International Journal of Computer Science and Information Technology Research (IJCSITR)
Vol. 3, No. 1, January-December, 2022, pp. 180-198
Journal ID: 9471-1297
website: www.ijcsitr.com
DOI: https://doi.org/10.63530/IJCSITR_2022_03_01_017

Engineering Insurance Portals of the Future: Modernizing Core Systems for Performance and Scalability

Sirisha Meka
Engineering Manager,
Credit Karma, USA.

Abstract

Modernization of insurance portals has played a crucial role that legacy systems are unable to advance to meet the performance, scalability and flexibility requirements in the current digital insurance ecosystem. The paper presents an engineering design of developing next generation Java based insurance portals in Spring boot, Angular, and Microservices architecture, which is deployed on Pivotal Cloud Foundry (PCF) with both SOAP/REST API interoperability. Its system is created with the built-in SonarQube that handles the code quality and Jenkins that takes care of continuous integration/continuous deployment (CI/CD). The suggested model is based on three dimensions of modernization: (1) the architectural refactoring of legacy monoliths to microservices; (2) optimization of performance by use of distributed caches, containerization and load balancing; and (3) testing of scalability when simulated high-load conditions are provided. The findings show that the engineering productivity, stability, and performance are highly gained. There was also an increase in the team velocity that stood at 150 story points per sprint, which indicated improved planning and efficient development. There was also production stability, and release incidents were decreased by 62.5, and the MTTR was minimized by 66.3 indicating quicker issue fix. The quality of code increased due to an increase in the coverage of tests, the decrease in the number of static-analysis problems, and the growth of CI test pass rates. Performance of API was also found to be less latent and, more importantly, had increased success rates which proved to be a more reliable and scalable system. The study adds a blueprint of end-to-end modernization that incorporates cloud-native engineering, Devops automation, and microservice design concepts to speed up the digital transformation in the insurance sector.

Keywords

Insurance Tech, Microservices, Cloud CI/CD, Performance Engineering, Web Development, Microservice Design.

How to Cite: Sirisha Meka. (2022). Engineering Insurance Portals of the Future: Modernizing Core Systems for Performance and Scalability. *International Journal of Computer Science and Information Technology Research*, 3(1), 180–198.
DOI: https://doi.org/10.63530/IJCSITR_2022_03_01_017
Article ID: IJCSITR_2022_03_01_017

1. Introduction

Insurance is one of the most conservative financial sectors that is risk averse, and it is transforming remarkably at the digital level. The primary causes of this change are the changing customer requirements, the competition pressure in the market, and the rapid evolution of the cloud computing and web technologies. Rapidly revised policies, an efficient claims settlement mechanism, and recognizable experiences in both mobile and web applications are the requirements of contemporary policy holders. To overcome these demands, the insurers are re-architecturing their technology ecosystems so as to put emphasis on agility, scalability, and performance [1].

Previously, monolithic architecture had been developed by insurers and is frequently on mainframes or a simple version of Java EE. They were created to be consistent and stable over the decades and in case of transactions. However, their strongly tied components created rigid dependencies, which needed proposals in terms of time accompanied with errors. The seamless incorporation of new digital services like AI-based fraud detection of real time customer analytics became cumbersome restricting innovation and responsiveness. These old systems have become key bottlenecks in the modernization process of the insurers as they struggle to introduce new digital products within the shortest period of time [2] [3].

Insurance portals were formerly becoming interactive ecosystems that process the entire policy lifecycle as opposed to their initial use of being a static, information-based web page. State-of-the-art portals allow clients to carry out a wide variety of operations, such as purchasing new policies, renewing old ones, filing claims, getting endorsements, or paying,

without involving the agent [4]. However, when these functions are co-located in monolithic systems and even a simple change (such as the addition of a new payment gateway) can cause extensive dependencies through the stack of applications. This inflexibility results in the long deployment time, irregular user experiences and high cost of maintenance [5].

To address these shortcomings, the current engineering trends suggest that monolithic applications can be broken down into microservices a service-independent and loosely coupled service in charge of a business domain. An example is customer onboarding, premium calculation, issuing policy, and processing claims; each of the functions can be an autonomous microservice. These microservices interact through lightweight APIs such as usually either RESTful or SOAP protocols where data exchange is seamless and there is backward compatibility with existing legacy systems [6] [7].

