International Journal of Emerging Research in Engineering and Technology
Pearl Blue Research Group| Volume 5, Issue 3, 73-83, 2024
ISSN: 3050-922X | https://doi.org/10.63282/3050-922X.IJERET-V5I3P108

Original Article

Secure Banking Microservices on AWS: A DevSecOps Framework

Karthik Allam
Big Data Infrastructure Engineer at JP Morgan &Chase, USA.

Abstract - *These days, it's hard to be open to new ideas and ways of doing things while simultaneously keeping your digital bank account safe. This post speaks a lot about a DevSecOps plan for keeping banking microservices safe on Amazon Web Services (AWS). The proposed method links the development, security, and operations teams by making security a part of the full development process instead of something that comes later. AWS Identity and Access Management (IAM), Key Management Service (KMS), Secrets Manager, and Cloud Trail are all AWS-native services that work together to make sure that identity management, data protection, and full audit trails are all strong. We deployed the system in containers using either Amazon Elastic Kubernetes Service (EKS) or ECS. We also automatically look for weaknesses and check for compliance on a regular basis to make sure the system is safe and can grow. The security model used by Amazon Guard Duty and Security Hub is "zero trust." This means that they only let people in who need to do their jobs. They also contain rules like "automated threat detection" and "encryption at rest and in transit." CI/CD pipelines work better when they have tools for analyzing both static and dynamic code, enforcing policy-as-code, and rolling back changes automatically. It's simple to get things properly the first time. Centralized logging, Cloud Watch for real-time monitoring, and proactive alerting systems all let you see what's going on and fix problems more quickly. A DevSecOps model that incorporates everything could help financial companies come up with new ideas faster, make sure they follow the rules (such PCI DSS and GDPR), build trust with customers, and lower the risks to their operations and security. In this article, we discuss best practices and why it's so crucial for everyone to know what they need to do. This implies that developers, security engineers, and operations teams all need to work together to make sure that security is a part of every stage of the financial application's life cycle. The architecture lets businesses offer microservices that are safe, fast, and reliable in a cloud environment that is always evolving. It also helps organizations quickly adjust to new policies and risks that come up in the digital world.*

Keywords - *Secure banking, microservices, AWS, DevSecOps, cloud security, CI/CD, infrastructure as code, compliance, container security, IAM, threat modeling, PCI DSS, AWS Well-Architected, observability, monitoring, automation, zero trust, security pipelines, secrets management, banking compliance.*

1. Introduction

The banking business has changed a lot in the previous ten years. It doesn't employ big, antiquated systems anymore. It leverages microservices solutions that may change and expand instead. Banking apps used to have a lot of code, which made them big and hard to use. This programming helped clients, maintained track of their accounts, and handled payments. These huge systems worked wonderfully, but they were very dependent on each other. Because of this, they couldn't change, come up with new ideas rapidly, or get things done swiftly. Banks started using microservices to improve their IT systems and make it easier for individuals to utilize digital banking. Microservices let you build, test, and alter parts of big apps without affecting the whole thing. This lets banks swiftly introduce new features without breaking anything. Microservices make security even more crucial, especially in the banking business, which has a lot of rules and deals with a lot of private information. Microservices include more APIs, distributed data flows, and linkages between services, which makes it easier for hackers to get into networks. This is not the same as monolithic systems, which let you use security solutions that are all in one spot. Every service needs its own security. The same measures should be followed to check who is who, provide permission, keep an eye on things, and protect data. Trust is very crucial in banking, therefore even a tiny error can have a big effect on your money and your reputation. When you make and use microservices, the first thing you should think about is security. Security should always be a top priority during the software development lifecycle. Traditional development methods often miss this difficulty.

As cloud-native programming grows more popular, having good security is increasingly more crucial. Many current banking solutions are built on cloud platforms like Amazon Web Services (AWS) because they can evolve and grow as needed and have all the features that microservices architectures need. AWS has a lot of tools that let developers construct and deploy a lot of

microservices at once. Some of these tools are Elastic Kubernetes Service (EKS), Elastic Container Service (ECS), Lambda, and API Gateway. Encryption, real-time monitoring, and identity and access management (IAM) are just a few of the many security features that come with these technologies. AWS only charges banks for what they use. This helps banks save money and handle workloads that change. This is really helpful when there are a lot of transactions. DevSecOps is a novel and crucial way to keep cloud-based microservices safe. DevSecOps is like DevOps, but it makes the whole CI/CD process safer. The development, security, and operations teams work together to make sure that security processes are always automated, proactive, and continuing. This strategy helps the company quickly introduce new products while still making sure it follows important banking laws like PCI DSS and GDPR.

Fig 1: Secure Banking Microservices on AWS: A DevSecOps Framework

This post is about the rules for safe banking microservices on AWS from a DevSecOps point of view. First, we'll talk about how microservices are made and how they help banks. Next, we'll discuss the different security problems that come up and the regulations that go with them. Next, we'll talk about the finest AWS services for using microservices and real-world DevSecOps strategies that will keep everything safe. We show you how to make a solid and flexible DevSecOps framework for banks.

