A black text with a white background

Description automatically generated

**Document Version Control Note**

|  |  |
| --- | --- |
| **Document Title** | DCC |
| **Document Code** |  |
| **Last Date of Release / Review** |  |
| **Document Superseded (Previous Version)** | Initial Release |
| **Document Version No. (Current Version)** | V1.0 |
| **Document Approver** |  |
| **Document Author(s)** | Raj Kansal , Vansh Shrivastava |
| **Applicable Environment** | CEQ Internal |
| **Service** | Observability |
| **Technology** | New Relic |
| **Classification** | Internal |

|  |  |
| --- | --- |
|  |  |

**Revision**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Version** | **Date** | **Revision Notes** | **Author** | **Reviewer** | **Approver** |
| 1.0 | 13thAugust 2024 | NA | Raj Kansal , Vansh Shrivastava | Kashish Pandita | Kashish Pandita |

**Requirements**

* NewRelic Account
* DCC NRQLs
* Dashboard
* GROK Debugger

**DDCC Jenkins Pipeline Overview**

**First, it clones the project's code from a GitHub repository. Then, it performs a code quality analysis using SonarQube. After that, it builds a Docker image of the project and logs into Docker Hub using stored credentials. The Docker image is then pushed to Docker Hub. Finally, the pipeline deploys the application to a Google Kubernetes Engine (GKE) cluster using specified environment settings. Each stage is carefully controlled by environment variables and credentials to ensure smooth execution.**

**DDCC Github Actions Pipeline Overview**

**This documentation shows an overview of how to create a CI/CD pipeline with AKS cluster.  We have created pipelines, these pipelines are used to deploy microservices in the AKS cluster and also sending data to new relic for observability . We are monitoring the deployment, error in the deployment and other errors in the pipeline.**

**DevOps Command Centre**

A **DevOps Command Centre** is a centralized platform or dashboard designed to provide a comprehensive view of your DevOps pipelines, infrastructure, and application performance. It serves as a hub for monitoring, managing, and optimizing DevOps processes, facilitating quick decision-making and issue resolution.

**Creating a DevOps Command Centre using a pipeline involves integrating various tools, automating data collection, and building a centralized dashboard to monitor and manage your DevOps processes. By consolidating data from CI/CD pipelines, monitoring systems, and logging tools, you can gain comprehensive visibility into your DevOps operations and make informed decisions to improve performance and efficiency.**

**1. Defining Your DevOps Command Centre Requirements -**

Before setting up the pipeline, determine what you want to monitor and manage from your command centre. Common requirements include:

**Pipeline Status:** Monitor the status and performance of CI/CD pipelines.

**Infrastructure Health:** Track server health, resource utilization, and performance metrics.

**Application Performance:** Monitor application performance, including response times, error rates, and throughput.

**Metrics and Analytics:** Aggregate and visualize key performance metrics and trends.

**2. Integrating Tools into the Pipeline**

To build the command centre, integrate the following tools into your pipeline:

**a. CI/CD Tools**

**Build and Deployment Status:** Use tools like Jenkins, GitLab CI, GitHub Actions, or Azure DevOps to manage and monitor your pipelines.

**Data Collection:** Configure these tools to send data about build status, deployment success/failure, and timing.

**b. Monitoring Tools**

**Application Performance Monitoring (APM)****:** Tools like New Relic, Datadog, or AppDynamics collect data on application performance.

**Infrastructure Monitoring:** Use tools like Prometheus, Nagios, or New Relic to monitor infrastructure metrics such as CPU, memory, and disk usage.

**c. Logging Tools**

**Log Management:** Integrate logging tools like ELK Stack (Elasticsearch, Logstash, Kibana), Splunk, or New Relic Logs to aggregate and analyze logs.

**3. Automating Data Collection and Reporting**

**Data Pipelines:** Set up automated data pipelines to gather data from CI/CD tools, monitoring tools, and logging systems. This can involve:

**APIs:** Using APIs to pull data from tools like Jenkins or New Relic.

**Scripts:** Writing scripts to collect and process data on a schedule.

**Data Aggregation:** Aggregate data from different sources into a centralized location, such as a data warehouse or a dashboard tool.

**4. Creating the Dashboard**

**a. Using a Dashboard Tool**

**New Relic Dashboards:** Create custom dashboards in New Relic to display metrics from your CI/CD pipelines, infrastructure, and applications.

**Widgets:** Add widgets for pipeline status, infrastructure health, application performance, and alerts.

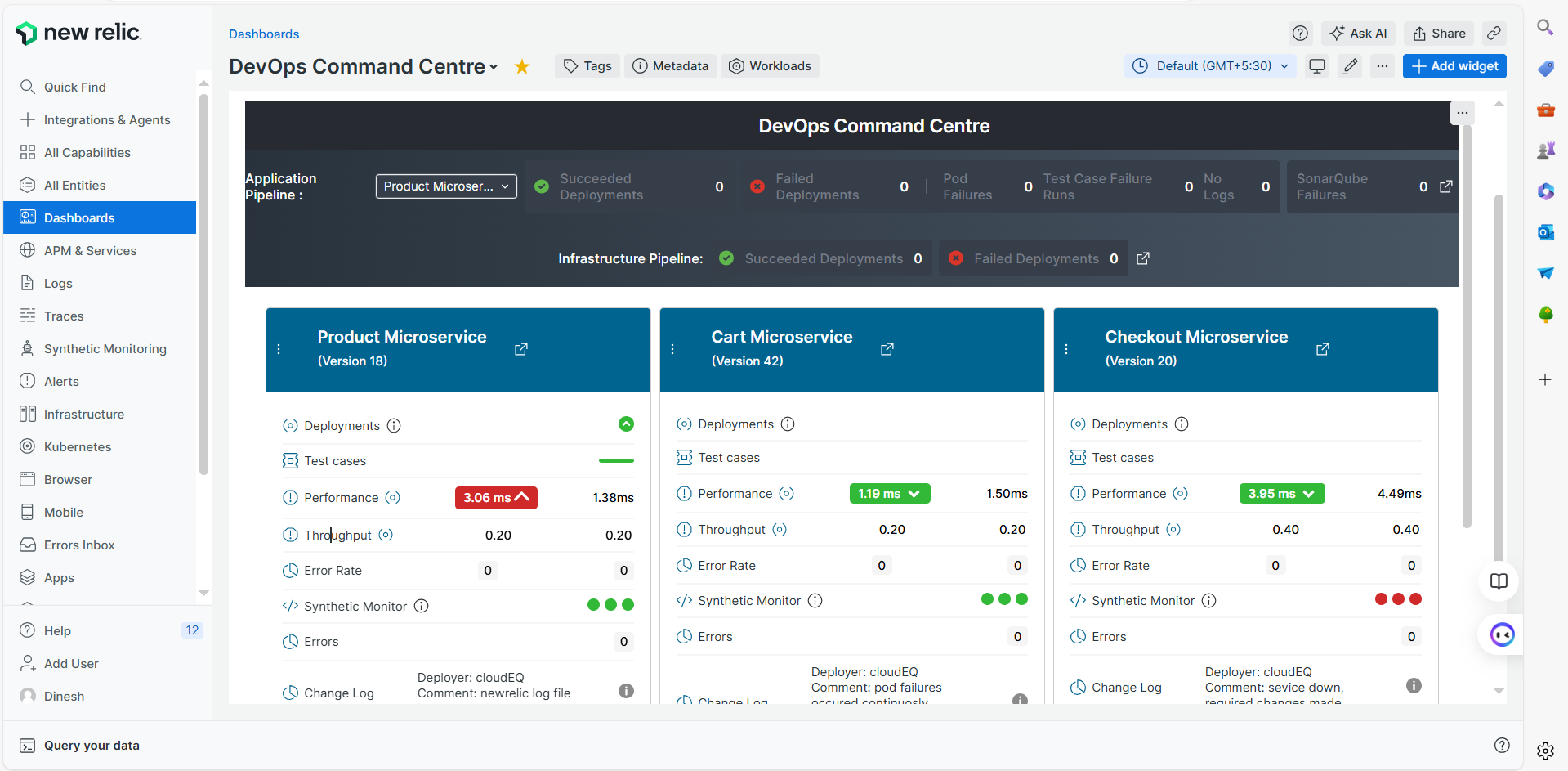
**Grafana:** If using Grafana, connect it to your data sources (e.g., Prometheus, New Relic) and create dashboards to visualize the data.

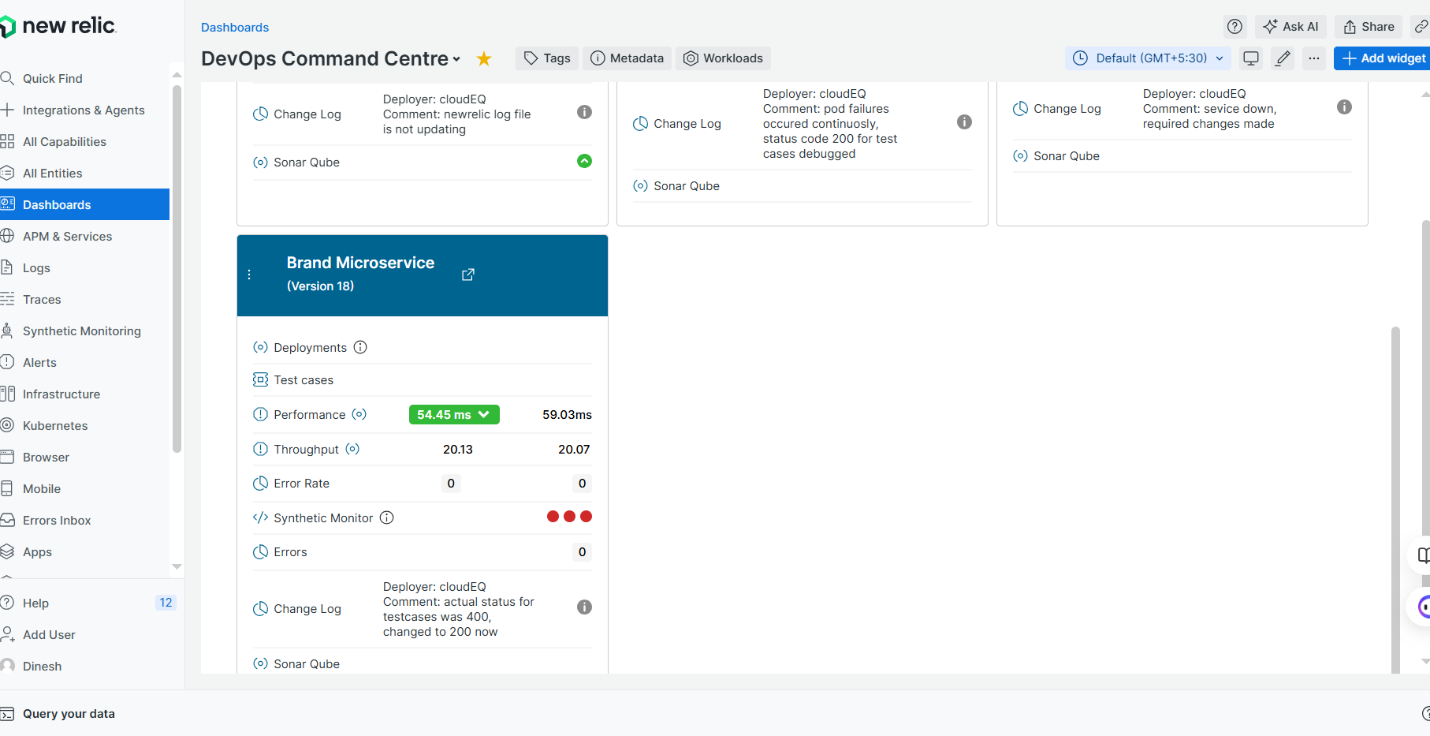
**Panels:** Use panels to show trends, charts, and tables for key metrics.

**b. Configuring Alerts**

Alerting: Configure alerts based on critical thresholds for CPU usage, build failures, or application errors. Ensure alerts are integrated with your command centre to provide real-time notifications.

We had created DCC for both Application and Infrastructure using Pipeline. In the Application Pipeline we have different microservices like **Product Microservices, Cart Microservices, Checkout Microservices, Brand Microservices.**





Like we can see in front of the **Application Pipeline** we have a dropdown for all the microservices to see all **Successful** **Deployments , False** **Deployments , Pod Failures, Test Case Failure Runs, No Logs, SonarQube Failures** and we can count them too, how many failures we got in every respective field.

In front of the Infrastructure Pipeline, we could see the total of **Successful Deployments, Failed Deployments.**

A **Product Microservice** is a specific type of microservice that manages and encapsulates the functionality related to the product domain within an application. In a microservices architecture, each microservice is designed to handle a distinct business capability or domain, and the Product Microservice is responsible for operations and data related to products.

A **Cart Microservice** is a specialized microservice within a microservices architecture that manages shopping cart functionalities. It handles operations related to the user's cart, including adding items, updating quantities, removing items, and calculating totals. This microservice is crucial for e-commerce platforms and any application that involves a shopping cart or similar concept.

A **Checkout Microservice** is a key component in an e-commerce or transaction-based application. It handles the final steps of the purchase process, where users review their cart, provide payment information, and complete their orders. This microservice manages various aspects of the checkout process, including payment processing, order creation, and transaction management.

A **Brand Microservice** manages brand-related data and functionality within an application, particularly in e-commerce or product-centric platforms. This microservice focuses on handling brand information, including brand names, descriptions, logos, and other attributes. It helps in organizing products by brand and providing brand-related insights.

**For every microservice we have different dashboards and different versions below them which display different signals for deployments, test cases, performance, throughput, error rate, synthetic monitor, errors, change log, SonarQube.**

**If the signal is green, it means that the respective field is working properly and if it is red, it is not working properly.**

**In front of every service there is a button, when it is clicked, we switch to the dashboard of the respective service and get to know all the widgets.**

**Product Microservice Detailed Analysis**

1. **Product Deployment Dashboard**
2. **Successful Builds**

This widget provides a count of successful builds from the log data within the past week. It is essential for monitoring the performance and reliability of the "Product Build and Deploy" workflow.

**Query-**

SELECT count(\*) AS 'Successful Builds' from Log WHERE action = 'completed' and workflow\_job.conclusion = 'success' and workflow\_job.workflow\_name = 'Product Build and Deploy' since 1 week ago

**Output-**

A green rectangle with white text

Description automatically generated

1. **Failed Builds**

This widget provides a count of failed builds from the log data within the past week. It is essential for identifying issues in the "Product Build and Deploy" workflow and assessing the stability and reliability of the deployment process.

**Query-**

SELECT count(\*) AS 'Failed Builds' from Log WHERE action = 'completed' and workflow\_job.conclusion = 'failure' and workflow\_job.workflow\_name = 'Product Build and Deploy' since 1 week ago

**Output-**

**A red square with white border

Description automatically generated**

1. **Deployments**

This widget provides a count of all deployments from the log data within the past week. It is essential for tracking the overall activity of the "Product Build and Deploy" workflow.

**Query-**

SELECT count(\*) as Deployments from Log WHERE action = 'completed' and workflow\_job.workflow\_name = 'Product Build and Deploy' since 1 week ago TIMESERIES

**Output-**

**A white background with black numbers

Description automatically generated with medium confidence**

1. **Failure Message by Version**

This widget provides detailed information about failures from the log data within the past week for a specific deployment. It selects the version, failure message, and GUID for entries where the workflow job steps include "CURRENT\_VERSION" and where a failure has occurred. The conditions ensure that only relevant failure messages are included, excluding empty or irrelevant messages.

