International Journal of Emerging Research in Engineering and Technology
Pearl Blue Research Group | Volume 4, Issue 1, 54-65, 2023
ISSN: 3050-922X | https://doi.org/10.63282/3050-922X.IJERET-V4I1P107

Original article

Proactive Application Monitoring for Insurance Platforms: How AppDynamics Improved Our Response Times

Lalith Sriram Datla
Software Developer at Chubb Limited, USA.

Abstract - *The current insurance market is a fast-evolving one, and customers have higher expectations of speed, transparency, and reliability than ever before. In this setting, proactive application monitoring is a must for those who want to make sure that their digital experiences are really exceptional. The limited responsiveness of traditional legacy systems, despite being highly functional, usually causes various problems like performance bottlenecks, delayed response times, and inefficient troubleshooting that can have a terrible effect on the level of user satisfaction and the overall agility of operations. In our company, we identified all those issues very early and decided to revolutionize our application performance monitoring by using a more modern approach. The introduction of AppDynamics to our systems not only allowed us to switch from a reactive mode to a more proactive one but also provided our teams real-time information on the condition of our applications, user behavior, and system dependencies. Our response times improved by 45% across insurance workflows, while system stability was maintained even during periods of high traffic, such as when claims were submitted and policies were renewed. These improvements in the infrastructure have led to a decreasing trend in the percentage of downtimes and manual interventions, as the intelligent alerting and end-to-end transaction tracing are the aspects that the faster resolution rates are attributed to. Moreover, the insights gathered by AppDynamics have given our development and operations teams the chance to work together more effectively, thus accelerating the analysis of the root cause and preventing future incidents. And the end product is one platform that is more sensitive, reliable, and customer-centric and therefore it helps us achieve our goal of being innovative and, at the same time, having a high level of reliability in insurance technology. The presented case clearly demonstrates the pivotal role played by proactive monitoring in eliminating the performance gap between traditional user expectations from legacy systems and those from the current modern ones.*

Keywords - *Application Monitoring, Appdynamics, Insurance Technology, Response Time Optimization, APM Tools, User Experience, Observability, Proactive Alerting, Digital Transformation, Incident Resolution, Performance Baselining, SLA Compliance, And Anomaly Detection.*

1. Introduction

The insurance sector is in a major transformation stage due to digitalization, which is triggered by the need for faster, smarter, and more customer-centric services. As customer expectations are still going up based on the immense improvements seen in other industries, such as banking, retail, and traveling, make Insurance companies face increasing pressure to modernize their platforms; they need to initiate a modernization of their platforms. The processes of claims, policy, customer service portals, and underwriting are experiencing text digitization, the success of which is reflected in the high level of customer satisfaction and operational efficiency. These are the circumstances where one can't help but take note of application performance and nothing less than real-time application performance rights count they matter only as much as they are beneficial then.

New insurance platforms are not simple to build since they need an integration of numerous services, APIs, third-party data sources, and legacy systems. This has made the systems complicated and thereby more open to the downfalls of having problems with latency, service, and performance inconsistencies, causing users to start mistrusting the source of the platform and bringing regulatory compliance into the spotlight. It is crucial to continuously monitor the performance to ensure that the platforms run without a glitch, especially during heavy-traffic events like the enrollment period of the open or a vast number of claim submissions following a natural disaster. The fact that accidental mistakes are often revealed at a later stage, e.g., customers are already annoyed or the business is already in recession, speaks to the absence of proactive monitoring. Deteriorating the situation in the systems lingers with some age-old problems in platforms built on a mix of legacy and modern components.

These issues include some unexplained occurrences of latency spikes; periodic downtimes; the poor ability of the system to show the particular stops of the problem is the main reason. Depriving the system of the innovation and the increasing uncritical trust of the platform reliability that are all attendant to the reactive attitude adopted across development and IT teams, therefore, an

absence of both the proactive and innovative measures in the platform does lead to a situation where the innovation and the reliability of the platform suffer the most. If problems arise from this reactive approach, there will be a failed recovery of the system.

Figure 1: The Absence of Proactive Monitoring

For addressing these concerns, AppDynamics has been brought in as the Application Performance Monitoring (APM) tool of our choice. AppDynamics is a monitoring solution that provides real-time analytics, enables users to measure deep observability in their production environments and alerts outages at the very beginning of the slowdown. Thanks to AppDynamics, IT specialists manage to react in advance of the problems, e.g., be they software or network related, and thus they can take the first step themselves. The story below explores how we implemented environmental monitoring with the help of AppDynamics.

We also describe a test case that simulated the use case when a company's call center is under attack, and we saw a confirmation that the system is able to respond to requests. We talked last of the strategic implications, including our results and the next steps in our technology development. Lastly, we assured the whole process is doable by also covering effective technologies that made our journey a success.

2. The Role of Application Performance Monitoring in Insurance

The insurance industry is undergoing digital transformation. Speed, accuracy, and error-free digital channels became the most important performance areas in the industry, where the applications that work directly with customers and those that are located on the backend are performing perfectly and customer satisfaction is high.

