International Journal of Emerging Research in Engineering and Technology
Pearl Blue Research Group| Volume 2, Issue 4, 30-38, 2021
ISSN: 3050-922X | https://doi.org/10.63282/3050-922X.IJERET-V2I4P104

Original Article

Designing for Defense: How We Embedded Security Principles into Cloud-Native Web Application Architectures

Lalith Sriram Datla[1], Rishi Krishna Thodupunuri[2]
[1]Independent Researcher, USA.
[2]Application Development Analyst at Accenture, India

Abstract *- As cloud-native development becomes the fundamental basis for present-day digital services, the securing of these evolving, distributed setups has been elevated to a topmost concern for developers and architects alike. In the following essay, we look into why the more secure your cloud-native web applications are, the more urgent and important it is. We suggest the major architectural tactics to be used for the creation of applications that are secure by default. These include zero-trust network design, policy-driven access controls, container and workload isolation, API security measures, and automated compliance enforcement. We provide a collection of practical design principles that take into account the specifics of each different cloud-native environment, based on the real-life implementation experiences from the industry. In order to make these ideas more concrete, we give a detailed description of a large-scale deployment that presents how the security aspect was introduced in every stage, from infrastructure provisioning to application runtime. Through the security lens, every single layer of the stack, from threat modeling, secure CI/CD pipelines, and runtime anomaly detection to identity-aware traffic segmentation, is viewed. The result: strengthened threat resilience, reduced attack surface, and greater compliance were the benefits derived—all while keeping agility and developer velocity intact. For those that are in the field, this piece provides a working scheme that can be used to make cloud-native applications that are not only designed but are secure. The example is of use to those who are giving technical guidance as well as those who have implementation solutions in mind. The advice and cases submitted will help them to establish trustful applications and systems most economically and effectively.*

Keywords *- Cloud-Native, Security Architecture, Devsecops, Microservices, Zero Trust, Secure Web Apps, Container Security, Kubernetes, Shift-Left Security, Identity And Access Management (IAM), TLS, Encryption, CI/CD Pipeline Security, Threat Modeling, Runtime Protection, Least Privilege, Service Mesh, Policy Enforcement, Secure Coding Practices, Vulnerability Management, Infrastructure As Code (IAC), Network Segmentation, Application Firewall, Secrets Management, Automated Compliance, Role-Based Access Control (RBAC), Observability.*

1. Introduction

The rapid proliferation of cloud-native web applications, characterized by the power of microservices, containers, and orchestration platforms like Kubernetes that keep the cover, is also exploited by the malicious actors. The traditional perimeter-based security architecture simply fails to deal with the unregulated and fluid nature of these infrastructures. In a reality with an increasing prevalence of ransomware, supply chain attacks, and sophisticated exploits directed at APIs and container runtimes, organizations have no other option but to reconsider the securing of their applications from the basics. The fact that third-party components are more and more used, the workloads are short-lived, and the deployment cycles are continuous makes the challenge even more difficult; therefore, conventional reactive methods are not a good choice for the time to come.

This new threat landscape is a clear signal that a more strategic approach is necessary: from successive fixes only invented after the threat has already become real to security inherent in the software itself throughout the entire lifecycle. Security-by-design is not just a good suggestion; it is an obligation for the protection of data and user privacy and the observance of the regulations in today's distributed architectures. Companies that do not take this direction are at risk of incurring regulatory fines, may be boycotted by clients, and are subject to the financial consequences of an unauthorized disclosure. On the other hand, those that decide to be on the initiative side shall benefit from being strategically favorable and turn security from a blocker to a driver of innovation.

This new threat landscape is a clear signal that a more strategic approach is necessary: from successive fixes only invented after the threat has already become real to security inherent in the software itself throughout the entire lifecycle. Security-by-design is not just a good suggestion; it is an obligation for the protection of data and user privacy and the observance of the regulations in today's distributed architectures. Companies that do not take this direction are at risk of incurring regulatory fines, may be

boycotted by clients, and are subject to the financial consequences of an unauthorized disclosure. On the other hand, those that decide to be on the initiative side shall benefit from being strategically favorable and turn security from a blocker to a driver of innovation.

