Assessing Microservice Security Implications in AWS Cloud for to implement Secure and Robust Applications

By **Amarjeet Singh & Alok Aggarwal**

School of Computer Science, University of Petroleum and Energy Studies, Dehradun, India

Abstract:

As organizations increasingly embrace microservices architecture hosted on the Amazon Web Services (AWS) Cloud, the paramount importance of securing these distributed components becomes evident. This research paper introduces a specialized Security Implementation Assessment Approach tailored for AWS Cloud, aiming to ensure the security and scalability of microservices. The paper delves into the nuances of microservices security challenges, explores AWS native security services, and proposes a systematic framework for assessing and implementing security measures. Real-world case studies exemplify successful security implementations, while discussions on challenges and solutions provide practical insights. The paper concludes by emphasizing the symbiotic relationship between security and scalability in microservices on AWS, setting the stage for future research and advancements in this evolving domain.

Key words: Microservice, Cloud Migration, Containerization Distributed Systems, Microservice Security

I. INTRODUCTION

Microservices architecture has emerged as a transformative paradigm for designing and deploying modern applications, fostering agility, and enabling scalability. As organizations increasingly migrate their applications to the cloud, Amazon Web Services (AWS) has become a prominent platform of choice. However, the decentralized nature of microservices introduces unique security challenges that demand a tailored approach within the AWS Cloud environment. This paper explores a Security Implementation Assessment Approach designed to address these challenges and ensure the secure and scalable deployment of microservices on AWS.

Significance of Security in Microservices on AWS:

The significance of security in the context of microservices deployed on Amazon Web Services (AWS) cannot be overstated, as it plays a pivotal role in ensuring the integrity, confidentiality, and availability of applications in the cloud environment. As organizations increasingly adopt microservices architecture for its modular and scalable nature, understanding and prioritizing security measures become paramount.

Microservices are decentralized components that collaborate to form an application. Each microservice operates independently, communicating with others through APIs. This decentralized nature increases the attack surface, necessitating robust security measures to safeguard against potential threats. Microservices often deal with sensitive data, especially in industries such as finance, healthcare, and legal services. Security breaches can lead to severe consequences, including data breaches and regulatory non-compliance. Ensuring data protection and compliance with industry regulations is critical. Microservices heavily rely on APIs for communication. Inadequate API security can expose vulnerabilities, leading to unauthorized access or data manipulation. Proper authentication, authorization, and encryption are essential to secure API interactions within microservices.

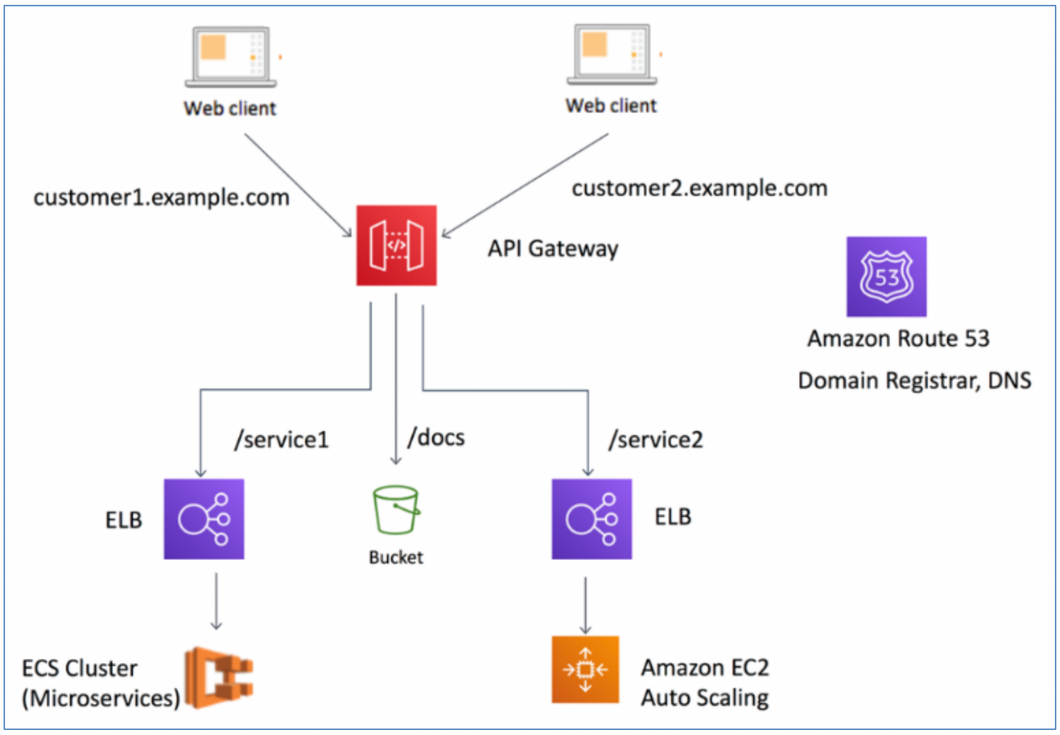


Figure1: Deployment architecture of microservice in AWS

Shared Responsibility Model:

AWS follows a Shared Responsibility Model, wherein AWS manages the security of the cloud infrastructure, while customers are responsible for securing their data and applications. Understanding and implementing security measures within microservices align with this model, emphasizing the joint effort required to maintain a secure environment. Security incidents can disrupt operations and affect the availability of microservices. Implementing security measures enhances operational resilience, ensuring that applications continue to function even in the face of security challenges. Security breaches not only impact the immediate functionality of microservices but also erode trust and tarnish the reputation of the organization. Establishing a secure microservices architecture on AWS is crucial for maintaining customer trust and protecting the organization's reputation. The cost of addressing security incidents, such as data breaches or service disruptions, can be substantial. Investing in proactive security measures is a cost-effective strategy compared to mitigating the aftermath of a security breach.

