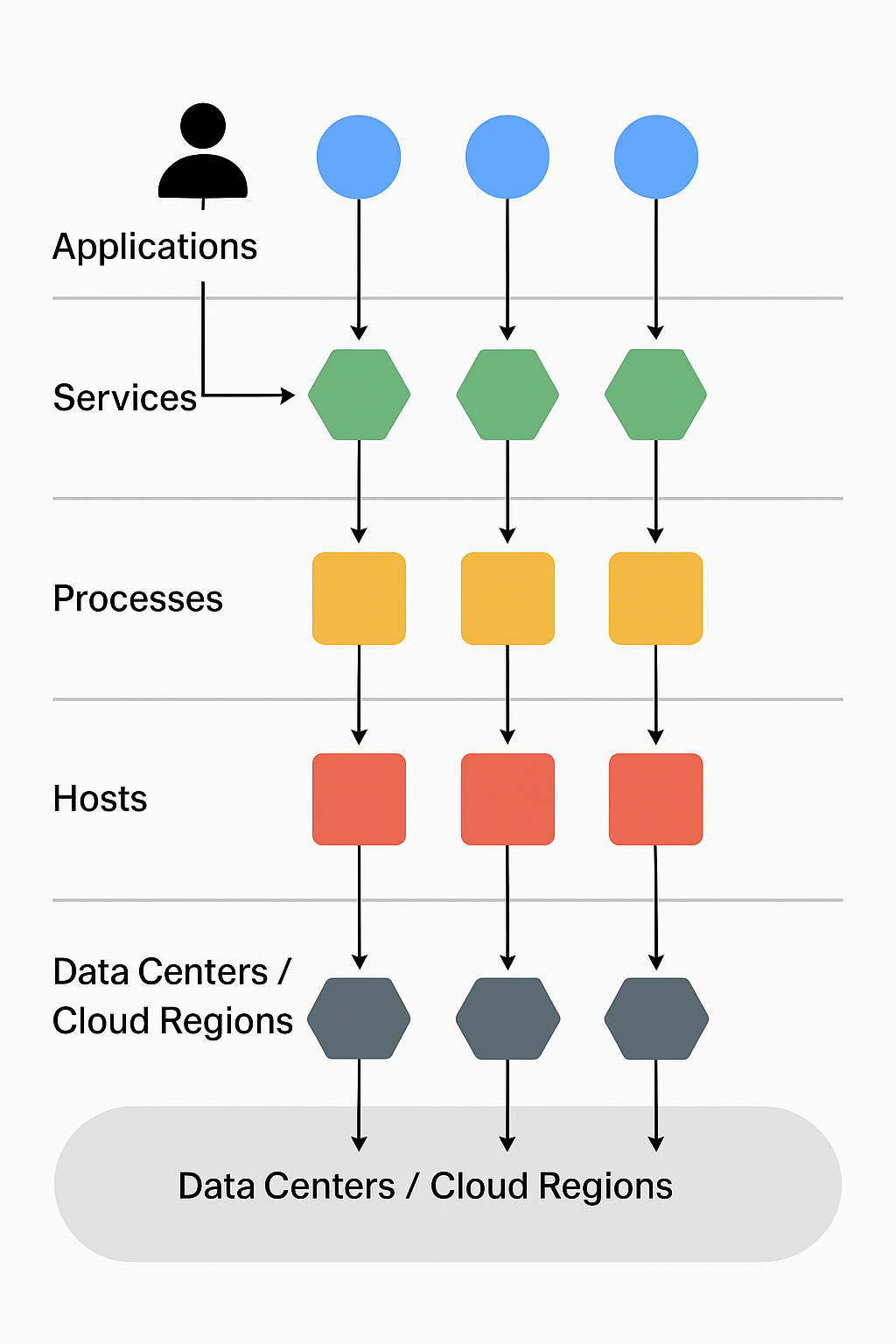
**Key Benefits of Smartscape**

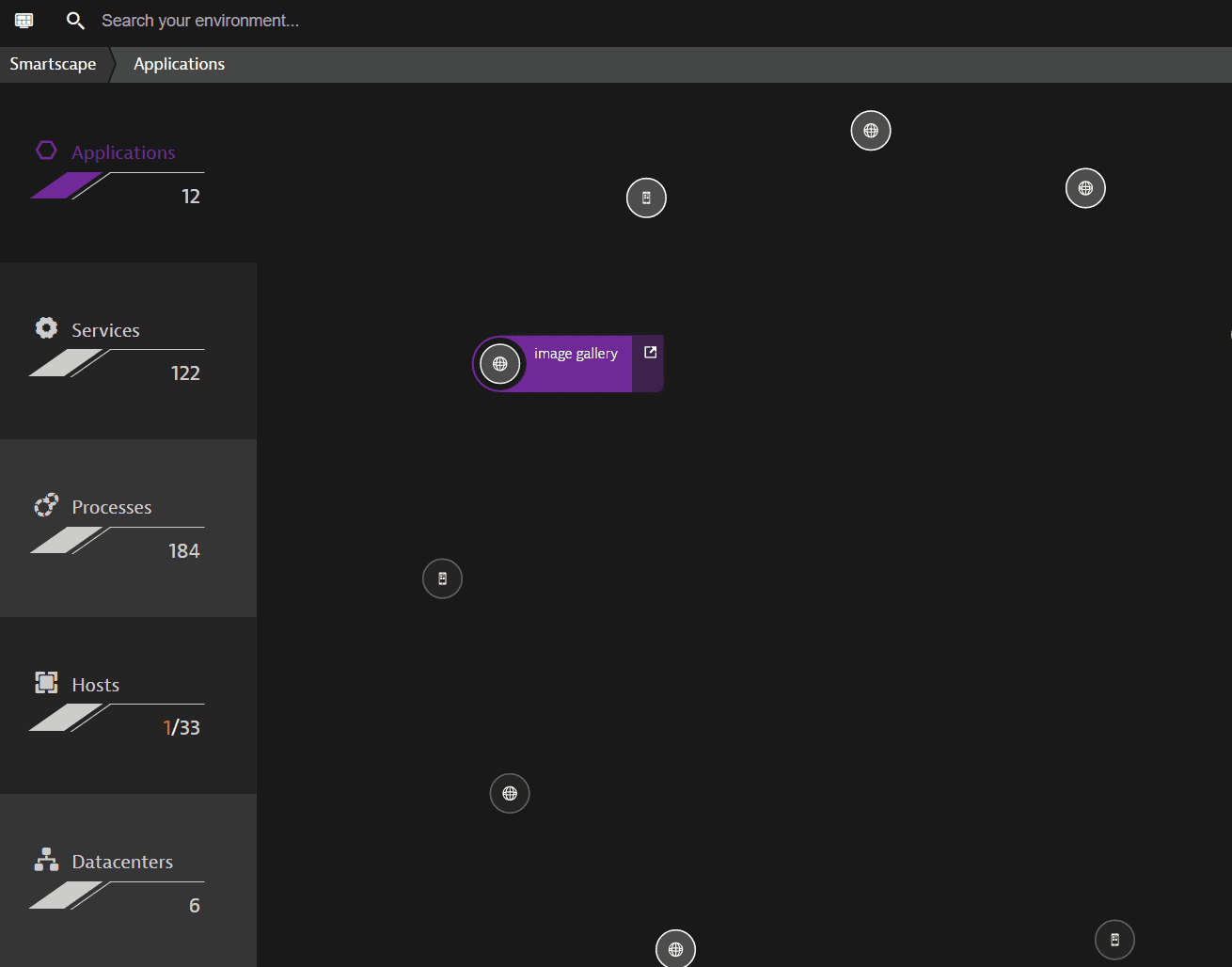
* **End-to-end visibility**: Full topology from users down to infrastructure.
* **Auto-updating**: Topology refreshes in near real-time when entities are added/removed.
* **Dependency mapping**: Shows inbound/outbound service calls.
* **Problem correlation**: Davis AI uses Smartscape to understand root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