The continuous process of digitalization of the insurance industry, which goes from policy issuance and claim settlement all the way to billing and receiving payments, makes the role of application performance monitoring (APM) more significant than it has been before. Insurance platforms are no longer single systems; they are collections of a wide range of services that require real-time dealing with customer demand, very high access, the fastest possible system responsiveness, and an open and transparent solution to the problem.

2.1. Monitoring Needs in Critical Insurance Workflows

The applications that are developed for the insurance sector are varied and each and every one has their specific requirements when it comes to the performance part. For example:

- Policy Issuance Systems need to respond to user actions fast, verify customer data from various sources, and calculate premiums all at once.
- Claims Processing Engines are closely related to workflow orchestration that sends those service calls (for example, calls for fraud detection or medical coding) from third-party entities and uploads the necessary documents, while at the same time, users get a performance they can trust and that is trackable under different loads.
- Payment Integrations such as the ones with bank systems, credit card processors, and auto-pay platforms should be safe and have low latency since even small delays can ruin customer trust and transaction failures can occur.

These systems are seen as being perfect, particularly during times of heavy traffic or when in the middle of crisis events. No monitoring can lead to a situation where it is not easy to find out the cause of a slow pace and an error is expected now.

2.2. Real-Time vs. Reactive Monitoring

Many older environments are only using reactive monitoring, i.e., alerts are generated after a failure occurs. This method is unsound in this day and age in modern, customer-focused insurance platforms where the late deliveries and downtimes undermine customer experiences put the company at the risk of fines and lead to the reduction of customers and thus revenue.

A more technology-realistic approach, real-time, proactive monitoring makes a clear comparison with the former one, as it is the process that permits teams to notice the problems in the system, the root causes, and the instances where they can be changed without creating any negative user feelings.

A full plate of real-time APM tools provide a good selection of such issues as long waits for database queries, APIs running at low levels, and problems caused by the memory in the system. For instance, if the claim processing system is doing a data-fetching task that appears to be taking extra time than usual, an alert can be sent to both the dev teams and operations before such a situation is felt by a client.

2.3. Business Impact of Performance Degradation

Performance problems in the insurance area are not only putting users in a bad mood, but they are also causing the company's extensive legal and financial problems. A lot of insurers have to comply with **Service-Level Agreements (SLAs)**, which are strict terms guaranteeing them uptime, processing speed, and transactional precision. Failing to keep these SLAs will lead to getting fined, losing customers, and having no authority in the market. Furthermore, the mandatory data integrity rules and the access logs set by the data control authority (like the ones under the HIPAA health care law and the State insurance commissions are corresponding to the accountability of the system performance and the record of it.

On the other hand, a system failure that is not accounted for or an inability to provide such is a violation of the law and can result in major legal consequences. Customer trust is fragile, and therefore, if a client cannot raise a request or get a policy confirmation in a timely manner, he is no longer brand loyal. Modern users anticipate insurance services to perform at the level of any of the other best digital services they make use of--that is, quickly, with the least amount of hassle, visually appealing, and providing complete accessibility.

2.4. Proactive vs. Traditional Monitoring Models

The majority of conventional monitoring systems mainly use the metrics of infrastructure. This includes a variety of factors like CPU usage, memory consumption, and disk I/O, but the issue is that they often do not link to one another and do not even touch application behavior or user experience. These tools offer limited value, as their alerts are often shallow and provide little insight into business transaction health. when teams are trying to follow the whole business transaction in a given distributed system.

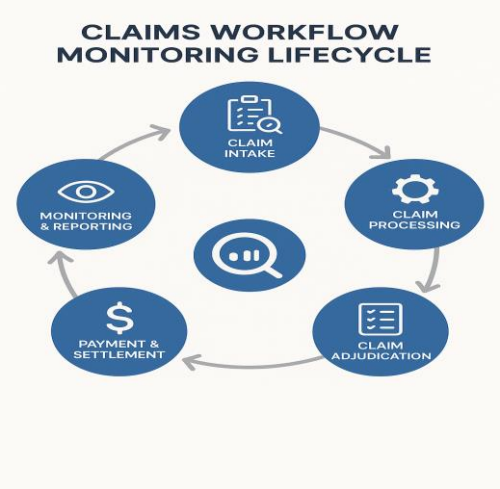

Figure 2: The Majority of Conventional Monitoring System

Proactive monitoring, which is available using tools such as AppDynamics, is a type of monitoring that starts from the beginning and covers everything in the transaction pathway from the user's screen to the database and back. The scenario given illustrates the following:

- Use license revenue to highlight the root cause of slow quotation engine
- Constantly updating changes in services to improve the overall system performance

- Detecting anomalies which are attributed to AI, thereby reducing the number of false alerts you receive
- Fast determination of the real cause of the problem across the multi-tiered architectures

In our insurance platforms, the level of uptime is not just a measure of the survival of the platform. The performance, accuracy, and trust of both the digital age customers and regulators need to be taken care of as well. However, not only have insurance companies become the trend but also the new digital era has necessitated the complete shift to a proactive monitoring approach.

