ESP-IJACT
ESP International Journal of Advancements in Computational Technology
ISSN: 2583-8628 / Volume 1 Issue 3 December 2023 / Page No: 180-189
Paper Id: IJACT-V1I3P119 / Doi: 10.56472/25838628/IJACT-V1I3P119

Original Article

Network isolation for cloud-native applications in multi-tenant Architectures

Khaja Kamaluddin
Masters in Sciences, Fairleigh Dickinson University, Teaneck, NJ, USA, Work: Aonsoft International Inc, 1600 Golf Rd, Suite 1270, Rolling Meadows, Illinois, 60008 USA.

Received Date: 07 November 2023 Revised Date: 10 December 2023 Accepted Date: 30 December 2023

Abstract - This research examines the evolution of network isolation techniques with-in kubernetes-based containerized environment with an emphasis on early architectural patterns and their associated security constraints. It investigates the use of iptables and ebtables for packet filtering processes, as well the use of VXLAN based overlay networks for cross node communication. While these mechanisms are instrumental in supporting basic network segmentation, they were often complex to manage at scale, and had limited integration with identity aware security models. A comparison of key CNI plugins Calico, Flannel and Weave Net shows their weaknesses, strength and suitability in different deployment scenario. While there have been efforts with the introduction of Kubernetes Network Policies and early use of service mesh technologies, the first iteration of Kubernetes often suffered due to weak workload identity verification, inconsistent policy enforcement and limited traffic observation of east-west traffic. These weaknesses led to lateral movement within clusters as well as high risk in multi-tenant mode. The paper synthesizes these lessons to outline the practices that will yield stronger segmentation and Defense in depth in Container Ecosystems. It concludes with a future outlook of which the work to move towards identity driven, policy rich models of workload isolation pave the way for a more resilient cloud native security posture.

Keywords – Network Isolation, VXLAN, Mechanisms.

I. INTRODUCTION

Currently, multi-tenant cloud-native environments are a fundamental pillar in the cloud computing evolution through the delivery of scalable and efficient services [1]. They provide the capability of multiple tenants such as users, applications or organizations to do share the same underlying infrastructure and still remain logical apart. Moreover, container orchestration platforms like Kubernetes have made even easier the process of deployment and management of applications in such shared environments [2].

On the other hand, the inherent characteristic of multi-tenant architecture is also an issue of network isolation which is not shared [3]. Therefore, the foremost prerequisite is to make sure that each of the tenant's data and operation does not come in touch, nor does it come in reach, from the hands of the other tenant. Boundary enforcement to prevent lateral movement of threats in the shared infrastructure is critical and is achieved by network isolation between tenants to isolate from each other [4].

Legacy security models, which were primarily designed for monolithic applications (Figure 1), often fall short in addressing the complexities of modern microservices-based, multi-tenant cloud-native environments [5]. As illustrated in Figure 1, monolithic architectures bundle all components into a single unit, relying on simple perimeter defenses. In contrast, microservices decompose functionality into distributed, interconnected services introducing dynamic east-west traffic flows that demand granular network isolation. Traditional perimeter-based approaches are inadequate for these scenarios, where the boundary between "inside" and "outside" is inherently blurred [6].

The security solutions are also expected to support fast provisioning and deprovisioning of resources in the cloud native environment. With the security demands for cloud native increased, implementing robust network isolation strategies, like micro segmentation and zero trust networking, becomes a necessity to keep security while maintaining the agility and scalability cloud native provides. These strategies make sure that even in a shared infrastructure, a tenant not only doesn’t share anything with every other tenant, but in fact runs his own workings in a secure and isolated environment where he doesn’t expose anything to any other tenant.

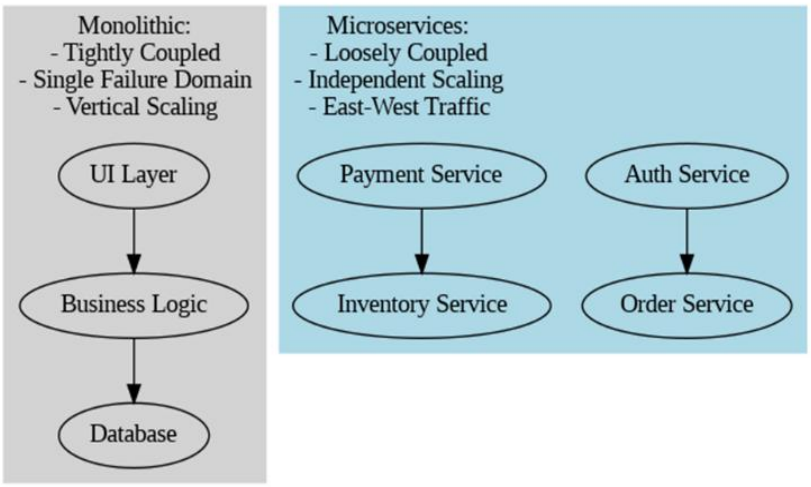

Figure 1: Comparison Of Monolithic Vs. Microservices Architectures.

