Guide

Container Segmentation Strategies and Patterns

Container Segmentation Strategies and Patterns

Segmentation, Container Segmentation, and Micro-Segmentation

Introduction

At a recent container security conference the topic of 'container segmentation patterns' came up, and it became clear that many security architects are wrestling with how to best segment workload communication in the dynamic environment of containers. The question was also raised "Is the DMZ dead?"

The concept of network segmentation has been around for a while and is considered a best practice to achieve 'defense in depth' for business critical applications. Proper segmentation can protect applications from hackers as well as limit the 'blast radius' in the case of a breach. So it makes sense that devops and security professionals would wonder if container segmentation would provide similar protections for a container network, and how this could be possible with network plug-ins, CNIs, and SDNs.

Old Ways of Segmentation – Patterns

Before we get into how container segmentation works let's review some of the common traditional patterns for achieving network segmentation:

- **The DMZ.** All external access including internet application front-ends such as web servers are placed in the DMZ, which uses perimeter firewalls to restrict inbound and outbound traffic.
- **Physical Network Segments.** Even behind the DMZ, different network segments are used for applications with different trust levels and to further segment communication to sensitive data such as databases. Perimeter firewalls are used to contain traffic in each segment.
- **Data Center Segmentation.** In extreme cases, segmentation is achieved by placing applications and infrastructure in separate data centers, each with its own security protections.
- **VPC and Security Groups.** More recently, in public cloud services VPCs and security groups are used to segment traffic with network segmentation policies and ingress/egress firewall rules easily applied to different VPCs. However, this is still a tedious manual configuration of L3/L4 policies that can't protect containers.

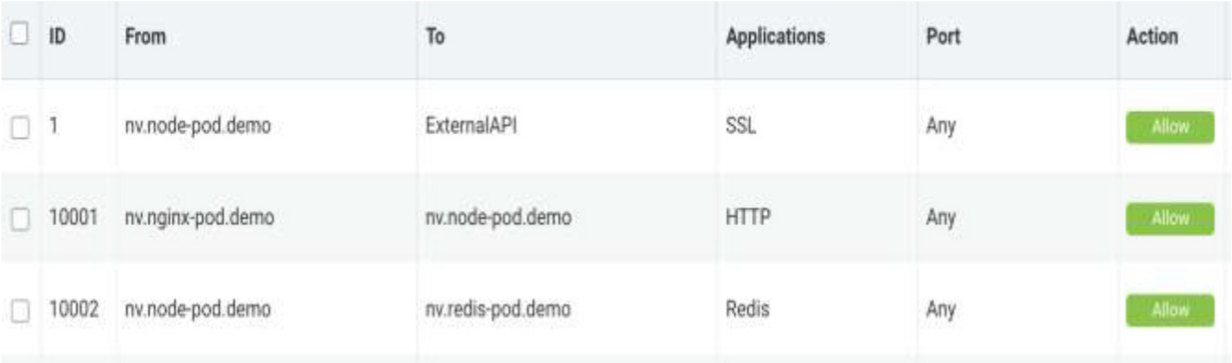
- **Separate Data and Control Networks.** Less common but with the same goal of separating traffic to control attacks, control plane and monitoring traffic is segmented on each server to separate them from data transmissions, minimizing the possibility of data breaches from monitoring and system tools.

The patterns above all attempt to segment network communications according to varying trust levels of the applications running in each segment. One commonality between them is the use of physical network controls and traditional firewalls to separate traffic. Even VPCs are based on traditional notions of a physical network segment. As we'll discover later, in a truly cloud native environment, these segmentation techniques become increasingly ineffective as workloads become dynamically deployed across traditional network boundaries.

The DMZ is Dead, Or Is It?

The general consensus around the room at this gathering of container security and operations people was that 'the DMZ is dead.' In this world of overlay networks, Kubernetes and public cloud providers, the old way of thinking of a DMZ to segment all internet facing applications is no longer relevant. In reality, DMZs will still exist, but they will be almost invisible, or irrelevant, to the security discussion, because they are not the primary way to protect access to applications and databases.