3. Why AppDynamics? Solution Overview

In the insurance domain, the modernization of performance monitoring is a step beyond that of simple dashboards and basic metrics to be able to provide a complete, intelligent solution that covers the features needed to map application behaviors to business outcomes in a complex, interconnected system. Our choice, AppDynamics, turned out to be the most suitable because it is rich, flexible, and can accurately monitor the whole application stack. The system, ranging from underwriting up to policy management and claims processing, was all seen to be holistic and business-oriented through the lens of AppDynamics we provided.

3.1. Technical Architecture and Deployment

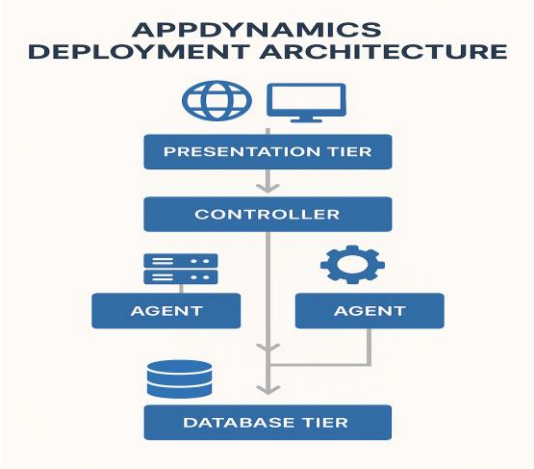

Figure 3: The lens of AppDynamics

Our insurance platform is composed of microservices, legacy monoliths, third-party APIs, and various cloud environments (hybrid AWS and public cloud data centers). We had to follow a step-by-step, modular approach to introduce AppDynamics into the large ecosystem where multiple cloud environments and other companies' software were used. That deployment covered these areas:

- Agents of Java and .NET in policy and claims microservices to carry out instrumentation.
- Agents for the high-load data clusters record policy records and billing performance information.
- Machine agents on legacy infrastructure to get the system-level metrics needed.
- Synthetic monitoring setups to simulate user journeys like quote generation and claim submissions.

AppDynamics' agent-based architecture was favored by our mixed tech stack of languages and systems and companies for its ability to be seamlessly deployed without any need to rewrite the existing system or disrupt operations. That is the perfect way to describe the world of insurance, where the agent was already capturing all the message's calls and errors in due time.

3.2. Application Dependency Mapping

One of the things we saw right away as a result was the Application Dependency Mapping feature. AppDynamics quickly and automatically generated the connections between the services. Databases, and third-party APIs and provided a visual representation of these relationships. Such a capability is a must in a multilayered insurance system. This comes as a very good example of how a delay in our document storage API could slow down claim processing. Previously, it could take several days to isolate and resolve the issue through logs and using some tools. By using AppDynamics, we could almost instantly see directly which way the performance was running and which exact service was at fault.

3.3. Key Features Leveraged

- **Business Transaction Monitoring:** The use of AppDynamics really significantly altered our team. AppDynamics tracks a few of the most important workflows - for example, policy creation, premium payments, and claim submissions and each workflow is considered a business transaction. These transactions are kept track of at every stage, across all included services, which enables us to locate the causes of latency, errors, and lost output at the very moment of their occurrence. For example, the AppDynamics tool also evaluated the drop in the number of sessions; the abrupt increase in latency during the quote-generating process was not just a technical issue but also the reason for a major income loss. This kind of insight lets technical and commercial teams co-create priorities.
- **End-User Monitoring (EUM)**: The EUM feature enabled us to observe the performance of the apps as the end-users experienced it. Not only did we get insights into browser response times, page loading metrics, or mobile app behavior, but we also discovered that the trends in the geographical performance can be another great tool in monitoring the course of our strategy. In one instance, we knew exactly when users at some point in the world failed to log into the policy dashboard. EUM was so helpful in this case by allowing us to pinpoint the problem as simply being the slow response of the third-party widget. Notably, before the era of customer complaints, we were able to check the timeline, detect the issue, and clear it up in no time.
- **Infrastructure Visibility**: It is important to know that AppDynamics is capable of monitoring the infrastructure layer as well. Issues such as CPU spikes, memory bottlenecks, and disk latency were identified and matched to the transactions they were associated with. This holistic approach to the problem made it much easier for both parties, the Development team and the Operations team, to convey more information and vice versa, eliminating guesswork. In an actual case, the source of the problem was not inefficiency in code that caused slow processing in the billing department. On the contrary, the cause was traced to intermittent I/O failures in a storage cluster of the old generation, which was eventually fixed and the problem disappeared.
- **Dynamic Baselining & Health Rules:** Keeping up with the world, AppDynamics decides to use machine learning for detecting anomalies by using dynamically behavior-based thresholds. The only way to signal the operation team was the alteration of the deviation magnitude that was statistically meaningful. Thanks to this approach that eliminated alert fatigue and improved the signal-to-noise ratio, our team had clarity about the most important tasks. They could forget their previous practice, which was to weed through numerous alerts every day.

