To compare **old Splunk dashboards** with **modern dashboards** (like in Dynatrace) and inform your Product Owner about what’s missing or needs improvement, you can follow this **step-by-step practical approach**, tailored for your **pharmacy claims application**:

**✅ Step 1: Understand the Purpose of Each Dashboard**

Split dashboards into **two categories**:

* **Business Metrics** (claims, volumes, errors, etc.)
* **Technical Metrics** (latency, errors, CPU, memory from Dynatrace)

| **Category** | **Examples from Old Splunk** | **Examples from Dynatrace** |
| --- | --- | --- |
| Business | Claim Volumes, Types (Paid, Reversed, Rejected), Rejection Trends | Mostly missing |
| Technical | Some exceptions, errors, latency logs | CPU, Memory, Request throughput, Service health, Error rate |

**✅ Step 2: Note Key Dashboards Available in Splunk**

From your message, the old Splunk dashboards include:

1. **Claim Volume Over Time**
2. **Claim Type Breakdown** (Paid, Reversal, Rejected)
3. **Rejection Rate Over Time**
4. **Universal Drilldown** – all transaction logs
5. **Historical Drilldown** – past claim trends
6. **Errors and Exceptions**

✅ **These are important** for understanding how your **claim system behaves from a functional/business view.**

**✅ Step 3: What Dynatrace Covers**

Dynatrace is **great for system health**, and gives you:

* Request rate, response time (per service)
* Failure rate, exception detection
* Infrastructure metrics: CPU, memory, disk
* Distributed tracing, service flow

🚫 **BUT** Dynatrace is weak at:

* Business-level insights (claim counts, types, specific rejections)
* Custom log searches unless explicitly piped

**✅ Step 4: Make a Simple Comparison Table**

| **Metric / View** | **Old Splunk** | **Dynatrace** | **Needed?** | **Comment** |
| --- | --- | --- | --- | --- |
| Claim Volume | ✅ Yes | ❌ No | ✅ Yes | Needed in new dashboard |
| Claim Types (Paid, Reversal, Rejected) | ✅ Yes | ❌ No | ✅ Yes | Business insights missing |
| Rejection Trend | ✅ Yes | ❌ No | ✅ Yes | Helps detect business issues |
| Universal Drilldown (all logs) | ✅ Yes | ❌ No | ✅ Optional | Useful for detailed support |
| Historical Drilldown | ✅ Yes | ❌ No | ✅ Optional | For long-term trends |
| Error Types per Service | ✅ Yes | ✅ Partial | ✅ Yes | Match and improve |
| Performance (response time, failure rate) | ❌ No | ✅ Yes | ✅ Already there | No new story needed |
| Service Health & Infra | ❌ No | ✅ Yes | ✅ Already there | Covered by Dynatrace |

**✅ Step 5: Create User Stories for Missing Dashboards**

Now give this to your PO in **simple story format** like:

**Title**: Add Claim Volume and Type Dashboard in Modern Stack  
**Description**:  
As a business analyst, I want to see a dashboard with daily/weekly claim volumes broken down by type (Paid, Reversed, Rejected) so that I can monitor business health.

**Title**: Rejection Rate Trend Dashboard  
**Description**:  
As a product owner, I want a dashboard showing rejection rate trends over time so that I can identify unusual spikes or drops in claim rejections.

**Title**: Error Type Classification View  
**Description**:  
As a developer/support engineer, I want to see error type trends per service (e.g., Validation errors, Auth errors, System failures) so that I can quickly identify recurring issues.

**Title**: Historical Claim Patterns Dashboard  
**Description**:  
As a stakeholder, I want to see long-term patterns in claim volumes and rejections to inform future business planning.

