Dynatrace Day 4: Advanced Monitoring & Integration – Detailed Notes

# 26. Network Monitoring

Dynatrace provides insights into the network layer to understand dependencies and identify latency or bandwidth bottlenecks.  
  
- Flow Analysis and Traffic Inspection:  
 - Monitors traffic between processes, services, and external endpoints.  
 - Visualizes packet flow direction and volume using Smartscape and service flow.  
 - Detects dropped packets, retransmissions, and connection errors.  
  
 Example: A spike in TCP retransmissions between two services can indicate degraded performance or a flaky network link, which Dynatrace can trace back to affected service calls.  
  
- Network Dependencies and Latency Insights:  
 - Shows how network issues impact service response time.  
 - Enables drill-down into slow external service calls.  
 - Correlates network metrics with CPU/memory to pinpoint performance bottlenecks.  
  
 Diagram: Smartscape map showing services A → B → C with network latency arrows highlighting delay on link B → C.

# 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 "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).

# 29. Diagnostic Tools

- Memory Dumps and CPU Sampling:  
 - Capture heap dumps for memory leak analysis.  
 - Perform CPU sampling to identify hotspots and long-running methods.  
  
- Thread and Heap Analysis:  
 - Visualize blocked or deadlocked threads.  
 - Analyze object allocation and GC pressure.  
  
- Problem Investigation with PurePath:  
 - Trace request lifecycle across services.  
 - Identify bottlenecks, errors, and abnormal behaviors.  
  
 Diagram: PurePath trace showing timing distribution across service A, DB, and external API with latency breakdown.

# 30. Reports & Alerts

- Custom Alerts and Problem Thresholds:  
 - Define static and dynamic rules on any metric.  
 - Leverage Davis AI for adaptive thresholds.  
  
- Alerting Profiles and Routing:  
 - Group alerts by severity, tag, or service.  
 - Route to email, Slack, ServiceNow, PagerDuty.  
  
- Scheduled and Ad Hoc Reports:  
 - Automate delivery of dashboards as PDFs.  
 - Export data from Data Explorer and DQL queries.  
  
 Example: Send weekly uptime and error reports to application owners.

# 31. Autosys Integration & Monitoring

- Use Cases for Autosys Monitoring:  
 - Monitor job execution time, failures, and scheduling delays.  
 - Correlate job failures with resource bottlenecks or application exceptions.  
  
- Job-Level Visibility and Correlation:  
 - Ingest Autosys logs and job status via APIs or log forwarding.  
 - Visualize impact on downstream services using PurePath.  
  
 Example: Job failure correlating with out-of-memory condition on the DB server.

# 32. Automation with APIs

- Configuration API Overview:  
 - Manage alerting profiles, management zones, dashboards, tags, and monitors.  
 - CRUD operations for environments and entities.  
  
- Automating Tagging, Dashboarding, Alerting:  
 - Auto-apply tags based on metadata.  
 - Create dashboards and alert rules via CI/CD scripts.  
 - Use Terraform or Ansible for infrastructure-as-code workflows.  
  
 Example: Use GitHub Actions to apply updated dashboards to all staging environments nightly.

# 33. Extending Dynatrace

- Custom Extensions (SDK):  
 - Use Extension Framework 2.0 for monitoring custom devices, protocols.  
 - Create plugin scripts to ingest data not natively supported.  
  
- Integration with Third-Party Systems:  
 - ServiceNow: CMDB sync, incident and event creation.  
 - Splunk: Export events or logs to SIEM.  
 - Prometheus/Grafana: Ingest metrics or visualize Dynatrace data externally.  
  
 Diagram: Architecture showing Dynatrace sending incidents to ServiceNow and metrics to Grafana.