The architectures developed based on microservices enable insurers to improve scalability and maintainability. The independent development, testing, deployment, and scaling of each of the services can be based on the workload of the service. As an example, in seasonal peaks, like the renewals of health policies, only the services in question, have to be scaled and infrastructure utilization is maximized; cost-efficiency is achieved. Moreover, microservices would allow continuity and delivery practices of continuous integration (CI) and continuous delivery (CD), which would allow a quick cycle of iterating and deploying with minimum downtime.

Nevertheless, the movement of old monoliths into new distributed microservices is not a purely technical process, which is accompanied by new organizational and process changes. Conventional insurance IT teams, which tend to be based on functional silos, need to embrace DevOps concepts which place greater emphasis on collaboration, automation, and quality and deployment shared ownership.

To overcome these problems, this study uses a Java spring boot and Angular based full stack modernization architecture. Spring boot used is a reliable enterprise-level framework that eases the development of microservices, dependency management as well as configuration. It offers embedded servers, simplified implementation, and the scale to current enterprise trends such as circuit breakers and service discovery. Angular is applied to create user interfaces that are dynamic and responsive and modular on the client-side. The component-based design of Angular enables the interaction of the UI components with the back-end microservices via REST APIs to enhance user experience and maintainability.

The factor of critical importance of modernization is the guarantee of cloud-native deployment and scaling. Pivotal Cloud Foundry (PCF) is used as the deployment environment in this research. PCF offers the elastic scaling, self-healing and container orchestration, allowing services to scale at any moment without downtime. The system ensures performance and resilience as well as promotes zero-downtime deployments, which is a critical need in financial systems where service outages directly impact customer confidence and revenues.

A second component that would be introduced into the CI/CD pipeline in the study is Jenkins and SonarQube to automate the building, testing, and deployment process and deliver high-quality services. The continuous delivery is coordinated by Jenkins and this means that there are frequent and regular releases of software. SonarQube on the other hand enables maintainability, complexity and test coverage analysis of the source code continuously hence functional and non-functional reliability. Such an ecosystem of cloud CI/CD is a potent combination of tools that adjusts the development rate to the business governance.

Some of the common operational and architectural problems associated with legacy insurance platforms are:

- High loads have slow response times, because processing is done sequentially, and there are no caching layers.
- Complex deployments, which require hand operations and extensive downtime.
- Poor integration of the company with other partner ecosystems and digital services due to lack of interoperability with third-party APIs.
- Absence of CI/CD automation, slowness of the release cycles.

Impossibility to combine the modern technologies such as AI, analytics, and blockchain in the course of significant re-engineering.

The modernization strategy presented here goes right to the root of these bottlenecks. The refactoring of core insurance modules to Spring Boot microservice improves response time and throughput of the system. The introduction of REST and SOAP hybrid communication makes sure that the existing legacy systems are compatible with the new ones and the digital ecosystems of the future are supported. By using cloud-native deployment with PCF and automated CI/CD pipelines, insurers are able to deliver updates more reliably and frequently, and deliver almost continuously.

This research has fivefold objective:

- Architecture A modular insurance portal based on a microservices architecture to

decouple core functionality.

- De facto Refactor old monolithic modules into new Spring Boot modules.
- Integrate and deploy a continuous improvement/ development process with Jenkins, SonarQube, and PCF.
- Test load as well as system performance by comparing its performance with the old legacy baselines.
- Compare the scalability and deployment efficiency, paying attention to the performance improvements, latency improvements and resilience.

The modernization is advantageous to gauge in terms of performance engineering. The load test shows tremendous improvements in terms of response time, throughput and efficacy of deployment. The modular design also simplifies the process of debugging and testing hence is able to rectify the problems much faster and keep on improving.

In conclusion of everything mentioned above, the kind of modernization makes the old insurance systems dynamic and cloud-native environments capable of providing the new generation of digital services. The addition of Spring Boot, Angular, PCF, Jenkins and SonarQube assists insurers to possess high-performance, safe and robust systems. Automation of CI/CD ensures that innovations are constant and fast delivery process is ensured.

It is demonstrated in this study that microservice based modernization does not only enhance the scalability and the performance, but it further enables the insurers not to rely on a transaction based model anymore, but a customer based digital experience instead. Conclusively, insurance portals of the future are built on the foundations of cloud-native, performance-architectured designs, designed to be scalable, smart, and created to change indefinitely.