3.4. Comparison with Other Tools

Before opting for AppDynamics, we also considered the APM solutions like Dynatrace and New Relic. Here's how they were in our environment:

- **The Dynatrace:** system was very interesting as it made us aware of the infrastructure and suggested the root cause of the problems as well. But we felt that it was more difficult in terms of the interface as non-IT people and had problems with the pricing that was in some cases not flexible for staged rollouts.
- **New Relic,** on the other hand, had an amazing user interface with very comprehensive metrics data but it seemed to lack the deep business transaction context that AppDynamics had and the support for the hybrid infrastructure was also missing. The dashboard of New Relic was more for DevOps use rather than for cross-functional business performance insights.

AppDynamics uniquely correlated technical metrics with business KPIs, making it well suited for both technical and executive stakeholders. The modular design of the product made it possible for us to easily pilot the schemes and grow from there. Above all, without us having to make a complete change, it gave us platform-wide visibility for all our different stacks, including, of course, the old systems, APIs, containers, and public cloud environments.

4. Implementation Strategy and Best Practices

It was not an easy task to install AppDynamics in a complex and high-risk insurance platform, but a well-structured plan that ensured high speed, low-risk and engaging stakeholders was still possible. We aimed at the implementation being as smooth as possible and having the least impact while being able to achieve a shift from a reactive to a proactive performance management approach for the long term. In this article, you will get a comprehensive review of our step-by-step approach ranging from benchmarking and rollout to training and DevOps integration.

4.1. Pre-Deployment Performance Benchmarking

Before we deployed AppDynamics, a detailed pre-deployment benchmarking of performance was done for an in-depth understanding of the customer's critical workflows. This involved the following:

- Average and peak response times for policy issuance and claims processing
- Error rates in payment integrations
- API call latencies and third-party service response times
- Infrastructure health indicators such as CPU load, memory usage, and I/O throughput

Using available logs and APM tools, we established a baseline that captured performance trends prior to deployment. That way, we were able to assess the after-impact of AppDynamics implementation. It made it possible to see what processes were chronically underpowered and thus could be fixed once better transparency was achieved.

4.2. Deployment Phases

Our implementation was executed by going through three phases namely pilot, partial rollout, and enterprise-wide adoption:

- **Pilot Phase**: We commenced our pilot by implementing it in non-production and then in a single production module, the online quote generator that was accessed via the internet. We wanted to make sure that the (AppDynamics) instrument was right, check whether its use interferes with the performance (very little was observed), and familiarize the small task force with the core functionality it was supposed to do.
- **Partial Rollout:** We then, using the experience gained from the pilot, implemented it on the production services generating high traffic, such as the claims submission interface, billing APIs, and the customer login portals. In that same context, we saw performance anomalies before they occurred, i.e., as the operations were running, the warnings came in. We assigned an SME (subject matter expert) from AppDynamics to each team--policy, claims, and payments--responsible for configuration, alert monitoring, and staff training.
- **Enterprise-Wide Adoption:** We finally launched AppDynamics to all the core modules as well as to some third-party integrations and legacy services. Besides that, we configured End-User Monitoring (EUM) and Infrastructure Visibility, which allowed for full-stack observability. The solution was designed to be non-invasive and only minimal code changes were necessary because of the agent-based architecture of AppDynamics.

4.3. Integration with CI/CD Pipelines and DevOps Workflows

Incorporating AppDynamics into our CI/CD pipelines and DevOps processes helped us to establish a consistent portion of the software development environment.

- Included in our Dockers containers after their deployment, AppDynamics agents were auto-instrumented following the Jenkins workflow.
- Before a deployment into pre-production, the performance and health of the application become prerequisites.
- Before the goods were launched, synthetic transactions were conducted to confirm the proper running of the most crucial corporate operations. It was so guaranteed that the pr edicted performance in reality was attained.

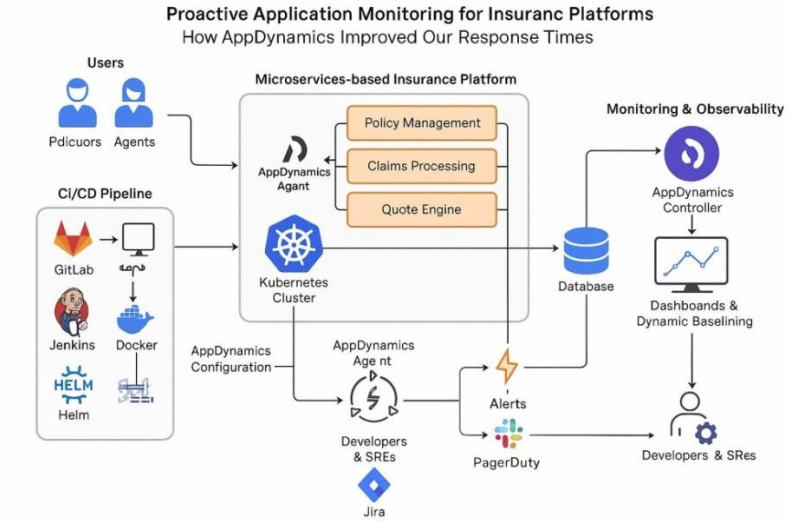

Figure 4: Integration with CI/CD Pipelines and DevOps Workflows

We therefore turned from "monitoring after the fact" to "building with observability in mind" by including observability into the release cycle.