**Query-**

FROM Log SELECT version, failure AS 'failure\_message', guid WHERE workflow\_job.steps LIKE '%CURRENT\_VERSION%' AND failure IS NOT NULL and failure not in (' ' ,' Actual Status 0 : 400, Actual Status 1 : null',' , ' ) and guid = 'Mzg4NDI0NXxBUE18QVBQTElDQVRJT058MTA0MjI5NjkyOA' since 1 week ago

**Output-**

**A screenshot of a computer

Description automatically generated**

1. **Successful Deploys by User**

This widget counts the number of successful builds from the log data within the past week, grouped by the user who initiated the builds. It filters for entries where the action is 'completed' and the workflow concluded successfully for the "Product Build and Deploy" workflow.

**Query-**

SELECT count(\*) from Log WHERE action = 'completed' and workflow\_job.conclusion = 'success' AND workflow\_job.workflow\_name = 'Product Build and Deploy' FACET sender.login since 1 week ago

**Output-**

**A red line on a white background

Description automatically generated**

1. **Failed Deploys by User**

This widget counts the number of failed builds from the log data within the past week, grouped by the user who initiated the builds. It filters for entries where the action is 'completed' and the workflow concluded with a failure for the "Product Build and Deploy" workflow.

**Query-**

SELECT count(\*) from Log WHERE action = 'completed' and workflow\_job.conclusion = 'failure' AND workflow\_job.workflow\_name = 'Product Build and Deploy' FACET sender.login since 1 week ago

**Output-**

**A close up of a logo

Description automatically generated**

1. **Error in Deployment**

This widget provides the failure messages from the log data within the past week for completed builds in the "Product Build and Deploy" workflow. It filters for entries where a failure has occurred and is not null.

**Query-**

SELECT failure as 'failure\_message'from Log WHERE action = 'completed' AND workflow\_job.workflow\_name = 'Product Build and Deploy'and failure is not null since 1 week ago

**Output-**

**A close-up of a computer screen

Description automatically generated**

1. **Error Deployment History**

This widget provides the number of completed builds that resulted in failure for the "Product Build and Deploy" workflow over the past week and compares this data with the same metric from one month ago.

**Query-**

SELECT count(\*) from Log WHERE action = 'completed' AND workflow\_job.workflow\_name = 'Product Build and Deploy' and workflow\_job.conclusion ='failure' since 1 week ago COMPARE WITH 1 month ago TIMESERIES

**Output-**

**A white background with black and red lines

Description automatically generated with medium confidence**

1. **Deployment Details**

This widget provides detailed information about completed deployments for the "Product Build and Deploy" workflow from the log data within the past week. It selects the workflow name, start and end times, conclusion, runner name, status, version, and the user who initiated the deployment.

**Query-**

SELECT workflow\_job.workflow\_name, workflow\_job.started\_at as 'Start Time', workflow\_job.completed\_at as 'End Time', workflow\_job.conclusion as 'Conslusion', workflow\_job.runner\_name as 'Runner Name', workflow\_job.status as 'Status', version as 'Version', sender.login as 'Deployer' from Log WHERE action = 'completed' AND workflow\_job.workflow\_name = 'Product Build and Deploy' and version IS NOT NULL since 1 week ago

**Output-**

**A group of text boxes

Description automatically generated**

1. **Failed Testcase Details**

This widget provides detailed information about failed deployments for the "Product Build and Deploy" workflow from the log data within the past week. It selects the version, deployer, test case status, test case title, workflow status, and conclusion for entries where the workflow concluded with a failure and the test case title is not null. T

**Query-**

SELECT version, sender.login as 'deployer' ,testcasestatus01 AS 'TestCaseStatus',testcasetitle01 AS 'TestCaseTitle',workflow\_job.status,workflow\_job.conclusion FROM Log WHERE workflow\_job.workflow\_name = 'Product Build and Deploy' and workflow\_job.conclusion = 'failure' and testcasetitle01 is not null SINCE 1 week ago LIMIT MAX

**Output-**

**A close-up of a white background

Description automatically generated**

1. **Pod Details**

This widget provides the information about failed deployments for the "Product Build and Deploy" workflow from the log data within the past 15 days. It selects the workflow name, pod name, pod status, version, and conclusion for entries where the workflow concluded with a failure and the pod status is not null.

**Query-**

FROM Log SELECT workflow\_job.workflow\_name,podname,podstatus,version,workflow\_job.conclusion Where workflow\_job.conclusion='failure' and workflow\_job.workflow\_name = 'Product Build and Deploy' AND podstatus IS NOT NULL since 15 days ago

**Output-**

**A close-up of a white background

Description automatically generated**

1. **Total Count Of Passed Workflows**

This widget provides the number of passed test cases from the log data for successful deployments in the "Product Build and Deploy" workflow within the past week. It filters for entries where the workflow concluded successfully, ensuring that the count includes only those with a non-zero and non-null number of passed test cases.

**Query-**

FROM Log SELECT count(passedtestcase) Where workflow\_job.conclusion = 'success' and workflow\_job.workflow\_name = 'Product Build and Deploy' and passedtestcase != 0 and passedtestcase IS NOT NULL since 1 week ago

**Output-**

**A close-up of a logo

Description automatically generated**

1. **Product Deployment Testcase Dashboard**

**1.Recent Product Build and Deploy Failures**

This widget provides the latest timestamps of failed deployments for the "Product Build and Deploy" workflow within the past week, filtered by a specific GUID. It facets the results by version, deployer, workflow conclusion, test case status, and test case title.

**Query-**

SELECT latest(timestamp) FROM Log WHERE workflow\_job.workflow\_name = 'Product Build and Deploy' and guid = 'Mzg4NDI0NXxBUE18QVBQTElDQVRJT058MTA0MjI5NjkyOA' AND workflow\_job.conclusion = 'failure' FACET version, sender.login, workflow\_job.conclusion,testcasestatus01 AS 'TestCaseStatus',testcasetitle01 AS 'TestCaseTitle' SINCE 1 week ago LIMIT 5

**Output-**

**A screenshot of a computer

Description automatically generated**

**2. Pod Details**

This widget displays detailed information about failed deployments from the log data within the past week. It includes the version, pod name, pod status, deployer, workflow status, and conclusion for entries where the "Product Build and Deploy" workflow concluded with a failure

**Query-**

SELECT version,podname,podstatus,sender.login as 'deployer',workflow\_job.status,workflow\_job.conclusion FROM Log WHERE workflow\_job.conclusion = 'failure' AND workflow\_job.workflow\_name = 'Product Build and Deploy' SINCE 1 week ago

**Output-**

**A screenshot of a computer

Description automatically generated**

**3. count of pod failure message 1 week vs 1 month**

This widget provides a count of failed deployments from the log data within the past week, specifically focusing on pods that are not in a 'running' status for the "Product Build and Deploy" workflow. It compares this count with the same metric from one month ago, displayed as a timeseries

**Query-**

SELECT count(podstatus) FROM Log WHERE workflow\_job.conclusion = 'failure' AND workflow\_job.workflow\_name = 'Product Build and Deploy'and podstatus != 'running' SINCE 1 week ago COMPARE WITH 1 month ago TIMESERIES

**Output-**

**A screenshot of a computer screen

Description automatically generated**

**4. podStatus count**

This widget provides a count of failed deployments from the log data within the past week for the "Product Build and Deploy" workflow, specifically focusing on the status of pods that are not null. It facets the results by pod status, allowing teams to see the distribution of different pod statuses associated with failures

**Query-**

SELECT count(podstatus) FROM Log Where workflow\_job.conclusion='failure' and workflow\_job.workflow\_name = 'Product Build and Deploy' AND podstatus IS NOT NULL FACET podstatus since 1 week ago

**Output-**

**A green circle with black text

Description automatically generated**

**5. Count by Test case failure message**

This widget provides a count of failures from the log data within the past 15 days for the "Product Build and Deploy" workflow, specifically focusing on the failure messages associated with each deployment. It facets the results by failure message, allowing teams to identify the different reasons for failures.

**Query-**

SELECT count(failure) FROM Log Where workflow\_job.conclusion='failure' and workflow\_job.workflow\_name = 'Product Build and Deploy' FACET  failure as 'failure\_message'since 15 days ago

**Output-**

**A purple circle with black text

Description automatically generated**

**6. Testcases Failures detailed view**

This widget displays detailed information about failed deployments from the log data within the past week for the "Product Build and Deploy" workflow. It includes the version, deployer, test case status, test case title, workflow status, and conclusion, specifically filtering for entries where the workflow concluded with a failure and the test case title is not null.

**Query-**

SELECT version, sender.login as 'deployer' ,testcasestatus01 AS 'TestCaseStatus',testcasetitle01 AS 'TestCaseTitle',workflow\_job.status,workflow\_job.conclusion FROM Log WHERE workflow\_job.workflow\_name = 'Product Build and Deploy' and workflow\_job.conclusion = 'failure' and testcasetitle01 is not null SINCE 1 week ago LIMIT MAX

**Output-**

**A screenshot of a computer

Description automatically generated**

1. **Product Performance Dashboard**
2. **Response time (ms)**

This widget displays the average response time in milliseconds for a specific application monitored by New Relic's APM over the past 8 hours. It includes data such as the average response time, specifically filtering for the application identified by its GUID. The widget also compares the current average response time with that from 10 minutes ago, allowing users to identify trends and potential performance issues effectively.

**Query-**

SELECT average(newrelic.goldenmetrics.apm.application.responseTimeMs) AS 'Response Time Ms' FROM Metric WHERE entity.guid in ('Mzg4NDI0NXxBUE18QVBQTElDQVRJT058MTA0MjI5NjkyOA') LIMIT MAX TIMESERIES SINCE 8 hours ago COMPARE WITH 10 minutes ago

**Output-**

**A graph with numbers and lines

Description automatically generated with medium confidence**

1. **Throughput**

This widget displays the throughput of a specific application monitored by New Relic's APM over the past 8 hours. It calculates the rate of requests per minute, specifically filtering for the application identified by its GUID. The widget also compares the current throughput with the value from 10 minutes ago, enabling users to assess changes in application traffic and identify any performance issues effectively.

**Query-**

SELECT rate(count(newrelic.goldenmetrics.apm.application.throughput), 1 MINUTES) AS 'Throughput' FROM Metric WHERE entity.guid in ('Mzg4NDI0NXxBUE18QVBQTElDQVRJT058MTA0MjI5NjkyOA') LIMIT MAX TIMESERIES SINCE 8 hours ago COMPARE WITH 10 MINUTES ago

**Output-**

**A screenshot of a computer

Description automatically generated**

1. **Error rate**

This widget displays the average error rate for a specific application monitored by New Relic's APM over the past 8 hours. It calculates the average error rate, specifically filtering for the application identified by its GUID. The widget also compares the current average error rate with that from 10 minutes ago, allowing users to identify trends in application errors and assess any recent changes in performance.

**Query-**

SELECT average(newrelic.goldenmetrics.apm.application.errorRate) AS 'Error Rate' FROM Metric WHERE entity.guid in ('Mzg4NDI0NXxBUE18QVBQTElDQVRJT058MTA0MjI5NjkyOA') LIMIT MAX TIMESERIES SINCE 8 hours ago COMPARE WITH 10 MINUTES ago

**Output-**

**A line of blue and white text

Description automatically generated with medium confidence**

1. **CPU utilization**

This widget displays the average CPU utilization for a specific application monitored by New Relic's APM over the past 8 hours. It calculates the average user CPU utilization, specifically filtering for the application identified by its GUID. The widget also compares the current average CPU utilization with that from 10 minutes ago, enabling users to track performance trends and identify any significant changes in resource usage

**Query-**

SELECT average(apm.service.cpu.usertime.utilization) AS 'CPU utilization' FROM Metric WHERE entity.guid in ('Mzg4NDI0NXxBUE18QVBQTElDQVRJT058MTA0MjI5NjkyOA') LIMIT MAX TIMESERIES SINCE 8 hours ago COMPARE WITH 10 MINUTES ago

**Output-**

**A graph with numbers and lines

Description automatically generated with medium confidence**

1. **Apdex score**

This widget displays the Apdex score for a specific application monitored by New Relic's APM over the past 8 hours. It calculates the Apdex score, which measures user satisfaction based on response times, specifically filtering for the application identified by its GUID. The widget also compares the current Apdex score with that from 10 minutes ago, allowing users to assess changes in user satisfaction and identify any potential performance issues

**Query-**

SELECT apdex(apm.service.apdex) AS 'Apdex score' FROM Metric WHERE entity.guid in ('Mzg4NDI0NXxBUE18QVBQTElDQVRJT058MTA0MjI5NjkyOA') LIMIT MAX TIMESERIES SINCE 8 hours ago COMPARE WITH 10 MINUTES ago

**Output-**

**A close-up of a graph

Description automatically generated**

1. **Time Spent by Component**

This widget displays the total duration of spans for the "Product Application" over the past week. It sums the duration of all spans, specifically categorizing the results by their respective categories. The widget provides insights into the performance and operational aspects of the application, allowing users to identify which categories contribute most to the overall span duration. The results are limited to the top 100 categories for clarity and focus on the most significant contributors.

**Query-**

SELECT sum(duration) from Span where appName = 'Product Application' facet category since 1 week ago limit 100

**Output-**

**A green and white circle with black text

Description automatically generated**

1. **Physical memory utilization**

This widget displays the average physical memory utilization for a specific application monitored by New Relic's APM over the past 8 hours. It calculates the average physical memory usage, specifically filtering for the application identified by its GUID. The widget also compares the current average physical memory utilization with that from 10 minutes ago, enabling users to track memory usage trends and identify any significant changes in resource consumption.

**Query-**

SELECT average(apm.service.memory.physical) AS 'Physical memory utilization' FROM Metric WHERE entity.guid in ('Mzg4NDI0NXxBUE18QVBQTElDQVRJT058MTA0MjI5NjkyOA') LIMIT MAX TIMESERIES SINCE 8 hours ago COMPARE WITH 10 MINUTES ago

**Output-**

**A graph with numbers and lines

Description automatically generated with medium confidence**

1. **Slowest Spans**

This widget displays the average duration of spans for the "Product Application" over the past day. It calculates the average duration, specifically categorizing the results by transaction name. The widget provides insights into the performance of different transactions within the application, allowing users to identify which transactions may be taking longer than expected. The results are displayed as a time series, enabling users to analyze trends in transaction durations over the specified time frame.

**Query-**

SELECT average(duration) from Span WHERE appName = 'Product Application' facet transaction.name since 1 day ago limit max TIMESERIES

**Output-**

**A close-up of a graph

Description automatically generated**

1. **Time consumed by top web transactions**

This widget displays the rate of transaction durations for web transactions in the "Product Application" over the past day. It filters the results to include only those transactions where the transaction name is not null. The widget calculates the rate of the summed transaction durations per second and categorizes the results by transaction name. This allows users to analyze the performance of specific web transactions, providing insights into which transactions are consuming more time and helping identify potential performance issues. The results are presented as a time series for detailed trend analysis.

**Query-**

SELECT filter(rate(sum(apm.service.transaction.duration), 1 second), WHERE transactionName IS NOT NULL) FROM Metric WHERE (entity.guid = 'Mzg4NDI0NXxBUE18QVBQTElDQVRJT058MTA0MjI5NjkyOA') AND (transactionType = 'Web') facet transactionName LIMIT max SINCE 1 day AGO TIMESERIESD

**Output-**

**A graph with purple and white lines

Description automatically generated with medium confidence**

1. **Top 20 Transaction**

This widget displays the count of transactions for the "Product Application" over the past day. It counts all transactions, categorizing the results by transaction name. The widget utilizes extrapolation to estimate transaction counts, providing a clearer view of transaction volume trends. The results are limited to the top 20 transaction names, allowing users to focus on the most significant transactions and gain insights into application usage patterns.