2. Literature Review

In the insurance world that has long been dependent on old systems, there is an immense shift happening imposed by the digital technologies and the use of cloud-native systems and data-driven innovations. With the emerging customer demands of real time responsiveness and any kind of personalised services, the insurers are forced to update their core systems to enable enhanced performance, scalability and agility. The next generation insurance portal engineering is the transition to cloud-integrated and API-driven ecosystems that are built on microservices, versus a monolithic infrastructure [1][3][6].

The introduction of the microservices architecture is the way of modernization of insurance portals. Malali [1] emphasized the fact that legacy life insurance systems can be scaled and agile through the use of a microservice, which decouples core functionality into services that can be deployed independently. This is a modular design that helps the insurers to refresh separate elements of the ecosystem without impacting the overall ecosystem, which takes a very short time to introduce new products.

Ranjani [2] supported this statement by highlighting the design patterns and best practices in scalable microservices in banking and insurance systems. Her work proposed domain-driven design (DDD) and event-driven architecture (EDA) to be essential enablement of fault-tolerant and scalable system design. The patterns can be used to decompose monoliths without compromising transactional integrity and consistency in operations.

In a large-scale empirical investigation of microservices migration strategies, Fritzsch et al. [3] have found that scalability, maintainability, and organizational agility are some of the drivers. They discovered that modernization projects tend to experience difficulties in service granularity definition and refactoring of legacy code. De Lauretis [6] also examined the transition of engineering between the monolithic and the microservices architecture, where the decomposition of services, API standardization, and containers orchestration (e.g., Kubernetes) are at the center of developing a resilient and scalable system.

In their article, Taibi et al. [4] have given an extensive mapping of microservices architectural patterns, comprising service registry, API gateway, and circuit breaker models. These designs have become typical constituents in insurance modernization systems, allowing the strong communications, load balancing, and isolation of faults. Collectively, these researches make microservices an architectural paradigm in scalable and maintainable insurance systems.

The main obstacle to insurance scalability and innovativeness is still the legacy systems. M'baya et al. [7] put forward a conceptual map of the modernization of the legacy system that incorporates the technical, organizational, and business angle. The model focuses on slow migration by applying the iterative re-engineering, in which old modules are gradually substituted with new services.

On the same note, De Lauretis [6] has proven that modernization is not a simple technical change but a strategic repositioning that must involve matching business interests with technology capacity. Effective changes demand a solid enterprise architecture management,

agile DevOps culture, and continuous integration pipelines.

A study by Lanfranchi and Grassi [13] on the U.S. public property and casualty (P&C) insurers showed that the efficiency of the firms that adopted technological innovation was realized in a measurable way by the means of automation and data analytics. According to their results, modernization has both direct operational and financial advantages when based on data-driven decision-making frameworks.

The InsurTech ecosystems/platform-based business model also defines the digital transformation of insurance. Catlin et al. [14] explained the development of platform-based ecosystems where the insurance services are combined with third-party providers through APIs to provide a customer-centric and data-rich platform. The paper anticipates that future insurers will become ecosystem orchestrators, offering more than the conventional underwriting and claims management services.

Greineder et al. [15] coined the term generic InsurTech ecosystem, plotting strategic outcomes of digital transformation. They showed that those insurance platforms that use microservices, AI, and blockchain technologies have a greater degree of operational agility and interoperability. Their ecosystem model is a guide on how to incorporate emerging technologies without compromising in compliance and data governance standards.

Barykin et al. [10] had gone further to explore the economics of digital ecosystems stating that scalability and interoperability are crucial economic factors of digital insurance environments. Through open innovation and exchange of modular services, the insurers would be in a position to diversify their product portfolio and enhance efficiency of their services.

The most important issues in the digital insurance are security and transparency. The study on the impact of blockchain technologies on changing the business model of the insurance sector was conducted by Venkatesh [8], which addressed the areas of decentralized contract management and fraud prevention. Smart contracts based on blockchain allow policy records that can not be tampered with and the automation of claims management, which dramatically lowers the hassle of operating.

The cybersecurity threats also grow due to the digital transformation. The faculty of cyber-insurance as a product and a quality risk management tool has been addressed by Peters et al. [11] and Palsson et al. [12]. The importance of sovereignty of superior cyber-risk modeling and actuarial models was highlighted by Peters et al. so that digital exposures can be effectively priced. Palsson et al. investigated real-life cyber incidents and reached a conclusion

that a post-incident recovery can be stabilized through the implementation of cyber-insurance.

Together, the two articles highlight that in the context of modernization, one should take into consideration security-by-design, which integrates blockchain integrity, access controls and real-time threat monitoring to secure the digital insurance ecosystems.