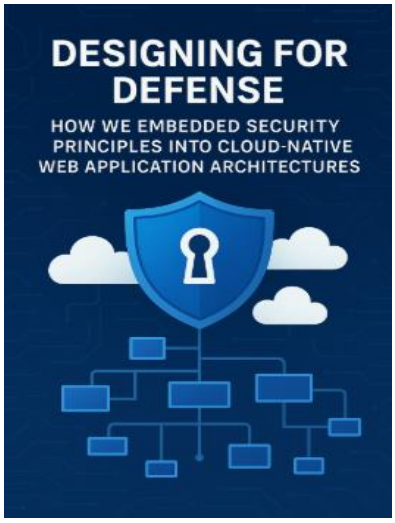

Fig 1: Designing For Defense

This new threat landscape is a clear signal that a more strategic approach is necessary: from successive fixes only invented after the threat has already become real to security inherent in the software itself throughout the entire lifecycle. Security-by-design is not just a good suggestion; it is an obligation for the protection of data and user privacy and the observance of the regulations in today's distributed architectures. Companies that do not take this direction are at risk of incurring regulatory fines, may be boycotted by clients, and are subject to the financial consequences of an unauthorized disclosure. On the other hand, those that decide to be on the initiative side shall benefit from being strategically favorable and turn security from a blocker to a driver of innovation.

This is an article that is structured following an approach that lays down the principles according to which security is deeply rooted in the architecture, development, and operations of cloud-native web applications. We aim to present a theoretical guide that sticks tightly to present-day DevSecOps methods while presenting some unusual use cases of the secure design architecture that covers the infrastructure, application logic, and runtime operations. We want to underline that every level has its unique pacing that contributes to the general infrastructure protection.

In order to achieve this goal, we have introduced a security engineering-based methodology that includes discipline, agile delivery, and cloud-native tooling. The design plan is uncovered in our work, including threat modeling sessions, security requirement mapping, and technology selection to risk profiles. The paper then covers architectural decisions, including zero-trust concepts, service meshes for safe communication, and IAM rules. Using tools for both static and dynamic analysis, automatic security scanning in CI/CD pipelines, and ongoing validation via anomaly detection, our approach for evaluating the product is in line with past concepts.

Thereby, the security gets woven in the fabric of the product that leads to resilient, scalable, and compliant applications. The flexibility and the effectiveness of the approach present a clear approach to architects, developers, and operations teams when establishing security in their cloud-native ecosystems.

2. Security by Design in Cloud-Native Context

Security by design is a key principle that underlines taking into account security aspects at the beginning of system architecture and development instead of dealing with security as a secondary or reactive response to vulnerabilities. In the context of cloud-native environments, this philosophy becomes even more necessary. These applications are often made up of various microservices, which execute in different parts of the dynamic infrastructure, such as Kubernetes clusters and public or hybrid clouds. When using CI/CD pipelines that operate automatically, containers that are only available for a short time, and frequent release cycles, the time for identifying and patching vulnerabilities after the deployment phase is getting shorter.

Embedding security from the beginning is a single scalable and sustainable way to deal with changing threats. The emergence of cloud-native systems brings up a set of problems that create the necessity and difficulty of security by design. Firstly, the magnitude of the number of deployments, which often goes beyond hundreds or thousands of containers and services, makes it necessary to introduce security mechanisms that are automated, policy-driven, and manageable without manual intervention. Secondly, the transient state of workloads, exemplified by the ability of containers to be started and stopped within a matter of seconds, totally precludes the use of traditional security tools and tactics like static firewalls or IP-based access lists.

Finally, the introduction of automation and continuous delivery, while boosting the speed and efficiency, could also lead to the spread of insecure configurations or vulnerable code unless they are controlled by strong governance. The project for the secure-by-design software ensures adjusting the old-school security rules and measures to the dimension where they can be handled in a decentralized, automated way. The sunglasses of security std that this endeavor requires are like implementation of checkpoints in DevOps flows, policy execution as code, IaC to preserve the configuration through version control, and handling security as a joint responsibility among developers, operations, and security teams. In order to be secure in such kinds of environments, observability and real-time threat detection also become extremely important.

In the cloud-native ecosystem, security by design is fundamentally about the ability of a system to persist through threats and uncertainties. The visibility of a situation and the speed of a response are more important characteristics here than a protective system that is perfect. This necessitates a change in culture from separated information security to an integrated part of the architecture, processes, and tools, where security becomes a culture in the organization. Besides that, with appropriate design patterns and principles, the enterprise can develop systems that are not only safe but also quick and efficient and can handle the advantages derived from cloud-native technology without any loss of trust or compliance.