**Query-**

FROM Transaction SELECT count(\*) WHERE appName = 'Product Application' FACET name SINCE 1 day ago EXTRAPOLATE LIMIT 20

**Output-**

**A green line with a white background

Description automatically generated with medium confidence**

1. **Product-Microservice-synthetic Dashboard**
2. **Product synthetics Success in last 1 hour**

This widget displays the success rate of synthetic checks for the "Product Microservice Synthetic" monitor over the past hour. It calculates the percentage of successful checks, specifically filtering for those where the result is 'SUCCESS'. This provides insights into the reliability and performance of the monitored service, allowing users to quickly assess its operational status and identify any potential issues affecting service availability.

**Query-**

FROM SyntheticCheck SELECT percentage(count(\*), WHERE result = 'SUCCESS') AS 'Success Rate' WHERE monitorName = 'Product Microservice Synthetic' SINCE 1 hour ago

**Output-**

**A screenshot of a phone

Description automatically generated**

1. **Product synthetics GB ingested of current month**

This widget displays the estimated amount of data ingested in gigabytes (GB) by the "Product Microservice Synthetic" monitor since the beginning of the month. It calculates the estimate by using the bytecountestimate() function and converting the result to gigabytes. This provides insights into the data usage of the synthetic checks, helping users understand the resource consumption associated with monitoring the performance and availability of the microservice.

**Query-**

SELECT bytecountestimate() / 10e8 as 'Ping GB ingested Estimate' FROM SyntheticCheck, SyntheticRequest, SyntheticsPrivateLocationStatus, SyntheticPrivateMinion SINCE this month where monitorName = 'Product Microservice Synthetic'

**Output-**

A black and white image of a number

Description automatically generated

1. **Latest 2 Product synthetics Checks**

This widget displays the most recent error details and results from synthetic checks for the "Product Microservice Synthetic" monitor over the past hour. It retrieves the error messages and results, specifically limiting the output to the latest two entries. This allows users to quickly identify any issues or failures in the synthetic checks, providing insights into the operational health of the monitored service.

**Query-**

SELECT error, result FROM SyntheticCheck WHERE monitorName = 'Product Microservice Synthetic' SINCE 1 hour ago LIMIT 2

**Output-**

**A screenshot of a computer

Description automatically generated**

1. **Product synthetics Number of Checks by Status**

This widget displays the count of synthetic checks for the "Product Microservice Synthetic" monitor over the past hour, categorized by the result of each check. It counts the total number of checks, providing insights into the performance and reliability of the monitored service. By faceting the results, users can easily see the distribution of outcomes (such as success or failure), helping to quickly assess the operational status of the microservice.

**Query-**

SELECT count(\*) FROM SyntheticCheck FACET result WHERE monitorName = 'Product Microservice Synthetic' since 1 hour ago

**Output-**

**A white background with black and red text

Description automatically generated with medium confidence**

1. **Product synthetics Common ErrorProduct Messages**

This widget displays the count of synthetic check results for the "Product Microservice Synthetic" monitor over the past 10 minutes, categorized by both error type and result. It counts the occurrences of each result, allowing users to identify how many checks succeeded or failed and to see specific error messages associated with failures. This provides a quick overview of the recent performance and reliability of the monitored service, helping to pinpoint any issues that may need attention

**Query-**

FROM SyntheticCheck SELECT count(result) WHERE monitorName = 'Product Microservice Synthetic' SINCE 10 minutes ago FACET error, result

**Output-**

**A white background with black and blue text

Description automatically generated**

1. **Product synthetics Request/response times**

This widget displays the average durations for sending, waiting, and receiving data for the "Product Microservice Synthetic" monitor over time, specifically in 1-minute intervals. It calculates:

Average Send Duration: The average time taken to send requests.

Average Wait Duration: The average time spent waiting for a response.

Average Receive Duration: The average time taken to receive responses.

By presenting these metrics as a time series, users can analyze trends in the performance of the synthetic checks, helping to identify any potential bottlenecks or issues affecting the microservice's responsiveness.

**Query-**

SELECT average(durationSend) AS 'Send', average(durationWait) AS 'Wait', average(durationReceive) AS 'Receive' FROM SyntheticRequest WHERE monitorName = 'Product Microservice Synthetic' TIMESERIES 1 minutes

**Output-**

**A graph with numbers and a number of points

Description automatically generated with medium confidence**

1. **Product synthetics Total Size (KB) over time (Request & Response)**

This widget displays the average request and response sizes for the "Product Microservice Synthetic" monitor over time. It calculates:

Average Request Size (KB): The average size of requests (combining both header and body sizes) divided by 1000 to convert bytes to kilobytes.

Average Response Size (KB): The average size of response bodies divided by 1000 to convert bytes to kilobytes.

By presenting these metrics as a time series, users can analyze trends in the sizes of requests and responses over time, providing insights into the data being transmitted during synthetic checks and helping to identify any changes in communication patterns with the microservice.

**Query-**

select average(totalRequestHeaderSize + totalRequestBodySize) / 1000 as 'Request Size (KB)', average(totalResponseBodySize) / 1000 as 'Response Size (KB)' from SyntheticCheck WHERE monitorName = 'Product Microservice Synthetic' TIMESERIES

**Output-**

**A close-up of a white background

Description automatically generated**

**Checkout Microservice Detailed Analysis**

1. **Deployments -** Deploying a Checkout Microservice involves preparing the environment, building and packaging the service, testing, and using appropriate deployment strategies. Implementing best practices for automation, security, scaling, and monitoring helps ensure a smooth and reliable deployment. Integrating with CI/CD tools and containerization platforms can streamline the process and improve deployment efficiency.

**(I) -** **Successful Builds -** A **successful build** in the context of software development and deployment refers to the process where the source code is compiled, packaged, and tested without errors, resulting in a deployable artifact. This means that the code has been validated through various stages and is ready for deployment to different environments (e.g., staging, production).

**Query-**  **SELECT count(\*) AS 'Successful Builds' from Log WHERE action = 'completed' and workflow\_job.conclusion = 'success' and workflow\_job.workflow\_name = 'Checkout Build and Deploy' since 1 week ago**

**(II) - Failed Builds -** A **failed build** occurs when the process of compiling, testing, or packaging code results in errors or issues that prevent the creation of a valid deployable artifact. This failure can halt the deployment process and requires attention to resolve the underlying problems.

**Query - SELECT count(\*) AS 'Failed Builds' from Log WHERE action = 'completed' and workflow\_job.conclusion = 'failure' and workflow\_job.workflow\_name = 'Checkout Build and Deploy' since 1 week ago**

**(III) - Deployments -** Deployments involve the process of making software or updates available in a production environment. It encompasses several stages and practices to ensure that code changes are delivered efficiently, reliably, and with minimal disruption.

**Query - SELECT count(\*) as Deployments from Log WHERE action = 'completed' and workflow\_job.workflow\_name = 'Checkout Build and Deploy' since 1 week ago TIMESERIES**

**(IV) - Failure Message by Version -** When a build or deployment fails, it's important to capture and manage failure messages effectively, especially when dealing with different versions of your software. This can help in diagnosing issues quickly and ensuring that the correct version is addressed.

**Query - FROM Log SELECT version, failure AS 'failure\_message', guid WHERE workflow\_job.steps LIKE '%CURRENT\_VERSION%' AND failure IS NOT NULL and failure not in (' ' ,' Actual Status 0 : 400, Actual Status 1 : null',' , ' ) and guid = 'Mzg4NDI0NXxBUE18QVBQTElDQVRJT058MTA0MjI5NTA0OQ' since 1 week ago**

**(V) - Successful Deploys by User -** Tracking **successful deployments by user** is important for understanding deployment activities and accountability in your development and operations processes. This helps in recognizing contributions, managing access controls, and auditing changes.

**Query - SELECT count(\*) from Log WHERE action = 'completed' and workflow\_job.conclusion = 'success' AND workflow\_job.workflow\_name = 'Checkout Build and Deploy' FACET sender.login since 1 week ago**

**(VI) - Failed Deploys by User -** Tracking **failed deployments by user** is crucial for identifying issues in the deployment process, understanding responsibility, and improving the deployment pipeline. It helps in analyzing failure patterns and attributing failures to specific users or actions.

**Query - SELECT count(\*) from Log WHERE action = 'completed' and workflow\_job.conclusion = 'failure' AND workflow\_job.workflow\_name = 'Checkout Build and Deploy' FACET sender.login since 1 week ago**

**(VII) - Error In Deployment -** Errors in deployment can occur due to various issues in the deployment process, affecting the stability and availability of the application. Identifying and resolving these errors promptly is crucial for maintaining a reliable deployment pipeline

**Query - SELECT failure as 'failure\_message'from Log WHERE action = 'completed' AND workflow\_job.workflow\_name = 'Checkout Build and Deploy'and failure is not null since 1 week ago**

**(VIII) - Error Deployment History - Error Deployment History** refers to the log or record of all failed deployments over time, including details about the errors encountered. Maintaining a history of failed deployments helps in analyzing recurring issues, improving deployment processes, and ensuring accountability.

**Query - SELECT count(\*) from Log WHERE action = 'completed' AND workflow\_job.workflow\_name = 'Checkout Build and Deploy' and workflow\_job.conclusion ='failure' since 1 week ago COMPARE WITH 1 month ago TIMESERIES**

**(IX) - Deployment Details - Deployment Details** encompass the comprehensive information about each deployment process, including the success or failure status, involved resources, configurations, and other pertinent metadata. Having detailed deployment information helps in tracking changes, troubleshooting issues, and ensuring that deployments are executed correctly.

**Query - SELECT workflow\_job.workflow\_name, workflow\_job.started\_at as 'Start Time', workflow\_job.completed\_at as 'End Time', workflow\_job.conclusion as 'Conslusion', workflow\_job.runner\_name as 'Runner Name', workflow\_job.status as 'Status', version as 'Version', sender.login as 'Deployer' from Log WHERE action = 'completed' AND workflow\_job.workflow\_name = 'Checkout Build and Deploy' and version IS NOT NULL since 1 week ago**

**(X) - Failed Testcase Details - Failed Testcase Details** provide in-depth information about test cases that did not pass during automated testing. This information is crucial for identifying issues in the code, understanding failure patterns, and improving overall software quality.

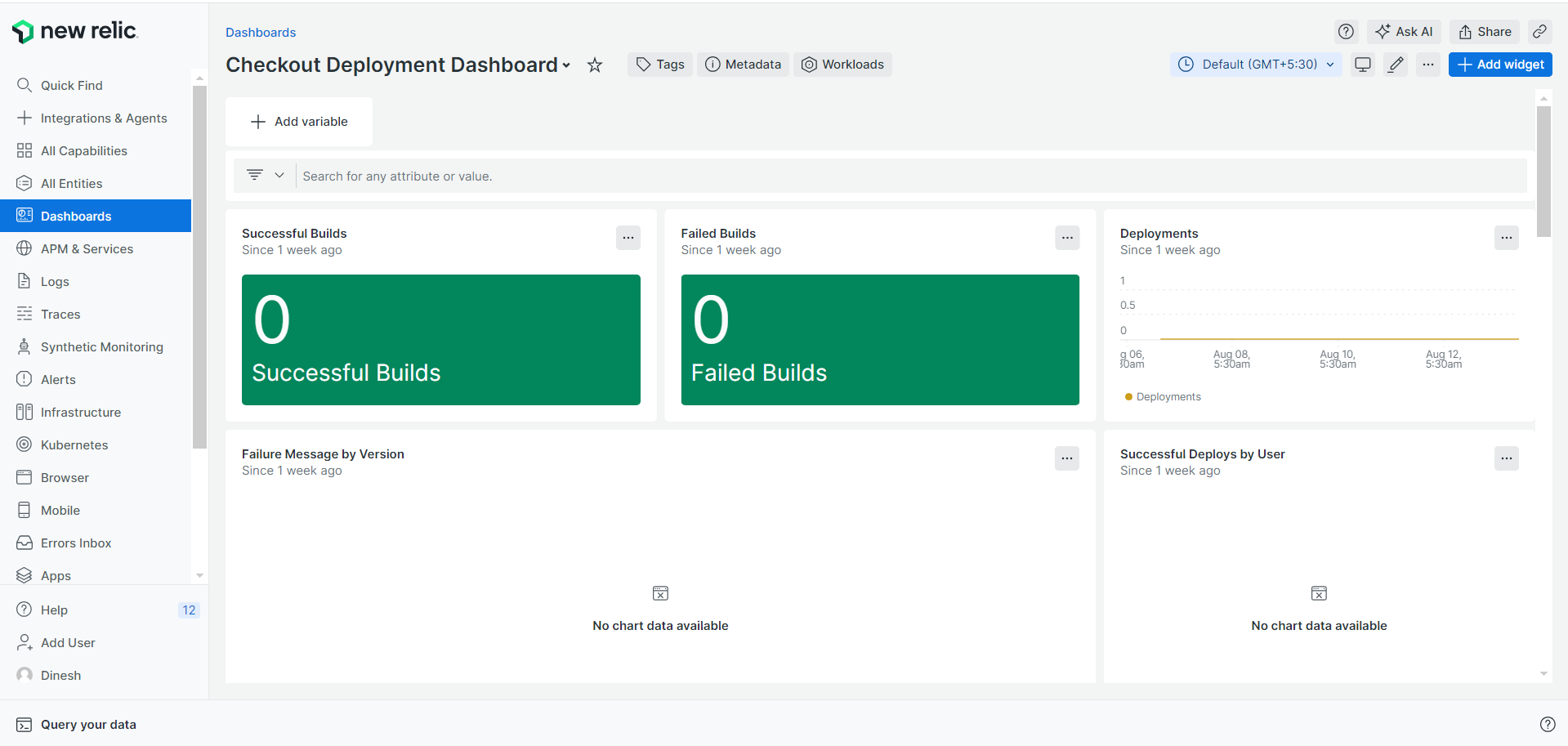
**Query - SELECT version, sender.login as 'deployer' ,testcasestatus01 AS 'TestCaseStatus',testcasetitle01 AS 'TestCaseTitle',workflow\_job.status,workflow\_job.conclusion FROM Log WHERE workflow\_job.workflow\_name = 'Checkout Build and Deploy' and workflow\_job.conclusion = 'failure' and testcasetitle01 is not null SINCE 1 week ago LIMIT MAX**

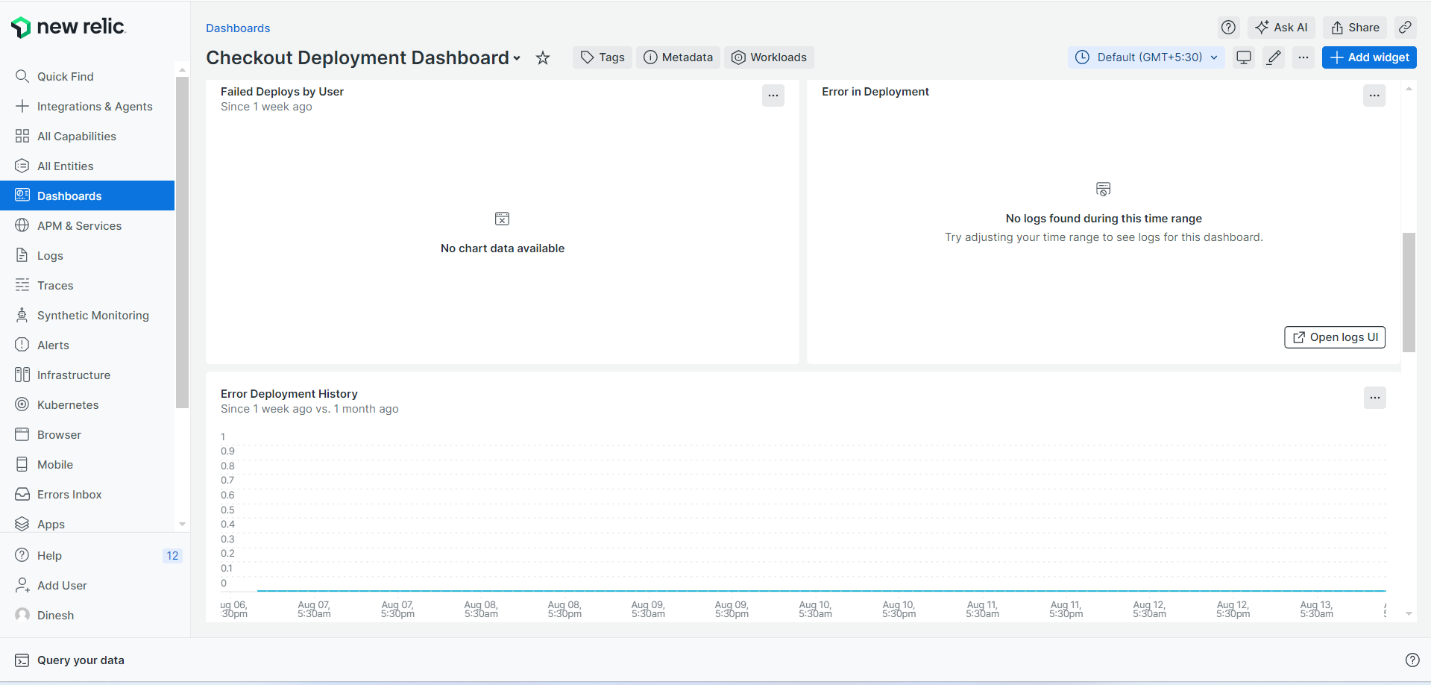
**(XI) - Pod Details - Pod Details** provide comprehensive information about the Kubernetes pods running in your cluster. This includes metadata about the pod, its status, and resource utilization. Understanding pod details is essential for monitoring, debugging, and managing your applications effectively in a Kubernetes environment.