1.1. Understanding Banking Microservices

Microservices for banking is a new way to build systems. They broke them up into discrete services that can work on their own and talk to each other using explicit APIs. All services should be able to make accounts, take payments, and discover fraud. Because they use simple protocols like REST or gRPC, these services can easily talk to each other and to other systems. Microservices are different from regular monolithic systems since they let developers, testers, and deployers work on their own. This helps businesses get things done more quickly and make new things more quickly. Using microservices in banking has a lot of advantages. One good thing about it is that it could grow. People can add to services when they require them. This lets schools and other groups make the most of what they currently have without spending too much money. The system is modular, so if one microservice has a problem or needs to be altered, the rest of the system won't be affected. This means that the system will be less likely to crash and more stable. Banks can quickly adapt to changes in the market, client needs, and the law since their deployment cycles are shorter. Microservices can assist create an innovative culture by letting smaller teams work on different aspects, test out new technologies, and make changes with less risk.

Microservices are great, but they also make security problems that businesses need to deal with. Microservices are easier to hack because they aren't all in one place. This means that each service has to be safe on its own. Attackers could sneak in through weak APIs, poorly constructed containers, or unsafe ways to talk to each other. It's important for all services to use the same procedures for authentication, authorization, and data encryption, but this could be challenging to do in a decentralized system. You also need full automation and monitoring to keep secrets, tokens, and certificates safe on a large scale. This will keep people who shouldn't be able to get in from getting in. It gets much harder to follow the rules issued by the industry, including PCI DSS

and GDPR. All of a financial institution's microservices must retain audit trails, secure customer data, and limit who may access it. This means that a security plan must be made that fulfills the rules and can also adapt to deal with new threats and changes in the law. There are a variety of tools in AWS that make it easier for developers to make microservices that are safe and can grow quickly. With Amazon ECS and EKS, it's simple to manage and deploy workloads in containers. AWS Lambda lets you run functions without a server, which is safer and easier. API Gateway is a flexible and safe solution to manage API traffic. AWS Shield and AWS WAF (Web Application Firewall) protect you against DDoS assaults and other typical online security issues. These tools work well together to let you build cloud-native banking microservices on a solid foundation.

2. DevSecOps Principles for Banking

In the financial industry, where safety and following the laws are very important, old ways of building software don't always keep up with the need for new ideas and the development of cyber threats. In this case, DevSecOps is a new way of doing things that modifies the rules. The main goals of DevOps are to work collaboratively, automate chores, and finish things more quickly. These ideas are what DevSecOps is based on. This is done by making sure that security is a part of the process of manufacturing anything. Everyone who works on the project, from developers to operational teams to security experts, is responsible for security from the start. DevSecOps doesn't see security as the final thing you do when you make anything. Instead, it makes sure that security is a part of every process, from designing the code and testing it to using it and keeping it operating all the time.

2.1. CI/CD Pipelines with Integrated Security

CI/CD pipelines are particularly important for building microservices these days since they make it easy and safe for teams to add new features. In the world of finance, safety should always come before speed. Companies may find security gaps before products go live by adding security testing to their CI/CD workflows. Static application security testing (SAST), dynamic application security testing (DAST), and infrastructure-as-code evaluations are all examples of processes that could be part of a shared pipeline. These tests that run on their own make sure that any improper code or incorrect settings are found right away. AWS Code Pipeline and Code Build are two technologies that make it simple to connect to security scanners and tests that make sure everything is up to code. This makes sure that the code is always tested for safety and quality.

2.2. Shift-Left Security and Early Threat Detection

Shift-left security is what DevSecOps is all about. It shows how important it is to fix the security holes as quickly as possible, ideally while the program is still being constructed, rather than after it has been released or while it is being used. Banks that use microservices need to use safe coding methods & do threat modeling from the very beginning of the design process. It's easier & cheaper to fix problems when you find them early. It also makes it less likely that hackers will be able to get into operating systems. Some of the technologies that can help you find weak libraries or code patterns before they can be used for undesirable things are automated dependency assessments, container image analysis & static analysis tools. Shift-left methods encourage a culture of proactive security by getting developers to think like an enemy & hunt for holes while they are working on new features.

2.3. Automated Testing for Vulnerabilities and Compliance

DevSecOps has a lot to do with automation. It helps find security gaps and make sure that rules like PCI DSS and GDPR are obeyed. Automated test suites might look for typical security weaknesses in code all the time, such as SQL injection, cross-site scripting (XSS), and APIs that aren't set up right. You can look at the base images of containerized banking microservices to see if there are any known security problems. This will help you remember any dependencies that could cause problems. You may also add compliance testing to the pipeline to check that systems are following the rules about who can access data, how it is stored, and how it is protected. By automating these tests, schools can keep their security procedures the same and reliable. This doesn't make it take longer to deploy.

2.4. Toolchains for Secure Pipelines

You need to use the right technologies to build a good DevSecOps pipeline for banking microservices. AWS Code Pipeline handles the stages of writing code, testing it, and deploying it. AWS Code Build handles the steps of compiling code, testing it, and detecting security holes. You can use these services with security products from other businesses. For instance, Aqua Security examines container images while they are operating to make sure they are safe. Snyk, on the other hand, checks open-source dependencies for problems. You can be guaranteed that the infrastructure, containers, and code that make up the microservices architecture are all being observed and protected when you employ these technologies together. AWS Security Hub, Guard Duty, and Inspector are built-in tools that can be added to the DevSecOps toolchain to keep an eye on security and discover threats all the time.