**✅ Step 6: Tools to Build Missing Dashboards**

You can tell the team:

* Either extend Splunk dashboards for business metrics
* Or **integrate Dynatrace logs to feed business data** into custom dashboards (if licensing permits)
* OR use **Grafana + Splunk/Dynatrace** as a unified view layer

**Summary of Actions:**

1. List old dashboards – categorize by business vs technical.
2. List what Dynatrace covers.
3. Build a table of coverage vs missing.
4. Write clear user stories for what’s missing.
5. Share with the PO and suggest which team (backend, infra, ops) will own each.

Would you like me to draft an email for you to send to your Product Owner with the summary and action items?

For business insights we must go with splunk logs

Exactly, you're absolutely right ✅ — **for business insights**, especially for your **pharmacy claims data**, **Splunk logs are the best choice**. Here's why, and how you can guide your Product Owner (PO) clearly:

### 🔍 Why Splunk for Business Insights?

Dynatrace is great for **infrastructure** and **application health**, but:

| **Business Requirement** | **Dynatrace** | **Splunk** | **Reason** |
| --- | --- | --- | --- |
| Claim Volume, Types | ❌ No | ✅ Yes | These come from your **application logs**, not system metrics |
| Rejections by Type | ❌ No | ✅ Yes | Detailed **error and claim status** info is logged |
| Drilldown into Transactions | ❌ No | ✅ Yes | Splunk can **search by claim ID, timestamp, etc.** |
| Trends & Business KPIs | ❌ No | ✅ Yes | Can build time-based **charts from logs** easily |

So:  
➡️ **Splunk = Business**  
➡️ **Dynatrace = System Health**

### ✅ Suggested Action Plan for You

#### 📌 Step 1: Confirm What's Needed

* Claim Volume Over Time
* Claim Types (Paid / Rejected / Reversed)
* Rejection Rate %
* Top Error Types from Logs
* Universal & Historical Drilldowns
* Timeline View of Business KPIs

#### 📌 Step 2: Create a Summary for Product Owner

You can say something like:

**Subject**: Dashboard Gaps for Business Insight – Splunk Stories Required

**Hi [PO Name],**

In our old Splunk dashboards, we had strong **business visibility** around pharmacy claim data — such as claim volumes, rejection rates, and type breakdowns. These are **not covered in Dynatrace**, which focuses on system health (CPU, latency, failure rate, etc.).

To ensure we don’t lose business monitoring capability in our modernization, I recommend creating user stories for the following dashboards in Splunk:

1. 📊 **Claim Volume Dashboard** – Total claims per day/week
2. 📉 **Rejection Rate Trend** – % of rejected claims over time
3. 📂 **Claim Type Breakdown** – Paid, Reversal, Rejected
4. 🧾 **Top Claim Error Types** – Grouped by service or code
5. 🔍 **Drilldown Dashboards** – Transaction-level logs for support
6. 📅 **Historical Trends** – Business KPIs over months

Let me know if I should help in creating these stories or reviewing backlog.

Thanks,  
[Your Name]

Would you like me to prepare these as **ready-made Jira story descriptions** for direct copy-paste too?

Claim types means what are they paid reversal rejected explain

Great question! In the context of **pharmacy claims**, the term **"claim types"** typically refers to the **outcome or action taken on a claim** after it's processed. Here's a simple explanation of the main types you mentioned:

**✅ 1. Paid Claims**

* **Meaning**: The claim was approved and the payment was successfully made to the pharmacy.
* **Business Impact**: Indicates successful transaction; good for revenue and customer satisfaction.
* **Logs Show**: Status like PAID, response code like A (Accepted), or payment confirmation.

**🔁 2. Reversal Claims**

* **Meaning**: A previously paid claim has been **cancelled or reversed**.
* **Why?**: This might happen due to:
  + Medication returned
  + Incorrect billing
  + Duplicate claim
* **Business Impact**: Adjusts previously paid amounts; important for auditing.
* **Logs Show**: Status like REVERSAL, response codes like R, or message Reversed.