**Query - FROM Log SELECT workflow\_job.workflow\_name,podname,podstatus,version,workflow\_job.conclusion Where workflow\_job.conclusion='failure' and workflow\_job.workflow\_name = 'Checkout Build and Deploy' AND podstatus IS NOT NULL since 15 days ago**

**(XII) - Total Count Of Passed Workflows -** To get the total count of passed workflows, use the respective API or CLI tools provided by your CI/CD system to query and filter the results based on status. Each CI/CD tool has its method for retrieving and counting successful workflows, allowing you to monitor and analyze the effectiveness of your build and deployment processes.

**Query - FROM Log SELECT count(passedtestcase) Where workflow\_job.conclusion = 'success' and workflow\_job.workflow\_name = 'Checkout Build and Deploy' and passedtestcase != 0 and passedtestcase IS NOT NULL since 1 week ago**





1. **Test Cases - Test** **Cases** for a checkout microservice are designed to validate that the service functions correctly, securely, and efficiently under various conditions. They help ensure that users have a smooth and reliable experience when completing their purchases.

**(I) - Filter by Version -** Filtering by version allows you to focus on specific versions of software or data, making it easier to track deployments, review changes, monitor performance, and analyze test results. This functionality helps in managing and diagnosing issues related to different versions effectively.

**Query - SELECT latest(timestamp) FROM Log WHERE workflow\_job.workflow\_name = 'Checkout Build and Deploy' and guid = 'Mzg4NDI0NXxBUE18QVBQTElDQVRJT058MTA0MjI5NTA0OQ' FACET version, sender.login, workflow\_job.conclusion ,testcasestatus01 AS 'TestCaseStatus', testcasetitle01 AS 'TestCaseTitle' SINCE 1 weeks ago LIMIT 10**

**(II) - Pod Details - Pod details** are crucial for managing and troubleshooting Kubernetes applications. Pods are the smallest deployable units in Kubernetes, and they represent a single instance of a running process in your cluster.

**Query - SELECT latest(timestamp) FROM Log WHERE workflow\_job.conclusion = 'failure' AND workflow\_job.workflow\_name = 'Checkout Build and Deploy' FACET version,podname,podstatus,sender.login as 'deployer',workflow\_job.status,workflow\_job.conclusion SINCE 1 week ago**

**(III) - Count of pod failure message 1 week vs 1 month -** Counting and analyzing pod failure messages over different time periods provides valuable insights into the stability and reliability of your Kubernetes environment. It helps you monitor recent and historical performance, identify patterns, diagnose issues, and make informed decisions about resource management and scaling. Understanding these metrics is essential for maintaining a robust and reliable application infrastructure.

**Query - SELECT count(podstatus) FROM Log WHERE workflow\_job.conclusion = 'failure' AND workflow\_job.workflow\_name = 'Checkout Build and Deploy' and podstatus != 'running' SINCE 1 week ago COMPARE WITH 1 month ago TIMESERIES**

**(IV) - podStatus count -** Counting pods by status helps you monitor and manage your Kubernetes environment effectively. Using kubectl, Kubernetes dashboards, or monitoring tools, you can get insights into the number of pods in different states (Running, Pending, Failed, etc.), which is useful for understanding the health and performance of your cluster.

**Query - SELECT count(podstatus) FROM Log Where workflow\_job.conclusion='failure' and workflow\_job.workflow\_name = 'Checkout Build and Deploy' AND podstatus IS NOT NULL FACET podstatus since 1 week ago**

**(V) - Count by Testcasefailure message -** To count the number of test case failures based on failure messages, you typically need to analyze test results and logs.

**Query - SELECT count(failure) FROM Log Where workflow\_job.conclusion='failure' and workflow\_job.workflow\_name = 'Checkout Build and Deploy' FACET failure as 'failure\_message'since 15 days ago**

**(VI) - Testcases Failures Detailed View -** When you're examining **test case failures** in detail, you are essentially analyzing the specific reasons why certain tests did not pass during a test run. This detailed view is crucial for diagnosing issues and improving the quality of your software.

**Query - SELECT version, sender.login as 'deployer' ,testcasestatus01 AS 'TestCaseStatus',testcasetitle01 AS 'TestCaseTitle',workflow\_job.status,workflow\_job.conclusion FROM Log WHERE workflow\_job.workflow\_name = 'Checkout Build and Deploy' and workflow\_job.conclusion = 'failure' and testcasetitle01 is not null SINCE 1 week ago LIMIT MAX**

1. **Performance -** The performance of the Checkout microservice is vital for the user experience and the overall success of your application. By monitoring key metrics such as response time, throughput, error rates, and resource utilization, and by conducting load and stress tests, you can ensure that the service is reliable, scalable, and efficient. Continuous monitoring, along with optimization techniques like caching and database tuning, will help maintain high performance even under heavy load.

**(I)- Response time (ms) -** Response time in milliseconds (ms) is a critical performance metric that measures the time it takes for the Checkout microservice to respond to a request. This includes the time from when the client requests to when the response is receive.

**Query - SELECT average(newrelic.goldenmetrics.apm.application.responseTimeMs) AS 'Response Time Ms' FROM Metric WHERE entity.guid in ('Mzg4NDI0NXxBUE18QVBQTElDQVRJT058MTA0MjI5NTA0OQ') LIMIT MAX TIMESERIES SINCE 8 hours ago COMPARE WITH 10 minutes ago**

**(II) - Throughput - Throughput** is a key performance metric that measures the number of transactions or requests that a system, like the Checkout microservice, can handle within a specific time frame, usually measured in requests per second (RPS) or transactions per second (TPS).

**Query - SELECT rate(count(newrelic.goldenmetrics.apm.application.throughput), 1 MINUTES) AS 'Throughput' FROM Metric WHERE entity.guid in ('Mzg4NDI0NXxBUE18QVBQTElDQVRJT058MTA0MjI5NTA0OQ') LIMIT MAX TIMESERIES SINCE 8 hours ago COMPARE WITH 10 MINUTES ago**

**(III) - Error rate -** The error rate is a vital metric for ensuring the reliability and user experience of the Checkout microservice. By monitoring and optimizing the error rate, you can quickly identify and resolve issues that might otherwise lead to user dissatisfaction or lost revenue. Maintaining a low error rate is key to ensuring a smooth and successful checkout process for users.

**Query - SELECT average(newrelic.goldenmetrics.apm.application.errorRate) AS 'Error Rate' FROM Metric WHERE entity.guid in ('Mzg4NDI0NXxBUE18QVBQTElDQVRJT058MTA0MjI5NTA0OQ') LIMIT MAX TIMESERIES SINCE 8 hours ago COMPARE WITH 10 MINUTES ago**

**(IV) - CPU utilization -** CPU utilization is a critical metric for understanding how efficiently the Checkout microservice is using available processing power. By regularly monitoring and optimizing CPU usage, you can ensure that the microservice performs well under load, scales effectively, and operates within cost-efficient parameters. Keeping CPU utilization within an optimal range helps maintain a responsive and reliable service, especially during peak traffic periods.

**Query - SELECT average(apm.service.cpu.usertime.utilization) AS 'CPU utilization' FROM Metric WHERE entity.guid in ('Mzg4NDI0NXxBUE18QVBQTElDQVRJT058MTA0MjI5NTA0OQ') LIMIT MAX TIMESERIES SINCE 8 hours ago COMPARE WITH 10 MINUTES ago**

**(V) - Apdex Score -** The **Apdex score** is a user satisfaction metric that measures how well an application or service, like the Checkout microservice, performs from the perspective of its users.

**Query - SELECT apdex(apm.service.apdex) AS 'Apdex score' FROM Metric WHERE entity.guid in ('Mzg4NDI0NXxBUE18QVBQTElDQVRJT058MTA0MjI5NTA0OQ') LIMIT MAX TIMESERIES SINCE 8 hours ago COMPARE WITH 10 MINUTES ago**

**(VI) - Slowest Traces over last Hour -** In New Relic, you can identify the **slowest traces** over the last hour to pinpoint performance bottlenecks within your application, such as the Checkout microservice. Traces provide detailed information about individual transactions or requests, allowing you to see exactly where delays are occurring.

**Query - SELECT average(duration) FROM Span WHERE appName = 'Checkout Application' FACET traceId since 1 hour ago limit max**

**(VII) - Physical memory utilization - Physical memory utilization** refers to the amount of physical RAM (Random Access Memory) being used by the system or a specific application, such as the Checkout microservice. This metric is crucial for understanding how much memory is consumed by your service and how efficiently it's managing its memory resources.

**Query - SELECT average(apm.service.memory.physical) AS 'Physical memory utilization' FROM Metric WHERE entity.guid in ('Mzg4NDI0NXxBUE18QVBQTElDQVRJT058MTA0MjI5NTA0OQ') LIMIT MAX TIMESERIES SINCE 8 hours ago COMPARE WITH 10 MINUTES ago**

**(VIII) - Slowest Spans - Slowest spans** in a distributed tracing system, such as the one you might monitor using New Relic, refer to the individual operations or segments of a transaction that take the longest to execute. A "span" represents a single unit of work done by a service or microservice as part of a larger transaction or trace. Identifying the slowest spans is crucial for understanding where delays are occurring within a distributed system, like your Checkout microservice.

**Query - SELECT average(duration) from Span WHERE appName = 'Checkout Application' facet transaction.name since 1 day ago limit max TIMESERIES**

**(IX) - Time Spent by Component - Time Spent by Component** refers to the amount of time a transaction or request spends in each individual component of a system during its lifecycle. In a distributed application, such as a microservices architecture, this metric helps you understand how different parts of the system contribute to the overall response time of a request.

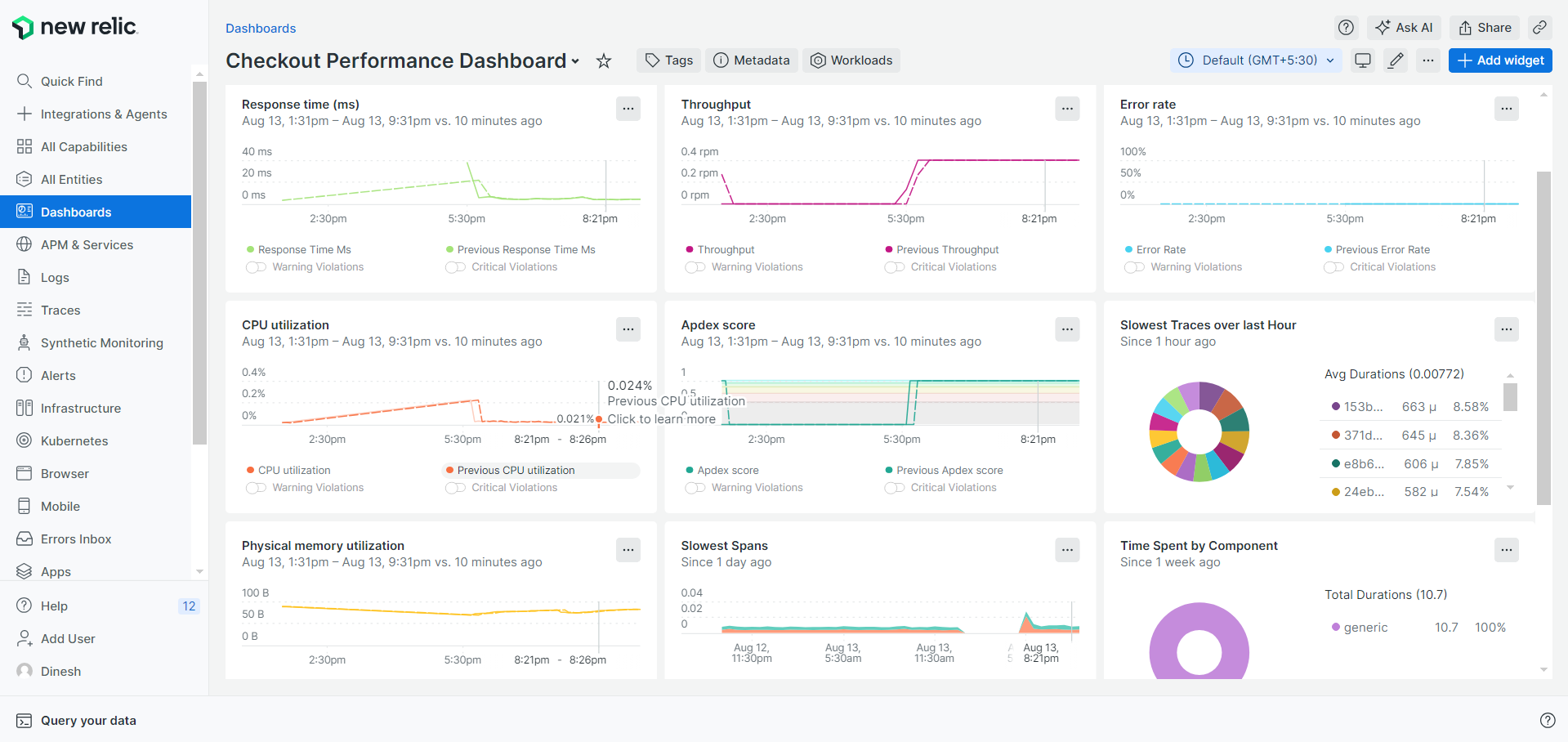
**Query - SELECT sum(duration) from Span where appName = 'Checkout Application' facet category since 1 week ago limit 100**