Given this realization, how do security architects provide network visibility and protection in an environment where external access frequently comes through an ingress into a container cluster directly from the internet? In addition, ingress/egress connections to container based api services must be allowed in what was traditionally considered an 'east-west' flow of traffic. It seems in cloud environments that the definitions of north-south and east-west traffic are becoming blurred.



□	ID	From	To	Applications	Port	Action
□	1	nv.node-pod.demo	ExternalAPI	SSL	Any	Allow
□	10001	nv.nginx-pod.demo	nv.node-pod.demo	HTTP	Any	Allow
□	10002	nv.node-pod.demo	nv.redis-pod.demo	Redis	Any	Allow

What is Segmentation in a Cloud- native, Container-based World?

Container segmentation is the practice of segmenting container communications so only authorized connections between containers are allowed. In practice, because containers are typically created from a service concept by orchestration tools such as Kubernetes, container segmentation can be enforced at the service level. Multiple containers scaling up from the same image/service should not require different network segmentation policies in most cases.

For example, these are layer 7 segmentation rules from the NeuVector console.

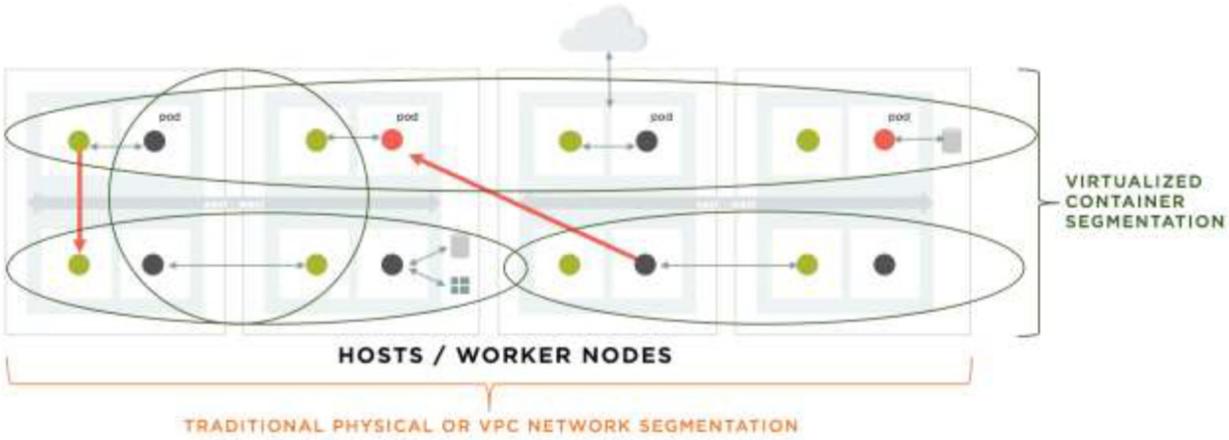
These whitelist rules allow connections from one Kubernetes service (all pods) to another and requires a certain protocol to be used. For example, rule 10002 requires the redis application protocol between the nodejs demo pods and redis demo pods.

Container segmentation is often called micro-segmentation or nano-segmentation because containers are often deployed as microservices which can be dynamically deployed and scaled across a Kubernetes cluster. Because different services can be deployed across a shared network and servers (or VMs, hosts), and each workload or pod has its own network addressable IP address, container segmentation policies can be difficult to create and enforce.

However, without the ability to segment container connections and enforce network restrictions the blast radius of an attack can be the entire cluster, or worse yet, the entire container deployment across clouds.

What's needed is more of a virtualized network segmentation capability that is aligned more tightly with how cloud-native container services are deployed, as shown below.

Container segmentation can provide the required protection regardless of where the workload is deployed and give confidence to the security and devops teams that unauthorized connections between segments can be prevented, or at least detected and alerted.


pod
pod
pod
pod
VIRTUALIZED CONTAINER SEGMENTATION
HOSTS / WORKER NODES
TRADITIONAL PHYSICAL OR VPC NETWORK SEGMENTATION

What About Namespaces? Network Policy?