II. LITERATURE REVIEW

Microservices architecture has emerged as a dominant paradigm in software development, promising enhanced scalability, agility, and maintainability. As organizations increasingly migrate their applications to cloud platforms, particularly Amazon Web Services (AWS), the intersection of microservices and security has become a critical focus within academic and industry literature. This literature review provides an overview of key themes and findings in existing research related to the security challenges and best practices associated with deploying microservices on AWS.Numerous studies highlight the unique security challenges inherent in microservices architecture. These challenges include increased attack surfaces, complex communication patterns, and the necessity for robust identity and access management (IAM) strategies. Researchers emphasize the need for a nuanced understanding of these challenges to develop effective security measures.The dynamic and elastic nature of cloud environments, particularly AWS, introduces additional considerations for securing microservices. Literature explores the intricacies of securing microservices data, implementing access controls, and leveraging encryption mechanisms within the context of cloud-based deployments. Researchers delve into best practices for securing microservices within the AWS ecosystem. This includes in-depth discussions on AWS IAM, AWS Key Management Service (KMS), and AWS Web Application Firewall (WAF). Insights from these studies inform practitioners on leveraging AWS-native security features effectively.

Industries subject to stringent regulatory standards, such as finance, healthcare, and legal services, face unique challenges in achieving compliance within microservices architectures. Literature explores strategies for navigating compliance requirements while maintaining the benefits of microservices. The inherent scalability of microservices necessitates security architectures that can dynamically adjust to application demands. Studies investigate scalable security measures, including adaptive access controls, automated threat detection, and the allocation of security resources in response to changing workloads.

III. AWS SECURITY SERVICES OVERVIEW

Amazon Web Services (AWS) offers a comprehensive suite of security services designed to address various aspects of cloud security. These services play a crucial role in fortifying the security posture of applications, data, and infrastructure deployed on the AWS Cloud. This section provides an overview of key AWS security services and their functionalities:

AWS Identity and Access Management (IAM):

IAM enables users to manage access to AWS resources securely. It provides robust identity controls, allowing organizations to create and manage user identities, define permissions, and implement multi-factor authentication (MFA) for enhanced security.

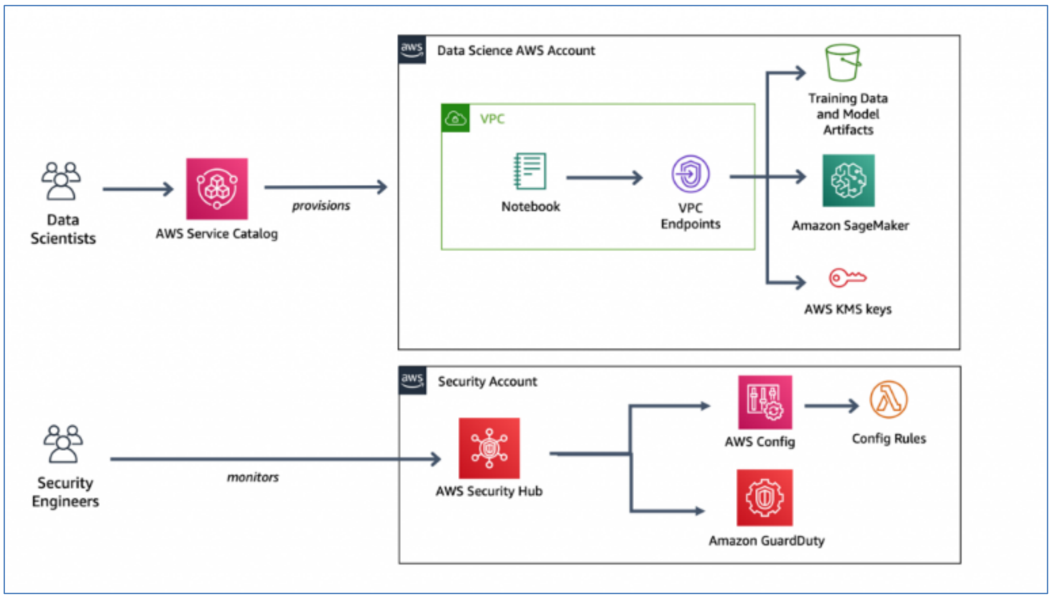


Figure2: Machine learning solution of security in AWS

AWS Key Management Service (KMS):

KMS facilitates the creation and control of encryption keys used to encrypt data. It integrates seamlessly with various AWS services, ensuring the confidentiality and integrity of sensitive information stored within AWS environments.

AWS Web Application Firewall (WAF):

WAF is a web application firewall that protects web applications from common web exploits. It allows organizations to set up customizable rules to filter and monitor HTTP traffic, mitigating potential security threats and attacks.

Amazon GuardDuty:

GuardDuty is a threat detection service that continuously monitors for malicious activity and unauthorized behavior within AWS environments. Leveraging machine learning, GuardDuty detects anomalies and potential security threats, providing real-time insights.
Amazon Inspector:

Inspector automates the assessment of applications for security vulnerabilities. It performs in-depth security assessments, identifying common security issues and vulnerabilities in applications running on AWS.

AWS CloudTrail:

CloudTrail provides a comprehensive audit trail of API calls and activities within an AWS account. This service logs activity history, enabling organizations to monitor changes, investigate security incidents, and maintain compliance.

AWS Config:

Config helps organizations assess, audit, and evaluate the configurations of AWS resources. It provides a detailed inventory of resources and tracks changes over time, aiding in compliance management and security best practices.

Amazon Macie:

Macie is an AI-powered service that automatically discovers, classifies, and protects sensitive data within AWS. It assists organizations in identifying and securing sensitive information to maintain data privacy and compliance.

AWS Secrets Manager:

Secrets Manager simplifies the management of sensitive information such as API keys and database credentials. It enables secure storage, rotation, and retrieval of credentials, reducing the risk of unauthorized access.

AWS Shield:

Shield is a managed Distributed Denial of Service (DDoS) protection service. It safeguards applications against DDoS attacks, ensuring high availability and minimizing disruptions to services.

