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)**

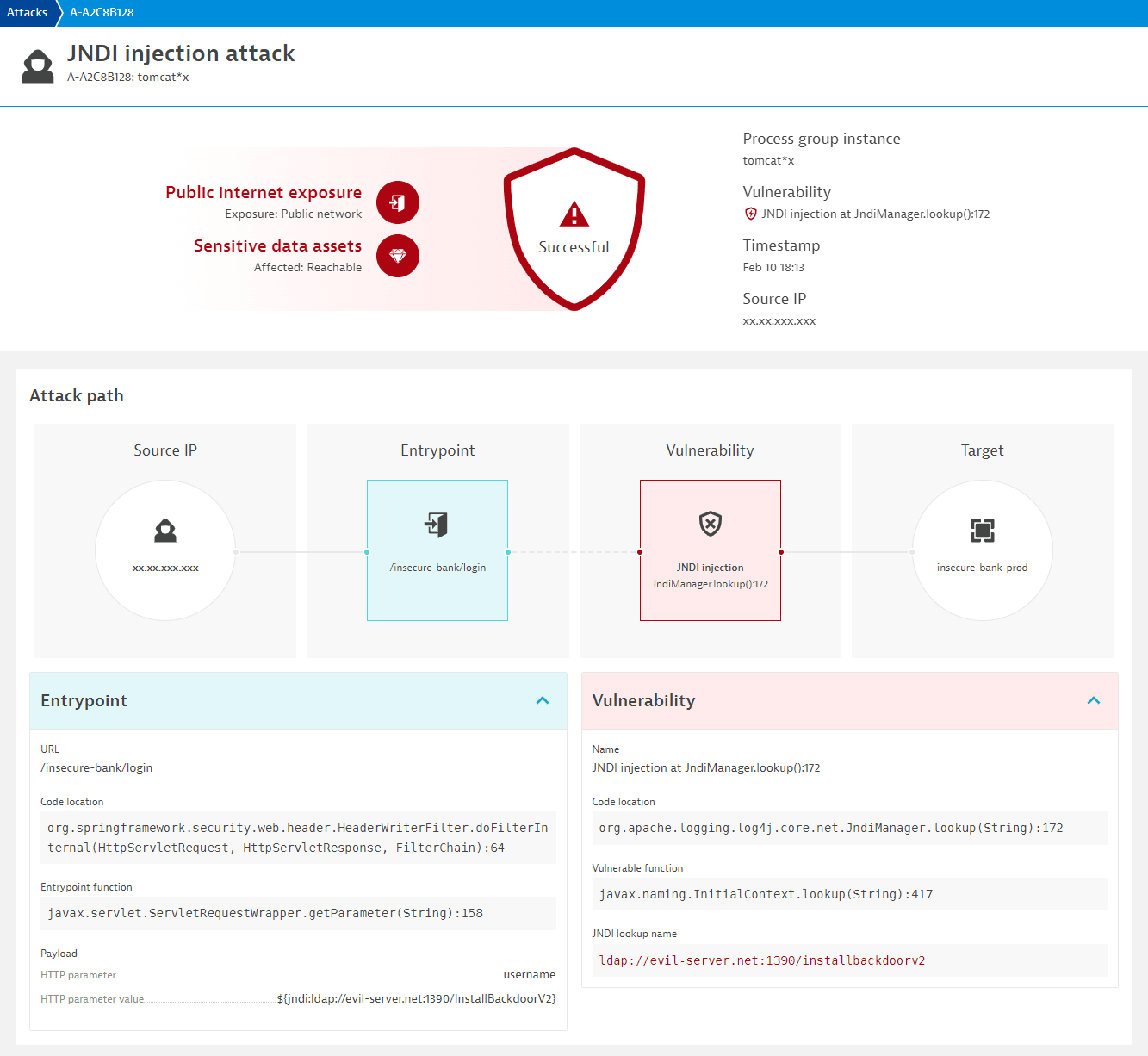
* Automatically detects **third-party vulnerabilities** (e.g., vulnerable libraries) and **code-level vulnerabilities** (issues originating in your own code) at runtime.
* Visualizes vulnerabilities in the **Security Overview dashboard**, showing counts by risk and status, host coverage, impacted process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