Going back to the container security gathering of experts, the general agreement was that namespaces **can NOT be trusted** to enforce container segmentation policies. While namespaces do provide some level of segmentation between containerized services, security teams should not rely of them for defense in depth. The built-in Network Policy features of Kubernetes were also deemed to be not practical for most business critical deployments. These opinions were due to a number of cited reasons:

1. Recent demonstrations of breaching namespace boundaries.
2. Cumbersome granularity of segmentation policies.
3. Lack of policy management framework.
4. Lack of visibility and monitoring.
5. Inability to detect network attacks within trusted connections.

Namespaces were found to be useful for organizing services and to ease the management of such services where each service in a namespace has some attribute in common with others to make them manageable as a group. But, don't use namespaces for your container segmentation strategy.

Will a Service Mesh Do Segmentation?

The excitement about service mesh technologies like Istio and Linkerd2 is driven by the promise of an application discovery and routing layer for containers which has some security features built-in. But there is a difference between a Layer 7 load balancer with security features and a true security product like a Layer 7 container firewall. Security features in service meshes include the ability to do authentication, authorization, and encryption of connections. By authorizing connections between containers based on defined policies, a service mesh has the ability to do segmentation for certain HTTP protocols.

Limitations to keep in mind about using a service mesh for segmentation include:

- **Does not support all HTTP protocols, nor ICMP, or UDP.** If you have applications requiring other protocol support you will need a multi-protocol container firewall.
- **Has no visibility into policy violations.** If a connection attempt is blocked, is there logging of the event which makes it easy to drill down into the source and destination service names and IP addresses, network payload used, and other forensic details?
- **Can't detect embedded network based attacks within trusted connections.** A container firewall should be able to inspect network payloads for embedded application attacks such as SQL injection, even in authorized trusted connections.
- **Can't perform DLP functions.** Can't inspect connections for sensitive data such as credit cards and PII.

- **Does not provide other alerting and network forensic capabilities.** There are many required features in a true container security product such as alerting, response rules, packet captures and enterprise integration hooks.
- **Lacks management and automation.** Authorization policies must be defined manually. Although these can be automated during deployment, the creation and management of these policies is difficult to centrally administer and review.

It is therefore important to consider the protocol requirements for containerized applications today as well as for the next few years, as well as the desired level of security required for your applications. Modern container security tools should be integrated with service mesh technology to provide defense in depth, and enhance those built-in security features.

Layer 7 Container Segmentation

Segmentation of network traffic can be done at Layer 3/4 based on IP addresses and ports, but in cloud-native environments this is best done at Layer 7 to detect and verify the application protocol used. This provides better scalability, manageability, and flexibility for deployments to change without needing to change security rules. An added benefit of Layer 7 deep packet inspection is the ability for the container firewall to inspect network traffic for hidden, or embedded attacks, even within trusted connections between workloads.

Multi-protocol Layer 7 segmentation provides detection and enforcement of connections across multiple application protocols and should also support non-HTTP protocols such as ICMP and UDP.

True Workload Segmentation Across Clusters, Clouds

With a cloud-native, Layer 7 container segmentation solution, workloads can be segmented even if they are running on the same host, network, or cluster. The ability to mix workloads of different required trust levels on the same infrastructure provides the ultimate flexibility for architects and devops teams to maximize performance, resource utilization, and speed up the pipeline. It also limits the blast radius if one set of services is hacked from spreading laterally onto other workloads, even if running on the same host.

Container Segmentation Patterns

Containers and orchestration tools like Kubernetes are relatively new, so there will be many experiments using combinations of old and new technologies to achieve container segmentation. Use of traditional segmentation patterns based on physical networks as described above may provide temporary protections for containers while sacrificing many of the main benefits such as scalability and resource optimization. Here are a few example patterns, some a mix of old and new, and some which can only be achieved with cloud-native container firewalls.