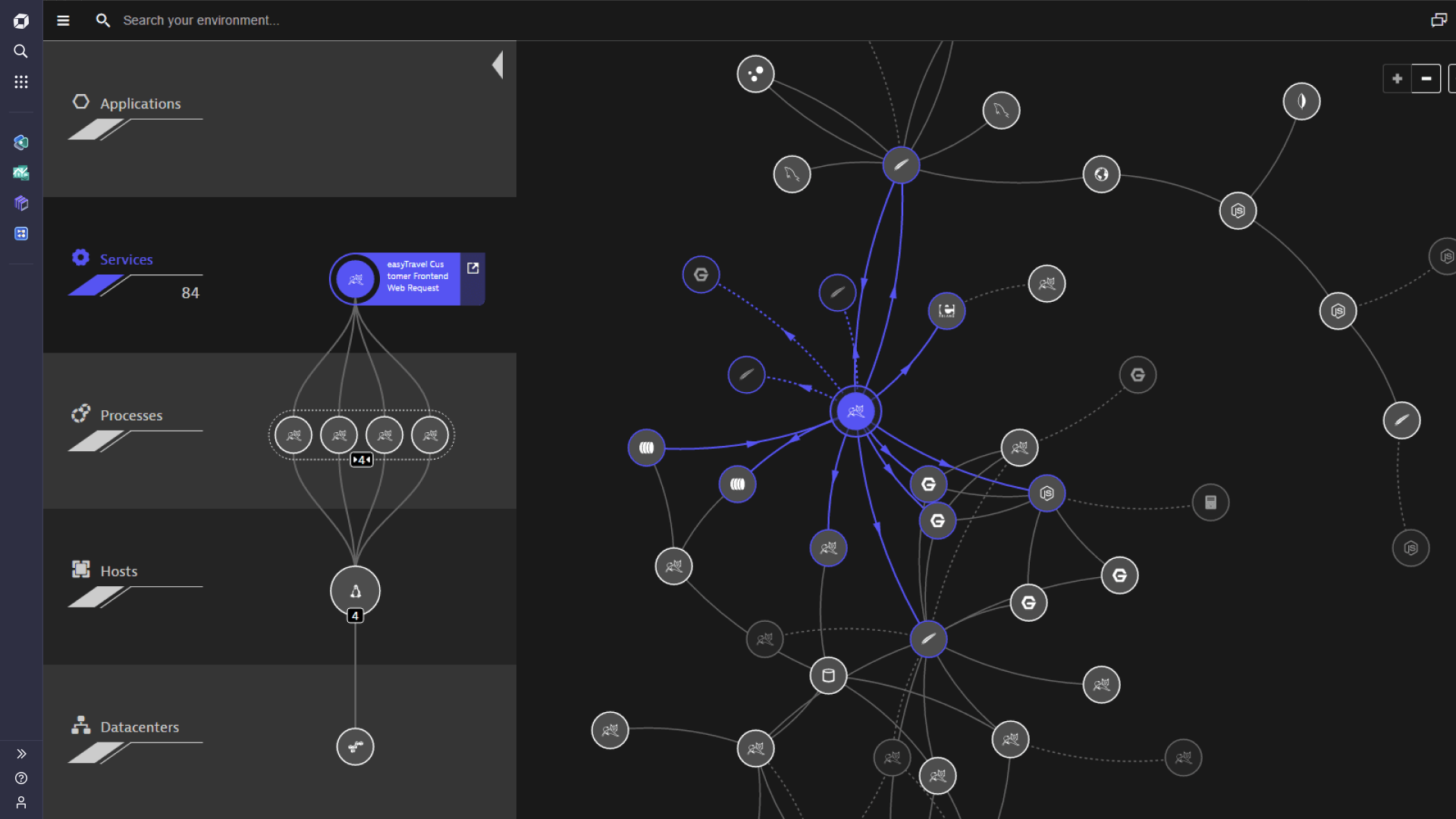
**Visual Representation of Smartscape**

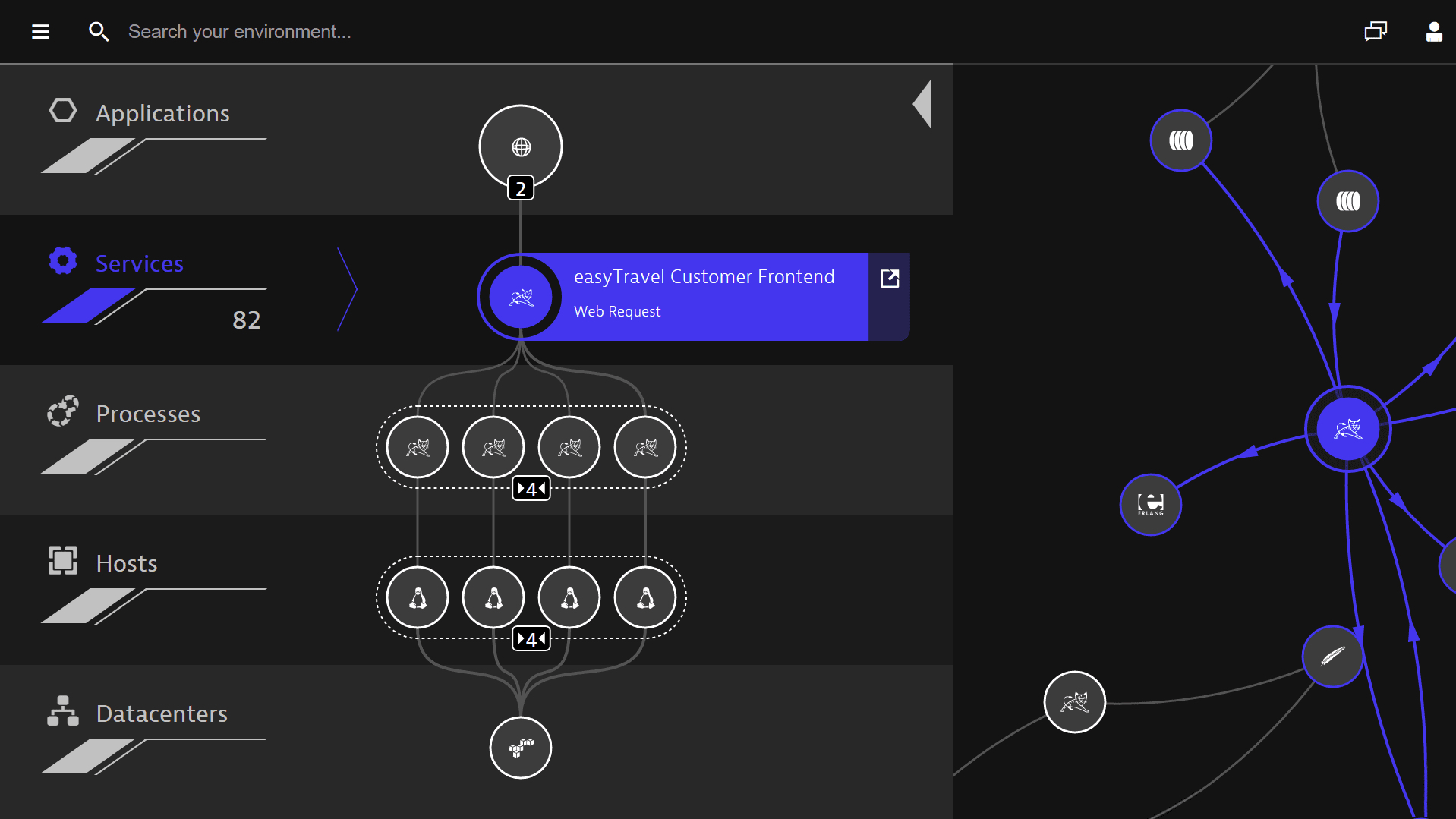
Here are some illustrative diagrams I can create to replicate the **Smartscape views**:

**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**

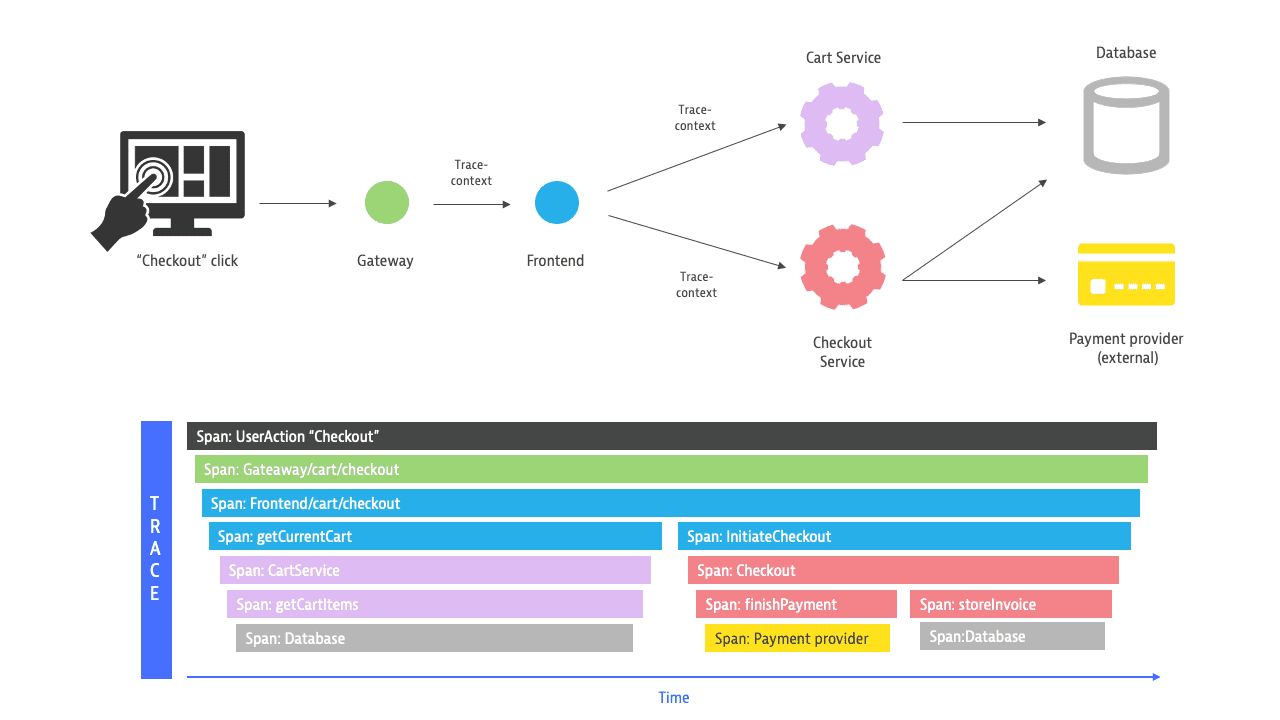
| **Feature** | **Description** |
| --- | --- |
| **PurePath® / Service Backtrace** | Visual “red thread” of call chains from UI click to backend. Drill into method-level details. |
| **Full‑Stack Monitoring** | Auto-instrumentation with OneAgent; captures traces, code-level info, sampling intelligently.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