Network isolation in multi-tenant cloud native architectures is explored in this paper with an eye towards current problems, old solutions and new approaches to enhance security sans jeopardizing the advantages of cloud native deployment.

II. BACKGROUND AND EVOLUTION OF CLOUD-NATIVE ARCHITECTURES

Cloud native architecture is a shift in paradigm in terms of software development and it is an architecture designed to enjoy all cloud computing capabilities [7]. The approach promotes building scalable and resilient, manageable and thriving applications in public clouds, private clouds and hybrid clouds. Cloud native architectures use technologies such as containers, microservices and declarative APIs to help organizations quickly deliver high impact changes rapidly and effectively [8].

Visualizing the progression of cloud-native architectures may help explain them through four phases as architecture evolves from monolithic applications to the virtual machine, containers, and finally serverless computing (as shown in Figure 2). The capabilities introduced at each stage of each approach addressed limitations of the previous approaches.

A. From Monolithic to Microservices: The Rise of Docker and Kubernetes

Application architectures evolved from the monolithic systems wherein all the components are woven inside a single codebase to microservices which split the application into separate independent modular services [9]. Both scalability, maintainability and deployment agility get enhanced due to this. In 2013, Docker first arrived on the scene, fundamentally changing this paradigm by providing lightweight containerization that could be relied on to deliver the same environments both during development, testing, and production [10]. Shifting ahead, Kubernetes, an orchestration platform born from open sourcing of Google's Kubernetes project in 2014, has evolved into a powerful platform for automating deployment, scaling and management of containerized applications. Docker and Kubernetes have come together as foundational to modern cloud native development and enabled the adoption of the microservices architecture everywhere [11].

B. Key Technologies: Kubernetes v1.20-, Docker Swarm, Mesos, and Others

Many of the technologies involved in shaping the container orchestration and management were many. Features like better support for stateful applications, improved security policies and scalability was introduced to/Kubernetes versions up to v1.20. It was simple and would fit quite well with Docker CLI and Dockers native tool for clustering, Docker Swarm for smaller deployments [12]. Apache Mesos had a more generalized resource scheduler and could handle workloads which are not containers like big data processing. Also included are CoreOS Fleet and HashiCorp Nomad, both of which provide innovative ways of orchestrating and managing your workloads. They were the basis for the cloud native ecosystems of today that are robust, scalable, and flexible.

C. Legacy Multi-Tenancy Models: Shared VM, OS-Level, and Container-Level Isolation

Through various isolation models, multi tenancy has grown from the practice of serving different clients (tenants) with shared resources [13]. At the outset, virtual machines (VM) were shared initially, which was then made by hypervisors to create hardware isolation for tenants. However, this was a very secure approach but very resource intensive. OS level isolation

appeared subsequently, by using the mechanisms such as chroot jails and namespaces to separate tenant environments onto a single operating system while being more efficient but with security trade-offs [10]. Containerization brought with it container level isolation on where the tenants' application runs in separate containers which share the host OS kernel with isolated user space. The resulting model is reasonable in a sense, but isolates less than resource efficient enough and this is the preferred method in modern multi-tenant cloud native architecture.

III. TYPES OF MULTI-TENANCY MODELS

Cloud native architectures are built on the concept of multi-tenancy where multiple tenants (readers, teams or organizations) share the same infrastructure, but with some level of logical or physical isolation [14]. There were prevalent several multi-tenancy models, each provides separate approaches for resource sharing and isolation.

A. Infrastructure-level multi-tenancy

This model has tenants functioning independently in shared physical hardware between different virtual machines (VMs). Strong isolation between tenants is provided by a dedicated operating system environment on each VM. However, the tradeoff of this approach is that it provides robust security boundaries, yet tends to offer poor resource utilization, as well as increases the operational overhead of managing a separate VM for each application.

B. Container-level multi-tenancy

The advantage of container level multi tenancy is that tenants are allowed to run their applications inside a container on a shared Kubernetes cluster [15]. Containers are lightweight and efficient but share the same underlying kernel, which can be a security problem if not properly isolated. Isolation can be accomplished only if strict access controls, resource quotas, and network policies are implemented.