2.5. Secure Development Lifecycle (SDLC) for Financial Services

In the Secure Development Lifecycle, banks require more than simply new technology. You also need to modify the way people work. DevSecOps makes sure that all part of the Software Development Life Cycle is safe, from gathering requirements to running production. People make a list of things they need to do to keep safe and then do those things to get ready. During the design and development stages, developers use safe coding guidelines, automated linters, and code review methods to lower the risk. Automated security scans, penetration tests, and compliance checks are all part of the testing process. They do things for them all the time, even when they don't have to. Finally, tools for continuous monitoring watch the program as it is being installed and used, look for faults, and make sure that security standards are met. This version of the SDLC can only be used by banks who follow industry standards and rules. It's crucial to talk about how to safely design cloud resources, how to encrypt data correctly, and how to keep logs that meet audit criteria from the beginning. Adding DevSecOps to the SDLC helps financial companies be more open, respond more quickly to new risks or changes in the law, and save time and money on compliance audits.

3. AWS Security Foundations for Microservices

You need to know all of AWS's security requirements to keep banking microservices safe in the cloud. According to the Shared Responsibility Model, AWS and its customers are both in charge of security. This method works with a lot of AWS security solutions and best practices. According to this concept, AWS is in charge of keeping the cloud infrastructure safe. This comprises network layers, hardware, and data centers. The bank is responsible for keeping the cloud safe. This includes installing apps, keeping data safe, and controlling who can use it. This disparity shows that banks and other financial institutions need to keep an eye on their workloads to protect their consumers, stay within the law, and protect themselves from growing cyber threats.

3.1. IAM and Least Privilege Strategies

In a framework for banking microservices, AWS Identity and Access Management (IAM) is a key aspect of keeping track of who may do what. One of the most critical parts of cloud security is keeping track of who can get in. Following the principle of least privilege is very important. In other words, a person, service, or program should only be able to do what it needs to perform and nothing extra. A payment microservice should only be able to see a customer's private information if it needs to. AWS IAM roles, groups, and rules let you decide who may see what. Single sign-on (SSO) and multi-factor authentication (MFA) are two more ways to keep private financial transactions safe. You can make sure you don't allow someone too much access or put things in the incorrect place by regularly checking IAM rules, automatically discovering roles that are too open, and using tools like AWS IAM Access Analyzer.

3.2. Network Segmentation with VPC, Subnets, and Security Groups

To construct microservices, you need to know how to divide your network up properly. This keeps services separate and makes sure that communication is safe. Banks can establish their own cloud spaces with AWS Virtual Private Cloud (VPC). In these situations, they can set up their own IP ranges, route tables, and gateways. People who are not on private subnets can't see microservices that are on the public internet. This is possible because of load balancers and public subnets that don't have many ways to get in. You can use security groups and Network Access Control Lists (NACLs) to decide what traffic is allowed to enter and depart each microservice. A database service can stop all other inquiries and just let requests from the application layer through on its own. This layered approach to network security makes it less likely that a breach will enable someone to move sideways.

3.3. Secrets Management Using AWS Secrets Manager and Parameter Store

Apps that deal with money need to keep things like passwords, API keys, and encryption keys safe. If you don't encrypt your credentials or arrange them in a way that makes them hard to find, you're nearly sure to get into problems. You may safely store, change, and get secrets with AWS Secrets Manager and Parameter Store. Secrets Manager connects to AWS services and automatically updates things like database passwords. This makes sure that microservices always use safe, up-to-date credentials. AWS Systems Manager's Parameter Store lets you store secrets and configuration data in a safe and organized way. This is done by using the AWS Key Management Service (KMS) to encrypt the data. By adding these tools to their CI/CD pipelines, businesses can keep important information safe while it is being built and sent out.

3.4. Encryption at Rest and In Transit

Apps that deal with money need to keep passwords, API keys, and encryption keys safe. If you don't encrypt your credentials or arrange them in a way that makes them hard to find, you're nearly sure to get into problems. You may safely store, change, and get secrets with AWS Secrets Manager and Parameter Store. Secrets Manager connects to AWS services and keeps things like database passwords up to date on its own. This makes sure that microservices always use passwords that are safe and up to date. AWS Systems Manager's Parameter Store is a safe and clean place to keep secrets and configuration data. To do this, the AWS Key Management Service (KMS) is used to encrypt the data. By incorporating these solutions to their CI/CD pipelines, companies can keep important data safe while it is being built and sent out.

3.5. Logging and Monitoring with CloudWatch, Guard Duty, and Security Hub

Cloud security has to always be able to see everything. Amazon CloudWatch gathers logs and analytics from microservices as they happen. This allows teams keep an eye on performance, find problems, and fix them quickly. Amazon Guard Duty employs machine learning to look through logs and network activity for possible threats. It can find possible security problems, like API requests that don't seem right or access that shouldn't be authorized. The AWS Security Hub lets you examine all of your security findings in one place. It uses Guard Duty, Inspector, and other tools to provide you a full picture of your security. Banks can resolve problems more quickly if they set up systems that automatically send out alerts and fixes. This suggests that you probably won't be attacked for a long time.