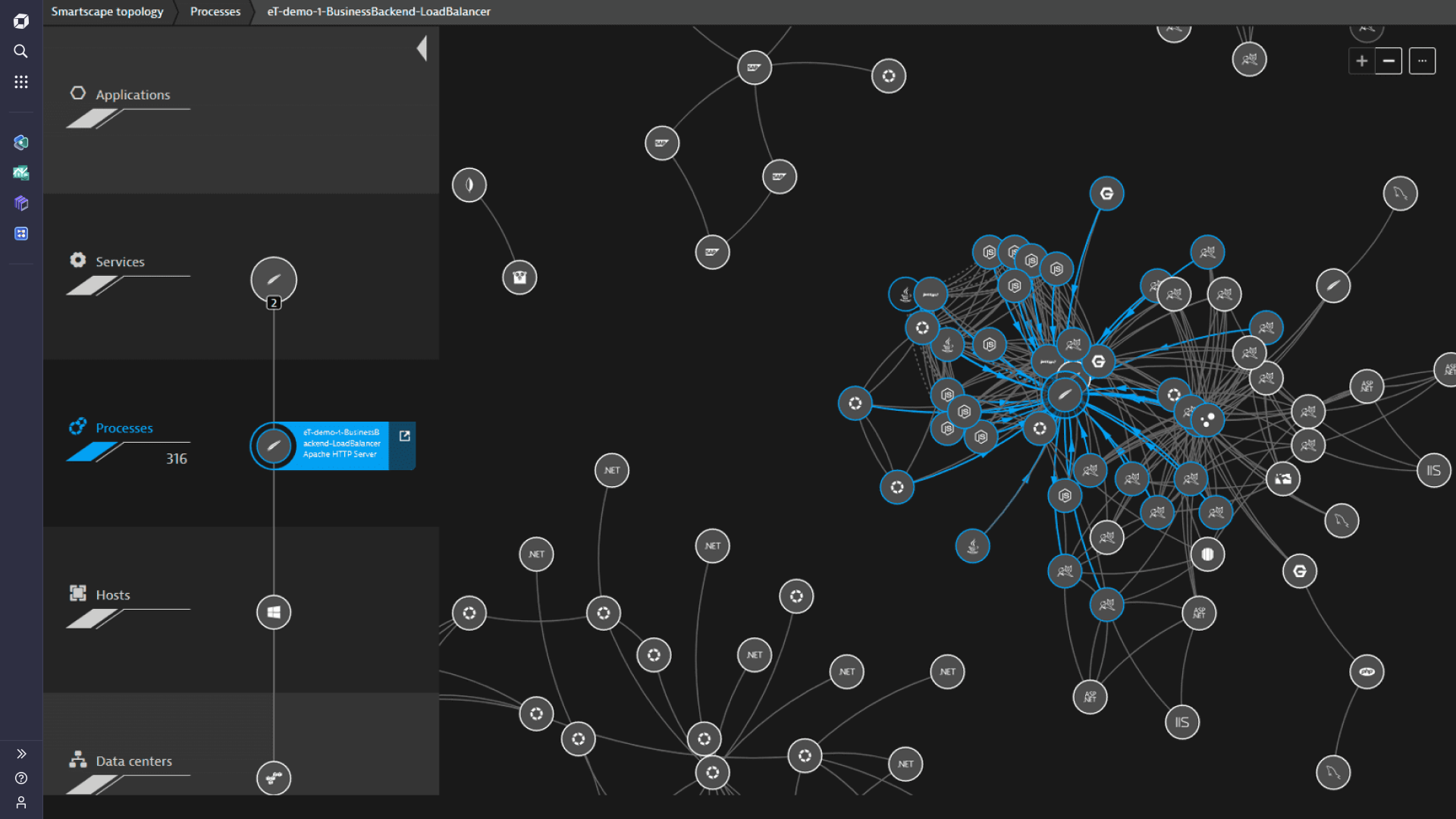
Let me know if you'd like help going through a specific trace in your environment—happy to walk you through the UI or help you interpret call stacks further.