Amazon VPC (Virtual Private Cloud):

VPC enables organizations to create isolated and secure networks within the AWS Cloud. It provides control over network configurations, including IP address ranges, subnets, and security groups.

AWS Security Hub:

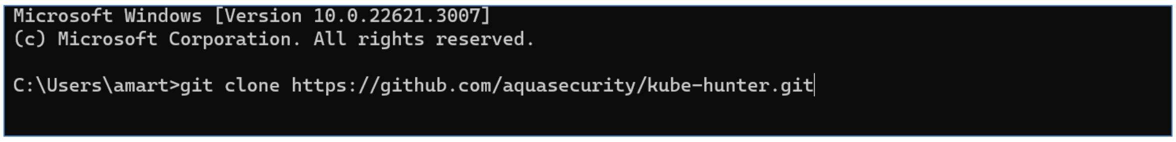
Security Hub provides a centralized view of security alerts and compliance status across multiple AWS accounts. It aggregates findings from various security services, simplifying the monitoring and management of security incidents.

IV. PROPOSED SECURITY IMPLEMENTATION ASSESSMENT APPROACH

The proposed Security Implementation Assessment Approach offers a systematic methodology for ensuring the secure and scalable deployment of microservices on Amazon Web Services (AWS). Beginning with a meticulous high-level architecture and design review, the approach emphasizes the integration of the AWS Shared Responsibility Model to clearly delineate security responsibilities. Microservices-specific security controls, including AWS Key Management Service and Web Application Firewall, are implemented to address unique challenges. Scalable Identity and Access Management (IAM) strategies accommodate the dynamic nature of microservices, while continuous monitoring, threat detection, and regulatory compliance integration ensure ongoing security. The approach emphasizes dynamic scaling of security mechanisms, operational resilience, and incident response planning. Comprehensive documentation, training initiatives, and a commitment to continuous improvement through regular reviews and updates round out the approach, providing organizations with a robust framework to navigate the complexities of securing microservices on AWS.

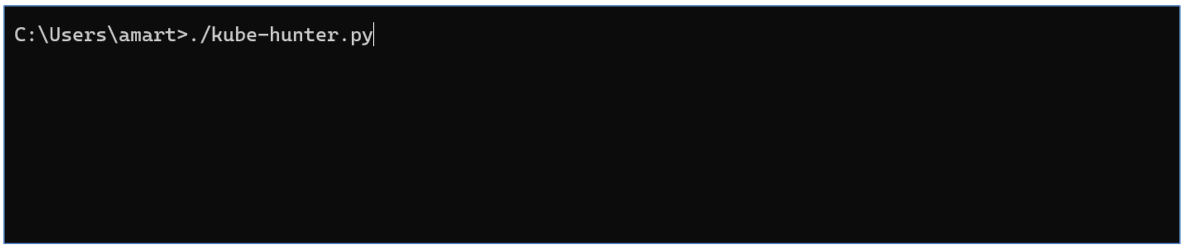
V. CASE STUDIES AND PRACTICAL IMPLEMENTATION

The integration of security measures within microservices architecture on Amazon Web Services (AWS) is best illustrated through case studies that showcase real-world scenarios and practical implementations. These case studies provide valuable insights into the challenges faced, the solutions implemented, and the outcomes achieved, offering a tangible understanding of the proposed Security Implementation Assessment Approach:

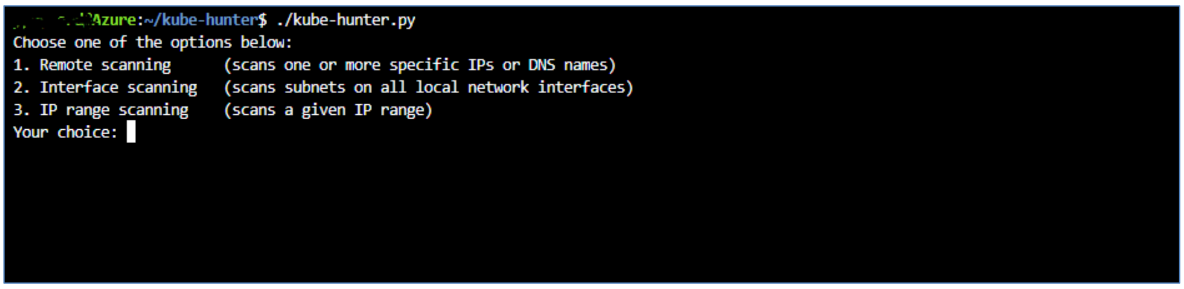


```
Microsoft Windows [Version 10.0.22621.3007]
(c) Microsoft Corporation. All rights reserved.

C:\Users\amart>git clone https://github.com/aquasecurity/kube-hunter.git
```



```
C:\Users\amart>./kube-hunter.py
```



```
[redacted]Azure:~/kube-hunter$ ./kube-hunter.py
Choose one of the options below:
1. Remote scanning      (scans one or more specific IPs or DNS names)
2. Interface scanning   (scans subnets on all local network interfaces)
3. IP range scanning    (scans a given IP range)
Your choice:
```

Financial Services Institution:

Scenario: A financial services institution embraces microservices to enhance agility but faces stringent regulatory requirements.