3.6. Zero Trust in AWS Environments

The idea of Zero Trust is particularly appealing to banks because it is getting tougher and harder to resist cyberattacks. You can't trust anything or anyone on the network using a Zero Trust architecture. They always check your ID, your behavior, and your location to see if you can go in. AWS could use IAM policies, micro-segmentation, encryption, and constant monitoring all at once to make Zero Trust work. Temporary tokens or certificates let microservices talk to each other. There are strict criteria about whether or not an API call can go through. You can construct stronger walls of trust with AWS Private Link and service-to-service mutual Konkret TLS (mTLS). A Zero Trust framework can help banks keep their microservices architecture safe from threats that emerge from inside the bank and from people who are always on the move. These AWS security rules can help banks and other financial institutions build a secure and safe structure for their microservices. IAM, VPC architecture, secrets management, encryption, and real-time monitoring all work together to provide a security-focused framework that follows the best practices and policies in the business Using DevSecOps methods, the next section will show you how to build on and connect these basic parts over time in the microservices. This will make sure that security is constantly on and works by itself.

4. Designing a Secure DevSecOps Framework

To create a safe DevSecOps architecture for banking microservices, architecture, operations, and culture all need to work together very carefully. In conventional development, security is a separate thing. DevSecOps ensures that security is included into the development process from the start and stays a top focus after deployment. This method makes your microservices system strong, scalable, and very safe from the rising number of cyber threats to financial systems. The architecture has to be flexible and big enough to fulfill the strict rules that banks have to follow and the fact that cloud-native systems are always changing.

4.1. Architectural Overview: Bringing together Microservices and DevSecOps

Automation, cooperation, continual feedback, and following the rules are the four main parts of a secure DevSecOps system. Microservices are modular and distributed by nature, which is in line with the ideas of DevSecOps. Each microservice may have its own CI/CD pipeline, and security may be a part of it. This lets teams add things faster without sacrificing quality. This AWS architecture usually uses a combination of managed services, such ECS/EKS for managing containers, API Gateway for handling traffic, and AWS WAF for security at the edge. It checks and keeps an eye on security at every step of the pipeline. Infrastructure as Code (IaC) ensures that environments are always set up the same way. CloudWatch and GuardDuty are two examples of observable technologies that let you keep an eye on things and find hazards in real time.

4.2. Developing a Methodical Framework

The DevSecOps technique comprises four steps: Code, Build, Deploy, and Run. There are also continuous feedback loops and compliance checks at each step.

4.2.1. Code Phase: Linting, scanning, and secure programming

The developers are the first people responsible for making sure that microservices are safe. The team has to always apply safe coding techniques. They should utilize static code analysis and linting tools to make sure they follow coding standards and find any security holes early on. Adding tools like SonarQube or Checkmarx to a developer's IDE may help them find security holes, such as hardcoded secrets and not checking input properly.

- Code Reviews and Working Together on Code: Peer reviews make it easier for people to work together to fix security problems. Automated pull request verifications may set quality standards, including not letting sensitive information into repositories.
- Dependency Management: Because open-source libraries are so important, technologies like Snyk and OWASP Dependency-Check are needed to find known security holes in third-party components.
- Shift-Left Security: At this stage, threat modeling must be done to find prospective attack points before development moves further. This proactive approach meets regulatory standards and cuts down on the need for modifications later on.

4.2.2. Construction Phase: Checking the Container Image and Making Sure It Works

Containerized workloads are common in microservices, and the build phase is when these containers are produced. But containers generally hold not just the application code but also parts of the operating system, which might make them less secure.

- Container Image Scanning: You may add tools like Aqua Security, Trivy, or Clair to the CI pipeline to check container images for security holes in both the application and base layers. When high-severity issues are found, the scans must be automated and employ fail-build methods.
- Immutable Artifacts: Using immutable container images makes sure that the same artifact is deployed in the same way in all environments, which reduces the chance of discrepancies.
- Automated Unit and Integration Testing: Security-focused test suites must go beyond functional testing by finding weaknesses like weak authentication methods or unsafe data handling.
- Managing Secrets: Don't put secrets in containers. Use AWS Secrets Manager or Parameter Store to securely add runtime credentials.

4.2.3. The Deployment Phase: Infrastructure as Code (IaC)

The deployment stage is when you plan and manage the infrastructure that the microservices run on. In financial systems, human settings are very dangerous and often make mistakes, which is why Infrastructure as Code (IaC) is so important.

- Terraform with AWS Cloud Formation: These solutions automatically set up secure places like VPCs, subnets, IAM roles, and security groups. Templates may have defined security baselines that make sure compliance from the start.
- Policy-as-Code: Tools like Open Policy Agent (OPA) or AWS Config Rules may check that infrastructure settings follow compliance rules before they are put into use.
- AWS Code Pipeline and Code Deploy may include security approval steps as part of their automated deployment pipelines. This makes sure that no changes are made without proper validation.
- Deployments with no downtime and blue/green: Dependability is just as important as security in financial systems. Blue/green or canary deployments cause very little disruption during upgrades and let you reverse back changes to fix any problems or issues.

4.2.4 Execution Phase: Protecting the runtime and finding anomalies

When you deploy microservices, security at runtime is really important. This means protecting workloads from imminent threats and keeping an eye on conduct at all times.

- Finding threats in real time: Instruments such as AWS Guard Duty, AWS Inspector, or external runtime security solutions (e.g., Aqua or Sysdig) discover anomalies, unlawful acts, and malware.
- Service-to-Service Authentication: Microservices must use mTLS or temporary tokens to prove who they are to each other. This is based on a "zero confidence" model that does not presume that services trust each other.
- API Protection: AWS API Gateway may help protect against API-based attacks by limiting the number of requests, verifying users, and authenticating requests.
- Cloud Watch, Security Hub, and Kinesis make sure that all requests and events are logged, creating a full audit trail for compliance and incident response.
- Automated Incident Response: AWS Step Functions or Lambda functions may start automated steps, such as isolating workloads that have been hacked or deleting suspicious access.