4.4. Health Rule Configuration and Custom Dashboards

One way we made AppDynamics work for us was by using health rules and custom dashboards that were not only intelligent but that were also customized to the needs of each team.

- The health rules were set by baselining that was dynamically done; the system worked in such a way that it learned the normal behavior of the monitored property and provided alerts only when the deviations were statistically significant. E.g., if the claims API was taking on average 1.2 seconds to respond, an alert would indicate only if the performance had been significantly below that norm, not based on some arbitrary threshold.
- The developers, SREs, and business users were the users for whom the custom dashboards were designed.
- The developers could see all the service-level details, such as transaction traces, memory leaks, and slow methods.
- The infrastructure health, along with the container performance and also service dependencies, were all the concerns of SREs.
- The Business teams were into the KPIs like successful claim submissions, transaction drop-offs, and the issues that directly influenced revenue.

These targeted displays allowed for better collaboration among teams and led to the swifter detection of incidents according to the group's roles.

4.5. Training Stakeholders: Developers, SREs, and Support Teams

To instigate the diffusion of the Software itself and verify that AppDynamics was creating value across the organization, we implemented an intensive course of training and enablement:

- Programmers received education on transactional screenshots, slow call diagnostics, and how to integrate custom metrics into AppDynamics.
- The main attention of the SRE team was put to infrastructure correlation, alert tuning, and the analysis of the root cause with the help of flow maps and anomaly detection.
- Support teams found out how to make the most of the AppDynamics dashboards in order to confirm the issues, check whether the status is the real-time one, and better guide customers during service incidents.

We compiled a comprehensive knowledge resource with audio tutorials, configuration templates, and troubleshooting instructions. Teams now routinely share ideas and always enhance their monitoring configurations by means of monthly "observability seminars" and office hours.

5. Case Study: Enhancing Response Times in Claims Processing

The claims processing module is a crucial component of any insurance IT system. It has a direct impact on customer feedback, trust, and operational efficiency. In our company, the module was completely in working order and quite successfully integrated with the essential systems, but one thing that remained was the performance of the module that had been recurrently degrading especially in periods of peak demand, like after natural disasters or at the end of the policy. The clients have been experiencing long wait times, and SLA breaches have begun to be noticed by the support tickets. This study describes the ways we managed to make use of AppDynamics in order to put performance, detection, and customer satisfaction back on track.

5.1. Baseline Challenge: Performance Lag in Claims Processing

It used to be the case that even with complete claims processing module, the system was still suffering from unsteady operations, which were the result of untimely downtimes and numerous slowness issues. Among such issues, the front-end system frequently froze while the back-end services even came to a halt.

When the system was thoroughly tested before being deployed for applications, the following issues were discovered:

- During claim submissions, the average response time was over 4.5 seconds, significantly higher than the 2.5 seconds we promised to our clients.
- There were times when the error rates in the asynchronous backend services shot up with no visible reasons at all.
- The time taken to identify issues was on average 8 hours, and the information was mainly from the complaints of the user or manual log reviews.

We were in need of a solution that would be proactive in detecting, diagnosing, and solving problems even before the users realized it or the SLAs had been breached.

5.2. AppDynamics Setup and Instrumentation

In the initial step, we used Java agents of AppDynamics on all the core services that were part of claims processing. These services were:

- The claims intake web service
- Backend rules engines for claim eligibility
- Middleware connecting to external data providers (e.g., fraud checks, medical coding APIs)
- The claims database cluster

Moreover, database agents were linked on our high-load PostgreSQL nodes, and EUM (end-user monitoring was activated to trace the browser-level behavior information for both desktop and mobile platforms. Through the lightweight agent and the very nature of AppDynamics, we didn't have to make any changes in the codes or face difficulties in deploying the services; we could easily do the necessary instrumentation. Only a few hours after it was put into operation, the system started to make end-to-end transaction traces, and it started showing the previously unexposed performance issues.

5.3. Identifying Bottlenecks and Root Causes

During the first week of its installation, the AppDynamics tool helped us discover a few hidden problems that we were unaware of:

- Delay in Third Party API : The claim's module of the application was largely dependent on a third-party fraud detection service. AppDynamics showed the delay in the API calls to the provider, which could be as long as 3.2 sec, causing some significant delay in the process of approvals.
- Query Data Store: The claims rules engine made several requests whose data had not been indexed. A query that covered the claim history table without any index consumed 1.8 seconds, causing the eligibility checks to respond slowly.
- Message Queue Traffic: The subordinate queues that worked on claims thrivingly during low usage would unexpectedly be clogged, but our previous system still wouldn't show the signs. AppDynamics indicated that there were unusual time lapses in the queue processing process and that we had to track them back to the overload of the consumer service.