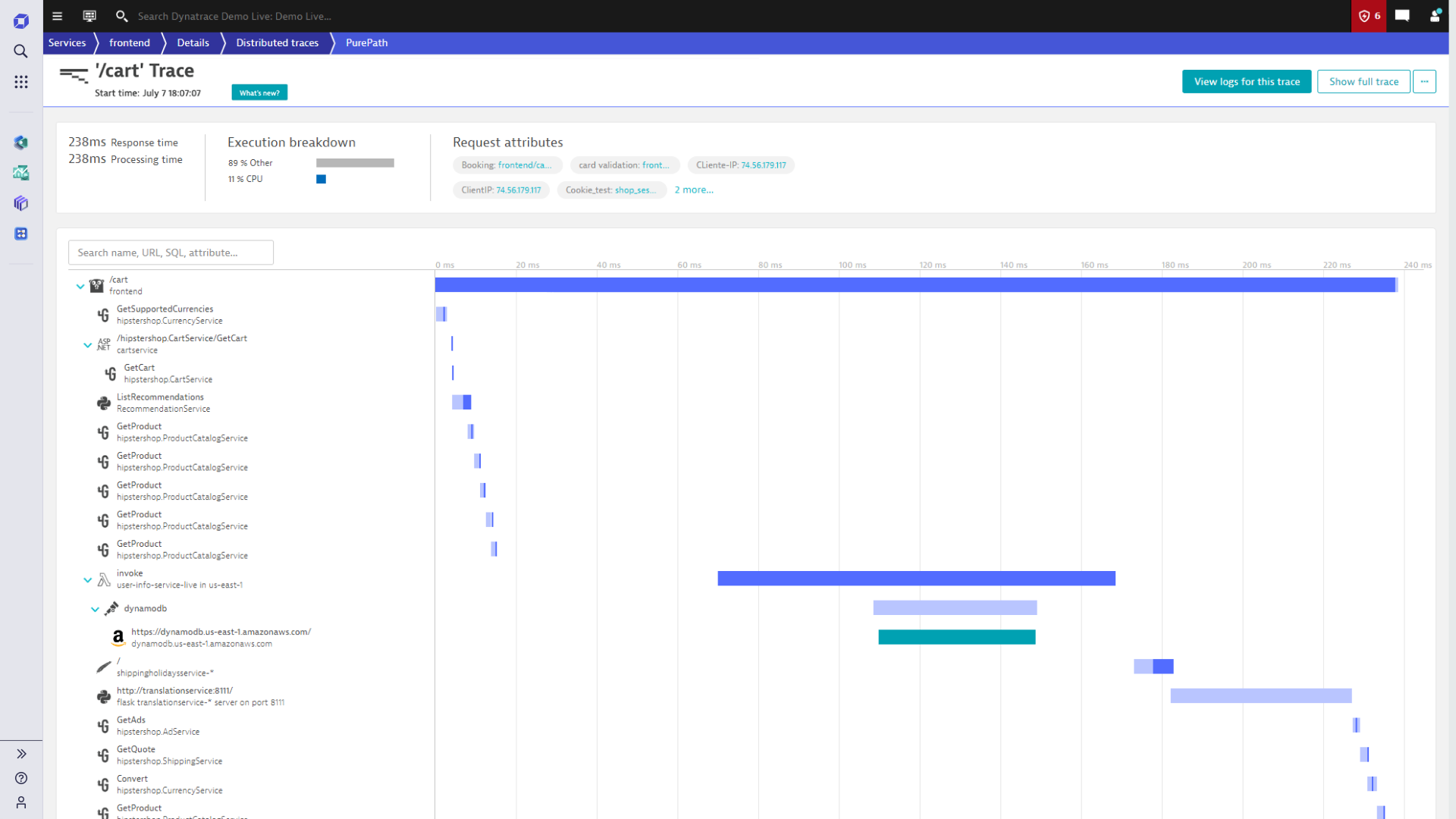
**You said:**

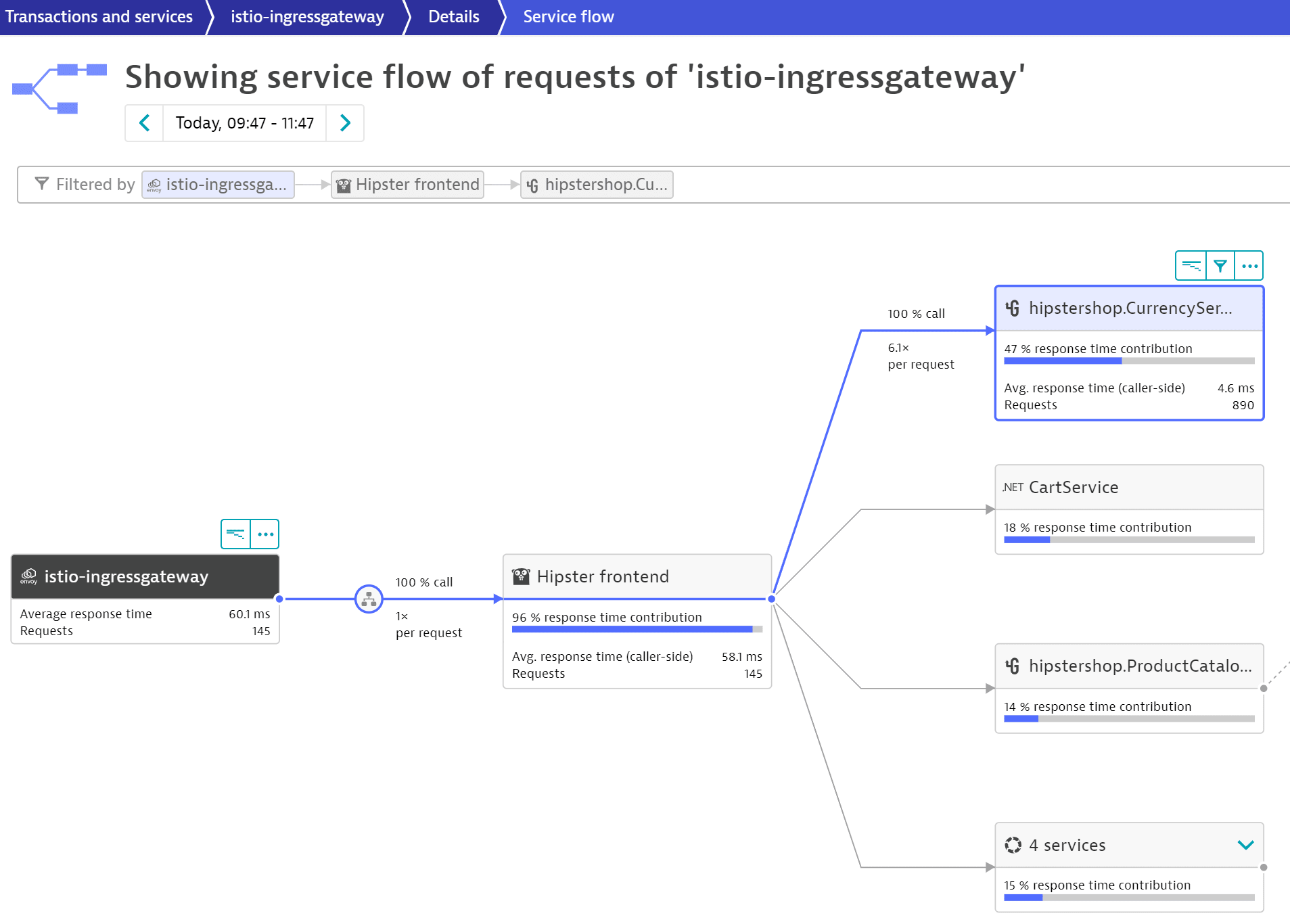
diagram or pictures

**ChatGPT said:**





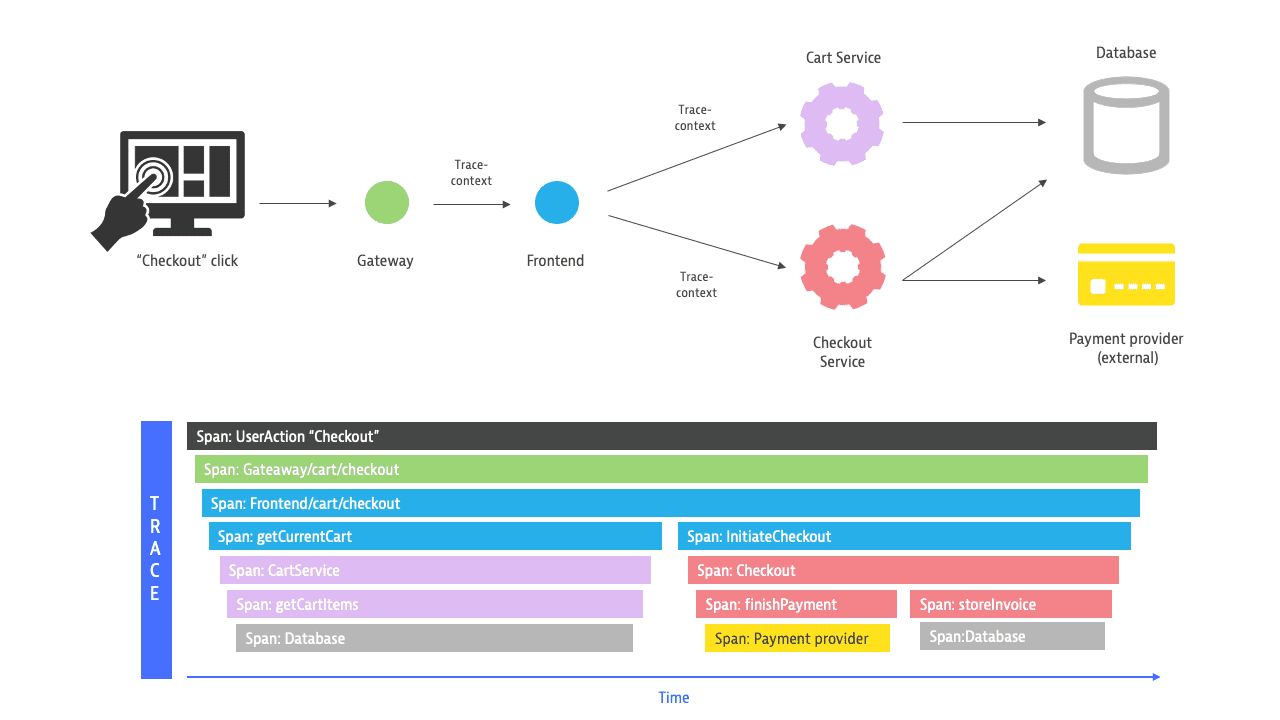




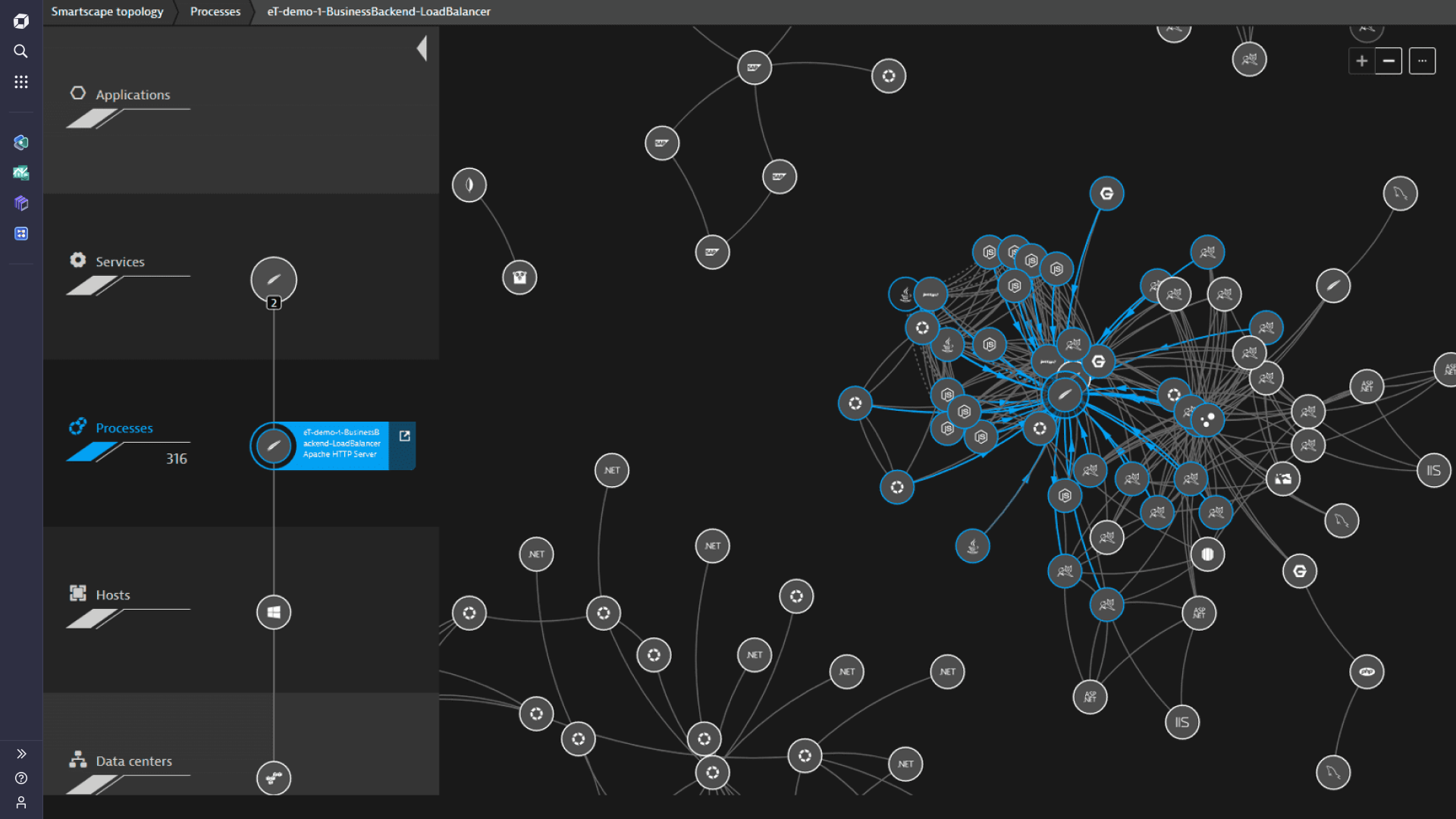
Here are some helpful illustrations from Dynatrace to visually explain how request call stacks and distributed traces (PurePath®) are represented:

**Image Breakdown**

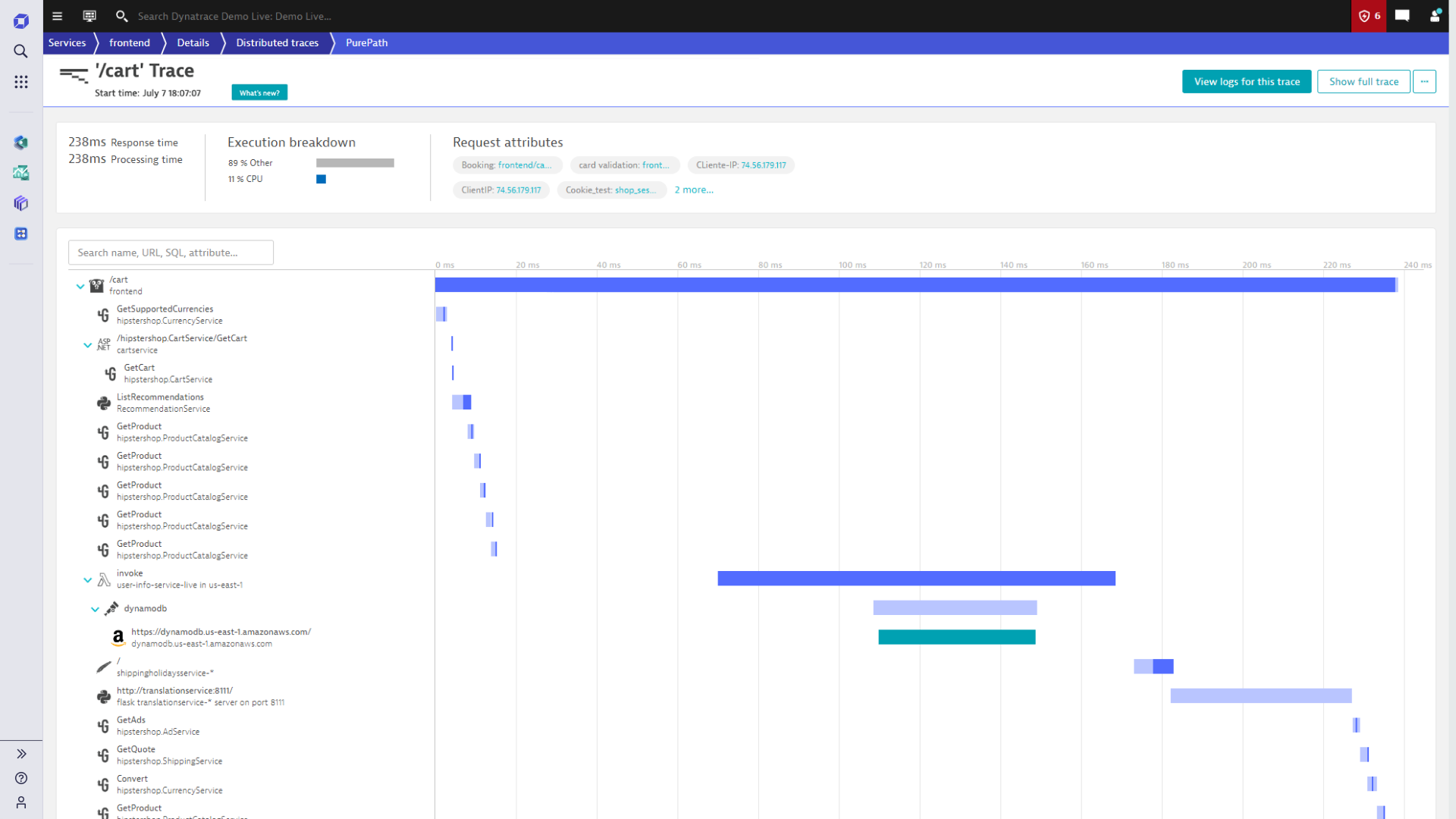
1. **Distributed Trace & Span Anatomy (1st image)**  
   Shows how a single request (trace) spans multiple services, broken down into spans—each capturing service-level timing and context.



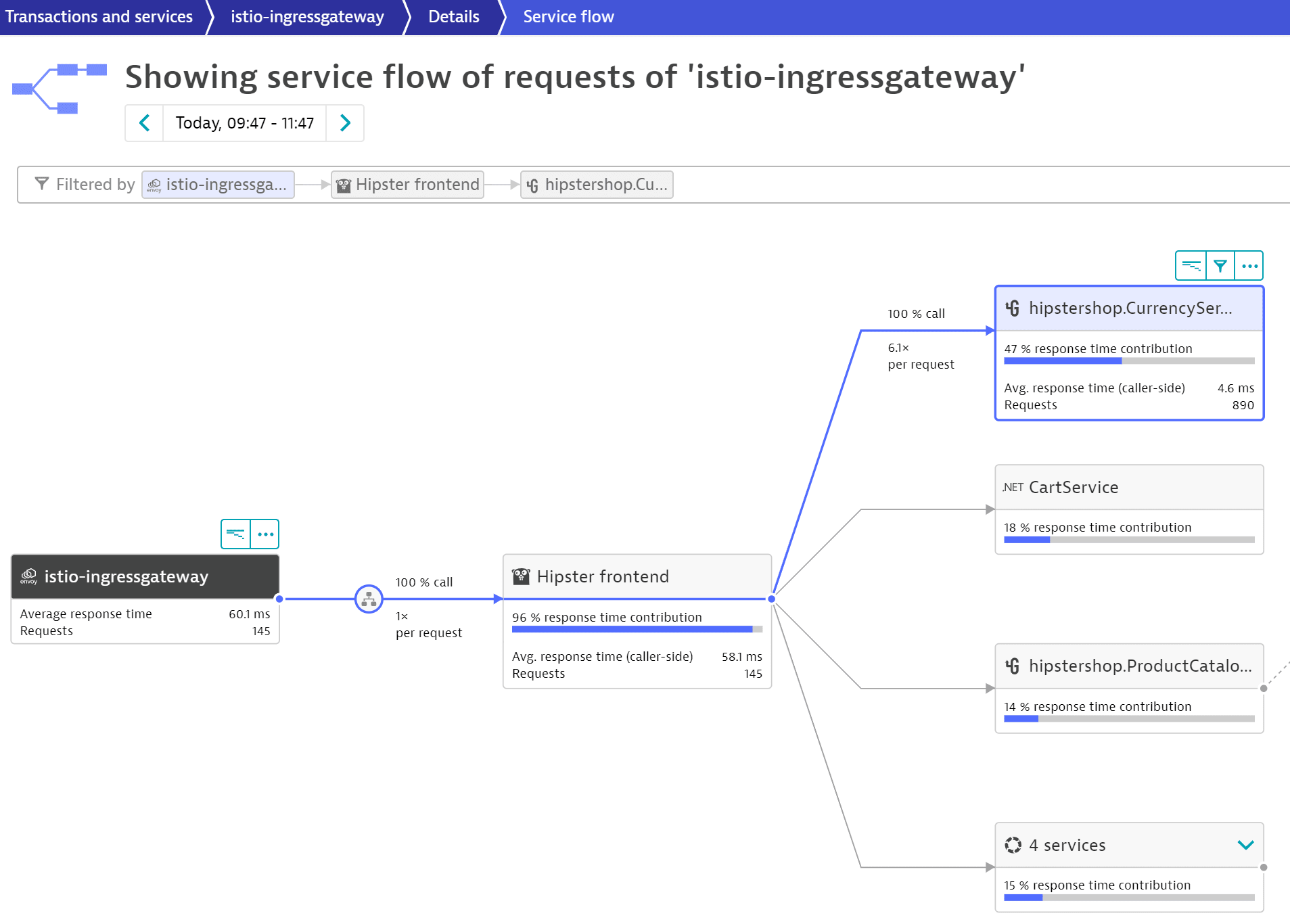
1. **PurePath Trace Visualization (2nd image)**  
   Demonstrates a call flow or “service map” representing how a PurePath traverses through services and microservices, ideal for understanding call patterns.