4.3. Adding Compliance Checks to Pipelines

For financial companies, following the rules is just as important as keeping their technology safe. PCI DSS, GDPR, and ISO 27001 are examples of rules that put strict limits on data, access, and infrastructure. To avoid delays during audits, compliance checks may be done automatically within pipelines.

- Frameworks like HashiCorp Sentinel or AWS Config can make sure that rules like encryption requirements, secure key rotation, and proper resource labeling are followed.
- Audit Readiness: Using technologies like AWS Audit Manager to make compliance reports all the time makes sure that evidence is always available for regulatory examinations.
- Automated Data Privacy Assessments: Static analysis methods may find probable data leaks or bad data storage practices that can break compliance rules.

4.4. Ways to get ongoing security feedback

A strong DevSecOps architecture is always changing and learning. Feedback loops make sure that every security incident, test result, or strange behavior makes the system stronger as a whole.

- Threat Intelligence That Keeps Coming: Information from AWS services like Guard Duty or outside threat feeds might help you change your firewall rules, IAM policies, or application safeguards.
- Post-event Reviews: After a security incident or a known weakness, teams should have blameless retrospectives to improve processes and, if possible, automate fixes.
- Monitoring measurements like mean time to detect (MTTD), mean time to recover (MTTR), and the number of vulnerabilities found before production are used to see how well the DevSecOps architecture works.
- Developer Empowerment: Teams can quickly fix problems and build a culture of shared security responsibility by giving developers actionable security feedback via dashboards and automated reporting.

5. Case Study: Secure Banking Application on AWS

5.1. Background: A Fictional Banking Scenario

NextGen Bank is a medium-sized bank that wants to improve its online offerings. It does a good job of explaining how to apply a DevSecOps architecture for microservices in actual life situations. Before, NextGen Bank utilized a single core banking technology for everything, from managing their accounts to processing payments. The monolith had worked well in the past, but it was evident that it was holding back innovation and not meeting customers' needs for mobile-first, real-time financial services. The bank couldn't take on new customers because it was too busy with service requests and it would take too long to improve its system. They asked whether using microservices on AWS would be a good idea. The hardest part was making sure that the new design could be changed without breaking safety and compliance rules.

5.2. Things that get in the way Faced with Legacy Systems

There are a number of problems with legacy systems in banking:

- Problems with scalability: The monolithic architecture made it exceedingly hard for components to scale on their own. An unexpected rise in payment transactions might cause problems with many other modules, such as loan processing & account statements.
- Longer Development Cycles: Adding the latest features meant testing and redeploying the complete monolithic application, which led to longer release cycles & more downtime.
- Security Risks: The previous system utilized outdated encryption methods, didn't fix things regularly, and didn't provide enough information about prospective weaknesses.
- Lack of Compliance: Following stricter rules like PCI DSS and GDPR was hard since the settings were too strict & there were no automatic compliance checks.

NextGen Bank's IT management have decided to move to a microservices architecture on AWS and use a DevSecOps approach to make sure security is included into every stage of a project.

5.3. Step-by-Step Process Move to Microservices on AWS

The migration procedure was broken down into many other important steps:

5.3.1. Assessment and Strategy Formulation:

A detailed look at the previous system was done to find out what important business operations it could do. These were put into groups of microservices domains, such as onboarding clients, payments, fraud detection & these account services. Requirements for security and compliance were set ahead of time. Using Docker, components were changed into their containerized services. This modularization made it easier for services to be deployed & scaled on their own. The main orchestration platform was chosen to be Amazon Elastic Kubernetes Service (EKS), which made automated scaling and rolling updates possible.

- API-Centric Design: The latest API layer, controlled by AWS API Gateway, was added to make it safe for services & outside clients to talk to each other. This made it possible to carefully handle requests, authentication & rate limiting.
- Infrastructure as Code: AWS Cloud Formation templates made it easier to automate the creation of VPCs, subnets, IAM roles & these security groups. Terraform was used for complex deployments across several regions to make sure they were always the same and could be done again.
- Incremental Migration: Services were moved over time, beginning with less important ones like the account statement service. Blue/green deployments made sure that customers didn't have any downtime throughout the transfer.

5.4. Added layers of security

Because financial information is so sensitive, numerous layers of security were put in place:

- Perimeter Security with WAF and Shield: AWS online Application Firewall (WAF) was put in place to protect against common online security holes like cross-site scripting and SQL injection. AWS Shield Advanced protects against DDoS attacks, making sure that services are always available even when there is a lot of traffic or someone tries to attack them.
- Network Segmentation: Services were set up on private subnets within a VPC. Only the API Gateway and load balancers could be reached from public subnets. Security groups made it so that microservices could only talk to each other in the least-privileged way.
- Identity and Access Management (IAM): Each microservice was given its own IAM role, which made sure that no service had more privileges than it needed. Administrative access has to use Multi-Factor Authentication (MFA).
- Secrets and Key Management: AWS Secrets Manager and KMS were used to maintain & change the passwords for the database, the API keys & the encryption keys. All sensitive information was secured using TLS while it was being stored & sent.
- Surveillance and Threat detection: AWS Guard Duty helped with ongoing threat detection, while AWS Config made sure that all resources met with compliance criteria.