- **Separate Clusters.** It is probably most common to see multiple clusters being deployed. This is due to different reasons, with security focused network segmentation being only one of them.
  - **Security focused.** Application workloads with different security protection levels can be separated by Kubernetes clusters. This makes isolating traffic easier by using traditional firewalls or VPCs to prevent cross-cluster communication. If connections between clusters are required then it can be manually allowed but management can become cumbersome and error prone.
    - For example, one cluster runs the application workloads and a separate one running databases, file storage (such as S3/minio) and other persistent storage for the same project because different security profiles are required for each cluster.
- **Cluster Manageability.** More often, separate clusters are deployed primarily for manageability reasons, with security being a secondary consideration. Separation can be based on:
  - **Application Characteristics.** For example, separate stateless and stateful clusters where management of services and workloads follows different processes. In the same example above separating application workloads from databases, the reason may include or be only due to the fact that each cluster requires different workload management approaches for rolling updates, backups, persistent data, etc.
  - **Platform Management.** Separating update and maintenance of the orchestration platforms and tools. For example, updating the Kubernetes version with all system containers and integrations may require a different process depending on the application workload requirements in the cluster.
  - **Organizational.** Separate clusters for divisions, departments, development teams or other reasons tied to how teams are organized.
  - Other deployment patterns we've seen could be based on availability of cloud resources in specific regions for public cloud providers, for example applications requiring GPU instances.
- **Container Zones.** Many companies think of clusters as zones, with each zone representing a collection of related services and/or services with similar security requirements. Although typically one cluster is deployed per zone, a container cluster could span multiple zones. The segmentation policies are based on the connection requirements in each zone, but typically focus on ingress and egress policies between zones and to the internet.
- **One Large Cluster.** Multiple application stacks, services, and workloads can be dynamically deployed in a large shared cluster. While this may present manageability issues described above, it may simplify maintenance of the orchestration platform and optimize resource utilization. Security issues, especially network segmentation policies for each service running in the cluster must be carefully managed and monitored to protect against lateral movement of attacks between workloads.

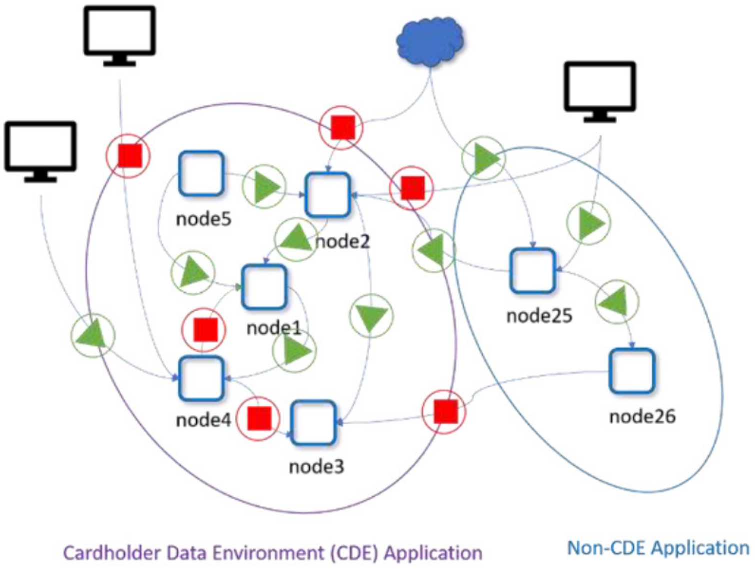
- **Cross cluster routing with Service Mesh.** Cross cluster connections are made more dynamic with service mesh technologies like Istio. While cross cluster routing can be secured to some extent by using the authorization features of the service mesh, business critical applications will need true Layer 7 container firewalling described above to protect against embedded attacks, detect multiple protocols, and make container segmentation policies manageable and scalable.

Segmentation for Compliance – PCI- DSS

One segmentation pattern of particular interest is for PCI-DSS compliance. Sections 1.2 and 1.3 of PCI-DSS require in-scope CDE traffic to be firewalled and segmented from all other connections. Traditionally, this was accomplished by using separate networks separated by traditional firewalls.