3. Core Security Principles Embedded

3.1. Zero Trust Architecture (ZTA)

Unlike in the case of the traditional perimeter security where Zero Trust security is implemented, the Zero Trust security model denies the view that a user, a system, or a service can be naturally trusted either inside or outside the network boundary. The suggestion here is that all requests are considered negative until there is evidence that the opposite is true.

3.2. Defense in Depth

Cloud-native environments benefit Through specific steps concerning our case, like the strict access control policy, the reinforcement of both external and internal factors, and the change of the previous status to the one that is aligned with the policy or ethical guidelines, strong authentication and authorization became reality and service-to-service communication became mTLS protected, attracting identity-aware proxies and service meshes to work. This approach led to access to sensitive data and services only if the identity, the context, and compliance with the company's policy was confirmed.

3.3. Least Privilege and IAM

By using the Role-Based Access Control (RBAC) method and partnering it with a federation IdP for the authentication of services as well as users, we managed to enforce the principle of 'least privilege' to all users and services. In particular, the Kubernetes RBAC roles were not only closely connected to those services but also to other resources as well. With identities of services that were removed from the previously used identities only to be rotated at regular intervals, this would make it a lot harder to move laterally, and, suppose in the case of one service being hacked, that would limit the blast radius. By using monitoring agents, one could predict any malicious actions of the system and even after bypassing the first security layers, the threats could still be monitored and stopped.

3.4. Encryption Everywhere

Data security was ensured with end-to-end encryption techniques. We employed TLS for all data in transit, both externally (HTTPS) and internally (service mesh-based mTLS). At rest, sensitive data, such as secrets, tokens, and personal information, were encrypted using cloud-native KMS (Key Management Services), thus making them only accessible to the right people. The access policy was further made strict to ensure only the right people accessed the data. Frequent changing of keys and the recording of all the events (audit logging) were therefore carried out to guarantee compliance with the HIPAA and GDPR industrial regulations.

3.5. Immutable Infrastructure and Patch Automation

Under the immutable infrastructure principles, the networks were not patched in place but rather fully redeployed after updates. This way, we guaranteed that they remained the same from one environment to the other and saved lots of extra work. Not only the application-level but also Operating System (OS) images were transformed into code and then tested for weaknesses using tools such as Trivy and Clair. To enable the latest security updates to be applied without developer intervention, CI/CD pipelines

incorporated automatic patching and rebuild workflows. What's more, the continuous integration of the Software Composition Analysis (SCA) eases the detection and fixing of open-source library risks during the build stage.

The embedding of these defining principles has given us an overall security position that is detailed, scalable, and able to maintain functionality through failures, yet there is no slowing in innovation. This not only helped in dealing with technical threats but also in the direction of governance and compliance, thus allowing for secure-by-design cloud-native development at scale.

4. Architectural Patterns and Tools

Architectural patterns and tool choices are very important in the process of the design of cloud-native web applications that are secure by default. These choices are the ones that have to support the modern application workloads' characteristics of being dynamic, distributed, and ephemeral. In this article, we are going to represent the architectural patterns and tools that we choose in order to have secure micro services, containers, orchestration platforms, and runtime environments from attackers—preferring the one that works well in terms of safety and operation.