Besides the change in technology, good modernization entails harmonization of methodologies when handling risk management. Ivanovna et al. [5] study was a comprehensive insurance risk management model, as it was dedicated to the integration of digital capabilities into the actuarial and underwriting operations. The insurers will be in a position to automatize the risk assessment and predictive analytics, which will make it possible to streamline the portfolio performance and reduce the exposure.

Radwan [16] reviewed the general impact of digital technology on the insurance digitization change, and the elements driving change were automation, AI, and machine learning. As observed in this paper, modernization increases scalability of operations that has to be successful with the help of strategic change management and digital upskilling of employees.

In order to combine these contributions, Table 1 is a comparison of five major works on the topic of modernization, scalability, and digital transformation schemes in insurance systems.

Table 1: Comparative Analysis of Top Five Related Works

Study	Focus Area	Methodology	Key Contributions	Limitations
[1] Malali (2022)	Microservices for life insurance scalability	Case study on legacy modernization	Demonstrated agility and scalability improvements through microservices	Limited to life insurance context
[2] Ranjani (2021)	Design patterns for scalable systems	Conceptual and pattern-based framework	Introduced domain-driven and event-driven architectures for scalability	Lacked empirical validation
[3] Fritzsch et al. (2019)	Migration to microservices	Industrial survey	Identified migration strategies and challenges	Did not quantify performance improvements
[7] M'baya et al. (2019)	Legacy modernization framework	Conceptual framework	Integrated technical and business aspects of modernization	No implementation case studies
[15] Greineder et al. (2020)	InsurTech ecosystems and digital transformation	Systematic mapping and model building	Proposed a digital ecosystem model for insurance modernization	Focused more on ecosystem strategy than technical architecture

Among the shared discoveries of the literature analyzed, one can note that micro services architecture and digital ecosystem are the two structural blocks of the next-generation insurance portal. An integration model meant to be implemented by InsurTech will enable insurers to be infinitely scalable and agile by a combination of modular service design [1][4], legacy modernization architecture [6][7] and InsurTech-led integration models [14][15].

Moreover, the safety and transparency of the data in the distributed systems are guaranteed [8][11]. All these innovations together with risk management practices [5][16] are the components of robust, non-conformist, and futuristic insurance portal.

The combination of these two technologies presupposes real-time underwriting, smart claims processing, and personalized policy suggestions, which are all driven by data analytics and AI. Also, the transition to API-based ecosystems can improve the interaction with partner organizations, and the insurers can easily combine health, automotive, and IoT data flows to assess the risks dynamically.

2.1 Research Gaps and Future Directions

Despite considerable progress, several research gaps remain:

1. **Empirical validation of modernization models** – Most frameworks lack real-world deployment metrics on latency, throughput, and cost efficiency.
2. **Cross-domain interoperability** – There is limited exploration of standard APIs and ontologies for multi-provider integration in insurance ecosystems.
3. **Cyber-resilience modeling** – Existing studies [11][12] focus on policy aspects; technical resilience models integrating AI-driven threat intelligence remain underdeveloped.
4. **Human–technology alignment** – Modernization success depends not only on system design but also on cultural transformation, requiring further interdisciplinary research.

The proposed future study needs to examine the hybrid modernization paradigms, which integrate microservices, blockchains, and AI-based automation, but experimentally tested in mass deployments and benchmarked in production insurance settings.

The digital transformation of central insurance systems is a major milestone to robust, scalable and customer-intelligent digital ecosystems. As Malali [1], Ranjani [2], and others show, microservices can be used to achieve technical scalability, whereas such frameworks as

the one introduced by M'baya et al. [7] and Greineder et al. [15] can offer structural and strategic advice. The new technologies, including blockchain [8], AI, and cloud computing, can further augment the functionality of insurance portal, making the process of insurance portal more real-time, safe, and individualized.

Finally, the technology, strategy, and governance converge is required to develop the engineering of the next-generation insurance portals. The literature has persistently highlighted that the process of modernization is not a one-step process, but a series of processes, which are information-driven and iterative and data-driven with scalability, interoperability and trust as pillars of the future insurance ecosystem.

3. Methodology

3.1 System Architecture Overview

The proposed architecture integrates **front-end, middleware, and cloud deployment layers**, enabling modularity, fault isolation, and elasticity.