While it is certainly possible to repeat this pattern for cloud native applications, doing so will ultimately add more friction to the modern CI/CD and deployment pipelines, as well as increase costs and reduce resource utilization of separate clusters. This means all of the potential benefit of cloud-native applications will not be possible to be realized.

The better solution is to achieve network segmentation automatically between CDE and non-CDE workloads, even if they are running on the same host, network, cluster, or cloud, as shown below.

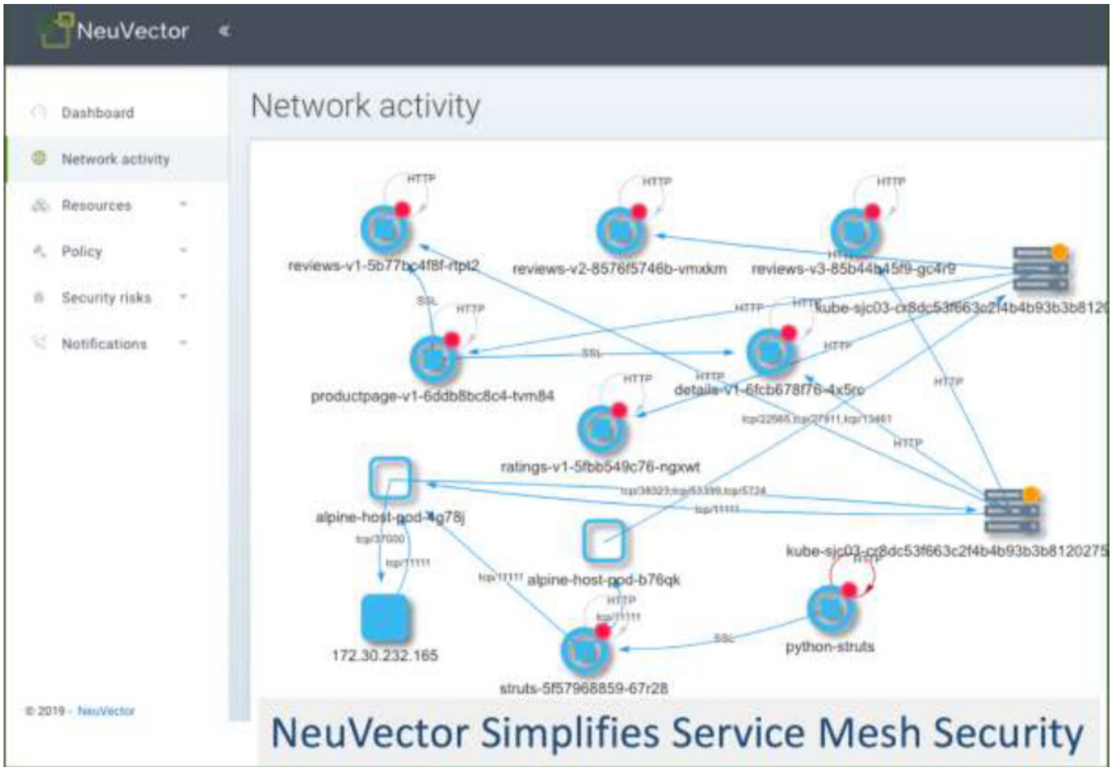


In the diagram above, the nodes are containers (not hosts) which can run dynamically across any host within the cluster. They can be segmented virtually by service names, labels, application protocols or other application metadata.