**❌ 3. Rejected Claims**

* **Meaning**: The claim was **denied** and no payment was made.
* **Why?**
  + Invalid member ID
  + Expired prescription
  + Missing authorization
  + Pricing mismatch
* **Business Impact**: Signals problems that must be fixed; key metric for quality and performance.
* **Logs Show**: Status REJECTED, rejection reason codes, rejection message, etc.

**🧠 Summary Table**

| **Claim Type** | **Meaning** | **Business Impact** | **Log Clues** |
| --- | --- | --- | --- |
| **Paid** | Approved & paid | Positive; revenue | PAID, status: A, paymentId |
| **Reversal** | Cancelled payment | Adjusts revenue; must track | REVERSAL, reversalReason |
| **Rejected** | Denied claim | Negative; fix needed | REJECTED, errorCode, reasonMessage |

**💡 Why Important in Dashboards?**

Visualizing claim types helps you:

* Spot spikes in rejections (e.g. system bug or policy change)
* Track how many reversals happen (maybe training issue)
* Understand performance (rejected vs paid ratio)

Would you like a **Splunk query example** to pull claim type counts over time?

To analyze a traffic spike for a specific request in Dynatrace and test a 110% traffic increase within a 30-minute window, follow these steps:

### 📊 Part 1: Analyze Traffic Spike in Dynatrace for a Particular Request

#### ✅ Step 1: Identify the Request

1. Go to Dynatrace > Transactions & services.
2. Select the service (e.g., pharmacy-claim-service).
3. Click on **"View requests"**.
4. Search for the specific endpoint (e.g., /claimsubmit).

#### ✅ Step 2: Use Multidimensional Analysis

1. Click **"Create multidimensional analysis view"**.
2. Add dimensions like:
   * Request name or URL
   * Response time
   * HTTP method
   * Status code
3. Set timeframe to **"Last 30 minutes"**.
4. Pin to dashboard if needed.

#### ✅ Step 3: Enable Baseline Comparison

1. In the same view, enable **baseline comparison**.
2. Choose a previous window (e.g., same time yesterday or same day last week).
3. Look for spikes in:
   * Request count
   * Response time
   * Error rate

### 🚀 Part 2: How to Simulate 110% Spike

#### ✅ Goal:

Simulate traffic load that's 110% higher than your baseline using Python or JMeter.

### ♻️ Option 1: Using JMeter (Preferred for Load Testing)

#### Step-by-Step:

1. Open Apache JMeter.
2. Add **Thread Group**:
   * Set baseline: e.g., 100 users in 30 minutes.
   * To simulate 110% spike: set 210 users in 30 minutes.
3. Add **HTTP Request Sampler**:
   * URL: your target endpoint (e.g., /claimsubmit)
   * Method: POST/GET as applicable
4. Add **Summary Report** & **Graph Results** listener.
5. Run test.
6. Monitor in Dynatrace.

### ♻️ Option 2: Python Script (Lightweight Testing)

import requests

import threading

import time

# Configuration

URL = "https://your-api/claimsubmit"

NUM\_REQUESTS = 210 # 110% increase from baseline of 100

THREAD\_COUNT = 50

# Function to send requests

def send\_request():

try:

response = requests.get(URL) # or .post with data

print(f"Status: {response.status\_code}")

except Exception as e:

print(f"Error: {e}")

# Start load

threads = []

for i in range(NUM\_REQUESTS):

t = threading.Thread(target=send\_request)

threads.append(t)

t.start()

time.sleep(30 / NUM\_REQUESTS) # spread over 30 mins

# Wait for all to complete

for t in threads:

t.join()

### 📊 Monitoring in Dynatrace:

1. Open **Service flow** or **Request view**.
2. Filter for /claimsubmit.
3. Use **"Failure rate", "Throughput", and "Response time"** metrics.
4. Enable **Anomaly detection** to see automatic spike alerts.

Let me know if you need help generating a JMeter .jmx file or visualizing the spike in Dynatrace!

enable baseline com parison