5.5. Example of a CI/CD Pipeline Using Code Pipeline and Container Security

NextGen Bank set up a CI/CD pipeline with the following phases to speed up the deployment of the latest features while still meeting strict security standards:

- Source Stage: Developers put code in a safe Git repository. Pull submissions begin automated tests for code quality & security holes using tools like Snyk and SonarQube.
- Building Phase: AWS Program The build created the code, performed unit tests & used Aqua Security to scan the container images. Builds were stopped right away when high-severity vulnerabilities were found.
- During the test phase, dynamic application security testing (DAST) was done in a controlled setting. Automated integration tests showed that the API was secure and followed PCI DSS criteria.
- Deployment Phase: AWS CodePipeline used blue/green deployment methods to send the containers to EKS clusters. Terraform scripts made sure that infrastructure provisioning was safe & also consistent.
- Post-Deployment Security: Runtime protection programs kept an eye on how containers acted to look for more problems, such as processes that shouldn't be there or suspicious network activity.

5.6. Framework for Observability (Cloud Watch, X-Ray)

To improve speed and keep an eye on security, it was important to understand how microservices worked. The observability stack was made up of:

- Amazon Cloud Watch: Provided logs & custom metrics for tracking their application performance, resource use & error rates.
- AWS X-Ray: Enabled request tracing across microservices, which helps developers quickly find bottlenecks & potential security holes in service-to-service connections.
- Security Hub Integration: AWS Security Hub brought together their information from Guard Duty, Inspector, and other security products to provide a complete picture of the bank's security.

This observability setup enables the DevSecOps team to promptly notice anomalies, analyze issues, and execute fixes prior to harming users.

5.7. Insights Gained and Optimal Strategies

The process of moving NextGen Bank gave us a lot of important information:

5.7.1. Adding security to the design process.

Putting security measures in place early on in the migration process, such during API design or Infrastructure as Code development, turned out to be considerably more effective than adding them later.

- Automation is Key: Automating security checks, compliance checks & deployments cut down on human error by a lot and sped up delivery times.
- Input that keeps coming Mechanisms: The team's ongoing improvement of security procedures was made easier by consistent feedback from monitoring tools, audits, and threat detection systems.

5.7.2. Helping Developers Get More Done:

Teaching developers how to write safe code and giving them automatic scanning tools improved the overall security of the system without slowing down development.

- Resilience and Scalability: The bank was able to handle both expected and unexpected traffic surges by using blue/green deployments, auto-scaling EKS clusters, and distributed architectures.
- Compliance as Code: By adding compliance checks to CI/CD pipelines, every release was ready for audits and met all regulatory requirements.

NextGen Bank, a bank, has changed an old system into a secure, modern, and scalable platform by using AWS-native services and a DevSecOps strategy. Microservices, layered security, and automated processes that run all the time have all improved performance and built trust with customers and regulators. This tale shows that even the banking sector, which is heavily regulated, can come up with new ideas while still keeping cloud protection secure.

6. Conclusion

Banks are using Amazon Web Services cloud-native environment & DevSecOps approaches, especially microservices, to build & protect their digital services in a whole new way. IAM, VPC, Secrets Manager, CloudWatch, & Security Hub are all Amazon Web Services -native solutions that can help banks & other financial firms keep their networks safe, regulate who can access what, & track what's occurring on all of their networks in real time. DevSecOps makes the whole process of building software safer, which makes these benefits even better. It finds problems early, makes sure everything is up to code on its own, & keeps security information coming in. Even so, banks can still follow the rules & win their clients' trust by offering services that are flexible, dependable, & can change. It's very crucial to remember that safety is the most important factor in a culture. Infrastructure as Code could help locations obey the rules better & be more reliable. It might also help to scan containers & protect APIs to keep things safe. The financial industry is turning digital, & new technologies like quantum-safe encryption, Artificial intelligence-powered threat detection, & serverless microservices will make it safer & more creative. Everyone should do everything they can to protect their money apps. Your cloud systems may perform better & be safer if you automate them, keep a watch on them effectively, and continually search for methods to improve them.

References

[1] Das, BK Sarthak, and Virginia Chu. *Security as Code: DevSecOps Patterns with AWS*. "O'Reilly Media, Inc.", 2023.

[2] Chandramouli, Ramaswamy. "Implementation of devsecops for a microservices-based application with service mesh." *NIST Special Publication* 800 (2022): 204C.

[3] Manda, Jeevan Kumar. "Augmented Reality (AR) Applications in Telecom Maintenance: Utilizing AR Technologies for Remote Maintenance and Troubleshooting in Telecom Infrastructure." *Available at SSRN 5136767* (2023).

[4] Jani, Parth. "Predicting Eligibility Gaps in CHIP Using BigQuery ML and Snowflake External Functions." *International Journal of Emerging Trends in Computer Science and Information Technology* 3.2 (2022): 42-52.

[5] Pakalapati, Naveen. *Blueprints of DevSecOps Foundations to Fortify Your Cloud*. Naveen Pakalapati, 2023.

[6] Shaik, Babulal. "Developing Predictive Autoscaling Algorithms for Variable Traffic Patterns." *Journal of Bioinformatics and Artificial Intelligence* 1.2 (2021): 71-90.

[7] Mulder, Jeroen. *Multi-Cloud Strategy for Cloud Architects: Learn how to adopt and manage public clouds by leveraging BaseOps, FinOps, and DevSecOps*. Packt Publishing Ltd, 2023.