C. Namespace-based isolation

The Kubernetes namespace provides a logical partitioning mechanism in a cluster to group and isolate the resources per tenant [16]. Depending on your needs, you can use Namespaces together with Role-Based Access Control (RBAC) and Network Policies to enforce access restrictions as well as communication boundaries [17]. Therefore, namespaces alone don't ensure complete isolation, and some cluster wide resources are still shared, but to add more isolation administrators usually deploy more controls on top of that such as enabling Resource Quota and Limit Range that manage the used resources per namespace. While these are measures that have been taken, they still don't open the door to full isolation cases, particularly around the Kubernetes API server nor common resources at the node level. Therefore, namespaces are of great importance to multi tenancy, but they should not stand alone and need to be paired with a comprehensive set of policies and configs to create a strong boundary for each tenant.

D. Challenges with resource and security boundary enforcement

Implementing effective multi-tenancy faced several challenges:

- *Shared Control Plane:* Kubernetes does not have a strict tenant isolation which is due to the control plane components (API Server, Scheduler, etc) being shared amongst the tenants and making it harder to enforce the strict isolation between them.

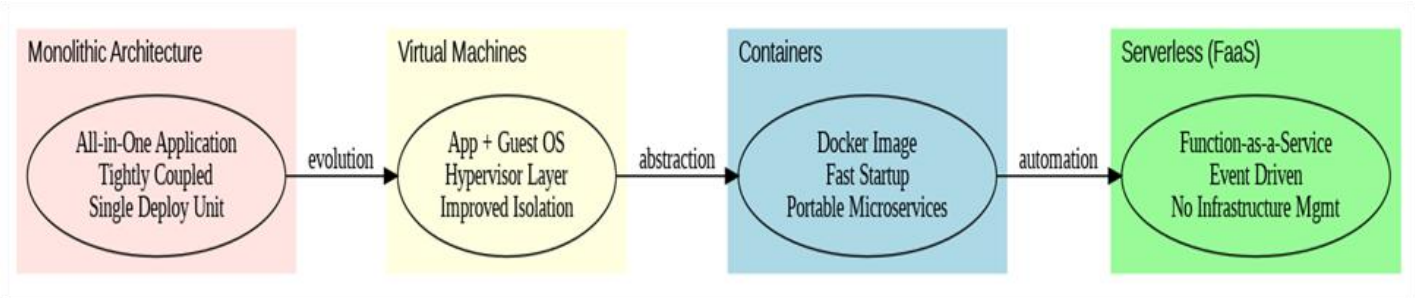

Figure 2: Evolution Of Cloud-Native Architectures.

- *Resource Contention:* Tenants could eat up disproportionate resources of their own, which would result in something called as 'noisy neighbor' problem.
- *Security Risks:* Unauthorized access or data leakage between tenants that originate from RBAC, network policy, and container runtime misconfigurations.

- *Complex Management:* It brought along operational complexity associated with complex management of isolation policies, resource allocations and access controls for multiple tenants.

Table 1: Comparison Of Multi-Tenancy Types

Model	Isolation Level	Resource Efficiency	Operational Complexity	Security Considerations
Infrastructure-Level (VMs)	Strong	Low	High	High
Container-Level (Kubernetes)	Moderate	High	Moderate	Moderate
Namespace-Based Isolation	Logical	High	Moderate	Moderate

IV. NETWORK ISOLATION TECHNIQUES (LEGACY APPROACHES)

Securing multi-tenant cloud native environments requires isolation in the underlying network as well. It used several techniques in order to isolate the network in Kubernetes clusters.

A. Flat Networks and Overlay Networks

In traditional Kubernetes deployments, we saw flat networking models wherein there are no restrictions on communicating between all the pods [18]. Overlay networks were used to introduce isolation, which could be implemented using tunneling protocols like VXLAN, IP-in-IP or GRE. They were overlays of pod traffic that allowed communication of pods across nodes without having to change any data plane or control plane. However, with overlays, the complexity was added, as well as additional room for performance overhead.

B. Kubernetes Network Policies

Network Policies in Kubernetes help in controlling the traffic flow between pods. However, there were only some compatible CNI plugins, and the definition of policy was still complicated [19]. However, before version 1.20, many clusters had not had full policy enforcement, and as such, vulnerabilities could be created.

C. Calico, Flannel, Weave – comparison and use-cases

There are several Container Network Interface (CNI) plugins developed for providing network isolation within Kubernetes clusters with unique features and usage cases. Calico provides BGP routing and comprehensive support for Kubernetes Network Policies, and offers advanced networking capabilities [20]. While this is not the best solution for a small general-purpose firewall, it is the right choice for large, security conscious deployments needing very granular control over network traffic. High performance and scalability, Calico's architecture enables complex and enterprise grade environments. It uses a simple overlay network created using VXLAN to deploy flannel easily. However, it does not natively support Kubernetes Network Policies that may be a limiting factor for the environments where very strict network segmentation and security controls are required.