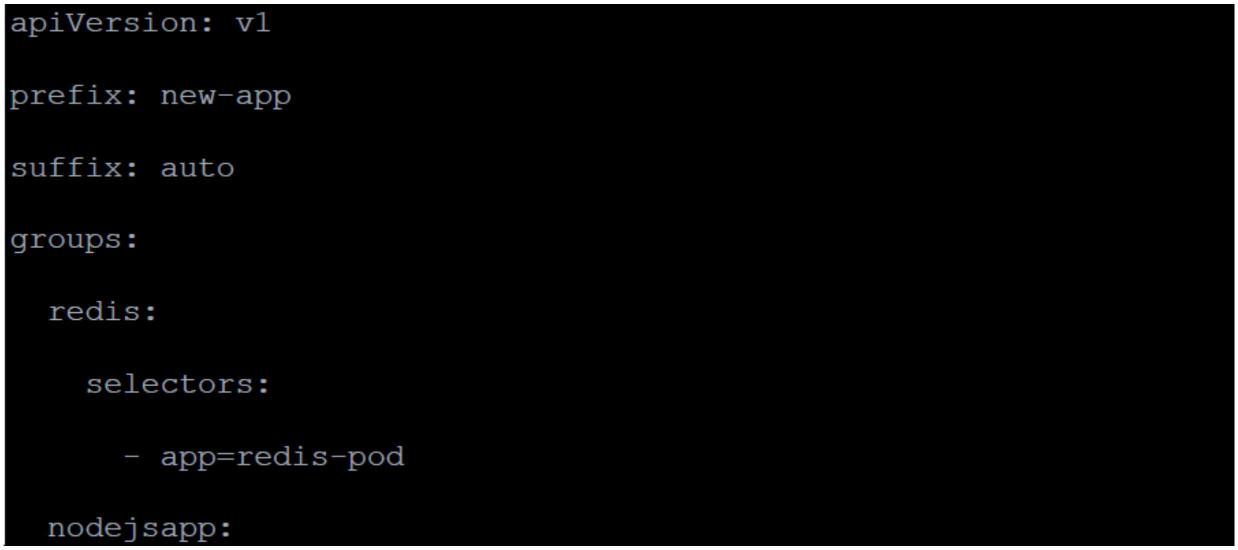
NeuVector Container Segmentation

NeuVector provides a true cloud-native Layer 7 container firewall which does network segmentation automatically. By using behavioral learning, connections and the application protocols used between services are discovered and whitelist rules to isolate them are automatically created. This means that container segmentation is easy and automated, without requiring knowledge of connections beforehand or the manual creation and maintenance of segmentation rules.

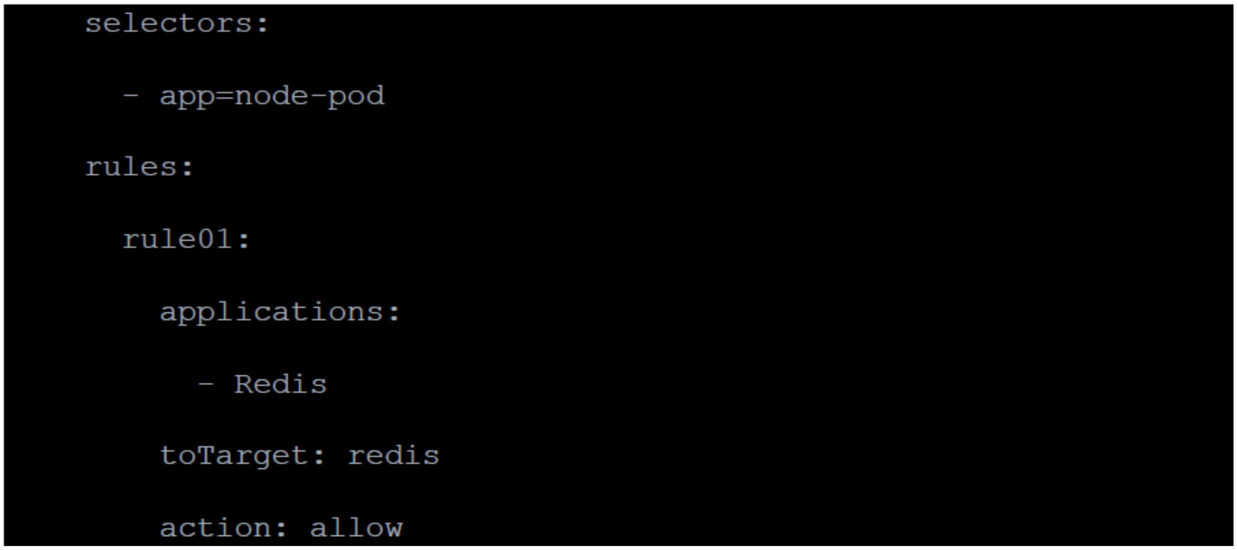
In the screenshot below, NeuVector provides a virtual view of container segmentation rules, violations, attacks and vulnerabilities regardless of the physical hosts in use. This also shows service mesh enabled pods where an Istio sidecar container is used for encryption.