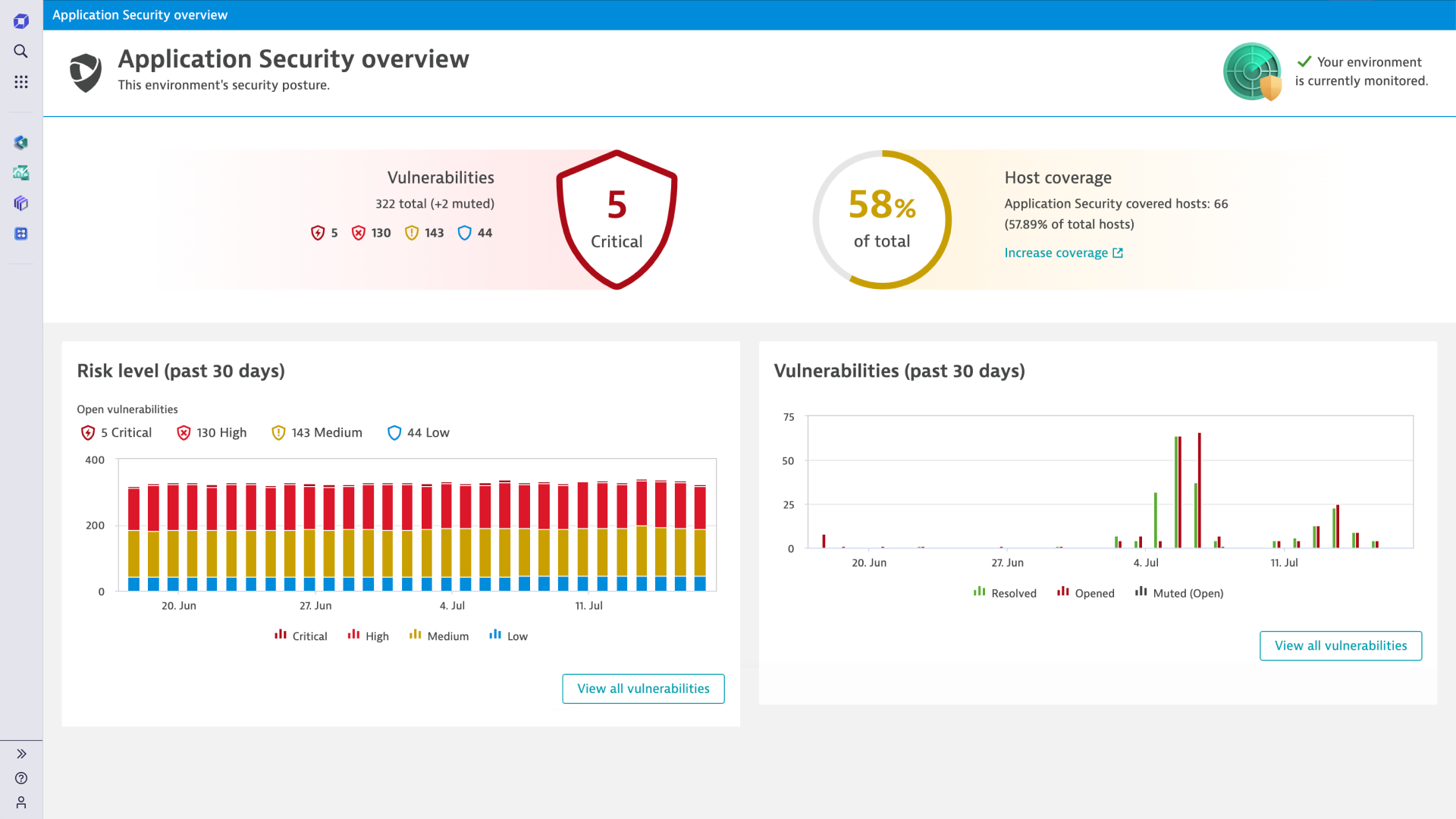
**2. Runtime Application Protection (RAP)**

* 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.