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

**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

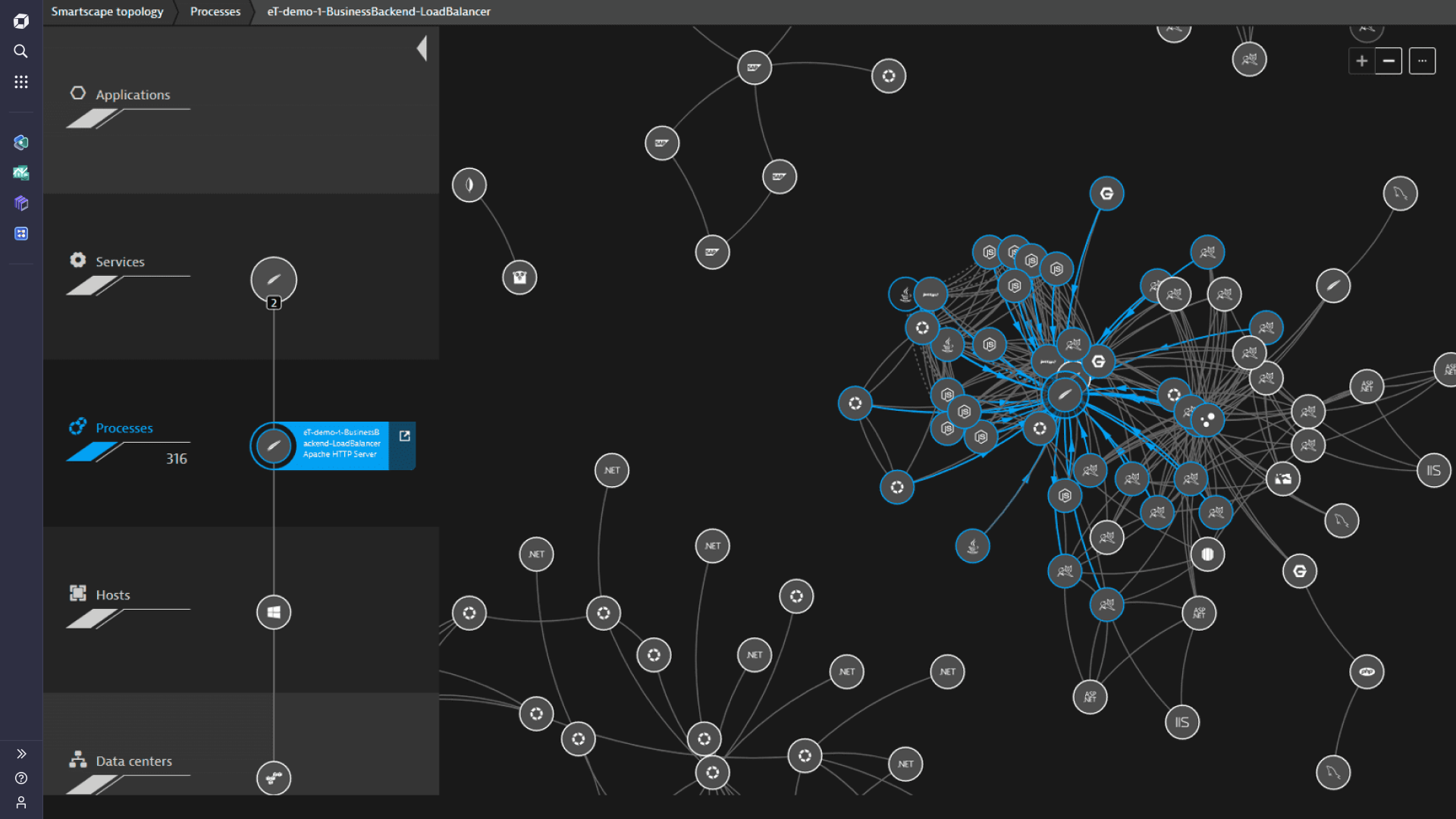
**Use Cases**

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

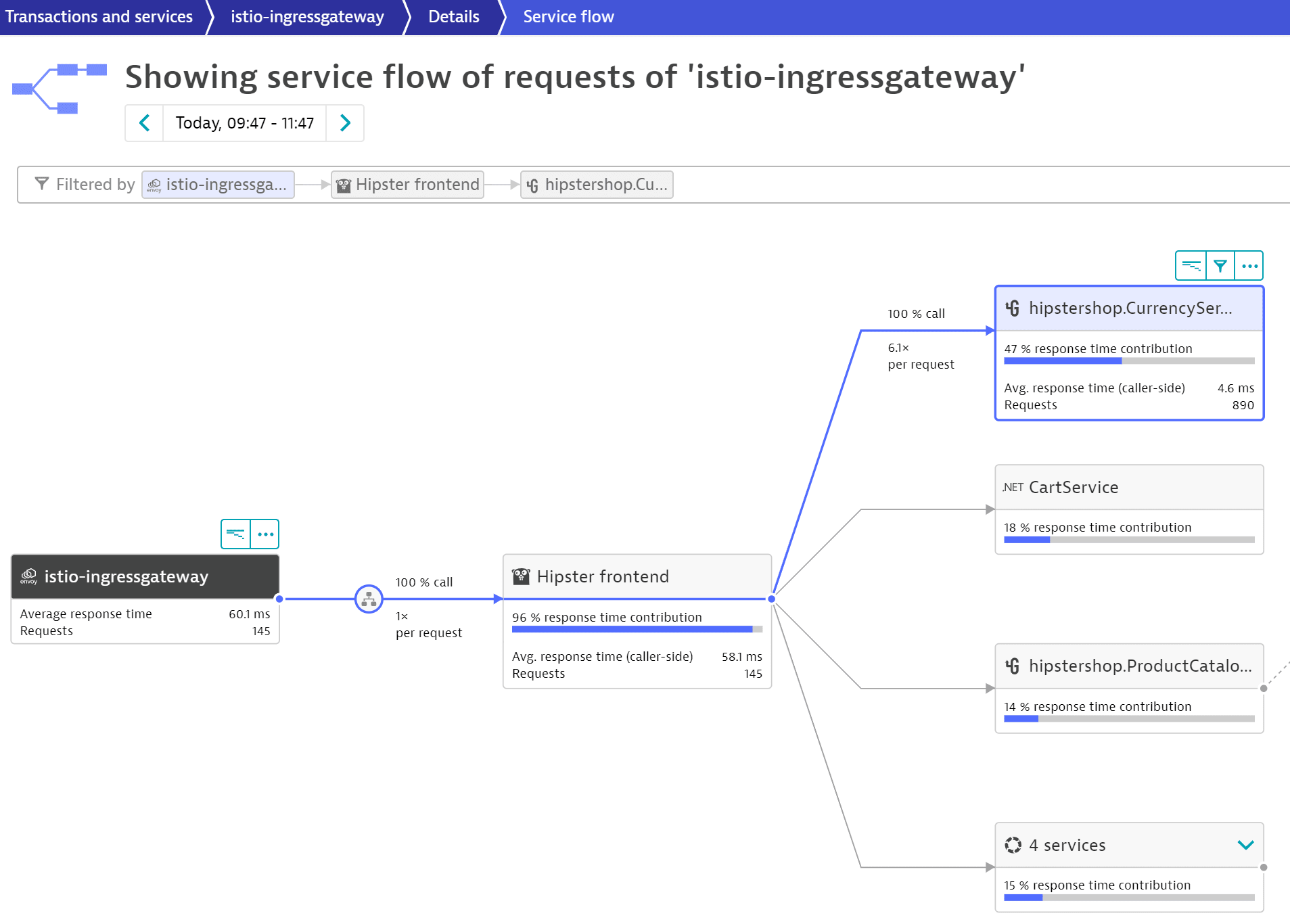
#### 1. ****Next-Gen PurePath Visualization****

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.



