Transaction-Based Monitoring Using Multiple Monitor Types

This chapter outlines a real-world transaction monitoring scenario using a combination of synthetic monitor types. The transaction consists of five stages that simulate an order lifecycle. Each stage uses a specific monitor to validate part of the process, and all are orchestrated to work in sequence.

---

1 Use Case Overview

Transaction Flow:

1. **Place an order on a website** (URL Sequence Monitor)
2. **Verify order status update** (URL Sequence Monitor)
3. **Check confirmation email received** (Mail Monitor)
4. **Confirm database entry** (Database Query Monitor)
5. **Validate legacy system entry** (Script Monitor)

Each of these steps represents a monitor type that can be configured individually and chained externally for full transaction validation.

---

2 Step-by-Step Monitor Implementation

Step 1: Place an Order (URL Sequence Monitor)

Objective: Simulate a user placing an order on the application frontend.

Steps:

1. Create a **Clickpath Monitor** or **Single-URL Monitor**.
2. Simulate user interaction: navigate to product page → add to cart → proceed to checkout → place order.
3. Insert **validations**:
    - Confirmation message ("Order placed successfully")
    - Redirect URL match (e.g., `/order/complete`)
4. Save the monitor and schedule it for regular execution.

Step 2: Verify Order Status (URL Sequence Monitor)

Objective: Confirm that the order status updates correctly in the UI.

Steps:

1. Create a **new URL monitor or Clickpath**.
2. Simulate login → navigate to “My Orders”
3. Add validation:
    - Verify text: “Order #12345 - Status: Processing”
    - Optionally capture status value from page

Step 3: Confirmation Email Validation (Mail Monitor)

Objective: Ensure an email confirmation is sent to the user.

Steps:

1. Set up a **Mail Monitor** (requires external integration or third-party email test service).
2. Configure:
    - Mailbox access credentials
    - Keyword validation in subject and body (e.g., “Thank you for your order”)
    - Email arrival timeout
3. Trigger this monitor after a successful order placement.

Step 4: Database Entry Check (Database Query Monitor)

Objective: Verify the order entry is added to the database.

Steps:

1. Create a **Database Monitor** (requires connectivity to internal DB).
2. Define query:

```
SELECT status FROM orders WHERE order_id = '12345';
```

3. Set result expectation (e.g., status = 'Processed')
4. Enable secure DB authentication using credential vault.

Step 5: Legacy System Update (Script Monitor)

Objective: Confirm that the legacy system received the order.

Steps:

1. Create a **Script Monitor** (e.g., shell or Python script).
2. Script should:
    - Call the legacy system API or run a log parser
    - Validate response or presence of new log entry
3. Use Dynatrace API or ActiveGate-based extension to execute it.

---

3 Orchestrating the Monitors

Dynatrace does not support native sequencing of multiple monitor types in a single UI transaction. Use external orchestration such as a **CI/CD pipeline, Jenkins job, or custom script** to:

1. Trigger Monitor 1 (Order placement)
2. On success, trigger Monitor 2 (Order status)
3. Followed by Mail Monitor, Database Monitor, and Script Monitor
4. Collect all responses and summarize in a custom dashboard or report

You can also use **Dynatrace API** to:

- Trigger each monitor execution
- Check status/results
- Pull metrics for consolidated view

---

4 Dashboard & Alert Configuration

Build a custom dashboard to track transaction health:

- Monitor tile for each step
- SLA widget for end-to-end response time
- Fail/success trends per step

Configure alerting:

- Failure of any step triggers an event via email, Slack, or webhook
- Include contextual metadata for easy troubleshooting

Summary

By combining URL monitors, database queries, script checks, and mail validations, you can simulate an end-to-end transaction in Dynatrace. Though these monitors are executed separately, they can be effectively coordinated using automation to simulate real business workflows. This approach enables complete observability across systems and ensures every component of the transaction pipeline is functioning correctly.