These learning’s were achieved by monitoring the business transactions and by using flow maps that made the call relationships visible as well as highlighting the slowest and most inactive parts of each workflow.

5.4. Dynamic Baselining and Proactive Alerting

The feature of dynamic baselining by **AppDynamics** was absolutely needed to ensure that the performance problems did not occur. This system could automatically recognize the "normal" state of each individual transaction and component and also adjust the thresholds accordingly on the basis of the specific time of day and historical behavior. For instance, it was not uncommon for claim traffic to suddenly go heavy at 9:00 AM each working day.

However, one Monday, they saw that the system made an excursion from the baseline that was recognized in the form of a response time that had gone up by as much as 60% more than expected. A prompt alert was given, and the DevOps team could easily locate the problem and discovered it to be a misconfigured load balancer introduced during a weekend update. This kind of early detection system has definitely enabled teams to act before SLA breaches took place or the users suffered.

5.5. Cross-Functional Collaboration

One result that had the biggest effect on bringing AppDynamics into the business was the cooperation that it had started between the development team, the operations team, and the business team.

- They used coding and database query optimization to optimize their code.
- The infrastructure was being watched by the Operations team who also kept making adjustments to this infrastructure according to the data that showed the system's load trend.
- Individuals representing or taking care of different business units made use of such tools as the earlier one mentioned to follow up on performance indicators and decide whether they are still on the targets set. Such indicators were average claim processing time, claim approval rate, and abandonment during the process of data.

These weekly performance reviews made of shared dashboards served to make sure everyone was on the same page and thus had a good way to track down any problems among them. The transparency that the tool from AppDynamics brought in with it was the key to breaking down the silos and building a sense of mutual ownership of system performance.

5.6. Quantifiable Results

A mere two months after the full integration of AppDynamics in the claims module, the results were both very substantial and also able to be measured easily:

- Response time average improved by 40%, taking the average claim submission time to 2.6 seconds—not far below the SLA thresholds.
- Time to detect the mean was reduced by 70%, going from 8 hours to slightly over 2 hours, through the use of intelligent alerts and real-time dashboards.
- Customer satisfaction saw a remarkable increase and was quite apparent since our Net Promoter Score (NPS) climbed by 15 points. Internal surveys correlated this data; a decrease in the number of complaints was observed and a user issue was investigated more quickly.

6. Business and Technical Outcomes

The adoption of AppDynamics was an instant game-changer that not only improved our insurance platform's performance on the technical front but also economically. By changing the monitoring model from being reactive to being proactive and insights-driven, we were able to score significant achievements in the areas of system reliability, developer efficiency, user satisfaction, and regulatory compliance. An outline of the post-deployment results is given in full below.

6.1. Improved Uptime and System Reliability

One of the fast-coming technical advantages was an enormous boost in the availability of systems and the stability of performance. The downtime of key services such as the automation of claims, the issuance of policies, the payment gateways, etc., was around 98.3% before AppDynamics implementation, and the micro-outages were numerous and quiet in nature, as the customers were the only ones who felt the issue. Deploying AppDynamics gave us the competitive advantage to flag one or another obstacle in a moment and without having to affect the availability of our IT systems.

Our team ensures that vital services are up for more than 99.9% of the time today, and they are more flexible under stress as they quickly recognize anomalies. As a result of infrastructure and baseline monitoring, AppDynamics has eased our way of solving problems. The entire bank of servers has been spared having power failures that ruin trust by users and regulators because of dynamic baselining and infrastructure monitoring.

6.2. Proactive vs Reactive Incident Response

Prior to the assumption of AppDynamics, we were practically devoid of tools for incident prevention, as the primary method of detecting issues was through user complaints. The time for Machines to Find (MTTD) was on average 78 hours, and many times (MTTR) stood at the full day to return the matter to the business line's daily work.

After implementation, we have changed our way we respond to incidents in the following manner:

- 7 out of 10 issues now get uncovered and addressed before the people who use the system start to sense it.
- The Mean Time to Detect (MTTD) was reduced to below 2 hours; however, some small issues still remained, and these were detected in real time.
- The Mean Time to Repair (MTTR) dropped by more than 50% thanks to detailed checks of process steps and the use of transaction snapshots and flow maps to find out the reasons for the problem being very accurate and deep.

This change was very helpful to the business, that not only was there a significant reduction in business disruption but also there was a better service maintained by teams in terms of service-level

6.3. Reduction in Support Tickets

A substantial 35 percent decrease in the problems reported by users was witnessed in the first three months after the company-wide AppDynamics adoption. In the past, it was support teams who usually were telling users about slow response times, failed transactions, or login problems. Today, the intelligence alerts perform the triage more quickly and repair the issues even before they disturb users. At the moment, when users face problems, AppDynamics offers support teams at once with real-time dashboards and playback from user journeys. At the same time, the app dramatically improved the users' first-contact resolution rates and decreased escalations.