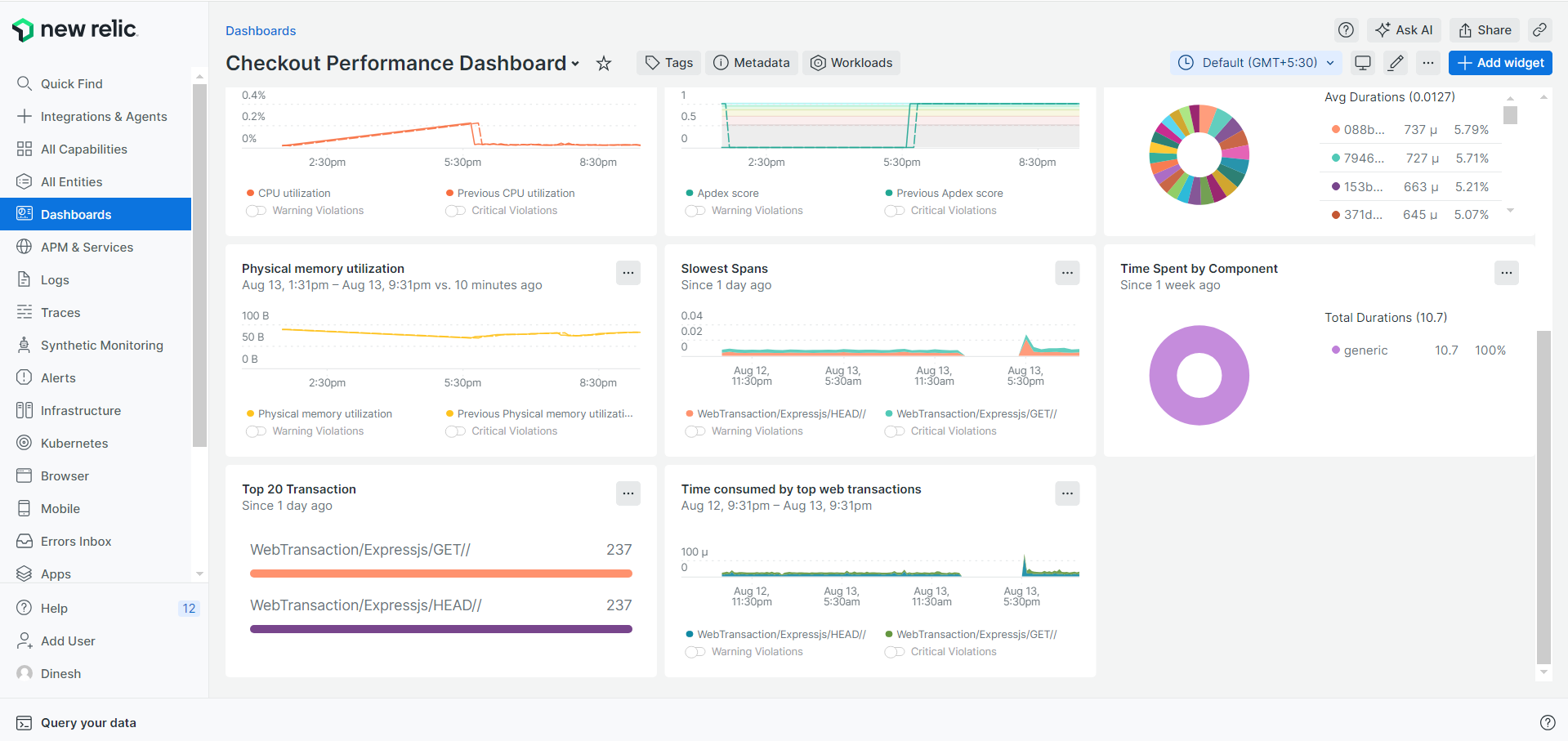
**(X) - Top 20 Transaction - Top 20 Transactions** typically refers to the 20 most critical or highest-impact transactions in terms of performance, volume, or user impact. In the context of application monitoring with New Relic, identifying the top transactions helps prioritize optimization efforts and monitor the most important interactions within your application.

**Query - FROM Transaction SELECT count(\*) WHERE appName = 'Checkout Application' FACET name SINCE 1 day ago EXTRAPOLATE LIMIT 20**

**(XI) - Time consumed by top web transactions - Time Consumed by Top Web Transactions** refers to the amount of time that each of the most significant web transactions takes to complete. In application monitoring, particularly in tools like New Relic, this metric helps you identify which web transactions are taking the most time and potentially causing performance issues.

**Query - SELECT filter(rate(sum(apm.service.transaction.duration), 1 second), WHERE transactionName IS NOT NULL) FROM Metric WHERE (entity.guid = 'Mzg4NDI0NXxBUE18QVBQTElDQVRJT058MTA0MjI5NTA0OQ') AND (transactionType = 'Web') facet transactionName LIMIT max SINCE 1 day AGO TIMESERIES**





**4 - Throughput - Throughput** in the context of a microservice, such as the Checkout microservice, refers to the number of requests or transactions processed by the service within a given period. Monitoring throughput helps you understand the load on your microservice and ensures that it can handle the volume of traffic it receives.

**(I)- Response time (ms) -** Response time in milliseconds (ms) is a critical performance metric that measures the time it takes for the Checkout microservice to respond to a request. This includes the time from when the client requests to when the response is receive.

**Query - SELECT average(newrelic.goldenmetrics.apm.application.responseTimeMs) AS 'Response Time Ms' FROM Metric WHERE entity.guid in ('Mzg4NDI0NXxBUE18QVBQTElDQVRJT058MTA0MjI5NTA0OQ') LIMIT MAX TIMESERIES SINCE 8 hours ago COMPARE WITH 10 minutes ago**

**(II) - Throughput - Throughput** is a key performance metric that measures the number of transactions or requests that a system, like the Checkout microservice, can handle within a specific time frame, usually measured in requests per second (RPS) or transactions per second (TPS).

**Query - SELECT rate(count(newrelic.goldenmetrics.apm.application.throughput), 1 MINUTES) AS 'Throughput' FROM Metric WHERE entity.guid in ('Mzg4NDI0NXxBUE18QVBQTElDQVRJT058MTA0MjI5NTA0OQ') LIMIT MAX TIMESERIES SINCE 8 hours ago COMPARE WITH 10 MINUTES ago**

**(III) - Error rate -** The error rate is a vital metric for ensuring the reliability and user experience of the Checkout microservice. By monitoring and optimizing the error rate, you can quickly identify and resolve issues that might otherwise lead to user dissatisfaction or lost revenue. Maintaining a low error rate is key to ensuring a smooth and successful checkout process for users.

**Query - SELECT average(newrelic.goldenmetrics.apm.application.errorRate) AS 'Error Rate' FROM Metric WHERE entity.guid in ('Mzg4NDI0NXxBUE18QVBQTElDQVRJT058MTA0MjI5NTA0OQ') LIMIT MAX TIMESERIES SINCE 8 hours ago COMPARE WITH 10 MINUTES ago**

**(IV) - CPU utilization -** CPU utilization is a critical metric for understanding how efficiently the Checkout microservice is using available processing power. By regularly monitoring and optimizing CPU usage, you can ensure that the microservice performs well under load, scales effectively, and operates within cost-efficient parameters. Keeping CPU utilization within an optimal range helps maintain a responsive and reliable service, especially during peak traffic periods.

**Query - SELECT average(apm.service.cpu.usertime.utilization) AS 'CPU utilization' FROM Metric WHERE entity.guid in ('Mzg4NDI0NXxBUE18QVBQTElDQVRJT058MTA0MjI5NTA0OQ') LIMIT MAX TIMESERIES SINCE 8 hours ago COMPARE WITH 10 MINUTES ago**

**(V) - Apdex Score -** The **Apdex score** is a user satisfaction metric that measures how well an application or service, like the Checkout microservice, performs from the perspective of its users.

**Query - SELECT apdex(apm.service.apdex) AS 'Apdex score' FROM Metric WHERE entity.guid in ('Mzg4NDI0NXxBUE18QVBQTElDQVRJT058MTA0MjI5NTA0OQ') LIMIT MAX TIMESERIES SINCE 8 hours ago COMPARE WITH 10 MINUTES ago**

**(VI) - Slowest Traces over last Hour -** In New Relic, you can identify the **slowest traces** over the last hour to pinpoint performance bottlenecks within your application, such as the Checkout microservice. Traces provide detailed information about individual transactions or requests, allowing you to see exactly where delays are occurring.

**Query - SELECT average(duration) FROM Span WHERE appName = 'Checkout Application' FACET traceId since 1 hour ago limit max**

**(VII) - Physical memory utilization - Physical memory utilization** refers to the amount of physical RAM (Random Access Memory) being used by the system or a specific application, such as the Checkout microservice. This metric is crucial for understanding how much memory is consumed by your service and how efficiently it's managing its memory resources.

**Query - SELECT average(apm.service.memory.physical) AS 'Physical memory utilization' FROM Metric WHERE entity.guid in ('Mzg4NDI0NXxBUE18QVBQTElDQVRJT058MTA0MjI5NTA0OQ') LIMIT MAX TIMESERIES SINCE 8 hours ago COMPARE WITH 10 MINUTES ago**

**(VIII) - Slowest Spans - Slowest spans** in a distributed tracing system, such as the one you might monitor using New Relic, refer to the individual operations or segments of a transaction that take the longest to execute. A "span" represents a single unit of work done by a service or microservice as part of a larger transaction or trace. Identifying the slowest spans is crucial for understanding where delays are occurring within a distributed system, like your Checkout microservice.

**Query - SELECT average(duration) from Span WHERE appName = 'Checkout Application' facet transaction.name since 1 day ago limit max TIMESERIES**

**(IX) - Time Spent by Component - Time Spent by Component** refers to the amount of time a transaction or request spends in each individual component of a system during its lifecycle. In a distributed application, such as a microservices architecture, this metric helps you understand how different parts of the system contribute to the overall response time of a request.

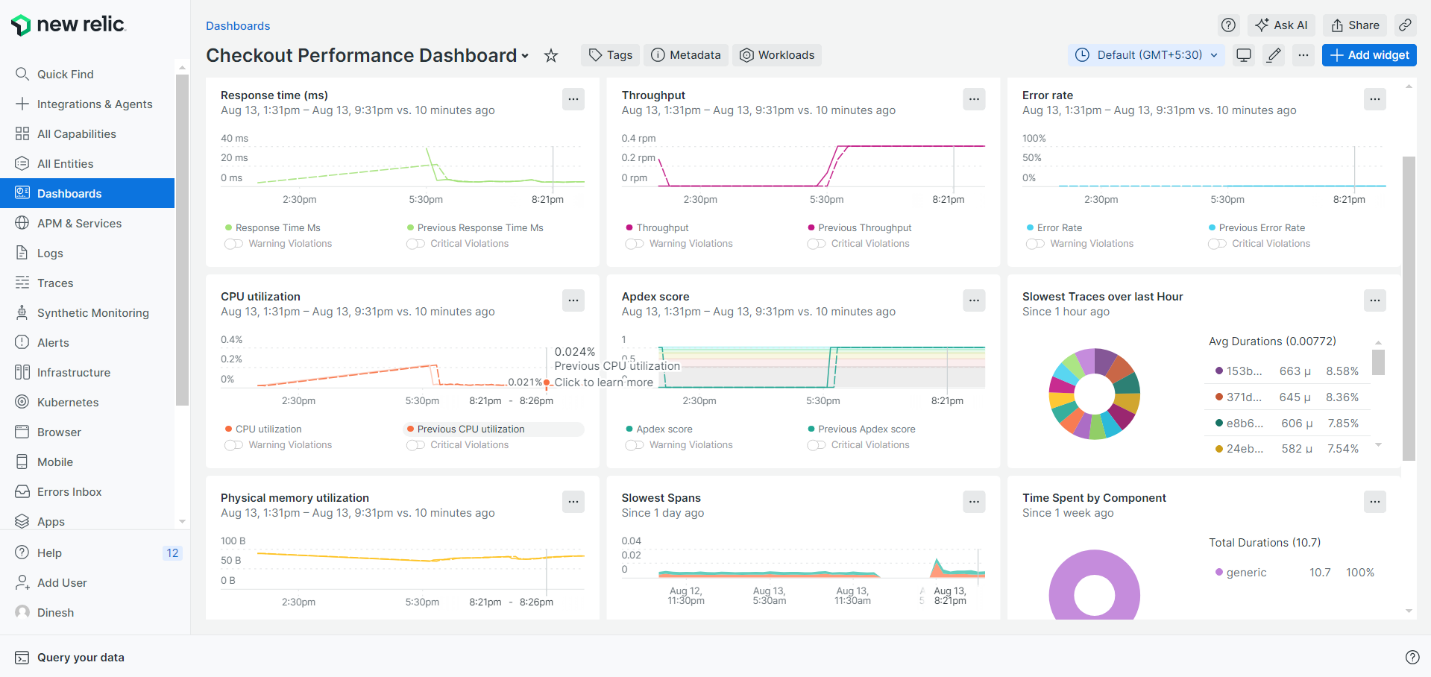
**Query - SELECT sum(duration) from Span where appName = 'Checkout Application' facet category since 1 week ago limit 100**

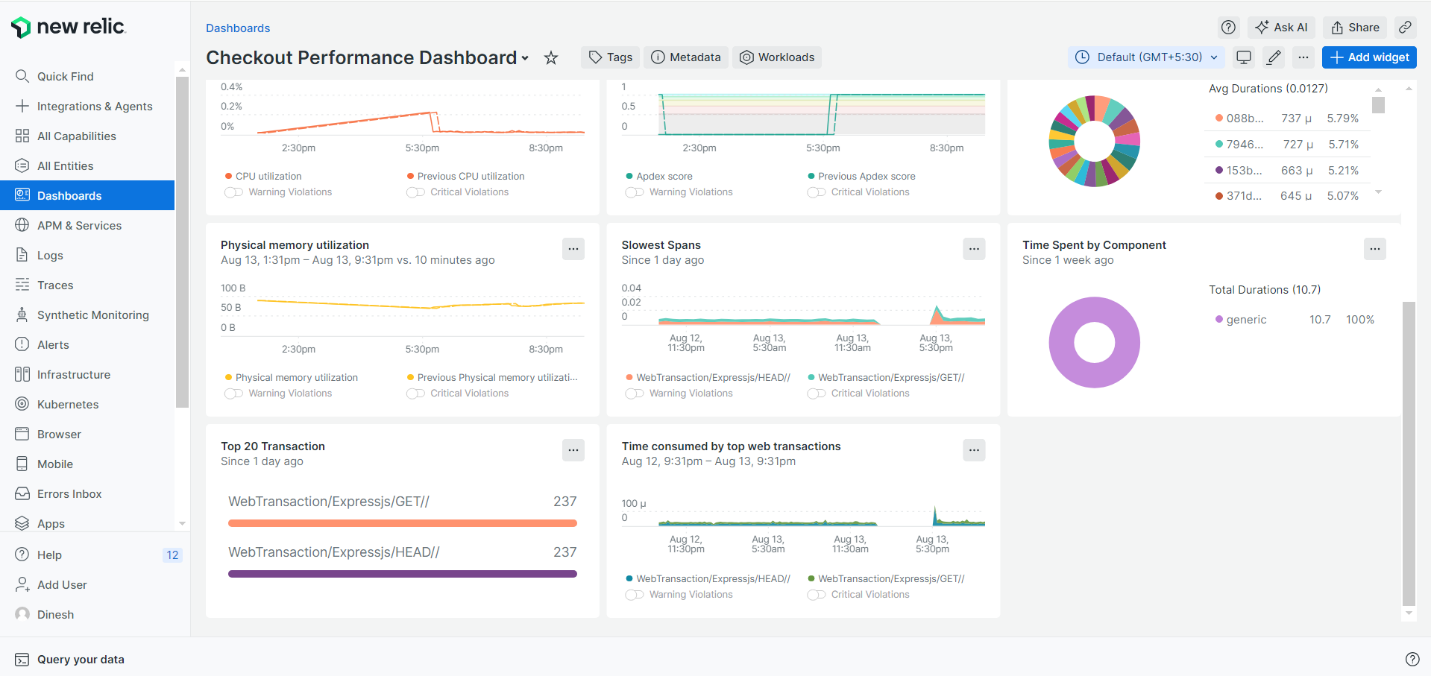
**(X) - Top 20 Transaction - Top 20 Transactions** typically refers to the 20 most critical or highest-impact transactions in terms of performance, volume, or user impact. In the context of application monitoring with New Relic, identifying the top transactions helps prioritize optimization efforts and monitor the most important interactions within your application.