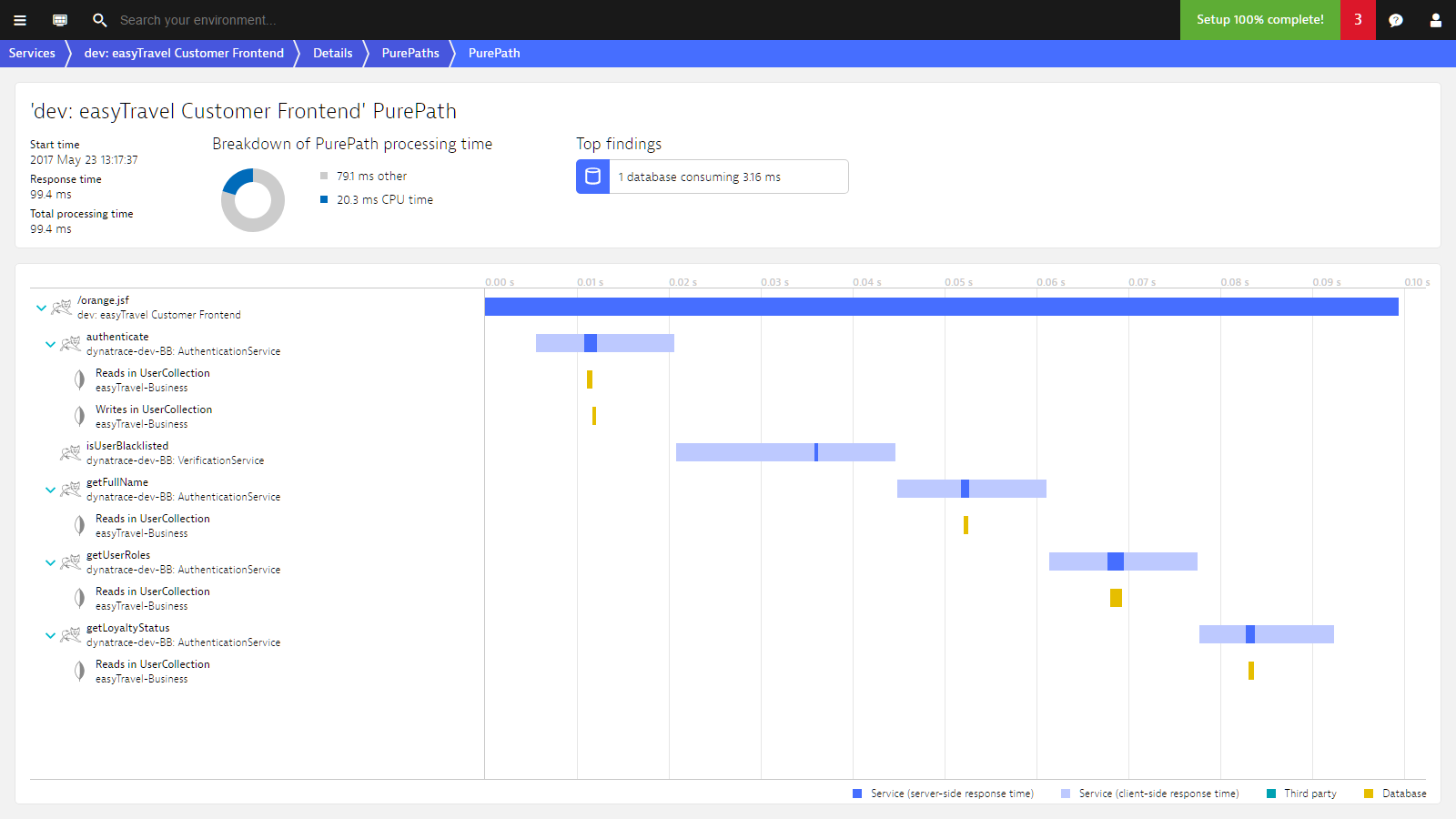
#### 2. ****Service Flow & Trace Mapping****

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.



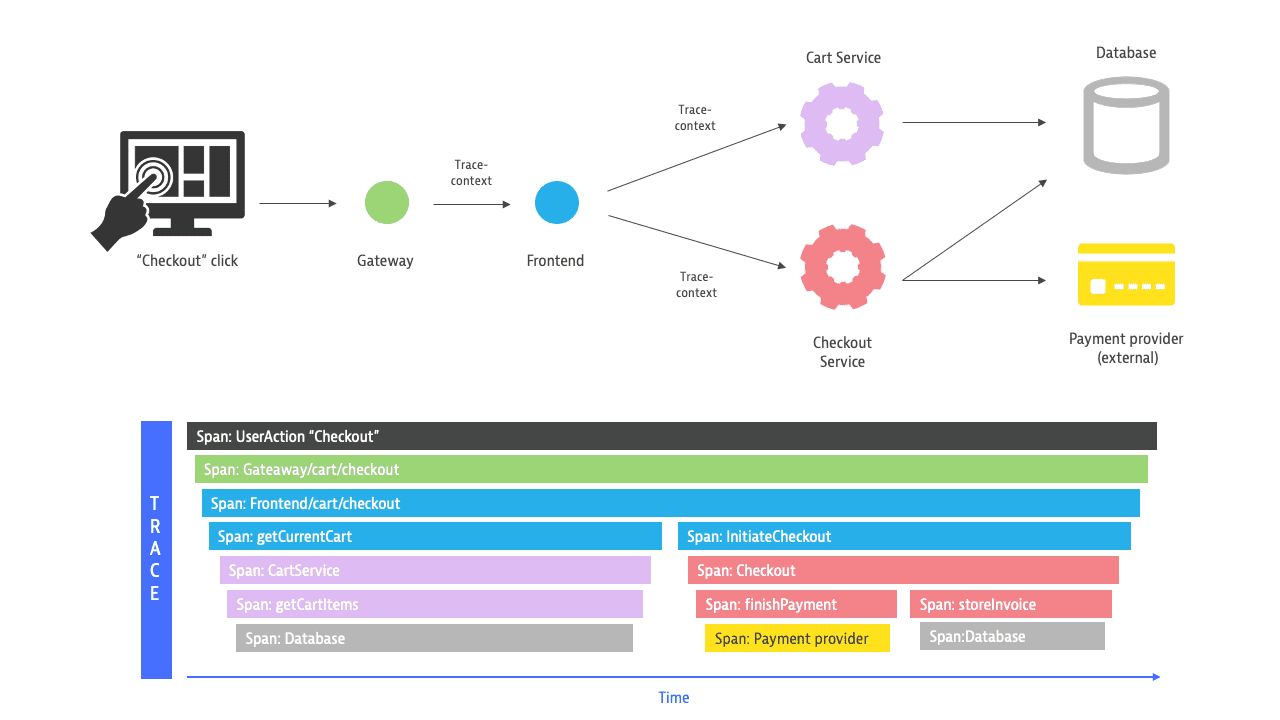
#### 3. ****PurePath Waterfall with Filtering****

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”