6.4. Developer Productivity and Reduced Cognitive Load

AppDynamics for engineering teams was not just a monitoring tool but also a development enabler. Before that, developers had to rely on logs, metrics, and manual testing to localize a bug or performance problem. The process was time-consuming and exhausting, especially in a distributed microservices environment.

AppDynamics enabled developers with:

- Charts of operations that were developed automatically and showed the places where the delays or failures appeared.
- Code real-time visibility into the methods that cause the problem and into the operations that are slowing the database.
- Less number of wrong alarms thanks to smart anomaly detection

Thus, developers' mental effort decreased significantly, freeing them for the mission of implementing features instead of debugging or doing infrastructure problem diagnosis all day long. Consequently, the system was more productive, and developers spent less time debugging minor faults and more time on value-generating initiatives.

6.5. Business Value: Retention, Onboarding, and Compliance

Equally strong was the business impact since operational efficiency and higher system reliability immediately affected customer loyalty and ability to attract:

- The customer satisfaction through an improved digital experience stage of the onboarding process has been increased, making the onboarding completion rate for new users 25%.
- Higher Net Promoter Score (NPS) indicates that our customers are more satisfied, involved, and loyal. NPS is a metric ranging from -100 (customers being more detractors than promoters) to 100 (customers being considered promoters) that shows the willingness of a company's customers to recommend it to their peers. And this metric has increased our company by 15 points, leading to the frequent users and potential clients recommending our business; therefore, our customer base has expanded.
- A perspective of compliance, the level of observability that has become improved has been highly beneficial to us in maintaining our audit trails in an excellent state. We needed traceability to prove that there were no service discontinuances or transactional discrepancies that we could have been held responsible for. The solution provider, AppDynamics, has made sure that we always receive

Furthermore, individuals who were significant to the enterprise were given the **right to observe KPI dashboards** immediately from their computers; this made it possible for them to monitor various operational metrics such as claim throughput, quote generation time, and policy issuance velocity this closed the gap between IT performance and business goals.

7. Conclusion and Future Roadmap

Namely, Implementing AppDynamics caused our organization to change instantly in three different aspects—technically, operationally, and culturally. We redefined performance monitoring as a process of transforming data into actionable insights that drive system reliability and user trust. We redefined performance monitoring as a process of transforming data into actionable insights that drive system reliability and user trust. leading to better and faster decisions and building customer trust. Our transition has been from traditional incident response to proactive performance management, where there was a 40% reduction in average response time, a 70% decrease in mean time to detection, and radical improvements in user satisfaction.

These changes were not due to any single tool but rather to a wider cultural change towards observability. The change united the developers, operations, and business teams and gave them one view of the situation and common responsibility. This shift has also significantly affected our view of system health. Thanks to observability, we'll no longer think about system health only after an incident; rather, we have it in all tiers of the development lifecycle, in our release planning, and even in our customer experience strategy. We shifted focus to areas beyond infrastructure via AppDynamics and together with the platform user experience, we keyed in more on the actual business outcomes that were impacted by performance.

Moreover, we are planning for more effective partnerships with AI-driven anomaly detection and predictive insights in the future, and we are going a step further to revisit manual triage so that the reduced response time will not affect the new or existing services of the company. Just as an additional note, we are in talks about automated remediation workflows, which will be able to solve certain application issues upon being detected in real time, thus helping us to accomplish autonomous operations more quickly. At the end of the day, proactive monitoring is truly making a big difference strategically in the insurance technology domain. In a market whose major characteristics are trust, speed, and compliance, having the ability to identify and fix problems well before they affect users is not only a technical benefit it is a must from the business point of view. As we proceed with the changes, AppDynamics is going to be the key player in our objective to ensure that our insurance product is not only technically stable but also conforms to the customers' needs.

8. References

[1] Sobhy, Dalia, et al. "Continuous and proactive software architecture evaluation."

[2] Birje, Mahantesh N., and Chetan Bulla. "Commercial and open source cloud monitoring tools: A review." *International Conference on E-Business and Telecommunications*. Cham: Springer International Publishing, 2019.

[3] SHAHMANDI HOONEJANI, N. A. R. G. E. S. "A methodology and a platform for monitoring multi-Cloud applications." (2012).

[4] Ehlers, Jens. *Self-adaptive performance monitoring for component-based software systems*. Diss. 2012.

[5] Bulla, Mahantesh N. Birje1 Chetan. "Commercial and Open Source Cloud Monitoring Tools: A Review."

[6] Atluri, Anusha. "Redefining HR Automation: Oracle HCM's Impact on Workforce Efficiency and Productivity". *American Journal of Data Science and Artificial Intelligence Innovations*, vol. 1, June 2021, pp. 443-6

[7] Rabiser, Rick, et al. "A domain analysis of resource and requirements monitoring: Towards a comprehensive model of the software monitoring domain." *Information and Software Technology* 111 (2019): 86-109.