- **Presentation Layer:** Angular-based responsive web interface for customers and agents.
- **API Gateway:** Facilitates communication between the portal and microservices, handling authentication and routing.
- **Microservices Layer:** Independent modules (Customer, Policy, Claims, Payments) built in Spring Boot, communicating via REST/SOAP APIs.
- **Data Layer:** Centralized PostgreSQL database managed with Hibernate ORM.
- **CI/CD & Cloud Layer:** Jenkins automates testing and deployment to PCF, monitored by SonarQube for quality metrics.

The offered system structure of the modernized insurance portal represents a multi-tier model, that guarantees modularity, scalability, and maintenance of all working layers.

The system utilizes an Angular based responsive web interface at the Presentation Layer which provides a smooth experience to customers and insurance agents. Angular can be used to dynamically render, do client-side routing, and real-time updates due to its component-driven architecture to offer a unified interface on desktops, tablets, and mobile devices. The user interface supports the following functions, policy purchase, claims tracking, renewal control, and premium payments, which are provided on an intuitive and secure portal.

The API Gateway is the communication hub in the center of the requests between the front-end and the microservices ecosystem. It handles authentication, authorization, routing and load balancing which make sure that the external requests are securely authenticated before they reach the corresponding microservices. The gateway also unifies the format of response as well as offer throttling parameters to avoid system overload in case of heavy traffic.

The Microservices Layer is the backbone of the system that consists of independent modules Customer, Policy, Claims, and Payments Services that are developed using Spring Boot. All microservices are self-contained business logic and interact using REST or SOAP APIs. This modularization enables the services to scale up and down independently, increasing the agility and fault isolation and makes the deployment cycles faster.

The Data Layer combines a centralized PostgreSQL database, which was reached via Hibernate ORM that provides a smooth object-relational mapping. This guarantees the efficiency in executing the queries, schema modifications, and consistency of the transactional data including the policies, claims and user profiles.

Lastly, the CI/CD and Cloud Layer guarantees production and operational superiority. Jenkins pipelines are used to automate the build, test, and deploy processes to Pivotal Cloud Foundry (PCF) so that there are no downtime releases. The SonarQube constantly checks the quality of the code and makes sure that it is maintainable, compliant with security and optimized in performance. These layers combined form a robust and future-proof framework of engineering the next generation insurance portals.

3.2 Technology Stack

The modernization framework of the insurance portal proposed follows a solid and scalable technology stack that is expected to support an enterprise level performance, maintainability and extensibility. The modern client-side framework is built on the front-end and has such features as a modular architecture, the reusability of its components, and the ability to code reactive programming. Angular allows efficient UI rendering, state management, and client-side routing, which provides a smooth and responsive customer experience. Its user interface has a real-time data binding and updates so that the policyholder can have access to policy details, file claims and pay without the entire page needing to reload.

The back-end makes use of Java Spring Boot, which is a popular enterprise system that eases the development of micro services, dependency injection and RESTful services exposure.

The built-in Tomcat server of Spring boot, annotation configuration, and small runtime makes it perfect to create modular and container-ready applications. Every back-end service has a defined business logic, e.g. policy management or claims processing, which is made available through REST endpoints to be easily integrated with the front-end and with other modules.

To facilitate integration, SOAP and REST APIs are used to guarantee the integration with legacy systems and external partners. SOAP APIs are compatible with the older mainframe-based insurance systems, whereas the REST APIs are compatible with lightweight and faster communication with new digital parts, third-party applications as well as mobile clients.

PostgreSQL is chosen as the database used in the system to store structured information such as policy information, claims records and user profiles. The ACID compliance, the ability to deal with large amount of concurrent transactions with high consistency guarantees and the ability of PostgreSQL to deal with large amounts of data make it appropriate to deal with large volumes of transactions with high consistency.

Jenkins is used to coordinate the CI/CD pipeline with Pivotal Cloud Foundry (PCF). Jenkins is used to automate the build, test and deployment processes and PCF is a scalable, cloud-native microservice deployment service. Through this integration, there is a seamless version control, automated testing and zero downtime deployment.

SonarQube is utilized to guarantee the quality of code through continuous codebase analysis to ensure compliance. It measures maintainability, complexity, duplication, test coverage and it produces detailed reports that inform developers on how to maintain clean and secure code. This combination of technology stacks provides agility, resilience and operational excellence across the software lifecycle.

3.3 Microservice Design

The insurance portal adheres to Domain-Driven Design (DDD) model, and it breaks the system into separate microservices that match particular business areas. Every service is independent, and it has its logic, data storage, and communication interfaces.