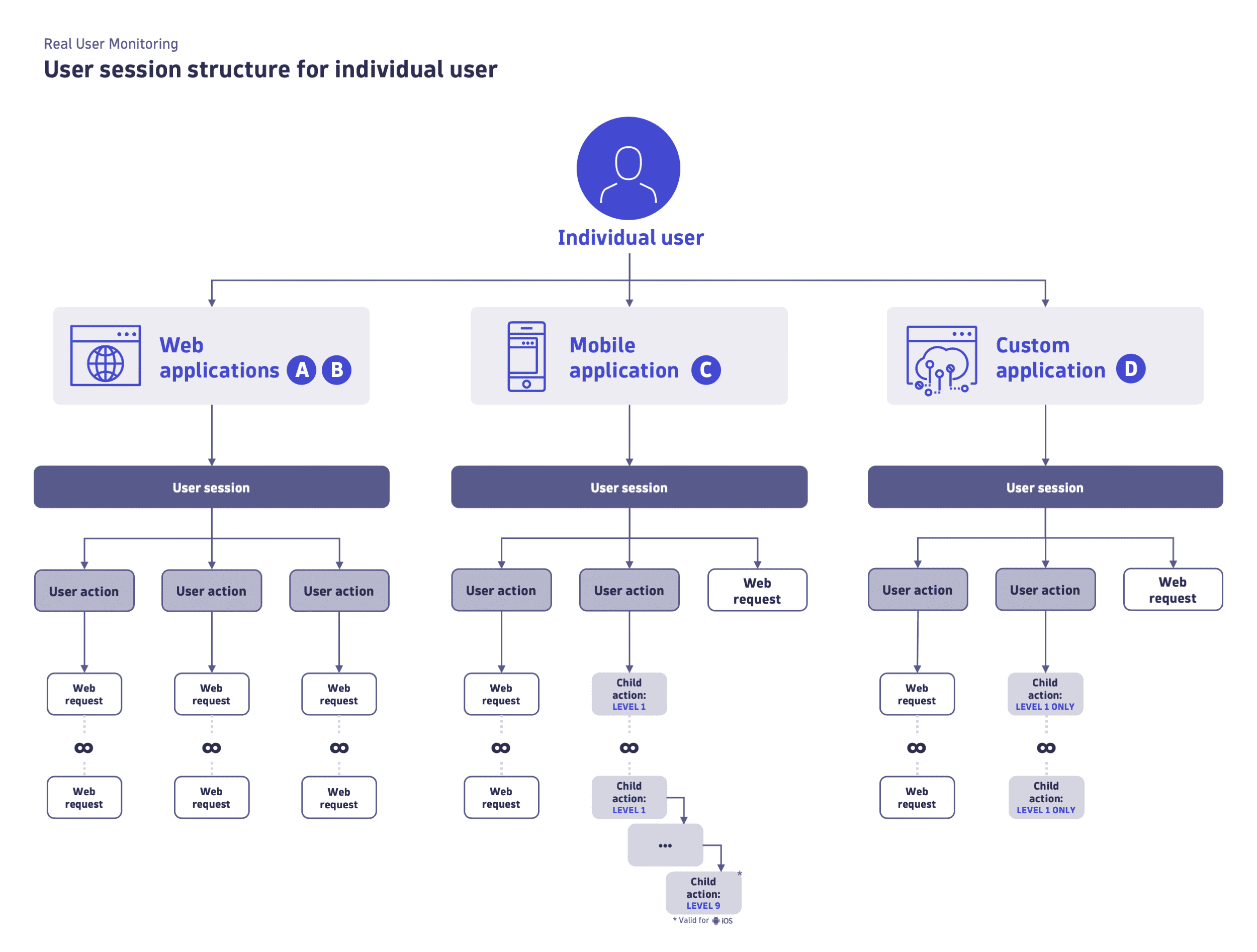
**Example RUM Views in Dynatrace**

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.

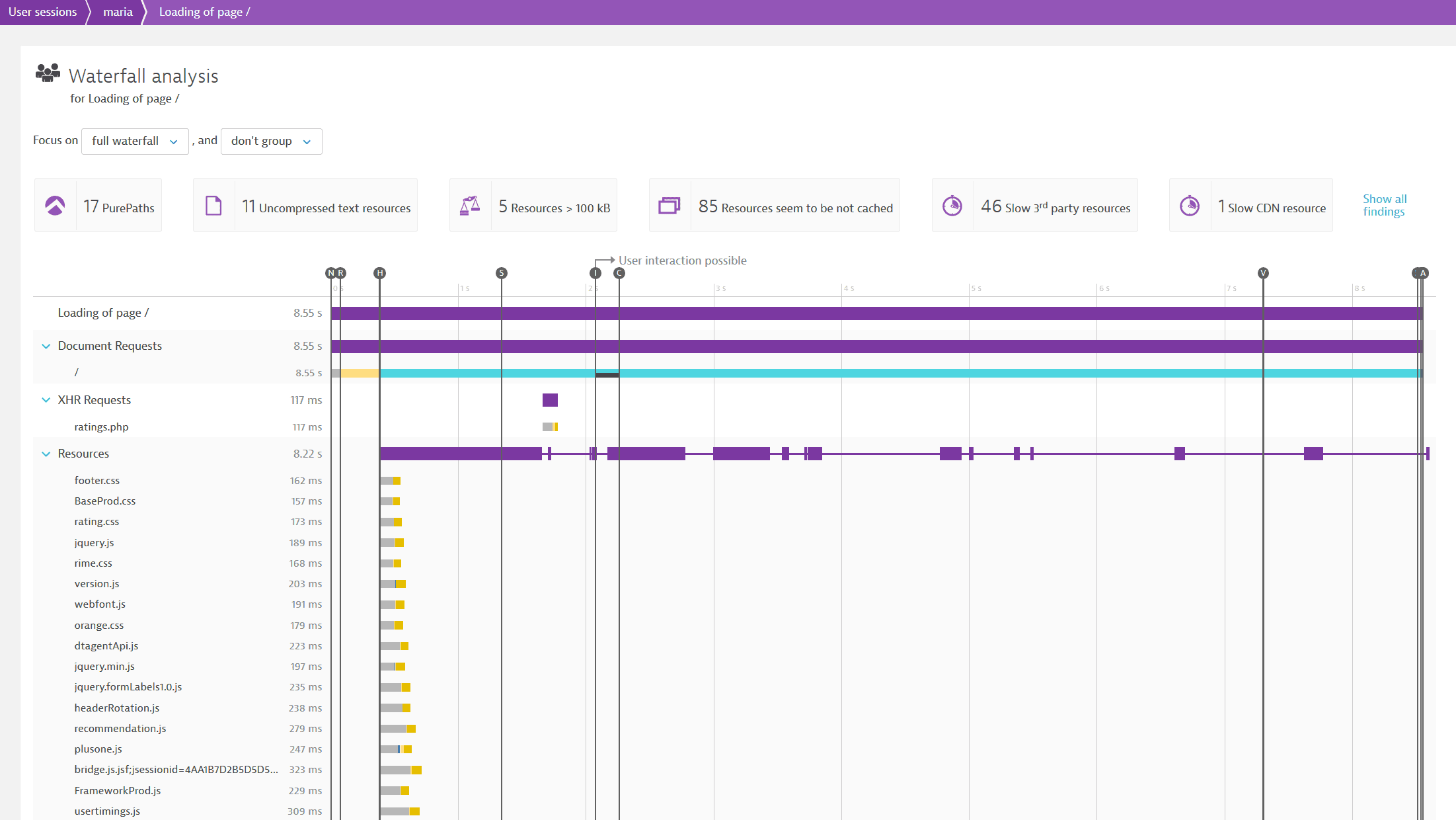
**Benefits of RUM**

* 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**.

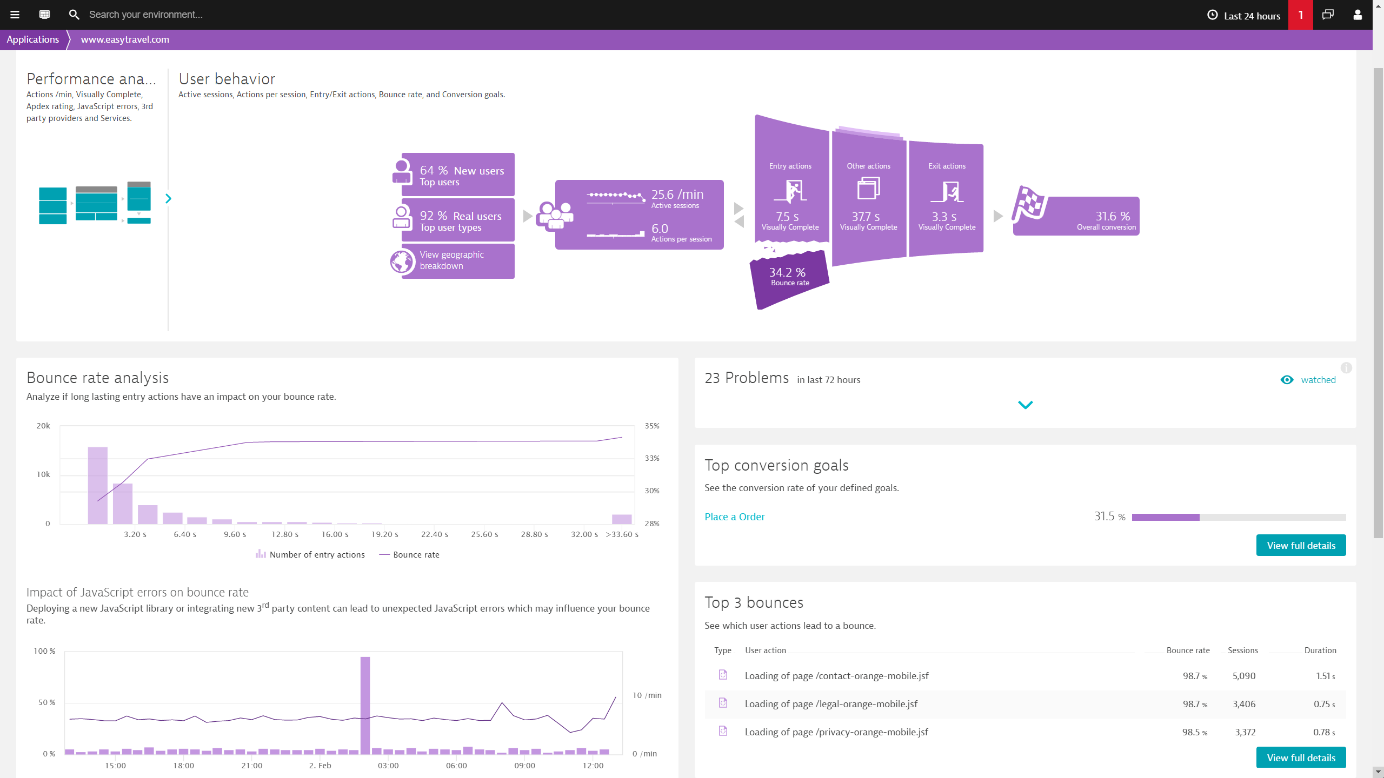
**1. RUM Conceptual Flow**   
A high-level diagram of how Dynatrace associates user activity across multiple devices (e.g., mobile → desktop) into a unified user session. This illustrates the backbone of end-to-end RUM tracking.



**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.