**Query - FROM Transaction SELECT count(\*) WHERE appName = 'Checkout Application' FACET name SINCE 1 day ago EXTRAPOLATE LIMIT 20**

**(XI) - Time consumed by top web transactions - Time Consumed by Top Web Transactions** refers to the amount of time that each of the most significant web transactions takes to complete. In application monitoring, particularly in tools like New Relic, this metric helps you identify which web transactions are taking the most time and potentially causing performance issues.

**Query - SELECT filter(rate(sum(apm.service.transaction.duration), 1 second), WHERE transactionName IS NOT NULL) FROM Metric WHERE (entity.guid = 'Mzg4NDI0NXxBUE18QVBQTElDQVRJT058MTA0MjI5NTA0OQ') AND (transactionType = 'Web') facet transactionName LIMIT max SINCE 1 day AGO TIMESERIES**





**5- Error Rate** - **Error Rate** in a microservice, such as the Checkout microservice, measures the percentage or number of failed requests relative to the total number of requests processed by the service. This metric helps in assessing the reliability and stability of the microservice, and identifying issues that may affect user experience.

**(I)- Response time (ms) -** Response time in milliseconds (ms) is a critical performance metric that measures the time it takes for the Checkout microservice to respond to a request. This includes the time from when the client requests to when the response is receive.

**Query - SELECT average(newrelic.goldenmetrics.apm.application.responseTimeMs) AS 'Response Time Ms' FROM Metric WHERE entity.guid in ('Mzg4NDI0NXxBUE18QVBQTElDQVRJT058MTA0MjI5NTA0OQ') LIMIT MAX TIMESERIES SINCE 8 hours ago COMPARE WITH 10 minutes ago**

**(II) - Throughput - Throughput** is a key performance metric that measures the number of transactions or requests that a system, like the Checkout microservice, can handle within a specific time frame, usually measured in requests per second (RPS) or transactions per second (TPS).

**Query - SELECT rate(count(newrelic.goldenmetrics.apm.application.throughput), 1 MINUTES) AS 'Throughput' FROM Metric WHERE entity.guid in ('Mzg4NDI0NXxBUE18QVBQTElDQVRJT058MTA0MjI5NTA0OQ') LIMIT MAX TIMESERIES SINCE 8 hours ago COMPARE WITH 10 MINUTES ago**

**(III) - Error rate -** The error rate is a vital metric for ensuring the reliability and user experience of the Checkout microservice. By monitoring and optimizing the error rate, you can quickly identify and resolve issues that might otherwise lead to user dissatisfaction or lost revenue. Maintaining a low error rate is key to ensuring a smooth and successful checkout process for users.

**Query - SELECT average(newrelic.goldenmetrics.apm.application.errorRate) AS 'Error Rate' FROM Metric WHERE entity.guid in ('Mzg4NDI0NXxBUE18QVBQTElDQVRJT058MTA0MjI5NTA0OQ') LIMIT MAX TIMESERIES SINCE 8 hours ago COMPARE WITH 10 MINUTES ago**

**(IV) - CPU utilization -** CPU utilization is a critical metric for understanding how efficiently the Checkout microservice is using available processing power. By regularly monitoring and optimizing CPU usage, you can ensure that the microservice performs well under load, scales effectively, and operates within cost-efficient parameters. Keeping CPU utilization within an optimal range helps maintain a responsive and reliable service, especially during peak traffic periods.

**Query - SELECT average(apm.service.cpu.usertime.utilization) AS 'CPU utilization' FROM Metric WHERE entity.guid in ('Mzg4NDI0NXxBUE18QVBQTElDQVRJT058MTA0MjI5NTA0OQ') LIMIT MAX TIMESERIES SINCE 8 hours ago COMPARE WITH 10 MINUTES ago**

**(V) - Apdex Score -** The **Apdex score** is a user satisfaction metric that measures how well an application or service, like the Checkout microservice, performs from the perspective of its users.

**Query - SELECT apdex(apm.service.apdex) AS 'Apdex score' FROM Metric WHERE entity.guid in ('Mzg4NDI0NXxBUE18QVBQTElDQVRJT058MTA0MjI5NTA0OQ') LIMIT MAX TIMESERIES SINCE 8 hours ago COMPARE WITH 10 MINUTES ago**

**(VI) - Slowest Traces over last Hour -** In New Relic, you can identify the **slowest traces** over the last hour to pinpoint performance bottlenecks within your application, such as the Checkout microservice. Traces provide detailed information about individual transactions or requests, allowing you to see exactly where delays are occurring.

**Query - SELECT average(duration) FROM Span WHERE appName = 'Checkout Application' FACET traceId since 1 hour ago limit max**

**(VII) - Physical memory utilization - Physical memory utilization** refers to the amount of physical RAM (Random Access Memory) being used by the system or a specific application, such as the Checkout microservice. This metric is crucial for understanding how much memory is consumed by your service and how efficiently it's managing its memory resources.

**Query - SELECT average(apm.service.memory.physical) AS 'Physical memory utilization' FROM Metric WHERE entity.guid in ('Mzg4NDI0NXxBUE18QVBQTElDQVRJT058MTA0MjI5NTA0OQ') LIMIT MAX TIMESERIES SINCE 8 hours ago COMPARE WITH 10 MINUTES ago**

**(VIII) - Slowest Spans - Slowest spans** in a distributed tracing system, such as the one you might monitor using New Relic, refer to the individual operations or segments of a transaction that take the longest to execute. A "span" represents a single unit of work done by a service or microservice as part of a larger transaction or trace. Identifying the slowest spans is crucial for understanding where delays are occurring within a distributed system, like your Checkout microservice.

**Query - SELECT average(duration) from Span WHERE appName = 'Checkout Application' facet transaction.name since 1 day ago limit max TIMESERIES**

**(IX) - Time Spent by Component - Time Spent by Component** refers to the amount of time a transaction or request spends in each individual component of a system during its lifecycle. In a distributed application, such as a microservices architecture, this metric helps you understand how different parts of the system contribute to the overall response time of a request.

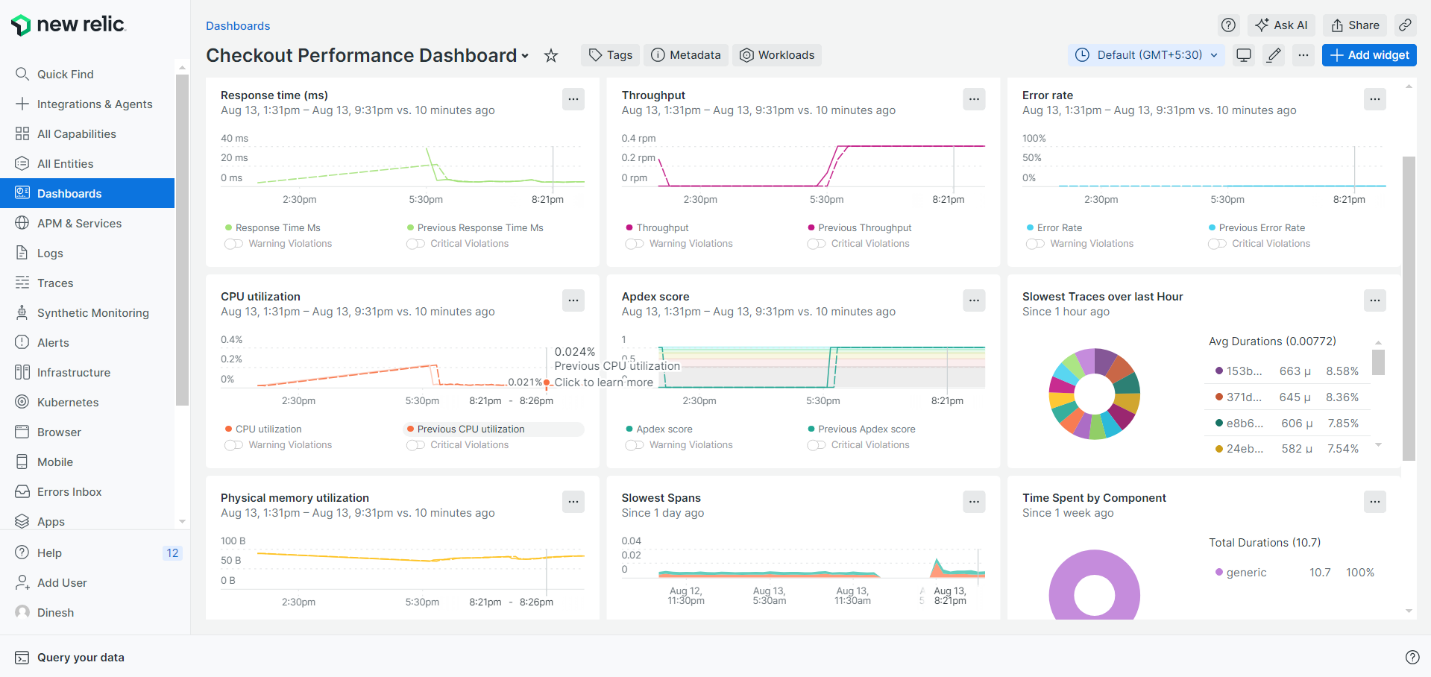
**Query - SELECT sum(duration) from Span where appName = 'Checkout Application' facet category since 1 week ago limit 100**

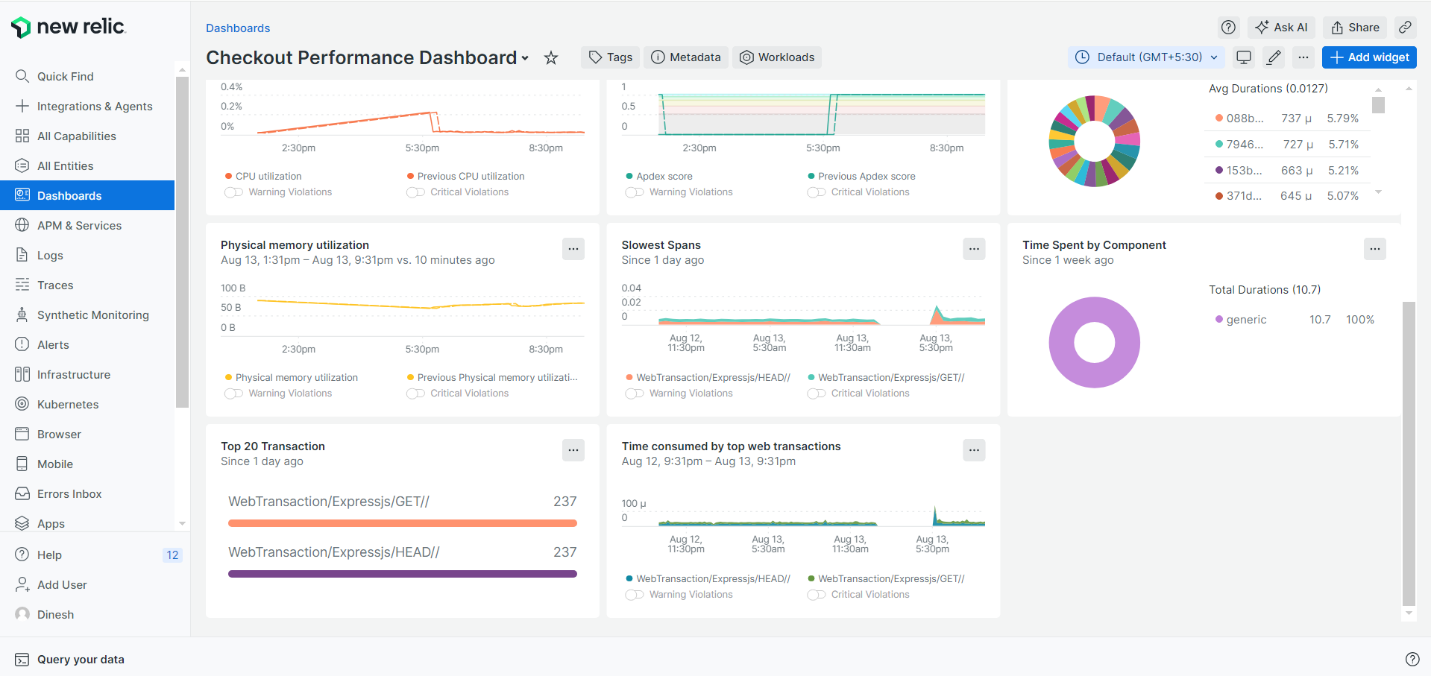
**(X) - Top 20 Transaction - Top 20 Transactions** typically refers to the 20 most critical or highest-impact transactions in terms of performance, volume, or user impact. In the context of application monitoring with New Relic, identifying the top transactions helps prioritize optimization efforts and monitor the most important interactions within your application.

**Query - FROM Transaction SELECT count(\*) WHERE appName = 'Checkout Application' FACET name SINCE 1 day ago EXTRAPOLATE LIMIT 20**

**(XI) - Time consumed by top web transactions - Time Consumed by Top Web Transactions** refers to the amount of time that each of the most significant web transactions takes to complete. In application monitoring, particularly in tools like New Relic, this metric helps you identify which web transactions are taking the most time and potentially causing performance issues.

**Query - SELECT filter(rate(sum(apm.service.transaction.duration), 1 second), WHERE transactionName IS NOT NULL) FROM Metric WHERE (entity.guid = 'Mzg4NDI0NXxBUE18QVBQTElDQVRJT058MTA0MjI5NTA0OQ') AND (transactionType = 'Web') facet transactionName LIMIT max SINCE 1 day AGO TIMESERIES**





**6 - Synthetic Monitor - Synthetic Monitoring** in the context of microservices, such as the Checkout microservice, involves simulating user interactions with the service to monitor its availability, performance, and functionality from an external perspective. Synthetic monitoring helps proactively identify issues before they affect real users by running automated tests and simulations.

**(I) -** **Checkout synthetics Success in last 1 hour - Synthetic Monitor Success Rate** refers to the percentage of successful tests conducted by synthetic monitors compared to the total number of tests performed.

**Query - FROM SyntheticCheck SELECT percentage(count(\*), WHERE result = 'SUCCESS') AS 'Success Rate' WHERE monitorName = 'Checkout Microservice Synthetic' SINCE 1 hour ago**

**(II) - Checkout synthetics GB ingested of current month - GB Ingested** in the context of synthetic monitoring refers to the amount of data collected and sent to New Relic by the synthetic monitors. This typically includes the data generated by running synthetic tests, such as the size of test results, scripts, and any associated logs.

**Query - SELECT bytecountestimate() / 10e8 as 'Ping GB ingested Estimate' FROM SyntheticCheck, SyntheticRequest, SyntheticsPrivateLocationStatus, SyntheticPrivateMinion SINCE this month where monitorName = 'Checkout Microservice Synthetic'**

**(III) - Latest 2 Checkout synthetics Checks -** By reviewing the latest two synthetic checks for the Checkout microservice, you can quickly assess recent performance and reliability. This helps in identifying.

**Query - SELECT error, result FROM SyntheticCheck WHERE monitorName = 'Checkout Microservice Synthetic' SINCE 1 hour ago LIMIT 2**

**(IV) - Checkout synthetics Number of Checks by Status -** Viewing the number of synthetic checks by status for the Checkout microservice helps you gauge the overall performance and reliability of your service. By analyzing these metrics, you can identify trends, detect issues early, and ensure that the checkout process remains functional and efficient. Regular monitoring of these statistics is crucial for maintaining service quality and improving user experience.

**Query - SELECT count(\*) FROM SyntheticCheck FACET result WHERE monitorName = 'Checkout Microservice Synthetic' since 1 hour ago**

**(V) - Checkout Common Error synthetics Messages - Common Error Messages** from synthetic monitoring for the Checkout microservice typically indicate issues or failures encountered during automated tests. These errors can provide valuable insights into problems with the checkout process, helping you diagnose and address issues before they affect real users.