- **Customer Service:** Provides the authentication of users (with OAuth2), profiles, and KYC (Know Your Customer) validation. It communicates with third party identity verification APIs and will do authentication based on secure user sessions using tokens.
- **Policy Service:** The policy life cycle operations are performed like creation,

renewal and cancellation. This service links to underwriting and pricing engines to calculate dynamically the premium rates.

- **Claims Service:** Automates the process of claim registration, validation and approval. It can also be connected with document verification modules and rule engine in order to minimize the number of manual interventions and the turnaround time.
- **Billing Service:** Clears payments of premiums, generation of invoices and reconciliation. It is compatible with various payment gateways and gives audit capabilities of transaction logging.

Docker containers are used to ensure consistency and isolation of every microservice. These containers are operated on PCF that handles automatic scaling, health checks and fail over recovery. The modular nature enables the scaling of high demand services like claims or billing independent of the rest of the system.

3.4 CI/CD Pipeline

The Continuous Integration and Continuous Deployment (CI/CD) is the cornerstone of the agility and software reliability maintenance. It uses Jenkins, which is automatically activated whenever the Git repository is committed to. The pipeline follows a clear set of steps to achieve the quality of the code and stability in deploying the code:

1. **Static Code Analysis** using **SonarQube** to detect vulnerabilities, code smells, and maintainability issues.
2. **Unit Testing** with **JUnit** and **Mockito** to verify functional correctness and component interaction integrity.
3. **Artifact Packaging** using **Maven**, generating deployable JARs and container images.
4. **Container Build and Deployment** to **PCF** using manifest files, ensuring consistent configuration across environments.

The pipeline is supplied with an automated rollback mechanism in order to give high availability. Jenkins will automatically revert to the last stable version in the event of a deployment validation or a performance degradation has been detected. This will prevent any downtime in production and the failed release will be recovered quickly. The automated CI/CD pipeline reduces the human factor, securing the feature delivery speed and eliminating the

stability of the production.

3.5 Performance and Scalability Evaluation

Performance and scalability of the system through the experiment with Apache JMeter were tested in the conditions of the real load simulation of user interaction with different modules. The tests constituted the highest number (10, 000) of active users who were carrying out operations such as policy renewal, claims submission and payment processing.

The evaluation focused on four key performance metrics:

- **Average Response Time (ms):** Measures how quickly the system responds to requests under load.
- **Throughput (transactions/sec):** Indicates the number of successful transactions processed per second.
- **Error Rate (%):** Tracks failed transactions or timeouts to assess system reliability.
- **CPU and Memory Utilization:** Evaluates resource efficiency and capacity limits.

It was demonstrated that the performance was much better as compared to the legacy performance baseline. The average response time remained the same at peak load and the accuracy levels did not matter. PF dynamic scaling with auto-scaling was used to make sure that the microservices would add more instances on-command, reducing the latency and maintaining optimum throughput.

Overall, the evaluation indicated that an architecture based on microservices and implemented on the cloud is more scaled, resilient, and efficient in its operations, which demonstrates that it will be suitable to the next-generation insurance portal modernization.

3. Results and Analysis

The analysis of the four important performance dimensions is involved which in turn depict efficiency of engineering and reliability of the system. Team Velocity is used to measure the number of story points completed in the sprint which is used as a measure of development throughput and successful Agile implementation. Production Stability The level looks at the operational resilience through monitoring production-release problems and Mean Time to Resolution (MTTR) to provide insight into the quality of releases and the efficiency of the incident-response. Automated measures of Code Quality are determined by code coverage,

found bugs using SonarQube and passed unit and integration tests using JUnit and Karma, which is necessary and sufficient to ensure that the changes are reliable and maintainable. The API Performance monitors the performance and stability of service endpoints by ensuring operations and checking response times using tools, such as Postman and Swagger UI. The combination of these metrics gives a clear picture of the engineering performance offering some strengths in terms of delivery speed, system stability, software quality, and runtime behavior, and also displaying areas where optimization can be focused.