4.1. Micro services Security: Sidecar Pattern and API Gateways

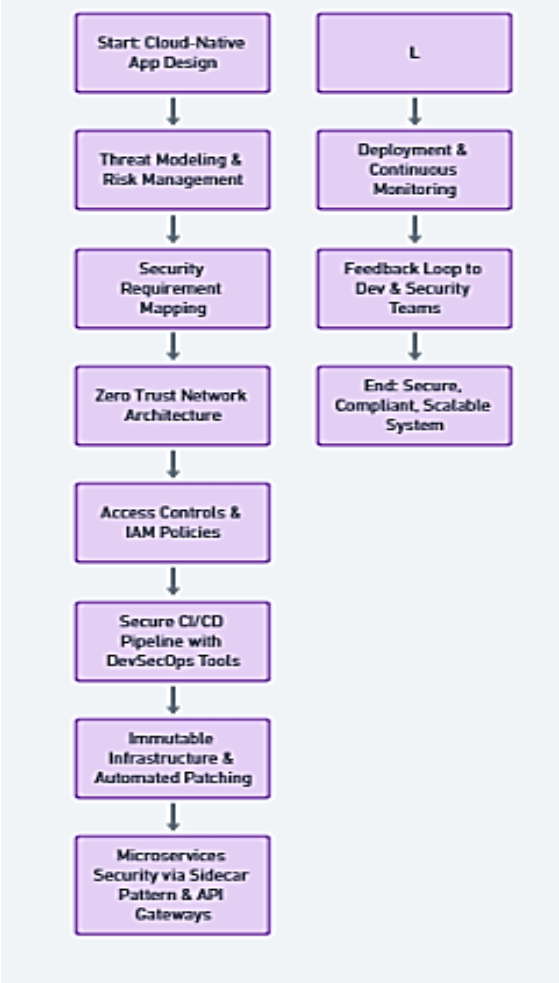

Fig 2: Micro Services Security

The deconstruction of applications into loosely coupled, independently deployable services, which in turn allows for the improvement of agility but also comes with the baggage of an increase in complexity and risk, is typical microservices architecture. To ensure that these challenges are handled safely, we decided to use the sidecar pattern, i.e., a security agent or proxy that is specially dedicated and deployed alongside each service container within a pod. Such a sidecar is responsible for the interception of all the traffic in and out, as well as the policing of policies like encryption (mTLS), rate limiting, and authentication.

Moreover, we have made use of API gateways that facilitated the source of control for the external traffic that comes into the microservices architecture. Among the tools that we utilized were Kong and AWS API Gateway, which in turn provided capabilities like SSL termination, request validation, IP whitelisting, and integration with identity providers (e.g., OAuth2, OpenID Connect). By having the management of these resources in a central location, we eliminated threats from no foolproof security controls being in use or security logic being mismatched by service operations.

4.2. Container and Kubernetes Security

The main thing here is that the locking down of containers and their orchestration layers is vital in the cloud-native system, where ensuring security is the first step. In the picture of the side of the container, we constructed Pod Security Policies (PSPs) and Pod Security Standards (PSS) to establish what tasks a pod could execute, such as privilege escalation, network access on the host, and device mounts. These policies also made sure that only the necessary permission was given to the operations.

Runtime protection was ensured by the use of several tools, such as Falco a runtime security engine that is open source and is capable of detecting anomalous behavior in Kubernetes environments. With Falco, the primary task was to create alerts that were supposed to occur, such as insider a container or unexpected file writes. Also, we have utilized AppArmor and Seccomp, which had the benefit of creating container security profiles with very low syscall execution, which decreased the chances of exfiltrating the rest of the host system.

4.3. Secure Service Meshes: Istio and Linkerd

In our system, to ensure that the service-to-service communication is safe, we turned to the service mesh paradigm. Using Istio, we took the opportunity to implement mTLS without needing to change the application code, and this was our main option. Other than that, it was a source of traffic control, observability and policy enforcement through the control plane. Linkerd, a similar but lighter solution, was also examined for places where the undesirable performance overhead is minimized.

These service meshes came with the sidecar model to ensure that the processes of encryption, authentication, and authorization will be performed in a hidden and seamless manner. Moreover, they also offered a fair amount of telemetry data that could be used to feed the observability pipeline, thus enriching the visibility into the traffic patterns and detecting potential threats.

4.4. Secrets Management

Handling sensitive credentials like API keys, tokens, and database passwords in a secure way is of utmost importance. Our choice for centralized, auditable secret management was HashiCorp Vault and AWS Secrets Manager. By using these ones, the keys were stored in a secure manner, access controls were strictly enforced and rotation of the keys was done on a regular basis.

When talking about the functionalities of Vault, the dynamic secrets feature stood out by its great capability of supplying services with credentials that are short-lived. As a result, the chances of the token leaking due to being long-lived were not only reduced but also eliminated. The process of secrets handling focused on the concept of providing secrets as services instead of having the secrets hardwired into the app or config files.

4.5. Monitoring and Observability

Security and monitoring are like two sides of the same coin; in the absence of one, you can only expect half of the whole. A proper observability stack was set up, which consisted of well-structured logging, metrics collection, and real-time alerting. The logs were consumed by Fluent Bit and then sent to Elasticsearch and Amazon CloudWatch to be stored securely, and they were also in-transit and at-rest encrypted.