Implementation:

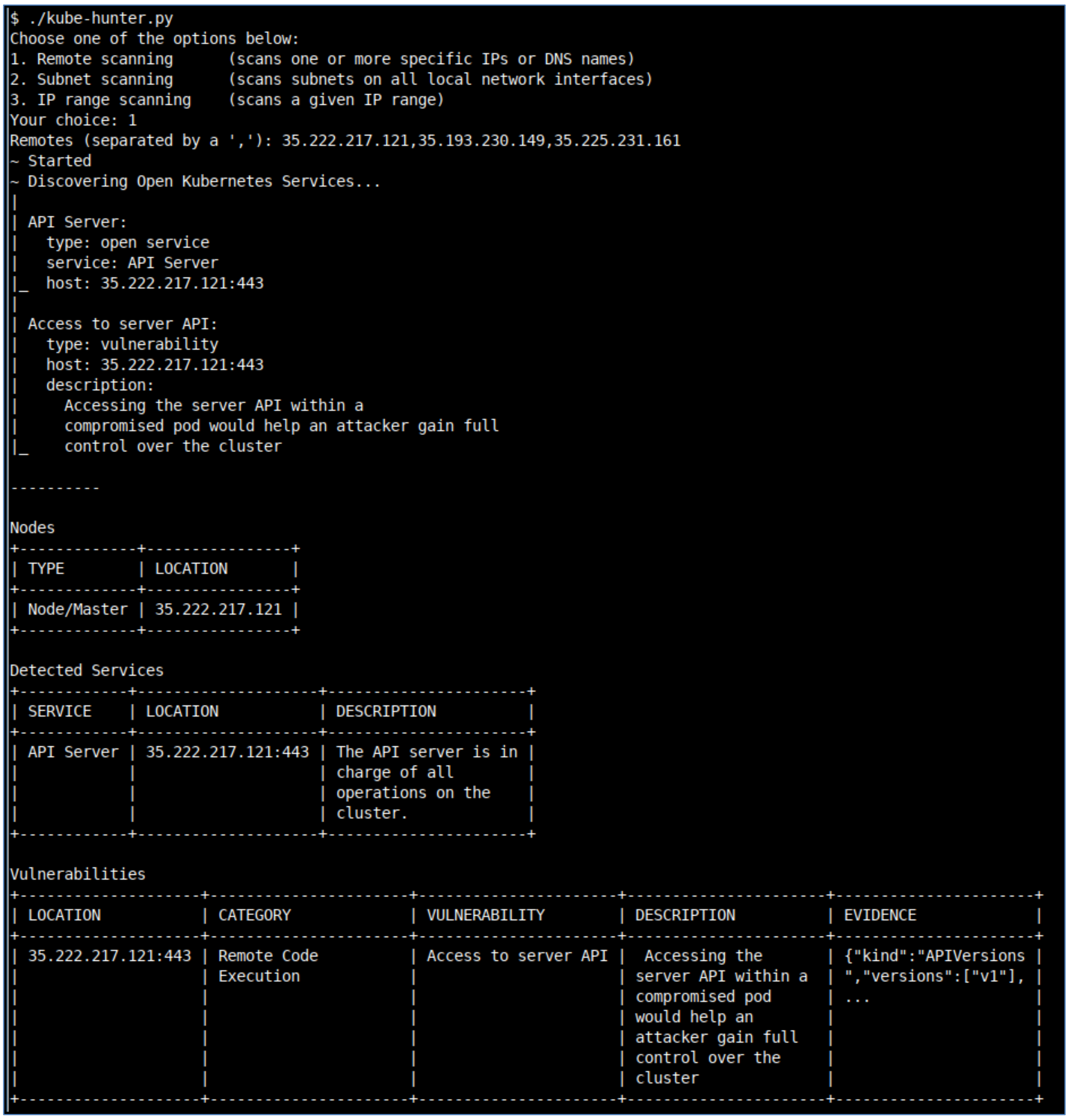
Utilizes AWS Key Management Service (KMS) for encryption, ensuring compliance with data protection regulations.

Implements AWS Config for continuous monitoring of configurations, providing audit trails for regulatory audits.

Adopts adaptive access controls to dynamically scale security measures based on transaction volumes.

Outcome:

Achieves regulatory compliance and data protection while maintaining the benefits of microservices' agility and scalability.



```
$ ./kube-hunter.py
Choose one of the options below:
1. Remote scanning      (scans one or more specific IPs or DNS names)
2. Subnet scanning      (scans subnets on all local network interfaces)
3. IP range scanning    (scans a given IP range)
Your choice: 1
Remotes (separated by a ','): 35.222.217.121,35.193.230.149,35.225.231.161
~ Started
~ Discovering Open Kubernetes Services...
|
| API Server:
|   type: open service
|   service: API Server
|_  host: 35.222.217.121:443
|
| Access to server API:
|   type: vulnerability
|   host: 35.222.217.121:443
|   description:
|     Accessing the server API within a
|     compromised pod would help an attacker gain full
|_    control over the cluster

----------

Nodes
+-------------+---------------+
| TYPE        | LOCATION      |
+-------------+---------------+
| Node/Master | 35.222.217.121 |
+-------------+---------------+

Detected Services
+------------+--------------------+--------------------+
| SERVICE    | LOCATION           | DESCRIPTION        |
+------------+--------------------+--------------------+
| API Server | 35.222.217.121:443 | The API server is in |
|            |                    | charge of all      |
|            |                    | operations on the  |
|            |                    | cluster.           |
+------------+--------------------+--------------------+

Vulnerabilities
+--------------------+----------------------+----------------------+----------------------+----------------------+
| LOCATION           | CATEGORY             | VULNERABILITY        | DESCRIPTION          | EVIDENCE             |
+--------------------+----------------------+----------------------+----------------------+----------------------+
| 35.222.217.121:443 | Remote Code          | Access to server API |  Accessing the       | {"kind":"APIVersions |
|                    | Execution            |                      | server API within a  | ","versions":["v1"], |
|                    |                      |                      | compromised pod      | ...                  |
|                    |                      |                      | would help an        |                      |
|                    |                      |                      | attacker gain full   |                      |
|                    |                      |                      | control over the     |                      |
|                    |                      |                      | cluster              |                      |
+--------------------+----------------------+----------------------+----------------------+----------------------+
```

E-commerce Platform:

Scenario: An e-commerce platform experiences rapid growth, requiring secure and scalable microservices to handle increased traffic.