[8] Balkishan Arugula, and Vasu Nalmala. "Migrating Legacy Ecommerce Systems to the Cloud: A Step-by-Step Guide". *Los Angeles Journal of Intelligent Systems and Pattern Recognition*, vol. 3, Dec. 2023, pp. 342-67

[9] Jani, Parth. "Embedding NLP into Member Portals to Improve Plan Selection and CHIP Re-Enrollment". *Newark Journal of Human-Centric AI and Robotics Interaction*, vol. 1, Nov. 2021, pp. 175-92

[10] Mishra, Sarbaree, et al. "A Domain Driven Data Architecture for Improving Data Quality in Distributed Datasets". *International Journal of Emerging Trends in Computer Science and Information Technology*, vol. 2, no. 3, Oct. 2021, pp. 81-90

[11] Bayya, Anil Kumar. "Cutting-Edge Practices for Securing APIs in FinTech: Implementing Adaptive Security Models and Zero Trust Architecture." *International journal of applied engineering and technology (London)* 4 (2022): 279-298.

[12] Guntupalli, Bhavitha, and Surya Vamshi ch. "Designing Microservices That Handle High-Volume Data Loads". *International Journal of AI, BigData, Computational and Management Studies*, vol. 4, no. 4, Dec. 2023, pp. 76-87

[13] Shmeleva, Ekaterina. "How microservices are changing the security landscape." (2020).

[14] Manda, Jeevan Kumar. "Privacy-Preserving Technologies in Telecom Data Analytics: Implementing Privacy-Preserving Techniques Like Differential Privacy to Protect Sensitive Customer Data During Telecom Data Analytics." *Available at SSRN 5136773* (2023).

[15] Vasanta Kumar Tarra. “Claims Processing & Fraud Detection With AI in Salesforce”. *JOURNAL OF RECENT TRENDS IN COMPUTER SCIENCE AND ENGINEERING (JRTCSE)*, vol. 11, no. 2, Oct. 2023, pp. 37–53
[16] Smith, Bridger A. *A DEVSECOPS APPROACH FOR DEVELOPING AND DEPLOYING CONTAINERISED CLOUD-BASED SOFTWARE ON SUBMARINES*. Diss. Monterey, CA; Naval Postgraduate School, 2021.
[17] Mishra, Sarbaree. “Comparing Apache Iceberg and Databricks in Building Data Lakes and Mesh Architectures”. *International Journal of AI, BigData, Computational and Management Studies*, vol. 3, no. 4, Dec. 2022, pp. 37-48
[18] Mohammad, Abdul Jabbar, and Seshagiri Nageneini. “Temporal Waste Heat Index (TWHI) for Process Efficiency”. *International Journal of Emerging Research in Engineering and Technology*, vol. 3, no. 1, Mar. 2022, pp. 51-63
[19] Allam, Hitesh. "Declarative Operations: GitOps in Large-Scale Production Systems." *International Journal of Emerging Trends in Computer Science and Information Technology* 4.2 (2023): 68-77.
[20] Talakola, Swetha. “Leverage Microsoft Power BI Reports to Generate Insights and Integrate With the Application”. *International Journal of AI, BigData, Computational and Management Studies*, vol. 3, no. 2, June 2022, pp. 31-40
[21] Mishra, Sarbaree, et al. “Leveraging In-Memory Computing for Speeding up Apache Spark and Hadoop Distributed Data Processing”. *International Journal of Emerging Research in Engineering and Technology*, vol. 3, no. 3, Oct. 2022, pp. 74-86
[22] Datla, Lalith Sriram. “Optimizing REST API Reliability in Cloud-Based Insurance Platforms for Education and Healthcare Clients”. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, vol. 4, no. 3, Oct. 2023, pp. 50-59
[23] Abdul Jabbar Mohammad. “Leveraging Timekeeping Data for Risk Reward Optimization in Workforce Strategy”. *Los Angeles Journal of Intelligent Systems and Pattern Recognition*, vol. 4, Mar. 2024, pp. 302-24
[24] Bird, Jim, and Eric Johnson. "A SANS survey: rethinking the Sec in DevSecOps: Security as Code." *SANS Institute Reading Room, SANS Institute* (2021).
[25] Guntupalli, Bhavitha. “Data Lake Vs. Data Warehouse: Choosing the Right Architecture”. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, vol. 4, no. 4, Dec. 2023, pp. 54-64
[26] Veluru, Sai Prasad. "Leveraging AI and ML for Automated Incident Resolution in Cloud Infrastructure." *International Journal of Artificial Intelligence, Data Science, and Machine Learning* 2.2 (2021): 51-61.
[27] Immaneni, J. (2022). Strengthening Fraud Detection with Swarm Intelligence and Graph Analytics. *International Journal of Digital Innovation*, *3*(1).
[28] Mishra, Sarbaree, and Jeevan Manda. “Building a Scalable Enterprise Scale Data Mesh With Apache Snowflake and Iceberg”. *International Journal of Emerging Research in Engineering and Technology*, vol. 4, no. 2, June 2023, pp. 95-105
[29] Sirigina, Praveen Varma. "Digital Payment Security: A Developer Framework." (2023).
[30] Shaik, Babulal. "Automating Compliance in Amazon EKS Clusters With Custom Policies." *Journal of Artificial Intelligence Research and Applications* 1.1 (2021): 587-10.
[31] Chaganti, Krishna Chaitanya. "The Role of AI in Secure DevOps: Preventing Vulnerabilities in CI/CD Pipelines." *International Journal of Science And Engineering* 9.4 (2023): 19-29.
[32] Nookala, G. (2023). Secure multiparty computation (SMC) for privacy-preserving data analysis. *Journal of Big Data and Smart Systems*, *4*(1).
[33] Mohammad, Abdul Jabbar. “Dynamic Labor Forecasting via Real-Time Timekeeping Stream”. *International Journal of AI, BigData, Computational and Management Studies*, vol. 4, no. 4, Dec. 2023, pp. 56-65
[34] SOLANKE, ADEDAMOLA ABIODUN. "Enterprise DevSecOps: Integrating security into CI/CD pipelines for regulated industries." (2022).
[35] Manda, J. K. "Data privacy and GDPR compliance in telecom: ensuring compliance with data privacy regulations like GDPR in telecom data handling and customer information management." *MZ Comput J* 3.1 (2022).
[36] Mishra, Sarbaree. “Scaling Rule Based Anomaly and Fraud Detection and Business Process Monitoring Through Apache Flink”. *International Journal of AI, BigData, Computational and Management Studies*, vol. 4, no. 1, Mar. 2023, pp. 108-19
[37] Arugula, Balkishan. “AI-Powered Code Generation: Accelerating Digital Transformation in Large Enterprises”. *International Journal of AI, BigData, Computational and Management Studies*, vol. 5, no. 2, June 2024, pp. 48-57
[38] Datla, Lalith Sriram, and Rishi Krishna Thodupunuri. "Designing for Defense: How We Embedded Security Principles into Cloud-Native Web Application Architectures." *International Journal of Emerging Research in Engineering and Technology* 2.4 (2021): 30-38.
[39] Mohammad, Abdul Jabbar. “Predictive Compliance Radar Using Temporal-AI Fusion”. *International Journal of AI, BigData, Computational and Management Studies*, vol. 4, no. 1, Mar. 2023, pp. 76-87
[40] Shaik, Babulal. "Network Isolation Techniques in Multi-Tenant EKS Clusters." *Distributed Learning and Broad Applications in Scientific Research* 6 (2020).
[41] Balkishan Arugula. “Personalization in Ecommerce: Using AI and Data Analytics to Enhance Customer Experience”. *Artificial Intelligence, Machine Learning, and Autonomous Systems*, vol. 7, Sept. 2023, pp. 14-39