We chose to use Prometheus and Grafana for system metrics and AWS GuardDuty and Datadog for behavioral anomaly detection. From these instruments, we received notifications when unusual activity occurred, like multiple failed authentication attempts or lateral movement attempts. Hence, it became possible for security analysts to promptly respond to the alerts and perform forensic analysis.

We put together a secure and resilient cloud-native environment by using the architectural patterns together with the specialized tools. Each layer was designed in a secure way so that the security was built-in; thus, we could set up policies consistently, monitor actively and answer effectively. This tool-based strategy addresses security by design in the cloud-native deployments and is a workable and scalable way to get there. without bringing it into physical reality.

5. Secure SDLC and CI/CD Integration

Where software is built continuously and on a cyclical basis, they pass the stages of development & deployment of the code, testing, and integration of continuous SDLC. This forward-thinking method, also called Left-Shift Security, brings the first steps of your security considerations to the development stage; thus, you put your teams in a position where they are able to discover and eliminate security vulnerabilities before the vulnerabilities insert into the product.

Left-Shift Security finds its place at the very beginning with source code analysis. We came up with Static Application Security Testing (SAST), regulated at the coding level, to pick up insecure coding patterns and logic traumas while the software is in the development stage. To satisfy the latter, we checked two solutions: Semgrep and SonarQube. With the help of Software Composition Analysis (SCA) provided by Snyk and Trivy, we were able to perform a dependency check of open-source vulnerabilities.

We used DevSecOps tools to automatically check security in our CI/CD pipelines—set up on services such as GitHub Actions and GitLab CI. Security checks were automated at the stage of infrastructure-as-code (IaC) configurations by using Checkov and TFSec, which confirmed that the cloud services were provisioned with the least privilege and hardened defaults. Furthermore, Aqua Security software was a significant layer of protection during container image scanning by getting the runtime configurations' integrity verified and identifying deviations from authorized baselines.

For the sake of ensuring compliance and putting the guardrails for better security into effect, the Policy as Code (PaC), with the help of Open Policy Agent (OPA) and Gatekeeper, was employed. The aforementioned software defined governance policies and delivered them directly in the clusters of our Kubernetes and CI/CD pipelines. To illustrate, the implemented policies were disallowing the use of root containers and also the use of unapproved base images.

Security issues that were found during the building process initiated autonomous loops of fixing the issues. Moreover, the developers were notified by the comments in pull requests and chats along with the context, thereby promoting quick fixes without the effect of velocity drop. In addition, the integration of issue trackers such as Jira allowed security defects to be triaged and prioritized in the same way as feature development; thus, the balance between delivery speed and the security posture was still maintained.

At the end of the day, the results from the scans of code, dependency, and configuration were all combined into dashboards that were used for the management of vulnerabilities. Regular inspections of the dashboards made it possible for us to read the exposure, detect the remediation trends, and expose the areas, if any, that require security education. It should be noted that through these tools and best practices that were weaved into the workflow of the developers, we were able to achieve a culture of continuous security, where the guardrails were the new normal, as no gatekeepers were necessary. Besides, the velocity of the development was ensured in safety at all times.

6. Threat Modeling and Risk Management

While working in the cloud-native environment, we realized that the threats change very quickly and the architecture is very complicated, which makes the threat modeling and the risk management the most important elements of a secure design process. We don\'t want to consider these activities as a one-time event so we have chosen a kind of continuous threat modeling approach that is able to adapt to the unpredictable characteristics of the environment and the interdependencies of the microservices.

We took into account the OWASP Top 10 and other popular frameworks to be the foundation of our threat modeling, in addition to the fact that we have optimized them to be relevant to both web applications of the traditional type and the cloud-native cases. Among the risks were inadequate access control, misconfigured APIs, cloud storage, and the exposure of third-party elements. These kinds of risks along with real-life attack vectors were converted to a diagram that enabled the developers and the people responsible for the architecture for the security of the control based on relevance and impact.

We adopted STRIDE to structure threat modeling workshops and at the same time, we gave an example from PASTA to illustrate how we are working at risk modeling. In the case of STRIDE, the use of such a tool helped the teams to go through threats between trust boundaries, while PASTA followed the malicious person's thinking that attempted to attack the trust boundaries, making the attack interconnected to technical controls and business risks. These workshops took place throughout sprint cycles, as well as release planning; thus, the security was on every iteration of the process.