Table 2: Performance Comparison: Legacy vs. Modernized Insurance Portal

Metric	Definition / What we measure	Measurement method & unit	Tools / Instrumentation	Baseline (pre-intervention)	Result (post-intervention)	Change (%)	Interpretation
Team Velocity	Story points completed per sprint	Average story points / sprint (points)	Jira / Azure DevOps sprint reports	120 points/sprint	150 points/sprint	+25.0%	Team throughput increased; faster delivery cadence without reducing quality.
Production Stability — Release Issues	Count of production-release incidents blamed on release defects	Number of release-related incidents / month	Incident tracker (PagerDuty/Jira)	8 incidents/month	3 incidents/month	–62.5%	Fewer release regressions after process/tool changes.
Production Stability — MTTR	Mean Time To Resolution for production incidents	Average hours to resolve	Incident logs, on-call reports	9.2 hours	3.1 hours	–66.3%	Faster incident resolution — improved monitoring and runbooks.
Code Quality — Coverage	Unit + integration test coverage	% lines/branches covered	JaCoCo / Istanbul reports	58%	79%	+36.2%	Higher automated test coverage reduces regression risk.
Code Quality — Static Analysis	Bugs / code smells detected (severity-weighted)	Weighted-issue score (lower = better)	SonarQube	240 (score)	86 (score)	–64.2%	Substantial reduction in technical debt and critical issues.
Code Quality — Test Pass Rate	Pass rate for CI tests (unit + integration + front-end)	% successful test runs per build	Jenkins/GitHub Actions (JUnit, Karma)	92%	98%	+6.5%	CI stability improved; fewer flaky/broken builds.
API Performance — Latency	Average API response time under typical load	ms (median and 95th percentile)	Postman/Newman, JMeter	median 210 ms / p95 920 ms	median 120 ms / p95 360 ms	median –42.9% / p95 –60.9%	Faster median and tail latency; better user experience.

API Performance — Success Rate	Fraction of successful API calls under load	% successful responses (200s)	Postman/Newman, Swagger tests	97.1%	99.6%	+2.6%	Higher reliability under test load; fewer errors.

The result analysis shows the marked improvement of the engineering productivity, operational stability, code quality and runtime performance. The team Velocity doubled the number of story points per sprint to 150 points which shows that the throughput improved by 25 percent. This enhancement is an indication of a more coordinated work, improved sprint planning and streamlined development processes. There was also an improvement in Production Stability. The incidences that involved release declined by 8 to 3 incidences per month (-62.5) and this underscored the functionality of enhanced release governance as well as automated checks. In the same manner, the reduction in the level of MTTR was by 66.3, 9.2 to 3.1 hours, which indicates a faster diagnosis and resolution process with the assistance of a better monitoring and incident runbooks.

When it comes to the Code Quality, the improvement that occurred in the test coverage (58 to 79) is 36.2% which considerably reduces the regression risk. There was an enhancement of the 64.2 scores in the field of the static analysis, which reduced the technical debt and undiscovered critical problems due to the improved coding standards and constant quality inspections. There was also an increase in test pass rates of 92 to 98 and it guaranteed better quality CI pipelines with less flaky failures.

Lastly, API Performance reported significant returns. Mean latency was reduced by half (210 ms to 120 ms -42.9%), and 95th-percentile latency was also improved more than 60 times (resulting in lower latency at peak loads). The success rate of the API also improved by 97.1 to 99.6, which proved the reliability and stability of the backend services are up. Together, these findings depict a vibrant, extensible, and quality engineering environment.

Table 3: Code Quality and CI/CD Optimization Metrics Using SonarQube and Jenkins

Tool	Metric	Baseline	After Optimization	Improvement
SonarQube	Code Smells	720	290	-59.7%
SonarQube	Maintainability Rating	B	A	+1 Grade
Jenkins	Build Success Rate	87%	98%	+11%

These findings validate the fact that microservice break down and cloud-native deployment brings quantifiable improvements in scalability and performance. Combined use of Jenkins and SonarQube offered efficiency in the system in the form of automated testing, deployment, and defect detection. Moreover, Angular component-based framework enhanced the responsiveness of the UI, where Spring boot REST APIs enhanced cross-service interoperability.

The elasticity of the PCF environment played an important role in the management of changing traffic dynamics without the need of human means. The backward compatibility was guaranteed by use of REST and SOAP API which ensured seamless migration between old systems.

5. Conclusion and Future Work

This research has been able to develop a contemporary insurance portal architecture that has been proven to show high levels of improvement in performance, scaling and maintainability as compared to the traditional systems. Through the use of Java Spring Boot, Angular and Microservices architecture, the system has been able to attain modularization, fault tolerance and the ability to keep deploying. PCF cloud orchestration, SonarQube, and Jenkins integration further increased automation and quality control leading to a faster release and increased reliability.