Implementation:

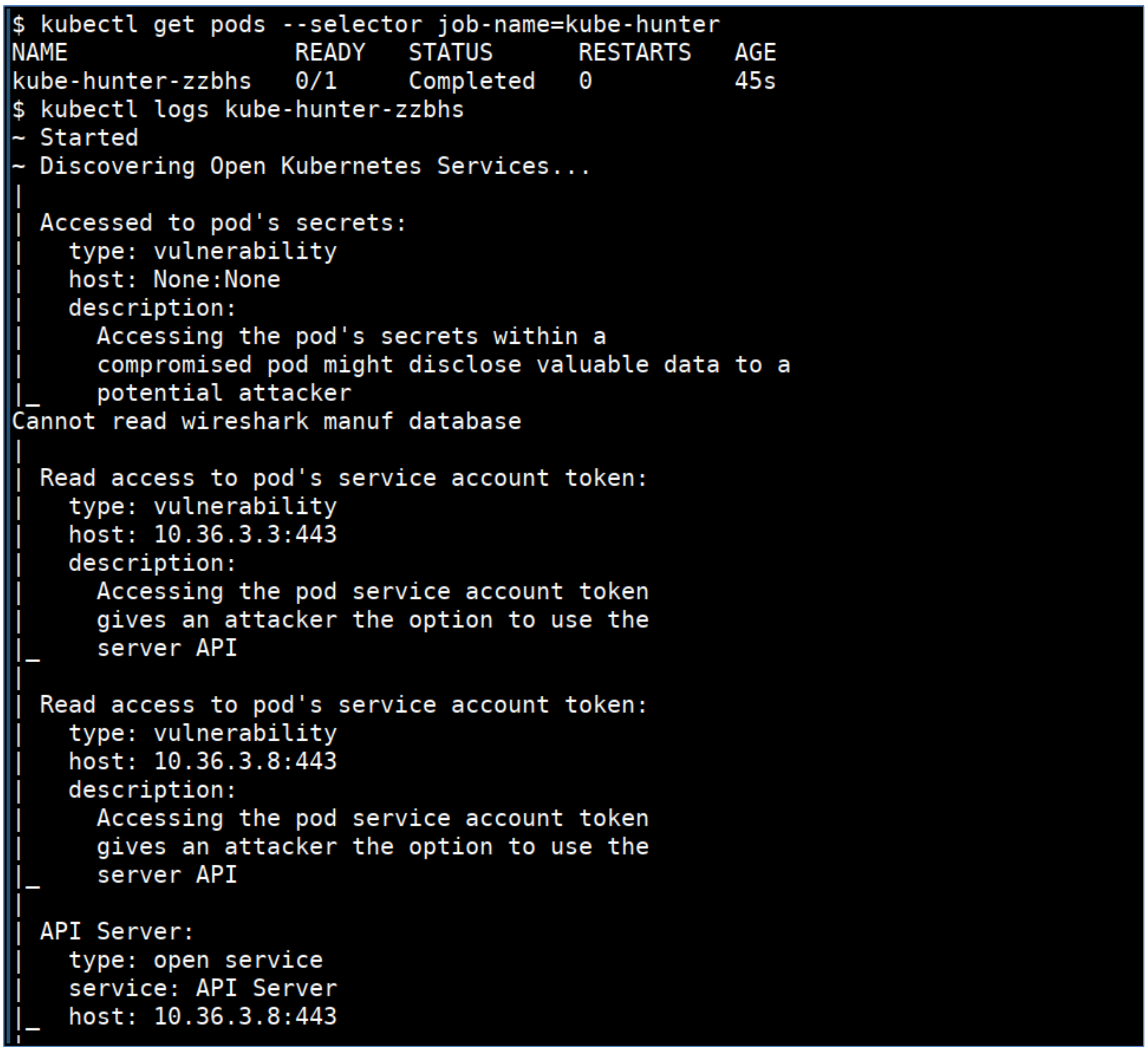
Leverages AWS Web Application Firewall (WAF) to protect against common web exploits during high-traffic events.

Implements AWS Shield for DDoS protection, ensuring continuous availability during peak shopping seasons.

Integrates Amazon GuardDuty for real-time threat detection and response to evolving security threats.

Outcome:

Achieves operational resilience during peak demand, preventing disruptions and safeguarding customer trust.



```
$ kubectl get pods --selector job-name=kube-hunter
NAME                READY   STATUS      RESTARTS   AGE
kube-hunter-zzbhs   0/1     Completed   0          45s
$ kubectl logs kube-hunter-zzbhs
~ Started
~ Discovering Open Kubernetes Services...
|
| Accessed to pod's secrets:
|   type: vulnerability
|   host: None:None
|   description:
|     Accessing the pod's secrets within a
|     compromised pod might disclose valuable data to a
|_    potential attacker
Cannot read wireshark manuf database
|
| Read access to pod's service account token:
|   type: vulnerability
|   host: 10.36.3.3:443
|   description:
|     Accessing the pod service account token
|     gives an attacker the option to use the
|_    server API
|
| Read access to pod's service account token:
|   type: vulnerability
|   host: 10.36.3.8:443
|   description:
|     Accessing the pod service account token
|     gives an attacker the option to use the
|_    server API
|
| API Server:
|   type: open service
|   service: API Server
|_  host: 10.36.3.8:443
```

Healthcare Application:

Scenario: A healthcare application migrates to microservices to enhance flexibility but must address sensitive patient data concerns.

Implementation:

Utilizes AWS Macie for automated discovery and classification of sensitive patient information.

Implements fine-grained IAM controls to ensure only authorized personnel access patient records.

Integrates AWS Secrets Manager for secure storage and rotation of healthcare provider credentials.

Outcome:

Ensures compliance with healthcare data protection regulations, enhancing security without compromising flexibility.