As a general rule, flannel is often chosen for smaller clusters or development environments where simplicity and easiness outweigh other advanced networking features. Weave Net is a mesh overlay network that provides encryption and DNS service discovery by default. Kubernetes Network Policies are supported and are used to guarantee secure communication between pods [21]. Weave Net is a good choice if what you need is something user friendly, that offers lots of nice features like automatic service discovery, and you do not expect to need to scale up to very large clusters. These CNI plugins are intended to achieve different needs in Kubernetes networking, and the selection of one of these CNI plugins over another should be based on each individual's specific security, scalability, performance, and ease of deployment requirements.

Table 2: Comparison Of Key Features Of CNI Plugins (Calico, Flannel, Weave Net) For Kubernetes Network Isolation

Feature	Calico	Flannel	Weave Net
Network Model	BGP-based routing and overlay network	Simple overlay network using VXLAN	Mesh overlay with encryption and service discovery
Kubernetes Network Policy	Full support for Kubernetes Network Policies	Limited support (lacked native policy enforcement)	Supports Kubernetes Network Policies
Scalability	High scalability, suitable for large-scale deployments	Suited for small to medium-sized clusters	Suitable for medium-sized deployments
Performance	High performance, efficient networking	Lower performance due to overlay complexity	Moderate performance, additional overhead from mesh
Security	Strong security features, supports	Basic network isolation with no	Encryption built-in, moderate

	network segmentation	security features	security support
Ease of Use	More complex, requires advanced configuration	Simple to deploy and configure	User-friendly, automatic service discovery
Use Cases	Large-scale, high-performance, security-focused environments	Small clusters, development environments	Medium-sized deployments, ease of use emphasized

D. Use of iptables, ebtables, and VXLAN

In particular, during the days before low level tools like iptables, ebtables, and tunneling technologies like VXLAN fell out of favor as a means to implement network isolation and traffic control in containerized, multi-tenant environments, they were critical to its success. iptables served as the primary packet filtering framework in Linux and was heavily used to enforce network policies, port forwarding, and NAT rules within Kubernetes clusters [22]. Container runtimes and network plugins leveraged iptables to programmatically manage pod-to-pod communication, service exposure, and traffic redirection. However, managing iptables at scale became increasingly complex as rule sets grew dynamically with each pod and service deployment, often leading to performance bottlenecks and debugging challenges.

Ebtables, a lesser-known but important counterpart to iptables, operates at Layer 2 of the OSI model [23]. It was occasionally used in container environments to enforce Ethernet-level filtering, particularly in bridge-based networking setups. In legacy Docker and Kubernetes networking, ebtables helped control ARP and MAC address spoofing, which were relevant security concerns in multi-tenant container bridges. VXLAN (Virtual Extensible LAN) emerged as a common encapsulation protocol for building overlay networks, especially in scenarios where Kubernetes clusters spanned multiple nodes or data centers. VXLAN allowed for Layer 2 tunneling over Layer 3 networks by encapsulating Ethernet frames in UDP packets. CNI plugins like Flannel (in VXLAN mode) and Calico (in its overlay mode) employed VXLAN to create logical pod networks without requiring underlying physical network reconfiguration. Despite its utility, VXLAN introduced additional latency and complexity in packet inspection and debugging, and was often less performant than native routing approaches.

While these technologies were instrumental in enabling early network isolation strategies in Kubernetes, they lacked integration with higher-level constructs like identity-aware policies or service-level security controls. As such, isolation mechanisms built solely on iptables, ebtables, and VXLAN were often static, hard to manage, and insufficient for achieving fine-grained, tenant-aware security in large-scale cloud-native environments.

E. Lack of Zero-Trust Enforcement and Workload Identity

Almost all cloud native environments did not have their own true zero trust network enforcement, instead relying on implicit trust models that presumed workloads could freely communicate between themselves until explicitly blocked. Kubernetes clusters tended to come with permissive networking by default, and Kubernetes Network Policies were available, but inconsistently used and hard to manage at large scale. One of the main shortcomings was lack of strong workload identity. Mutable things like IP addresses or labels were used for identifying workloads that could be spoofed or changed and that made them policies brittle and insecure. At the time, secure service to service authentication was not widely adopted because of standards such as SPIFFE/SPIRE in particular, which introduced the concept of cryptographic workload identities.

However, early service meshes (such as Istio, Linkerd v1) introduced some of these ideas (such as mutual TLS and identity-based routing) but their complexity and overhead of operation made them unusable in production across multi-tenant clusters. In general, the lack of identity aware enforcement and granular trust boundaries have rendered many environments susceptible to lateral movement and internal threats. Initial tools used for network isolation in the multi-tenant cloud-native environments came in form of iptables, ebtables and VXLAN, as well as early forms of Kubernetes network policies [24]. These did the trick for the basics of isolation, but the need for scalability and granularity was missing for large deployments. In addition, there was zero trust enforcement and strong workload identity, which further weakened the security. While these early techniques were foundational, they exposed gaps which lead to the push for more secure systems with service meshes and identity aware policies, both concepts are still evolving to further secure multi-tenant systems.