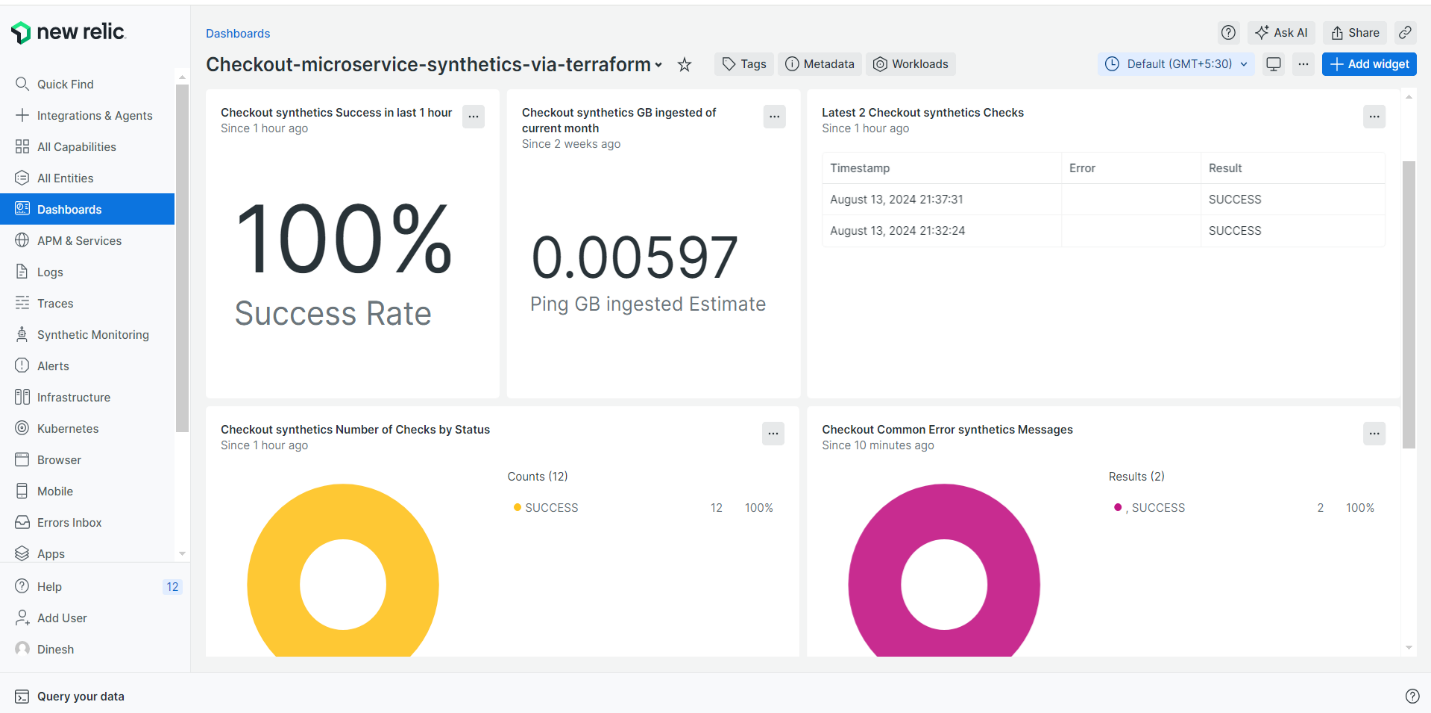
**Query - FROM SyntheticCheck SELECT count(result) WHERE monitorName = 'Checkout Microservice Synthetic' SINCE 10 minutes ago FACET error, result**

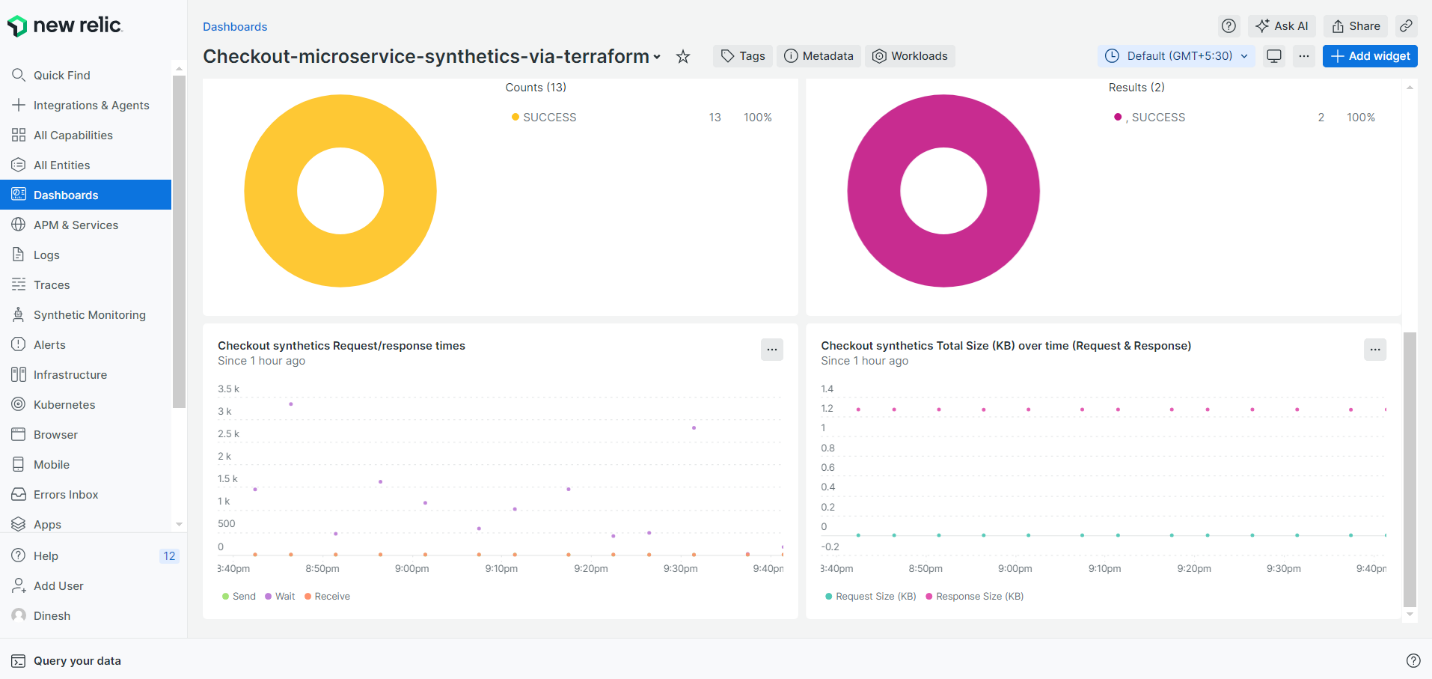
**(VI) - Checkout synthetics Request/response times - Request/Response Times** in synthetic monitoring for the Checkout microservice measure how long it takes for requests to be sent to the service and responses to be received. These metrics are crucial for understanding the performance and efficiency of the checkout process.

**Query - SELECT average(durationSend) AS 'Send', average(durationWait) AS 'Wait', average(durationReceive) AS 'Receive' FROM SyntheticRequest WHERE monitorName = 'Checkout Microservice Synthetic' TIMESERIES 1 minutes**

**(VII) - Checkout synthetics Total Size (KB) over time (Request & Response) -** Tracking the total size of request and response data for synthetic monitoring of the Checkout microservice provides insights into data usage and helps in managing bandwidth and storage. By analyzing these metrics over time, you can identify trends, optimize data payloads, and ensure efficient performance monitoring. Regular review and optimization are essential for maintaining effective monitoring practices and managing resource usage.

**Query - select average(totalRequestHeaderSize + totalRequestBodySize) / 1000 as 'Request Size (KB)', average(totalResponseBodySize) / 1000 as 'Response Size (KB)' from SyntheticCheck WHERE monitorName = 'Checkout Microservice Synthetic' TIMESERIES**





**7 - Errors -** To troubleshoot errors in your checkout microservice using New Relic, start by navigating to the APM section and selecting the relevant microservice. Check the "Errors" tab to view a summary of recent errors and their details. You can also examine transaction traces to understand the context of each error. Reviewing logs associated with the errors can provide deeper insights into their causes. Additionally, monitor related performance metrics like response times and throughput to see if there are any correlations. Setting up alerts for critical errors can help in catching issues early.

**8- Change Log -** Deployer: cloudEQ , Comment: sevice down, required changes made - The Change Log entry shows that the deployer, cloudEQ, identified a service outage. In response, they implemented required changes to restore functionality. The specific changes might involve configuration adjustments, code fixes, or updates to infrastructure components to resolve the issue and bring the service back online. The details of the modifications are likely documented further in the Change Log or deployment notes.

**9- SonarQube -** SonarQube is used in the checkout microservice to analyze code quality and identify potential issues such as bugs, vulnerabilities, and code smells. It continuously scans the codebase to ensure that the code adheres to best practices and maintains a high standard of quality. Regular analysis with SonarQube helps in detecting problems early in the development process, improving the overall stability and maintainability of the checkout microservice.

**LOG PARSING**

**Log parsing in New Relic involves extracting and structuring log data to make it searchable and easy to analyze. You can use log parsing to transform raw log data into meaningful fields, such as timestamps, error codes, or user IDs, enabling better filtering and querying. New Relic allows you to define custom parsing rules using Grok or Regex patterns, helping to organize unstructured log data into structured formats for more effective monitoring and troubleshooting. This enhances your ability to correlate log events with performance issues or errors in your applications.**

**1. Log Parsing Overview:**

* In New Relic, log parsing is used to extract structured data from unstructured logs. This process helps in creating searchable fields, making it easier to query and analyze logs within the New Relic UI.

**2. Setting Up Parsing Rules:**

* Navigate to the "Logs" section in New Relic.
* Select the specific log stream you want to parse.
* Click on "Parser" to create or edit a parsing rule.
* You can define parsing rules using either **Grok** or **Regex** patterns. These patterns help to match and extract specific parts of the log messages.

**3. Custom Parsing:**

* Custom parsers allow you to tailor the parsing process to your specific log formats.
* For example, you can create rules to extract timestamps, error levels, IP addresses, or any custom data embedded in your logs.

**4. Testing and Validating:**

* Before applying a parsing rule, you can test it directly in the New Relic UI to ensure it extracts the desired fields correctly.
* New Relic provides a preview of how the logs will look after parsing, making it easier to validate the correctness of your parsing rules.

### Grok Debugger in New Relic

**1. What is Grok?**

* **Grok** is a pattern matching syntax that simplifies the extraction of structured data from logs. It's particularly useful for parsing logs that follow consistent formats.

**2. Using Grok Debugger:**

* Grok Debugger is a tool within the New Relic UI that allows you to write and test Grok patterns interactively.
* You can input a sample log message and develop Grok patterns to see how they parse the log into different fields.
* The debugger highlights the parts of the log that match each part of your Grok pattern, showing how the log is being parsed in real-time.

**3. Building Grok Patterns:**

* Grok patterns consist of reusable expressions, making it easy to build complex parsers.
* For example, %{TIMESTAMP\_ISO8601:timestamp} would extract a timestamp in ISO 8601 format.
* You can combine multiple patterns to extract different parts of a log message into separate fields.

**4. Applying Grok Patterns:**

* Once you’ve tested and validated your Grok pattern in the debugger, you can apply it as a parsing rule in New Relic.
* This pattern will then be used to parse incoming logs automatically, converting them into structured data fields for easier analysis.

### Benefits of Log Parsing and Grok Debugging

* **Efficiency:** Automates the extraction of valuable information from raw logs.
* **Improved Searchability:** Parsed logs become easier to filter and search, enhancing your ability to diagnose issues.
* **Data Structuring:** Helps in structuring logs, making them more meaningful and usable for alerts, dashboards, and monitoring.
* **Error Reduction:** Testing with the Grok Debugger reduces the chances of errors in your parsing rules, ensuring accurate log analysis.

This combination of tools in New Relic allows you to create a highly customized and efficient log monitoring system.

Suppose you have a log entry like this:

**2024-08-13 12:34:56 ERROR User ID: 12345 - Checkout failed: Insufficient funds**

You could use a Grok pattern like this:

**%{TIMESTAMP\_ISO8601:timestamp} %{LOGLEVEL:level} User ID: %{NUMBER:user\_id} - %{GREEDYDATA:message}**

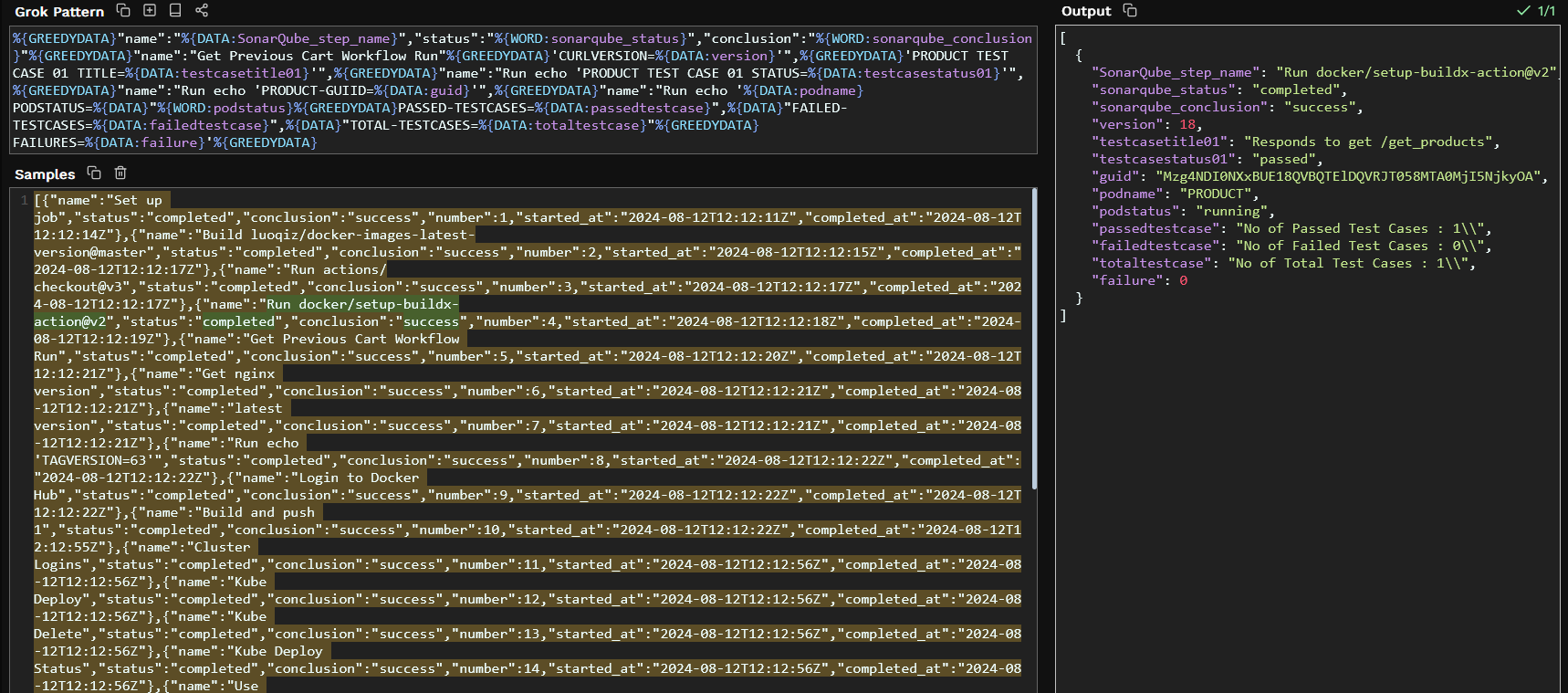
**You would create this Grok pattern in the New Relic UI, test it in the Grok Debugger, and then apply it to automatically parse incoming logs, making it easier to search for specific errors, user activities, or timestamps in your logs.**