One of the most fundamental parts of our risk management process was risk scoring, through which we could measure and estimate the probability and the impact of security risks. These scores resulted in the choice of clarification and facilitated the process of identifying issues and dealing with the most severe ones first. In terms of technical vulnerabilities, we used CVSS-based metrics, while with the contextual risk, we worked through business impact models. The progress was displayed on the dashboard and whenever there was something critical, there were alerts that notified the teams so they could solve the problem.

Crucially, feedback loops also enabled security engineers to communicate the threat intelligence they gathered to the developers directly. In-person security engineers and developers collaborated on improving threat mitigations, amending source code, and deploying automatic controls where applicable. This continuous togetherness allowed the joint ownership of security information and the constant alignment of risk management with agile development. Through an iterative process of threat modeling, risk scoring, and feedback embedding in workflows, we undertook the journey from being reactive to proactive, and we were able to build secure apps that can change and confront future threats.

7. Case Study: Secure Architecture for a FinTech SaaS Platform

7.1. Background

In the highly regulated field of financial services, the most important things are safety and observance of norms, not options. The given case study contains the subject of creating a secure architecture for a B2B FinTech SaaS platform that processes and stores sensitive financial data, such as payroll transactions, tax documents, and third-party integrations with banking systems. This software served the middle market, providing such services as automated reconciliation, real-time reporting, and safe data exchanges with regulators.

The risks were too high; the system had to grow with clients as well as reveal security that cannot be breached to satisfy the requirements of regulatory and corporate purchasing departments. Data protection, maintaining customers' trust, and getting compliance with regulations like SOC 2 Type II and ISO 27001 were the main goals that we had to achieve.

7.2. Architecture Overview

Using a simple design, the platform used serverless compute with AWS Lambda for quick tasks and Kubernetes clusters for longer services like APIs, data pipelines, and reporting engines. In this way, we considered speed, cost-effectiveness, and the isolation of workloads based on the data's regional fit, sensitivity, and compute patterns.

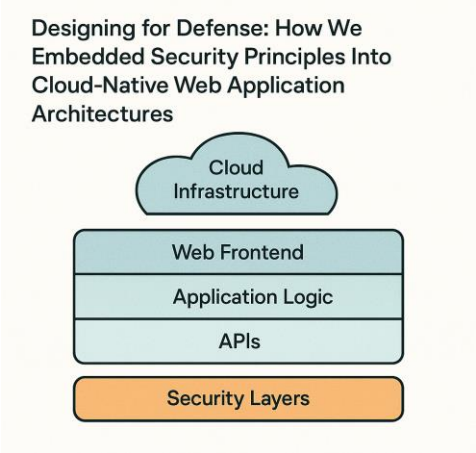

Fig 3: Designing For Defense

A multi-region spread ensured that traffic was routed through a global CDN and regional API gateways, which in turn made it possible for the system to efficiently give disaster recovery and keep data according to the rules of the region. The division of separate regional sectors into data residency and compliance boundaries was necessary to meet the various regulations of the financial sectors in different jurisdictions.

7.3. Security Implementation Highlights

- Zero Trust Networking Between Services All internal service communication was secured under a Zero Trust model. Zero Trust meant that all services were identified and authenticated based on the mTLS protocol provided by the Istio service mesh. Services shared only the minimum necessary information with each other while still adhering to the essential principles of the trust model. The identity of any client requesting services was examined, and authentication with SPIFFE

identities and short-lived certificates was enforced for each client that was truly safe. Verifying the client's identity and ensuring that the request is logical were essential requirements. Initially, all the necessary points that were agreed upon were provided with maximum service.