V. SECURITY RISKS AND GAPS IDENTIFIED

Some key security risks and gaps in multi-Tenant Kubernetes and containerized App environment were quite prominent. Mostly these risks were because of the lack of advanced security features like strong workload identity, east west traffic visibility, and overall comprehensive service mesh enforcement. The major security challenges in cloud-native environment are set out below.

A. No Strong Workload Identity

Among the most important gaps in security was a lack of an identity-based approach to workloads. In most Kubernetes environments selectors consisted of mutable and easily spoof able identifiers such as IP addresses, labels, or pod names [25]. The absence of any cryptographic identity to tie together inter service communication, makes securing communication between services and workloads hard, and also allows attackers to impersonate services or workloads. Many environments lacked these standards and thus, have remained exposed to unauthorized access and lateral movement: standards like SPIFFE (Secure Production Identity Framework for Everyone) and SPIRE (SPIFFE Runtime Environment) were not in mass adoption. Granular security for workloads couldn't be enforced without a well-defined identity model. This weakness made service impersonation easier and happened at the expense of not being detected by malicious attacks on the shared infrastructure.

B. East-West Traffic Visibility Issues

Communication between services or pods within the same cluster, i.e. East-West traffic, was not frequently monitored and much less governed [26]. In a cloud native world where the boundary between inside and outside is no longer based on a perimeter, traditional perimeter security models did not work when containers or services were dynamic. The available tools were not giving adequate visibility into internal communication tool allowing it to be difficult to detect lateral movement or find out compromised workloads performing some malicious actions. With a lack of visibility into east west traffic, there was a higher probability of breaches that went unnoticed within the cluster since the attacker was able to roam freely through different workloads and services without triggering conventional security measures.

C. Lack of Service Mesh Enforcement

Istio and Linkerd, each service meshes, were supposed to deliver secure communication and advanced traffic management for microservices architectures [27]. Early versions of these tools, for example Istio v1.0 and Linkerd v1, were not as mature, as well as had much less functionality than modern matured ones. They brought out the concepts of mutual TLSs for secure communication, but they were operationally complex and scaling limitations meant they were very hard to roll out at a large scale in multi-tenant environment. Many Kubernetes clusters lacked centralized control of traffic encryption, authentication and authorization between services without full-service mesh enforcement. At the same time, however, this trend left communication between workloads vulnerable to interception and attack, especially since adoption of these tools was to slow in many organizations.

D. Namespace Escape Vulnerabilities in Early Container Runtimes (runc, Docker)

Previously, many container runtimes such as runc and Docker have suffered from a number of security vulnerabilities that allowed the possibility of namespace escapes, where a container would escape its isolated environment and access resources outside of its assigned namespace. These vulnerabilities opened workloads to trials of unauthorized access or manipulation by other tenants sharing the same multi–tenant environment. For instance, a threat vector (CVE-2019-5736) well known to many people, that allowed attackers to execute arbitrary code on the host system with a flaw in how containers interacted with host filesystem. Exploitation of this vulnerability could lead to escape from container boundaries, gain root access, and affect other containers running on the same host.

E. Legacy CVE Examples or MITRE ATT&CK Mappings

Several CVEs (Common Vulnerabilities and Exposures) highlighted the security weaknesses in container and Kubernetes environments. Some notable examples include:

- *CVE-2019-5736:* Docker container escape vulnerability, allowing containers to break out of their isolation [28].
- *CVE-2020-8554:* Kubernetes API server vulnerability that allowed attackers to bypass authentication and perform unauthorized actions [29].
- *CVE-2020-8555: Kubernetes container runtime privilege escalation, which could lead to arbitrary code execution [30].*

Many of these vulnerabilities are a result of lack of sufficient network and workload isolation mechanisms, weak identity management, and poor adherence to security policies in cloud native environment. The mappings to container security tactics and techniques, like privilege escalation, lateral movement, service impersonation, are also present in the MITRE ATT&CK framework.

These include lack of strong workload identity, limited visibility into east west traffic, lack of service mesh, enforcement and security vulnerabilities in the early container runtimes, all adding significant security risks in multi tenant cloud native

environments. These gaps created vulnerabilities for cloud native architectures to attack, data breach, and unauthorized access which shows the necessity for better security model and tooling as cloud native technologies were evolving.