**Log Text -**

**[{"name":"Set up job","status":"completed","conclusion":"success","number":1,"started\_at":"2024-08-12T12:12:11Z","completed\_at":"2024-08-12T12:12:14Z"},{"name":"Build luoqiz/docker-images-latest-version@master","status":"completed","conclusion":"success","number":2,"started\_at":"2024-08-12T12:12:15Z","completed\_at":"2024-08-12T12:12:17Z"},{"name":"Run actions/checkout@v3","status":"completed","conclusion":"success","number":3,"started\_at":"2024-08-12T12:12:17Z","completed\_at":"2024-08-12T12:12:17Z"},{"name":"Run docker/setup-buildx-action@v2","status":"completed","conclusion":"success","number":4,"started\_at":"2024-08-12T12:12:18Z","completed\_at":"2024-08-12T12:12:19Z"},{"name":"Get Previous Cart Workflow Run","status":"completed","conclusion":"success","number":5,"started\_at":"2024-08-12T12:12:20Z","completed\_at":"2024-08-12T12:12:21Z"},{"name":"Get nginx version","status":"completed","conclusion":"success","number":6,"started\_at":"2024-08-12T12:12:21Z","completed\_at":"2024-08-12T12:12:21Z"},{"name":"latest version","status":"completed","conclusion":"success","number":7,"started\_at":"2024-08-12T12:12:21Z","completed\_at":"2024-08-12T12:12:21Z"},{"name":"Run echo 'TAGVERSION=63'","status":"completed","conclusion":"success","number":8,"started\_at":"2024-08-12T12:12:22Z","completed\_at":"2024-08-12T12:12:22Z"},{"name":"Login to Docker Hub","status":"completed","conclusion":"success","number":9,"started\_at":"2024-08-12T12:12:22Z","completed\_at":"2024-08-12T12:12:22Z"},{"name":"Build and push 1","status":"completed","conclusion":"success","number":10,"started\_at":"2024-08-12T12:12:22Z","completed\_at":"2024-08-12T12:12:55Z"},{"name":"Cluster Logins","status":"completed","conclusion":"success","number":11,"started\_at":"2024-08-12T12:12:56Z","completed\_at":"2024-08-12T12:12:56Z"},{"name":"Kube Deploy","status":"completed","conclusion":"success","number":12,"started\_at":"2024-08-12T12:12:56Z","completed\_at":"2024-08-12T12:12:56Z"},{"name":"Kube Delete","status":"completed","conclusion":"success","number":13,"started\_at":"2024-08-12T12:12:56Z","completed\_at":"2024-08-12T12:12:56Z"},{"name":"Kube Deploy Status","status":"completed","conclusion":"success","number":14,"started\_at":"2024-08-12T12:12:56Z","completed\_at":"2024-08-12T12:12:56Z"},{"name":"Use Node.js","status":"completed","conclusion":"success","number":15,"started\_at":"2024-08-12T12:12:57Z","completed\_at":"2024-08-12T12:12:57Z"},{"name":"Install npm","status":"completed","conclusion":"success","number":16,"started\_at":"2024-08-12T12:12:57Z","completed\_at":"2024-08-12T12:13:06Z"},{"name":"run curl","status":"completed","conclusion":"success","number":17,"started\_at":"2024-08-12T12:13:07Z","completed\_at":"2024-08-12T12:13:09Z"},{"name":"Run echo 'CURLVERSION=18'","status":"completed","conclusion":"success","number":18,"started\_at":"2024-08-12T12:13:09Z","completed\_at":"2024-08-12T12:13:09Z"},{"name":"Run echo 'CURRENT\_VERSION='","status":"completed","conclusion":"success","number":19,"started\_at":"2024-08-12T12:13:09Z","completed\_at":"2024-08-12T12:13:09Z"},{"name":"Run echo 'PRODUCT TEST CASE 01 TITLE=Responds to get /get\_products'","status":"completed","conclusion":"success","number":20,"started\_at":"2024-08-12T12:13:09Z","completed\_at":"2024-08-12T12:13:09Z"},{"name":"Run echo 'PRODUCT TEST CASE 01 STATUS=passed'","status":"completed","conclusion":"success","number":21,"started\_at":"2024-08-12T12:13:09Z","completed\_at":"2024-08-12T12:13:09Z"},{"name":"Run echo 'PRODUCT-GUIID=Mzg4NDI0NXxBUE18QVBQTElDQVRJT058MTA0MjI5NjkyOA'","status":"completed","conclusion":"success","number":22,"started\_at":"2024-08-12T12:13:09Z","completed\_at":"2024-08-12T12:13:09Z"},{"name":"Run echo 'KUBE APPLY STATUS='","status":"completed","conclusion":"success","number":23,"started\_at":"2024-08-12T12:13:09Z","completed\_at":"2024-08-12T12:13:09Z"},{"name":"Run echo 'PRODUCT PODSTATUS= {\"running\":{\"startedAt\":\"2024-08-12T11:55:22Z\"}}'","status":"completed","conclusion":"success","number":24,"started\_at":"2024-08-12T12:13:09Z","completed\_at":"2024-08-12T12:13:09Z"},{"name":"Run echo \"PASSED-TESTCASES=No of Passed Test Cases : 1\", \"FAILED-TESTCASES=No of Failed Test Cases : 0\", \"TOTAL-TESTCASES=No of Total Test Cases : 1\"","status":"completed","conclusion":"success","number":25,"started\_at":"2024-08-12T12:13:09Z","completed\_at":"2024-08-12T12:13:09Z"},{"name":"Run echo 'FAILURES= '","status":"completed","conclusion":"skipped","number":26,"started\_at":"2024-08-12T12:13:09Z","completed\_at":"2024-08-12T12:13:09Z"},{"name":"Run echo 'EXPECTED-TESTCASES='","status":"completed","conclusion":"skipped","number":27,"started\_at":"2024-08-12T12:13:09Z","completed\_at":"2024-08-12T12:13:09Z"},{"name":"Post Use Node.js","status":"completed","conclusion":"success","number":50,"started\_at":"2024-08-12T12:13:09Z","completed\_at":"2024-08-12T12:13:09Z"},{"name":"Post Build and push 1","status":"completed","conclusion":"success","number":51,"started\_at":"2024-08-12T12:13:09Z","completed\_at":"2024-08-12T12:13:09Z"},{"name":"Post Login to Docker Hub","status":"completed","conclusion":"success","number":52,"started\_at":"2024-08-12T12:13:09Z","completed\_at":"2024-08-12T12:13:09Z"},{"name":"Post Run docker/setup-buildx-action@v2","status":"completed","conclusion":"success","number":53,"started\_at":"2024-08-12T12:13:09Z","completed\_at":"2024-08-12T12:13:12Z"},{"name":"Post Run actions/checkout@v3","status":"completed","conclusion":"success","number":54,"started\_at":"2024-08-12T12:13:14Z","completed\_at":"2024-08-12T12:13:14Z"},{"name":"Complete job","status":"completed","conclusion":"success","number":55,"started\_at":"2024-08-12T12:13:12Z","completed\_at":"2024-08-12T12:13:12Z"}]**

**Rule Text –**

**%{GREEDYDATA}"name":"%{DATA:SonarQube\_step\_name}","status":"%{WORD:sonarqube\_status}","conclusion":"%{WORD:sonarqube\_conclusion}"%{GREEDYDATA}"name":"Get Previous Cart Workflow Run"%{GREEDYDATA}'CURLVERSION=%{DATA:version}'",%{GREEDYDATA}'PRODUCT TEST CASE 01 TITLE=%{DATA:testcasetitle01}'",%{GREEDYDATA}"name":"Run echo 'PRODUCT TEST CASE 01 STATUS=%{DATA:testcasestatus01}'",%{GREEDYDATA}"name":"Run echo 'PRODUCT-GUIID=%{DATA:guid}'",%{GREEDYDATA}"name":"Run echo '%{DATA:podname} PODSTATUS=%{DATA}"%{WORD:podstatus}%{GREEDYDATA}PASSED-TESTCASES=%{DATA:passedtestcase}",%{DATA}"FAILED-TESTCASES=%{DATA:failedtestcase}",%{DATA}"TOTAL-TESTCASES=%{DATA:totaltestcase}"%{GREEDYDATA}FAILURES=%{DATA:failure}'%{GREEDYDATA}**

****

**If the parsing rule is not working at all though it's likely either one of two things.**

1. **The messages are not matched / the rule doesn't work**
2. **More than one rule matches the same message**

**Specifically**

* **Parsing will only be applied once to each log message. If multiple parsing rules match the log, only the first that succeeds will be applied.**
* **Parsing rules are unordered. If more than one parsing rules matches a log, one is chosen at random. Be sure to build your parsing rules so that they do not match the same logs.**

**NRQLs OF DDCC**

|  |  |
| --- | --- |
| **Attribute** | **NRQL** |
| **Workload** | SELECT uniqueCount(entity.name),latest(workloadGuid) as wid  from WorkloadStatus where entity.name IN ('Checkout Microservice Workload','Brand Microservice Workload','Cart Microservice Workload','Product Microservice Workload')  FACET  entity.name limit Max |
| **Relationship** | SELECT uniqueCount(targetEntityGuid) from Relationship Where sourceEntityGuid IN (${workloadIds}) and targetDomain is NOT NULL facet targetEntityGuid,sourceEntityGuid,targetDomain limit MAX |
| **Deployment Version** | SELECT version,entity.name,version,entity.guid, deploymentId from Deployment  where entity.guid In (${apmIds})since 120 days ago limit max ORDER BY timestamp DESC |
| **Deployment Data** | SELECT  count(deploymentId) from Deployment where entity.guid IN ('${apmId}') since 120 days ago facet deploymentId,version Limit MAX |
| **Deployment Data** | FROM Log SELECT TestCaseStatus, TestCaseTitle, version,workflow\_job.conclusion as status,guid,pod\_failure\_message,pod\_status,podFailureReason  where workflow\_job.steps IS NOT NULL and guid IN ('${apmId}') and ((workflow\_job.conclusion ='success' and version ='${version}') OR (workflow\_job.conclusion = 'failure' and version ='${version}')) since 1 week ago ORDER BY timestamp DESC LIMIT 3 |
| **Dep Data** | FROM Log SELECT workflow\_job.workflow\_name, failure as 'failure\_message', guid, podstatus AS 'pod\_status', totaltestcase ,testcasetitle01 AS 'TestCaseTitle' ,testcasestatus01 AS 'TestCaseStatus',version, failedtestcase, failure, passedtestcase, podstatus, workflow\_job.conclusion as 'status'  where workflow\_job.steps IS NOT NULL and guid IN ('${apmId}') and ((workflow\_job.conclusion ='success' and version = '${version}') OR (workflow\_job.conclusion = 'failure' and version ='${version}')) since 1 week ago ORDER BY timestamp DESC LIMIT 3 |
| **Performance Query** | SELECT average(newrelic.goldenmetrics.apm.application.responseTimeMs) AS 'response\_time' ,rate(count(newrelic.goldenmetrics.apm.application.throughput), 1 MINUTES) AS 'through\_put',average(newrelic.goldenmetrics.apm.application.errorRate) \* 100 AS 'error\_rate'  FROM Metric WHERE entity.guid IN ('${apmId}') LIMIT MAX  SINCE ${currentVersionTime} UNTIL ${currentTime} |
| **Activity Stream** | SELECT changelog,deploymentId,description,entity.name,user,version,entity.guid from Deployment where entity.guid IN ('${apmIds}') and version = '${version}' since 120 days ago |
| **Synthetics Query** | Select result,error,entityGuid,monitorName from SyntheticCheck where entityGuid In (${synthIds}) limit Max since 120 days ago |
| **Exceptions Query** | From TransactionError SELECT  uniqueCount(error.class) as 'Error Count' where entity.guid IN (${apmIds}) and  error.class IS NOT NULL facet error.message,entity.guid,appName since 120 days ago |
| **Overall Health** | FROM Log SELECT filter(uniqueCount(guid), WHERE  workflow\_job.steps IS NOT NULL) AS 'Total pipelines' , filter(count(status), WHERE workflow\_job.conclusion LIKE '%success%') AS 'Successful Runs', filter(count(status), WHERE workflow\_job.conclusion LIKE '%failure%') AS 'Failure Runs', filter(count(pod\_status), WHERE pod\_status IS NOT NULL and pod\_status != 'running') AS 'Pod failure' ,  filter(count(failedTestcase), WHERE failedTestcase != '0') AS 'Test Case failure Runs'   WHERE workflow\_job.steps IS NOT NULL and guid IN ('Mzg4NDI0NXxBUE18QVBQTElDQVRJT058MTA0MjM2MTc2OA', 'Mzg4NDI0NXxBUE18QVBQTElDQVRJT058MTA0MjI5MjEwNw', 'Mzg4NDI0NXxBUE18QVBQTElDQVRJT058MTA0MjI5NTA0OQ', 'Mzg4NDI0NXxBUE18QVBQTElDQVRJT058MTA0MjI5NjkyOA', 'Mzg4NDI0NXxBUE18QVBQTElDQVRJT058MTA0MjI5MDcyNA') or ((workflow\_job.conclusion ='success' ) OR (workflow\_job.conclusion = 'failure')) since 120 days ago |
| **Overall Health** | FROM Log SELECT filter(count(\*),WHERE workflow\_job.conclusion IS NOT NULL) AS 'Total\_runs',filter(count(workflow\_job.conclusion),WHERE workflow\_job.conclusion = 'success') AS 'successful\_runs',filter(count(workflow\_job.conclusion),WHERE workflow\_job.conclusion = 'failure') AS 'failure\_runs', filter(count(pod\_status), WHERE pod\_status IS NOT NULL and pod\_status != 'running') AS 'podfailure', filter(count(TestCaseStatus), where TestCaseStatus like '%failed%') as 'testCasefailureRuns', filter(count(\*), where pod\_status IS NULL AND TestCaseStatus IS NULL) AS 'noLogs', filter(count(\*), WHERE sonarqube\_conclusion = 'failure') AS 'sonarqube' WHERE action = 'completed' AND workflow\_job.steps IS NOT NULL AND workflow\_job.steps like '%GUIID%' AND  workflow\_job.workflow\_name NOT IN ('Run Azure Login With a Service Principal Secret','Terraform pipeline') AND workflow\_job.workflow\_name = '${selectedString} Build and Deploy' SINCE 6 hour ago |
| **Pod Failure** | FROM Log SELECT credentialsFailure, guid, version  WHERE workflow\_job.steps LIKE '%CURRENT\_VERSION%' AND  guid IN ('${guid}') AND version ='${version}' and credentialsFailure IS NOT NULL since 120 days ago |
| **Infra Health** | SELECT filter(uniqueCount(workflow\_job.run\_id), WHERE workflow\_job.conclusion = 'success') as 'success', filter(uniqueCount(workflow\_job.run\_id), WHERE workflow\_job.conclusion = 'failure') as 'failure' FROM Log WHERE workflow\_job.steps LIKE '%terraform apply%' Since 6 hours ago |
| **SonarQube Health** | SELECT IF(latest(sonarqube\_health\_web\_status) = 1, 'Up', 'Down') AS 'sonarqube' FROM Metric |
| **SonarQube Task Data** | SELECT latest(ErrorCount) AS 'TaskErrors' , latest(SuccessCount) AS 'TasksSuccess' FROM JVMSampleSonarQubeComputeEngineMetrics |
| **SonarQube Data** | FROM Log SELECT sonarqube\_conclusion as status where workflow\_job.steps IS NOT NULL and guid IN ('${apmId}') and ((sonarqube\_conclusion ='success' and version ='${version}') OR (sonarqube\_conclusion = 'failure' and version ='${version}')) LIMIT 3 SINCE 1 week ago |