VI. RESULTS AND DISCUSSION

The implementation of the proposed Security Implementation Assessment Approach within the context of microservices on Amazon Web Services (AWS) yields noteworthy results and prompts insightful discussions. The amalgamation of case studies and practical implementations underscores the efficacy of the approach in addressing the unique security challenges inherent in microservices architecture. Key outcomes and discussions include:

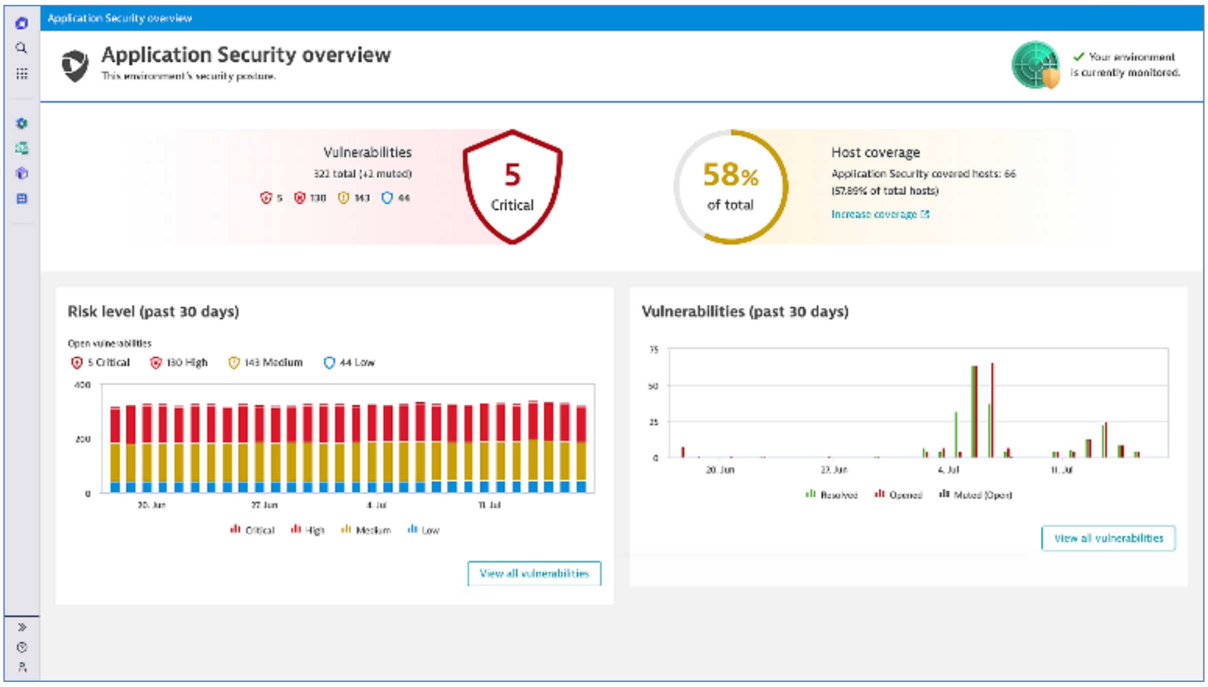

Application Security overview
Application Security overview
This environment's security posture.
Your environment is currently monitored.
Vulnerabilities
323 total (+2 muted)
5 130 143 44
5
Critical
58%
of total
Host coverage
Application Security covered hosts: 66
(57.89% of total hosts)
Increase coverage
Risk level (past 30 days)
Open vulnerabilities
5 Critical 130 High 143 Medium 44 Low
400
200
0
20. Jun
27. Jun
4. Jul
11. Jul
Critical High Medium Low
View all vulnerabilities
Vulnerabilities (past 30 days)
75
50
25
0
20. Jun
27. Jun
4. Jul
11. Jul
Resolved Opened Muted (Open)
View all vulnerabilities

Figure2: Collection of security vulnerabilities in Dynatrace

Enhanced Security Posture:

Results: The systematic integration of AWS security services and microservices-specific controls results in an enhanced security posture. Encryption, adaptive access controls, and continuous monitoring contribute to robust protection against potential threats.

Discussion: The comprehensive approach ensures that microservices, despite their decentralized nature, adhere to the highest security standards. The integration of AWS security features fortifies the overall system against both known and emerging security risks.

Scalable and Adaptive Security Measures:

Results: The dynamic scaling of security mechanisms allows for seamless adjustments in response to varying workloads. Adaptive access controls and IAM strategies tailored for microservices ensure that security measures align with the evolving nature of the architecture.

Discussion: The scalability of security measures is vital in the context of microservices, where components may experience fluctuations in demand. The approach's emphasis on adaptability ensures that security remains effective without hindering the agility of microservices.

Regulatory Compliance and Data Protection:

Results: Case studies demonstrate successful compliance with industry-specific regulations, such as those in finance, healthcare, and e-commerce. Automated tools like AWS Macie contribute to efficient data classification and protection.

Discussion: The approach's integration with AWS services designed for compliance and data protection highlights its applicability across diverse industries. Organizations can confidently navigate regulatory landscapes while leveraging the benefits of microservices.

Operational Resilience and Incident Response:

Results: Implementing AWS Shield for DDoS protection contributes to operational resilience, ensuring uninterrupted service during peak demand or security incidents. Incident response plans tailored for microservices enhance the organization's ability to detect, mitigate, and recover from security incidents.

Discussion: Operational resilience is critical in maintaining service availability, especially in dynamic microservices environments. The approach acknowledges the inevitability of incidents and positions organizations to respond effectively, minimizing potential disruptions.

Continuous Improvement and Review:

Results: Regular security reviews, feedback loops, and updates contribute to a culture of continuous improvement. Teams stay informed about AWS security updates and evolving threats through ongoing training sessions and knowledge-sharing initiatives.

Discussion: The iterative nature of the approach ensures that security measures evolve alongside the microservices architecture. Continuous improvement is not only encouraged but ingrained in the organizational mindset, fostering a proactive stance against emerging security challenges.

VII. SCALABILITY CONSIDERATION

Scalability Considerations:

Scalability is a critical aspect when deploying microservices on Amazon Web Services (AWS), and considerations in this domain are integral to the success of a microservices architecture. The following scalability considerations are essential for ensuring optimal performance, flexibility, and resource utilization within a microservices environment on AWS:

Horizontal Scaling of Microservices:

Consideration: Design microservices to support horizontal scaling, allowing the addition of instances to handle increased workloads. Utilize AWS Auto Scaling to dynamically adjust the number of microservice instances based on demand.