VI. RECOMMENDATIONS AND BEST PRACTICES

Several best practices were important to mitigate risks and provide proper configuration before a wide adoption of enhanced tools for safety and network isolation in cloud native environments: Below are the key recommendations:

A. Use of Dedicated Namespaces and Network Policies

Isolation at the Namespace Level: Leveraging An important layer of segmentation within Kubernetes clusters was accomplished with dedicated namespaces for segregating different workloads. Admin could control traffic flow across different services and pod by combining namespaces with Kubernetes Network Policies. This was founding in drastically reducing the blast radius in the case of a breach.

- Assign distinct namespaces for various application components.
- Define and enforce strict Network Policies to control communication between namespaces.

B. Segregating Environments via Separate Clusters

Environment Isolation for Enhanced Security: To Separating a development, staging, and production cluster would be highly recommended to mitigate the chance of cross-environment vulnerabilities. Physical isolation was the method used to contain security breaches within a single environment.

- Each environment (dev, staging, production) should have their own independent cluster.
- Isolate traffic in dedicated networking configurations (e.g., separate VPC or subnets).

C. Enforcing Ingress/Egress Controls Manually

Granular Control of External Traffic: Prior to automated solutions being more widely used, services were subjected to manual ingress and egress controls to limit which services could access external resources or talk to the wider internet. This was essential in order to minimize the attack surface exposure for data exfiltration or lateral movement.

- Implement firewalls and proxy servers to control outbound and inbound traffic.
- Apply strict egress rules to prevent sensitive data from leaving the network unmonitored.

D. Layered Firewall Rules, VPCs, and Security Groups

Multiple Layers of Defense: Layered security strategies such as firewalls, Virtual Private Clouds (VPCs), and security groups provided fine grained control of network traffic. The security was greatly improved by configuring firewalls at various levels (e.g. pod level, node level, cluster level etc.).

- Use VPCs and subnets to isolate different segments of the network.
- Configure firewall rules to restrict traffic between different parts of the network.
- Leverage security groups to enforce network-level access controls for specific resources.

E. Manual Network Segmentation Strategies

Purposeful Segmentation for Enhanced Isolation: In Without advanced measures, manual network segmentation was key to isolating between various workloads and tiers. Segregation of web servers, databases, and backend services into application components prevented unwelcome attack and allowed traversal to be blocked.

- Define and segment networks manually by role such as web servers, databases or internal services.
- Ensure that sensitive workloads are isolated by restricting between segments using access control lists (ACLs).

This wasn't all the advanced features we have today, the isolation practices didn't have the automation and X or Y, but these were solid security postures early in what we had. In a multi-tenant cloud native environment, focusing on the manual controls like namespace segmentation, separate clusters and ingress and egress controls would help organizations to create a solid security posture.

VII. LIMITATIONS OF LEGACY APPROACHES

Although the most important best practices mentioned previously are critical, there are many important limitations which lead to the failure of the network isolation strategies to work with cloud native environment. Although these limitations were quite clear with smaller Kubernetes deployments, as deployments of Kubernetes continued to scale other orders of magnitude, as well as more complex multi-tenanted clusters started creeping in, the limitations started to be felt more and more. The following are the major challenges of existing legacy network isolation techniques.

A. Scalability Bottlenecks

Challenges with Growing Environments: As When Kubernetes clusters grew more in size and more containers came to be there, it proved to be hard to scale these methods (firewall and iptables rules), so efficiently manual rule. This constant creation and destruction of services and pods in containerized environments made it difficult to keep the policies up to date.

- Static configuration, such as manually defined firewall rules and network policies, couldn't keep up with the growing complexity of large-scale environment.
- The use of iptables and other low-level tools introduced significant performance overhead, particularly in large cluster.

B. Complex Policy Management

Difficulty in Defining and Enforcing Policies Network policies were complex and error prone to manage across one or more clusters and environments. Without advanced tools, it was a laborious task to create and maintain a consistent set of security policies for different workloads, namespaces, and services.

- Policies had to be manually defined, monitored and updated, leading to increased operational overhead
- The Lack of centralized policy management often result in inconsistent enforcement and gaps in security coverage.

C. Lack of Integration with Identity Systems

Security Weakness Due to Mutable Workload Identification: Prior Prior to virtually every Kubernetes workload becoming hooked on cryptographic identity systems like SPIFFE/SPIRE, each Kubernetes workload identified itself via mutable attributes (e.g. IP addresses or labels). This did not prevent all attacks (an attacker could still use a valid identity) but it did increase the value of someone gaining the ability to spoof and bypass security policies.

- Insecure policies were created because a lack of strong workload identity mechanisms resulted in policies being susceptible to spoofing and their exploitation.
- Enforcing fine grained, identity aware security policies was not well integrated with identity management solution.