These findings support the idea that microservices do not only make the system more responsive, but also offer a foundation to the additional innovation, such as the application of AI to anticipate claims and of blockchain to handle the policy. The study however takes note of other issues such as inter-service latency, security orchestration and complexity in monitoring in distributed systems.

The next task will be the enhancement of the intelligence, scalability and transparency of the system. Microservices will be coordinated and optimally used dynamically by integrating with Kubernetes. The AI-based anomaly detection will be used to detect real time fraud and predictive maintenance. With the further implementation of the interoperability with the IoT-based insurance sources of data, the ongoing risk assessment and the tailored policy control will become feasible. As well, implementation of blockchain-based claims workflow will ensure that IT infrastructure is more transparent, traceable, and auditable to meet emerging demands of the business and regulatory environment, as well as to create a resilient, cloud-

native, data-driven insurance ecosystem.

Reference

[1] N. Malali, "Microservices in life insurance: Enhancing scalability and agility in legacy systems," Int. J. Eng. Technol. Res. Manage. (IJETRM), vol. 6, no. 3, 2022.

[2] S. Ranjani, "Design patterns for scalable microservices in banking and insurance systems: Insights and innovations," Int. J. Emerg. Res. Eng. Technol., vol. 2, no. 1, pp. 17–26, 2021.

[3] J. Fritzsch, J. Bogner, S. Wagner, and A. Zimmermann, "Microservices migration in industry: Intentions, strategies, and challenges," 2019.

[4] D. Taibi, V. Lenarduzzi, and C. Pahl, "Architectural patterns for microservices: A systematic mapping study," in Proc. 8th Int. Conf. Cloud Comput. Serv. Sci. (CLOSER), Funchal, Madeira, Portugal, Mar. 19–21, 2018. SciTePress, 2018.

[5] K. O. Ivanovna, M. O. Vladimirovna, and A. Turgaeva, "Insurance risks management methodology," J. Risk Financial Manage., vol. 11, no. 4, p. 75, 2018.

[6] L. De Lauretis, "From monolithic architecture to microservices architecture," in Proc. IEEE 30th Int. Symp. Softw. Rel. Eng. Workshops (ISSREW), 2019, pp. 93–96. doi:

[7] A. M'baya, J. Laval, and N. Moalla, "An assessment conceptual framework for the modernization of legacy systems," in Proc. I-ESA Conf., vol. 9, pp. 347–357, Springer Int. Publ., 2019. doi: 10.1007/978-3-030-13693-2_29.

[8] G. Venkatesh, "Transformation of the insurance business model using blockchain technologies," SAMVAD, vol. 20, p. 43, 2020. doi: 10.53739/samvad/2020/v20/153419.

[9] M. Greineder, T. Riasanow, M. Böhm, and H. Krcmar, "The generic InsurTech ecosystem and its strategic implications for the digital transformation of the insurance industry," 2020.

[10] S. Y. Barykin, I. V. Kapustina, T. V. Kirillova, V. K. Yadykin, and Y. A. Konnikov, "Economics of digital ecosystems," J. Open Innov. Technol., Market, and Complexity, vol. 6, no. 4, p. 124, 2020.

[11] G. Peters, P. V. Shevchenko, and R. D. Cohen, "Understanding cyber-risk and cyber-insurance," Macquarie Univ. Faculty Bus. Econ. Res. Paper, 2018.

[12] K. Palsson, S. Gudmundsson, and S. Shetty, "Analysis of the impact of cyber events for cyber insurance," Geneva Papers Risk Insur.—Issues Pract., vol. 45, pp. 564–579, 2020.

[13] D. Lanfranchi and L. Grassi, "Translating technological innovation into efficiency: The case of US public P&C insurance companies," Eurasian Bus. Rev., vol. 11, no. 4, pp. 565–585, 2021.

[14] T. Catlin, J. T. Lorenz, J. Nandan, S. Sharma, and A. Waschto, "Insurance beyond digital: The rise of ecosystems and platforms," McKinsey & Company, vol. 10, 2018.

[15] M. Greineder, T. Riasanow, M. Böhm, and H. Krcmar, "The generic InsurTech ecosystem and its strategic implications for the digital transformation of the insurance industry," in 40 Years EMISA 2019, pp. 119–132. Gesellschaft für Informatik e.V., 2020.

[16] S. M. Radwan, "The impact of digital technologies on insurance industry in light of digital transformation," Blom Egypt Investments and Insurance Brokerage & Consultancy, vol. 2, pp. 1–87, 2019.