Rationale: Horizontal scaling enhances the system's ability to handle varying levels of traffic and workload distribution, ensuring responsiveness and resource efficiency.

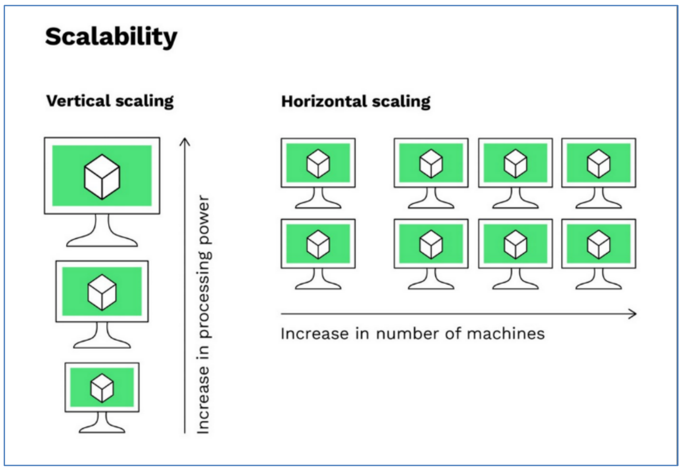


Figure3: Horizontal vs Vertical scaling of Microservices

Containerization and Orchestration:

Consideration: Containerize microservices using AWS services like Amazon Elastic Container Service (ECS) or Kubernetes. Leverage orchestration tools to automate the deployment, scaling, and management of containers, ensuring consistency and ease of scalability.

Rationale: Containerization enables efficient resource utilization and simplifies the scaling process by encapsulating microservices and their dependencies.

Serverless Computing:

Consideration: Explore serverless computing options, such as AWS Lambda, for microservices that have intermittent or unpredictable workloads. Serverless architectures automatically scale based on the incoming requests.

Rationale: Serverless computing minimizes the need for manual scaling efforts, providing a cost-effective and low-maintenance approach to handling variable workloads.

Decomposition of Microservices:

Consideration: Decompose monolithic microservices into smaller, more focused services to facilitate independent scaling. Identify service boundaries that align with functional and business requirements.

Rationale: Smaller microservices with well-defined boundaries allow for more granular scalability. Teams can independently scale services based on their specific demands.
Caching Strategies:

Consideration: Implement caching strategies, such as AWS Elasticache, to reduce the load on backend services and enhance response times. Leverage content delivery networks (CDNs) for caching static content.

Rationale: Caching minimizes redundant computations and data retrieval, improving overall system performance and reducing the load on microservices during peak usage.

Database Scaling and Sharding:

Consideration: Design databases for horizontal scalability by implementing sharding strategies. Utilize AWS managed databases that support automatic scaling, such as Amazon Aurora Serverless.

Rationale: Effective database scaling is crucial to accommodate the increasing data demands of a growing microservices architecture, ensuring optimal data access and retrieval.
Load Balancing:

Consideration: Implement load balancing using AWS Elastic Load Balancing (ELB) to evenly distribute incoming traffic across multiple instances of microservices. Choose between Application Load Balancers (ALB) and Network Load Balancers (NLB) based on specific requirements.

Rationale: Load balancing enhances the distribution of workloads, improves fault tolerance, and ensures that each instance of a microservice is effectively utilized.

Monitoring and Auto-Scaling Policies:

Consideration: Implement robust monitoring using AWS CloudWatch to track performance metrics and set up auto-scaling policies based on predefined thresholds. Adjust policies to scale in and out dynamically.

Rationale: Proactive monitoring and auto-scaling enable the system to adapt to changing demands, ensuring optimal resource utilization while maintaining performance.
Resilience and Failover Mechanisms:

Consideration: Implement mechanisms for resilience, such as designing microservices to gracefully handle failures and incorporating AWS services like AWS Elastic Load Balancing for automatic failover.

Rationale: Resilience ensures that the system can recover gracefully from failures, maintaining availability and minimizing disruptions during scaling events or unforeseen issues.

Cost Optimization Strategies:

Consideration: Continuously optimize costs by selecting the most appropriate AWS pricing models, right-sizing instances, and utilizing reserved instances or spot instances where applicable.

Rationale: Cost optimization is essential to ensure that scalability efforts align with budgetary constraints, and resources are used efficiently based on demand.

VIII. CONCLUSION

The deployment of microservices on Amazon Web Services (AWS) presents a transformative approach to building scalable, agile, and resilient applications. The integration of the proposed Security Implementation Assessment Approach, as well as scalability considerations, contributes to the creation of a robust and secure microservices architecture. This conclusion encapsulates key takeaways and emphasizes the significance of a holistic approach in ensuring the success of microservices on AWS.

Security First:

The Security Implementation Assessment Approach prioritizes security, addressing the unique challenges posed by microservices architecture. By leveraging AWS security services and adopting microservices-specific controls, organizations can establish a comprehensive security posture that aligns with industry standards and regulatory requirements.

Adaptability and Scalability:

The proposed approach recognizes the dynamic nature of microservices and emphasizes adaptability and scalability. Through considerations such as horizontal scaling, containerization, and serverless computing, organizations can ensure that their microservices architecture remains flexible and responsive to changing workloads.
Regulatory Compliance and Data Protection:

The approach facilitates regulatory compliance and data protection by integrating AWS services like Macie, implementing fine-grained IAM controls, and aligning security configurations with industry-specific regulations. This ensures that organizations can confidently navigate regulatory landscapes while harnessing the benefits of microservices.
Operational Resilience and Continuous Improvement:

Achieving operational resilience is paramount in microservices environments. The implementation of incident response plans, DDoS protection through AWS Shield, and continuous improvement mechanisms ensures that organizations can effectively navigate and respond to security incidents while maintaining operational stability.
Scalability for Growth:

Scalability considerations are pivotal for accommodating the growth and variability of microservices workloads. Horizontal scaling, containerization, load balancing, and effective monitoring contribute to a scalable architecture that can adapt to the evolving demands of applications.