D. Inconsistent Enforcement Across Platforms

Fragmented Security Policies: In However, network isolation and security policies were not been enforced consistently across different platform in the multi cloud or hybrid environment. To summarize, Each of their own set of tools, configurations and limitations, and consequently each had their own set of gaps in enforcement, and each could leave the workloads exposed and vulnerable to attacks.

- The lack of consistent enforcement of security policies in cloud provider environments as well as on the premises left us with a fragmented security posture.
- The reliance on platform-specific tools made it difficult to standardize and automated network practice across heterogeneous environment.

The approaches for network isolation within the Kubernetes domain was a welcome start, but were limited by various flaws. There were several scalability challenges, complex policy management, weak identification of workloads to support, and uneven enforcement of policy across platforms, which were large impediments to achieving a secure, multi-tenant cloud at scale. The past few years have also seen the emergence of cloud native security solutions that have largely solved most of these limitations towards more effective and automated network isolation strategies.

VIII. TRANSITION TO MODERN ISOLATION MODELS

The world of cloud native security evolved with various advancements that changed the game when it comes to how network isolation and workloads protection are done on Kubernetes and in the containerized environment. Most of the shortcomings of earlier approaches were addressed by these innovations, for example, with stronger, identity aware and scalable solutions.

A. Service Mesh

The modern service meshes such as Istio, Linkerd 2.x and Consul matured a lot – they have built in mutual TLS (mTLS), traffic policies on finest granularity and telemetry for observability. With these tools, we were able to perform transparent encryption; zero trust enforcement; and secure service to service communication with little application changes.

B. Zero-Trust Network Access (ZTNA)

More and more models began to adopt zero trust principles which meant that by default none of the workload nor user were trusted even within the network perimeter. They integrated solutions to tighter segment with integration of identity based access control along with integration of policy as code and continuous authentication.

C. eBPF-Powered Networking and Security

Extended Berkeley Packet Filter (eBPF) completely revolutionized observability and enforcement by allowing kernel programmability at kernel level without any changes to the kernel modules. Dynamic traffic control, policy enforcement and real time security monitoring were previously performed by projects like Cilium with high performance and context aware isolation, using eBPF. It is a big step away from static infrastructure: on information, to dynamic identity: on information, policy on the identity model. It also enables cloud native environments to scale securely, remains agile, visible, and compliant.

IX. CONCLUSION

Network isolation has always been a foundational requirement for securing containerized workloads in multi-tenant Kubernetes environments. Early efforts depended heavily on basic Linux networking tools such as iptables, ebtables, and overlay technologies like VXLAN.While these solutions appeared effective at abstracting traffic for static segmentation, complex to manage at scale due to static rules, had no integration with any security constructs beyond overall control, had gaps in fine grain control and visibility

Container Network Interface (CNI) plugins such as Calico, Flannel, and Weave Net provided different approaches to implementing networking in Kubernetes. Although they introduced capabilities like policy enforcement and encrypted communication, their effectiveness varied depending on deployment size, performance needs, and policy complexity. Even many clusters that have not posed as consistent or centralized control, lacked identity mechanisms and centralized control, and continued to be vulnerable to misconfigurations, lateral move.

Real life incidents that served as the go between network segments and isolation allowed the flood gates to open to significant breaches. The legacy strategy was when we relied on things like manual controls with things like static firewall rules and separate clusters, which were error prone and were hard to maintain. It is necessary to bring a more holistic and identity aware approach to reduce risk and protect cloud native applications from the both internal and external threats.

X. REFERENCES

[1] W. Cao et al., “Logstore: A cloud-native and multi-tenant log database,” in Proc. 2021 Int. Conf. Manage. Data, 2021.

[2] A. Khan, “Key characteristics of a container orchestration platform to enable a modern application,” IEEE Cloud Comput., vol. 4, no. 5, pp. 42–48, 2017.

[3] A. Ranjbar, M. Antikainen, and T. Aura, “Domain isolation in a multi-tenant software-defined network,” in Proc. 2015 IEEE/ACM 8th Int. Conf. Utility Cloud Comput. (UCC), 2015.

[4] M.-M. Bazm et al., “Isolation in cloud computing infrastructures: new security challenges,” Ann. Telecommun., vol. 74, pp. 197–209, 2019.

[5] S. G. Haugeland et al., “Migrating monoliths to microservices-based customizable multi-tenant cloud-native apps,” in Proc. 2021 47th Euromicro Conf. Softw. Eng. Adv. Appl. (SEAA), 2021.

[6] J. Dobaj et al., “A microservice architecture for the industrial internet-of-things,” in Proc. 23rd Eur. Conf. Pattern Lang. Programs, 2018.