- A unified Identity and Access Management (IAM) layer handled authentication and authorization. SAML was used for B2B clients and OAuth2 with OpenID Connect for third-party and mobile clients to simplify IdPs' (Identity Providers) interaction with the system. RBAC was a method by which the rules of the organization were implemented effectively through the delineation of who gets what entitlement based on identity traits only, thereby making sure the users and the services accessed only the required resources at the time.
- Data Tokenization and Encryption Modules To ensure that data is safe at all the points. we chose the following strategy: our data protection was structured using multiple layers of security. We applied data tokenization to PII (Personal Identifiable Information) and sensitive financial data before storing them. Removing data not needed in the analytics pipeline was another method to minimize data exposure via tokenization. For instance, we encrypted all data end-to-end using AWS KMS with envelope encryption, heavily relying on TLS 1.3 for data transit. Without high-level access to data from the service provider, the customer could become the ultimate decision-maker in every encryption scenario by using fine-grained key policies and customer-managed keys.
- Runtime Scanning and Audit Trails When we transitioned from continuous monitoring to real-time monitoring, we chose a container security solution from Falco and Aqua Security to oversee potential threats and assess the security levels of containers and serverless environments. The tools flagged and resolved any compromise of privilege or data by unauthorized access. Furthermore, a security system, with which a full history of events was captured, describing in detail who performed the action and the machine it came from, and then all this, including writing to the Immutable ReadOnly (Amazon S3 with object locking), was indexed in the query log, giving highly efficient access to data for compliance audits, incident response, and investigations.

7.4. Results

The secure design of the architecture yielded clear benefits in terms of security and business, as evidenced by the metrics obtained:

- An independent cybersecurity firm conducted third-party penetration tests, and no high or critical vulnerabilities were discovered, thus proving that the platform's defense-in-depth strategy was performing well.
- The time used for the response to a certain incident was decreased, and it was made possible by the centralized logging, the set of alerts, and a collection of security manual responses that were automatic players in the rapid identification and confinement of the anomalies.
- The platform accomplished the goal of having SOC 2 Type II and ISO 27001 certification in the very first year of its operation this gave it more validity to the enterprise customers and made it simple to go through the procurement cycle.

Being a case study, this is an example of how a modern FinTech platform that is located in the cloud can add the most secure and most compliant architecture patterns, the most unbreakable identity management, and the best proactive threat detection to be the best off in terms of security and compliance—yet the only thing that they can lose is speed or user experience. SaaS providers can use this case study as a model for managing highly demanding, data-related environments.

8. Conclusion

Designing secure cloud-native web applications is not only about securing tools onto applications but it rather requires a significant cultural shift besides the architectural change. In this security-by-design journey, we highlighted those strategies through which security was imbibed at every phase: the implementation of zero trust networking, the enforcement of defense-in-depth practices, the integration of strong IAM and encryption policies, and the use of contemporary DevSecOps tooling in CI/CD pipelines. Architectural patterns such as sidecars, secure service meshes, and hybrid serverless-Kubernetes deployments have been capable of building the execution environments that are both scalable and secure. Every layer, from secrets management to runtime scanning, was adding to the system's resilience.

Nevertheless, the most crucial discovery involved the realization that security is not only a technical pursuit but also a cultural transformation. Cooperation among developers, security engineers, and operations teams was indispensable. They participated in the cross-functional threat modeling sessions, owned the vulnerabilities of the system together, and there were continuous feedback loops that facilitated the process of security shifting from a bottleneck to a shared responsibility and velocity enabler. Lessons were not without pain. Performance and security, two opposite poles, were handled through careful tuning; hence, the challenge of mTLS enforcement that came with a minimal increase of latency was accepted due to the trust guarantee it came with. The cloud architecture with immutable infrastructure was a solution for the issue of irregular allocation of resources but it required that the teams be trained again, especially those who were used to manual hotfixes.

Ultimately, stability and compliance were given priority in each trade-off task. Engineers working on projects in the era of cloud-native are very clearly left with a clear message: Security should be upgraded from being one-dimensional to becoming a first-class citizen. In this way, security will not be limited to just being secure, but it will also be able to develop the required capabilities. Further, the author points out that the author also believes that systems are not only secure but also scalable, resilient, and trusted if we carry security along the entire way, from design and development to implementation. This means that there is no better time than the present to move the security process to the left, adopt secure design, and embed security deeply into the architecture itself.

References

[1] Torkura, Kennedy A., et al. "Leveraging cloud native design patterns for security-as-a-service applications." *2017 IEEE International Conference on Smart Cloud (SmartCloud)*. IEEE, 2017.

[2] Gilbert, John. *Cloud Native Development Patterns and Best Practices: Practical architectural patterns for building modern, distributed cloud-native systems*. Packt Publishing Ltd, 2018.

[3] Laszewski, Tom, et al. *Cloud Native Architectures: Design high-availability and cost-effective applications for the cloud*. Packt Publishing Ltd, 2018.