NeuVector
Dashboard
Network activity
Resources
Policy
Security risks
Notifications
© 2019 - NeuVector
Network activity
reviews-v1-5b77bc4f8f-rtpt2
reviews-v2-8576f5746b-vmxkm
reviews-v3-85b44b45f9-gc4r9
kube-sjc03-cr8dc53f663c2f4b4b93b3b812
productpage-v1-6ddb8bc8c4-tvm84
details-v1-6fcb678f76-4x5rc
ratings-v1-5fbb549c76-ngxwt
alpine-host-pod-4g78j
alpine-host-pod-b76qk
kube-sjc03-cr8dc53f663c2f4b4b93b3b8120275
172.30.232.165
python-struts
struts-5f57968859-67r28
HTTP
SSL
NeuVector Simplifies Service Mesh Security

For more advanced users, NeuVector supports a declarative security policy where application level (e.g. Layer 7) policies can be specified during the CI/CD process by devops teams in order to fully automate the new releases or updating of application services. For example, the following is an example of how DevOps can declare the security rules in NeuVector in a yaml file as part of the application deployment process.



```
apiVersion: v1

prefix: new-app

suffix: auto

groups:

  redis:

    selectors:

      - app=redis-pod

  nodejsapp:
```



```
    selectors:

      - app=node-pod

    rules:

      rule01:

        applications:

          - Redis

        toTarget: redis

        action: allow
```

The example above creates the simple whitelist rule to allow the nodejs pods to connect to the redis pods only using the redis application protocol. The simplicity of such a layer 7 rule makes it scalable, flexible, and easy to manage. It also supports the ‘shift-left’ movement to push security further into the DevOps part of the pipeline, supporting faster deployments with automation.

All segmentation policies are centrally viewed, managed, and monitored so that conflicting rules are not created or connections start failing due to a forgotten deployment manifest.

Beyond container segmentation, NeuVector provides a complete Kubernetes security platform to secure the CI/CD pipeline from build to ship to run. Image vulnerability scanning starts during the build process and continuously monitors them for new vulnerabilities as soon as they're deployed.

The run-time container security is provided by Layer 7 container firewall together with container process and file system security, as well as host security. The container firewall detects threats such as sql injections, DDoS, DNS attacks and other application layer attacks by inspecting the payload even for trusted connections. It is integrated with new service mesh technologies to provide threat detection and segmentation even if the connection between two pods is encrypted.

In this way, NeuVector can provide multi-vector threat protection with the combination of network security, application security, endpoint security, and host security.

The Ultimate Cloud Security Pattern – Container Segmentation by Workload

Ultimately, to give the business the most flexibility for rapid release and optimal resource utilization, container segmentation must be enforced on each pod and follow application workloads as they scale and move dynamically. In this micro-perimeter vision article, NeuVector CTO Gary Duan outlines a vision for cloud security where the protection perimeter surrounds the workload even as it moves across hybrid clouds.

Next Steps

Want to learn more?

Contact NeuVector at https://neuvector.com for more container security articles on our blog and to schedule a demo of the NeuVector Container Security Platform, including the Layer 7 Container Firewall.

2022

SUSE
Maxfeldstrasse 5
90409 Nuremberg
www.suse.com

For more information, contact SUSE at:
+1 800 796 3700 (U.S./Canada)
+49 (0)911-740 53-0 (Worldwide)

Innovate Everywhere