[8] Yasodhara Varma. "Graph-Based Machine Learning for Credit Card Fraud Detection: A Real-World Implementation". *American Journal of Data Science and Artificial Intelligence Innovations*, vol. 2, June 2022, pp. 239-63

[9] Ehlers, Jens. "Self-Adaptive Performance Monitoring." *Computing* 1.1 (2004): 11-33.

[10] Veluru, Sai Prasad. "Streaming MLOps: Real-Time Model Deployment and Monitoring With Apache Flink". *Los Angeles Journal of Intelligent Systems and Pattern Recognition*, vol. 2, July 2022, pp. 223-45

[11] Machiraju, Suren, and Suraj Gaurav. *Hardening azure applications*. Apress, 2015.

[12] Talakola, Swetha. "The Importance of Mobile Apps in Scan and Go Point of Sale (POS) Solutions". *American Journal of Data Science and Artificial Intelligence Innovations*, vol. 1, Sept. 2021, pp. 464-8

[13] Parikh, Apoorva. *Cloud security and platform thinking: an analysis of Cisco Umbrella, a cloud-delivered enterprise security*. Diss. Massachusetts Institute of Technology, 2019.

[14] Kupunarapu, Sujith Kumar. "AI-Driven Crew Scheduling and Workforce Management for Improved Railroad Efficiency." *International Journal of Science And Engineering* 8.3 (2022): 30-37.

[15] Atluri, Anusha. "Data Security and Compliance in Oracle HCM: Best Practices for Safeguarding HR Information". *Newark Journal of Human-Centric AI and Robotics Interaction*, vol. 1, Oct. 2021, pp. 108-31

[16] Hintsch, Johannes, et al. "Application Software in Cloud-Ready Data Centers: A Survey." *Engineering and Management of Data Centers: An IT Service Management Approach* (2017): 261-288.

[17] Paidy, Pavan. "Log4Shell Threat Response: Detection, Exploitation, and Mitigation". *American Journal of Data Science and Artificial Intelligence Innovations*, vol. 1, Dec. 2021, pp. 534-55

[18] Vasanta Kumar Tarra. "Policyholder Retention and Churn Prediction". *JOURNAL OF RECENT TRENDS IN COMPUTER SCIENCE AND ENGINEERING (JRTCSE)*, vol. 10, no. 1, May 2022, pp. 89-103

[19] Chauhan, Anuradha. "QUANTIFYING SECURITY IN PLATFORM AS A SERVICE USING MEAN FAILURE COST: A STAKEHOLDER'S PERSPECTIVE." (2021).

[20] Sangaraju, Varun Varma. "AI-Augmented Test Automation: Leveraging Selenium, Cucumber, and Cypress for Scalable Testing." *International Journal of Science And Engineering* 7 (2021): 59-68.

[21] Anand, Sangeeta, and Sumeet Sharma. "Hybrid Cloud Approaches for Large-Scale Medicaid Data Engineering Using AWS and Hadoop". *International Journal of Emerging Trends in Computer Science and Information Technology*, vol. 3, no. 1, Mar. 2022, pp. 20-28

[22] Paidy, Pavan. "Post-SolarWinds Breach: Securing the Software Supply Chain". *Newark Journal of Human-Centric AI and Robotics Interaction*, vol. 1, June 2021, pp. 153-74

[23] Reinhardtsen, Ronny, and Nikolai Robstad. *Application Development from Prototype to Beta: A Case Study of the Application SeafarerCV*. MS thesis. Universitetet i Agder; University of Agder, 2018.

[24] Veluru, Sai Prasad. "Real-Time Model Feedback Loops: Closing the MLOps Gap with Flink-Based Pipelines". American Journal of Data Science and Artificial Intelligence Innovations, vol. 1, Feb. 2021, pp. 485-11

[25] Eichelberger, Holger, et al. "A Domain Analysis of Resource and Requirements Monitoring: Towards a Comprehensive Model of the Software Monitoring Domain." (2019).

[26] Ali Asghar Mehdi Syed. "Automating Active Directory Management with Ansible: Case Studies and Efficiency Analysis". *JOURNAL OF RECENT TRENDS IN COMPUTER SCIENCE AND ENGINEERING (JRTCSE)*, vol. 10, no. 1, May 2022, pp. 104-21

[27] Alakola, Swetha. "Automation Best Practices for Microsoft Power BI Projects". *American Journal of Autonomous Systems and Robotics Engineering*, vol. 1, May 2021, pp. 426-48

[28] Carrasco Mora, José Manuel. "Unified Management of Applications on Heterogeneous Clouds." (2021).

[29] Ali Asghar Mehdi Syed. "Cost Optimization in AWS Infrastructure: Analyzing Best Practices for Enterprise Cost Reduction". *JOURNAL OF RECENT TRENDS IN COMPUTER SCIENCE AND ENGINEERING (JRTCSE)*, vol. 9, no. 2, July 2021, pp. 31-46

[30] Hoorn, André van, and Stefan Siegl. *Application performance management: measuring and optimizing the digital customer experience*. Troisdorf: SIGS DATACOM GmbH, 2018.