**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) }

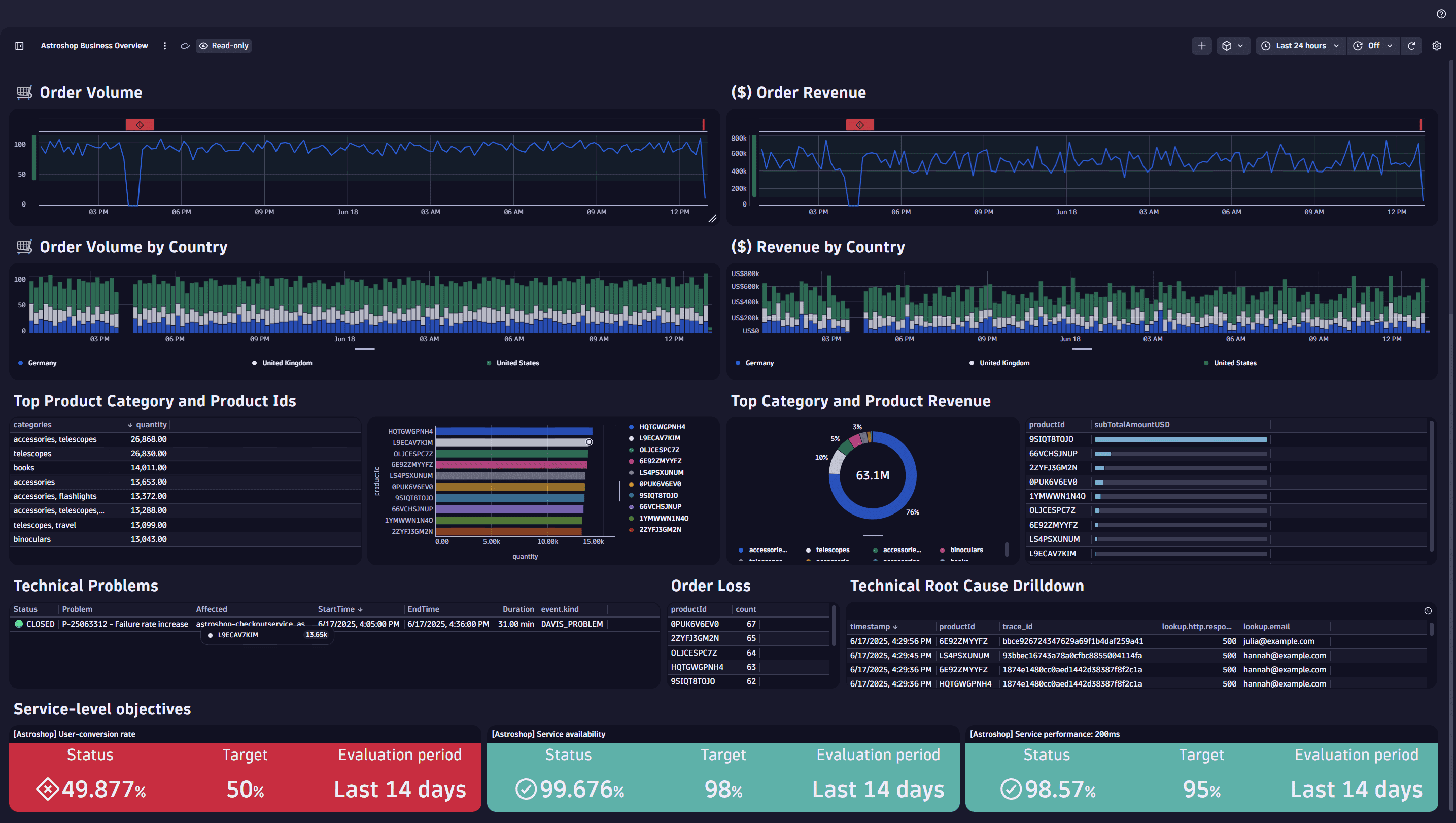
, by: { dt.entity.service }

| 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 |