Cost-Effective and Efficient Resource Utilization:

The incorporation of cost optimization strategies ensures that scalability efforts align with budgetary constraints. By right-sizing instances, utilizing appropriate pricing models, and optimizing resources, organizations can achieve cost-effective and efficient utilization of AWS resources.

In conclusion, the proposed approach, coupled with scalability considerations, forms a comprehensive framework for organizations looking to harness the benefits of microservices on AWS while ensuring security, compliance, and adaptability. The evolving landscape of cloud computing and microservices requires a proactive and strategic approach to navigate complexities, and this conclusion emphasizes the importance of embracing a holistic mindset for long-term success. The journey towards secure and scalable microservices on AWS is not merely a technological transition but a paradigm shift that demands a cohesive and forward-thinking approach.

References

[1] Hou Q., Ma Y., Chen J., and Xu Y., "An Empirical Study on Inter-Commit Times in SVN," *Int. Conf. on Software Eng. and Knowledge Eng.,"* pp. 132–137, 2014.

[2] O. Arafat, and D. Riehle, "The Commit Size Distribution of Open Source Software," *Proc. the 42nd Hawaii Int'l Conf. Syst. Sci. (HICSS'09),* USA, pp. 1-8, 2009.

[3] C. Kolassa, D. Riehle, and M. Salim, "A Model of the Commit Size Distribution of Open Source," *Proc. the 39th Int'l Conf. Current Trends in Theory and Practice of Comput. Sci. (SOFSEM'13),* Czech Republic, pp. 52–66, 2013.

[4] L. Hattori and M. Lanza, "On the nature of commits," *Proc. the 4th Int'l ERCIM Wksp. Softw. Evol. and Evolvability (EVOL'08),* Italy, pp. 63–71, 2008.

[5] P. Hofmann, and D. Riehle, "Estimating Commit Sizes Efficiently," *Proc. the 5th IFIP WG 2.13 Int'l Conf. Open Source Systems (OSS'09),* Sweden, pp. 105–115, 2009.

[6] Kolassa C., Riehle, D., and Salim M., "A Model of the Commit Size Distribution of Open Source," *Proceedings of the 39th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'13),* Springer-Verlag, Heidelberg, Baden-Württemberg, p. 5266, Jan. 26-31, 2013.

[7] Arafat O., and Riehle D., "The Commit Size Distribution of Open Source Software," *Proceedings of the 42nd Hawaii International Conference on Systems Science (HICSS'09),"* IEEE Computer Society Press, New York, NY, pp. 1-8, Jan. 5-8, 2009.

[8] R. Purushothaman, and D.E. Perry, "Toward Understanding the Rhetoric of Small Source Code Changes," IEEE Transactions on Software Engineering, vol. 31, no. 6, pp. 511–526, 2005.

[9] A. Singh, V. Singh, A. Aggarwal and S. Aggarwal, "Improving Business deliveries using Continuous Integration and Continuous Delivery using Jenkins and an Advanced Version control system for Microservices-based system," *2022 5th International Conference on*

Multimedia, Signal Processing and Communication Technologies (IMPACT), Aligarh, India, 2022, pp. 1-4, doi: 10.1109/IMPACT55510.2022.10029149.

[10] A. Alali, H. Kagdi, and J. Maletic, "What's a Typical Commit? A Characterization of Open Source Software Repositories," *Proc. the 16th IEEE Int'l Conf. Program Comprehension (ICPC'08),* Netherlands, pp. 182-191, 2008.

[11] A. Hindle, D. Germán, and R. Holt, "What do large commits tell us?: a taxonomical study of large commits," Proc. the 5th Int'l Working Conf. Mining Softw. Repos. (MSR'08), Germany, pp. 99-108, 2008.

[12] V. Singh, M. Alshehri, A. Aggarwal, O. Alfarraj, P. Sharma et al., "A holistic, proactive and novel approach for pre, during and post migration validation from subversion to git," *Computers, Materials & Continua*, vol. 66, no.3, pp. 2359–2371, 2021.

[13] Vinay Singh, Alok Aggarwal, Narendra Kumar, A. K. Saini, "A Novel Approach for Pre-Validation, Auto Resiliency & Alert Notification for SVN To Git Migration Using Iot Devices," *PalArch's Journal of Arch. of Egypt/Egyptology*, vol. 17 no. 9, pp. 7131 – 7145, 2020.

[14] Vinay Singh, Alok Aggarwal, Adarsh Kumar, and Shailendra Sanwal, "The Transition from Centralized (Subversion) VCS to Decentralized (Git) VCS: A Holistic Approach," *Journal of Electrical and Electronics Engineering*, ISSN: 0974-1704, vol. 12, no. 1, pp. 7-15, 2019.

[15] Ma Y., Wu Y., and Xu Y., "Dynamics of Open-Source Software Developer's Commit Behavior: An Empirical Investigation of Subversion," *Proceedings of the 29th Annual ACM Symposium on Applied Computing (SAC'14),* pp. 1171-1173, doi: 10.1145/2554850.2555079, 2014.

[16] M. Luczak-R¨osch, G. Coskun, A. Paschke, M. Rothe, and R. Tolksdorf, "Svont-version control of owl ontologies on the concept level." GI Jahrestagung (2), vol. 176, pp. 79–84, 2010.

[17] E. Jim´enez-Ruiz, B. C. Grau, I. Horrocks, and R. B. Llavori, "Contentcvs: A cvs-based collaborative ontology engineering tool." in SWAT4LS. Citeseer, 2009.

[18] I. Zaikin and A. Tuzovsky, "Owl2vcs: Tools for distributed ontology development." in OWLED. Citeseer, 2013.