[42] Edmundson, Chris, and Kenneth G. Hartman. "SANS 2022 DevSecOps Survey: Creating a Culture to Significantly Improve Your Organization's Security Posture." *URL https://www. sans. org/white-papers/sans-2022-devsecops-survey-creating-cultureimprove-organization-security* (2022).
[43] Patel, Piyushkumar. "Bonus Depreciation Loopholes: How High-Net-Worth Individuals Maximize Tax Deductions." *Distributed Learning and Broad Applications in Scientific Research* 5 (2019): 1405-19.
[44] Nookala, G. (2023). Serverless Data Architecture: Advantages, Drawbacks, and Best Practices. *Journal of Computing and Information Technology*, *3*(1).
[45] Tortoriello, Valentina. *Definition of a DevSecOps Operating Model for software development in a large Enterprise*. Diss. Politecnico di Torino, 2022.
[46] Guntupalli, Bhavitha. "ETL Architecture Patterns: Hub-and-Spoke, Lambda, and More". *International Journal of AI, BigData, Computational and Management Studies*, vol. 4, no. 3, Oct. 2023, pp. 61-71
[47] Allam, Hitesh. "Unifying Operations: SRE and DevOps Collaboration for Global Cloud Deployments". *International Journal of Emerging Research in Engineering and Technology*, vol. 4, no. 1, Mar. 2023, pp. 89-98
[48] Abdul Jabbar Mohammad. "Integrating Timekeeping With Mental Health and Burnout Detection Systems". *Artificial Intelligence, Machine Learning, and Autonomous Systems*, vol. 8, Mar. 2024, pp. 72-97
[49] Jani, Parth, and Sangeeta Anand. "Apache Iceberg for Longitudinal Patient Record Versioning in Cloud Data Lakes". *Essex Journal of AI Ethics and Responsible Innovation*, vol. 1, Sept. 2021, pp. 338-57
[50] Tan, Junsheng. "Ensuring component dependencies and facilitating documentation by applying Open Policy Agent in a DevSecOps cloud environment." (2022).
[51] Datla, Lalith Sriram. "Proactive Application Monitoring for Insurance Platforms: How AppDynamics Improved Our Response Times". *International Journal of Emerging Research in Engineering and Technology*, vol. 4, no. 1, Mar. 2023, pp. 54-65
[52] Patel, Piyushkumar. "The Role of AI in Forensic Accounting: Enhancing Fraud Detection through Machine Learning." *Distributed Learning and Broad Applications in Scientific Research* 5 (2019): 1420-35.
[53] Kamau, Eunice, et al. "Advances in Full-Stack Development Frameworks: A Comprehensive Review of Security and Compliance Models." (2023).
[54] Sreekandan Nair , S. (2023). Digital Warfare: Cybersecurity Implications of the Russia-Ukraine Conflict. International Journal of Emerging Trends in Computer Science and Information Technology, 4(4), 31-40. https://doi.org/10.63282/7a3rq622