[7] T. Laszewski et al., Cloud Native Architectures: Design High-Availability and Cost-Effective Applications for the Cloud. Packt Publishing Ltd, 2018.

[8] A. Balalaie, A. Heydarnoori, and P. Jamshidi, “Migrating to cloud-native architectures using microservices: an experience report,” in Eur. Conf. Serv.-Oriented Cloud Comput., Cham: Springer Int. Publishing, 2015.

[9] M. Tsechelidis, “Developing distributed systems with modular monoliths and microservices,” 2023.

[10] J. Watada et al., “Emerging trends, techniques and open issues of containerization: A review,” IEEE Access, vol. 7, pp. 152443–152472, 2019.

[11] K. Indrasiri and D. Kuruppu, gRPC: Up and Running: Building Cloud Native Applications with Go and Java for Docker and Kubernetes. O'Reilly Media, 2020.

[12] F. Soppelsa and C. Kaewkasi, Native Docker Clustering with Swarm. Packt Publishing Ltd, 2016.

[13] R. Krebs, Performance Isolation in Multi-Tenant Applications. Karlsruhe Institute of Technology, 2015.

[14] R. Jia et al., “A systematic review of scheduling approaches on multi-tenancy cloud platforms,” Inf. Softw. Technol., vol. 132, p. 106478, 2021.

[15] C. Zheng, Q. Zhuang, and F. Guo, “A multi-tenant framework for cloud container services,” in Proc. 2021 IEEE 41st Int. Conf. Distrib. Comput. Syst. (ICDCS), 2021.

[16] A. Terzolo, Enabling Multi-Tenancy and Fine-Grained Security in a Multi-Cluster Architecture. Politecnico di Torino, 2021.

[17] M. Uddin, S. Islam, and A. Al-Nemrat, “A dynamic access control model using authorising workflow and task-role-based access control,” IEEE Access, vol. 7, pp. 166676–166689, 2019.

[18] B. Burns et al., Kubernetes: Up and Running: Dive into the Future of Infrastructure. O'Reilly Media, Inc., 2022.

[19] R. Kumar and M. C. Trivedi, “Networking analysis and performance comparison of Kubernetes CNI plugins,” in Adv. Comput., Commun. Comput. Sci.: Proc. IC4S 2019. Springer Singapore, 2021.

[20] G. Budigiri et al., "Network policies in Kubernetes: Performance evaluation and security analysis," in Proc. 2021 Joint Eur. Conf. Netw. Commun. & 6G Summit (EuCNC/6G Summit), 2021.
[21] S. Raghunathan, "Optimizing container communication: Navigating challenges and solutions in Kubernetes networking," J. Sci. Eng. Res., vol. 8, no. 2, pp. 257–262, 2021.
[22] T. D. Zavarella, A Methodology for Using eBPF to Efficiently Monitor Network Behavior in Linux Kubernetes Clusters. Massachusetts Institute of Technology, 2022.
[23] A. K. Niazi and M. A. A. Usmani, "An analysis on scalable and faster iptables in Linux operating system," I-Manager's J. Comput. Sci., vol. 8, no. 2, 2020.
[24] X. Nguyen, "Network isolation for Kubernetes hard multi-tenancy," 2020.
[25] B. Creane and A. Gupta, Kubernetes Security and Observability: A Holistic Approach to Securing Containers and Cloud Native Applications. O'Reilly Media, Inc., 2021.
[26] Z. Liu, Z. Qian, and N. Li, "An East-West-traffic governance system based on eBPF and centralized gateway," in Proc. 2023 IEEE Int. Conf. Sensors, Electron. Comput. Eng. (ICSECE), 2023.
[27] Z. Butt, "Secure microservice communication between heterogeneous service meshes," 2022.
[28] National Institute of Standards and Technology, "CVE-2019-5736 Detail," Nat. Vulnerability Database, 2019. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2019-5736
[29] National Institute of Standards and Technology, "CVE-2020-8554 Detail," Nat. Vulnerability Database, 2020. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2020-8554
[30] National Institute of Standards and Technology, "CVE-2020-8555 Detail," Nat. Vulnerability Database, 2020. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2020-8555
[31] Jangid, J., Dixit, S., Malhotra, S., Saqib, M., Yashu, F., & Mehta, D. (2023). Enhancing security and efficiency in wireless mobile networks through blockchain. International Journal of Intelligent Systems and Applications in Engineering, 11(4), 958–969. https://ijisae.org/index.php/IJISAE/article/view/7309
[32] Yashu, F., Saqib, M., Malhotra, S., Mehta, D., Jangid, J., & Dixit, S. (2021). Thread mitigation in cloud-native application development. Webology, 18(6), 10160–10161. https://www.webology.org/abstract.php?id=5338s