[4] Davis, Cornelia. *Cloud Native Patterns: Designing Change-Tolerant Software*. Simon and Schuster, 2019.

[5] Anusha Atluri. "Data Migration in Oracle HCM: Overcoming Challenges and Ensuring Seamless Transitions". *JOURNAL OF RECENT TRENDS IN COMPUTER SCIENCE AND ENGINEERING (JRTCSE)*, vol. 7, no. 1, Apr. 2019, pp. 66–80

[6] Garrison, Justin, and Kris Nova. *Cloud native infrastructure: patterns for scalable infrastructure and applications in a dynamic environment*. " O'Reilly Media, Inc.", 2017

[7] Mammo, Kidus Wendimagegn. *rials: Cloud-Native Security*. Diss. Aalto University, 2020.

[8] Ali Asghar Mehdi Syed. "High Availability Storage Systems in Virtualized Environments: Performance Benchmarking of Modern Storage Solutions". *JOURNAL OF RECENT TRENDS IN COMPUTER SCIENCE AND ENGINEERING (JRTCSE)*, vol. 9, no. 1, Apr. 2021, pp. 39-55

[9] Sangeeta Anand, and Sumeet Sharma. "Big Data Security Challenges in Government-Sponsored Health Programs: A Case Study of CHIP". *American Journal of Data Science and Artificial Intelligence Innovations*, vol. 1, Apr. 2021, pp. 327-49

[10] Sethi, Manish. *Cloud Native Python*. Packt Publishing Ltd, 2017.

[11] 11. 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

[12] Keery, Sean, Clive Harber, and Marcus Young. *Implementing Cloud Design Patterns for AWS: Solutions and design ideas for solving system design problems*. Packt Publishing Ltd, 2019.

[13] Yasodhara Varma Rangineeni, and Manivannan Kothandaraman. "Automating and Scaling ML Workflows for Large Scale Machine Learning Models". *JOURNAL OF RECENT TRENDS IN COMPUTER SCIENCE AND ENGINEERING (JRTCSE)*, vol. 6, no. 1, May 2018, pp. 28-41

[14] Mammo, Kidus. "Online Platform for Interactive Tutorials: Cloud-Native Security." (2020).

[15] Veluru, Sai Prasad, and Mohan Krishna Manchala. "Federated AI on Kubernetes: Orchestrating Secure and Scalable Machine Learning Pipelines". *Essex Journal of AI Ethics and Responsible Innovation*, vol. 1, Mar. 2021, pp. 288-12

[16] Kumar, Tambi Varun. "Layered App Security Architecture for Protecting Sensitive Data." (2016).

[17] Anusha Atluri. "The Revolutionizing Employee Experience: Leveraging Oracle HCM for Self-Service HR". *JOURNAL OF RECENT TRENDS IN COMPUTER SCIENCE AND ENGINEERING (JRTCSE)*, vol. 7, no. 2, Dec. 2019, pp. 77-90

[18] Kothawade, Prasad, and Partha Sarathi Bhowmick. "Cloud Security: Penetration Testing of Application in Micro-service architecture and Vulnerability Assessment." (2019).

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

[20] Ingeno, Joseph. *Software Architect's Handbook: Become a successful software architect by implementing effective architecture concepts*. Packt Publishing Ltd, 2018.

[21] Veluru, Sai Prasad. "AI-Driven Data Pipelines: Automating ETL Workflows With Kubernetes". *American Journal of Autonomous Systems and Robotics Engineering*, vol. 1, Jan. 2021, pp. 449-73

[22] Emily Harris, and Bennett Oliver. "Event-Driven Architectures in Modern Systems: Designing Scalable, Resilient, and Real-Time Solutions." *International Journal of Trend in Scientific Research and Development* 4.6 (2020): 1958-1976.

[23] Paidy, Pavan. "Zero Trust in Cloud Environments: Enforcing Identity and Access Control". *American Journal of Autonomous Systems and Robotics Engineering*, vol. 1, Apr. 2021, pp. 474-97

[24] Langen, S. F. *An architectural design for LAN-based web applications in a military mission-and safety-critical context*. MS thesis. University of Twente, 2016.

[25] Hoffman, Kevin. "Programming WebAssembly with Rust: unified development for web, mobile, and embedded applications." (2019): 1-220.