TIGERA

Managing Application Connectivity in the Enterprise

Simplifying Policy Enforcement Across Heterogeneous Workloads

No cloud native environments exist in a vacuum, at least not for long. We live in a world in which we have to combine the new with what already exists. If not on day one, at least by day two there will be a requirement for a new cloud native environment to interact with a legacy component. The challenge is determining how to connect the old and the new in a scalable, automated and secure manner. This white paper describes a strategy for addressing this challenge.

Table of Contents

Introduction **3**

The Paradigm Impedance Mismatch **4**

Workload Lifecycles and Instance Cardinalities 4

Routers, Routers Everywhere 5

IP Addresses Are Not Identities 6

Green Fields are a Figment of Your Imagination **6**

A Policy Framework for Any Workload **7**

Mapping Workloads to Endpoints 7

Endpoint Types and Policy Rules 7

Selectively Applying Policy 9

Example: Kubernetes **9**

Protecting the Control Plane 9

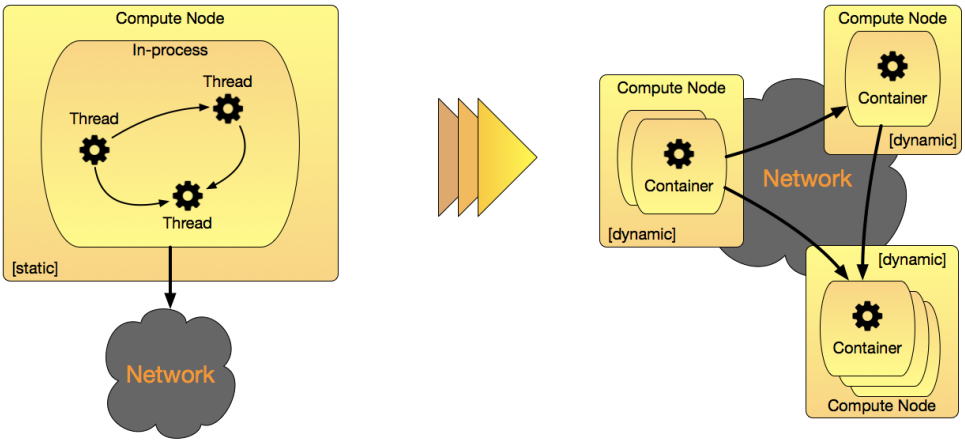
Protecting Services Exposed as Node Ports 10

Conclusion **10**

Introduction

High-profile breaches, compliance requirements, data governance rules and security best practices are motivating organizations to examine the rapid growth in east-west network traffic and search for ways to isolate all application interactions within the network. That's due in large part to the fact that there's been a rapid shift from monolithic architectures to microservices, and more specifically, containers, as the principal architectural building blocks for packaging and running modern applications. This shift has fundamentally changed the nature of network traffic:

- **Network as call boundary.** Rather than using in-process library calls, components exchange messages across the network as out-of-process workloads.
- **Ephemeral workloads.** Compared to static virtual machine (VM) and bare-metal approaches, workloads have relatively ephemeral life cycles.
- **Variability at scale.** Elastic scaling creates large volumes of workloads and interactions.



Network policies can provide an automated and scalable way to control traffic and isolate workloads for security and compliance. However,

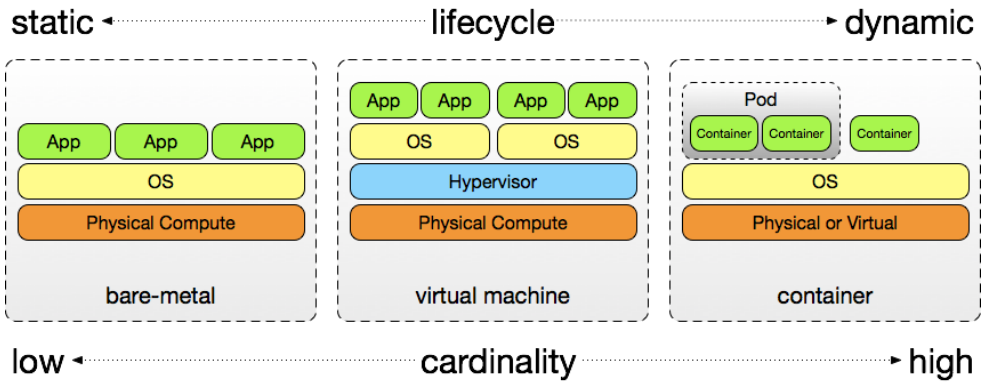
heterogeneous environments can pose challenges. How does an administrator describe and enforce policy in a consistent manner, even though workloads and interactions span different packaging, deployment and orchestration paradigms?

This white paper describes a workload abstraction and identification strategy that enables administrators to create a uniform policy framework that bridges multiple workload containment paradigms.

The Paradigm Impedance Mismatch

The never-ending drive to minimize computing costs has resulted in a progression of paradigm shifts in the workload containment arena. For applications running on general purpose computing platforms, we can distill these paradigms into three basic approaches:

- Applications installed on bare-metal
- Applications installed on VMs
- Applications packaged and deployed as containers (or pods as in the case for Kubernetes) on a bare-metal or virtual machine host



In each of these deployment scenarios, we assume that a workload accesses the network in a uniform way using standard operating system abstractions.

Workload Lifecycles and Instance Cardinalities

Each deployment model has a different lifecycle and instance cardinality (that is, the number of elements in a set or grouping).

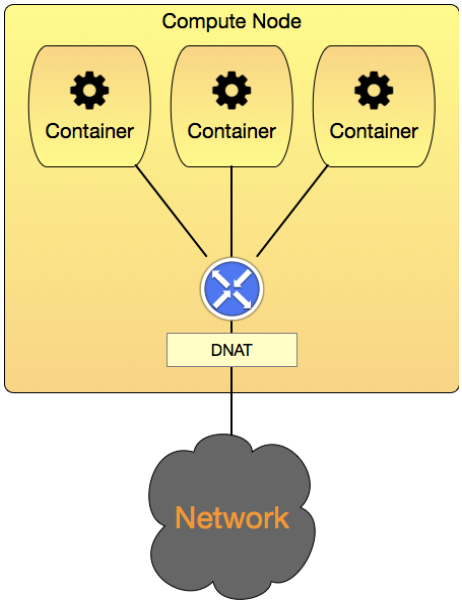
- **Bare-metal workloads** tend to be resource constrained and static in nature.
- **VMs** simplify the provisioning and deployment process enabling shorter life cycles and reduced resource usage.
- **Containers** are lightweight and typically orchestrated, introducing a dynamic lifecycle and high cardinality of instances.

A policy framework should treat these varied workloads in a consistent, uniform manner. Specifically, a framework should enable policy to be defined and applied across dynamic workloads in which endpoint coordinates or numbers of instances are not necessarily known beforehand.

Additionally, a policy framework should allow policies to be assigned in a modular way, so they can be aligned with workloads that have both large (monolith) and small (microservice) functional domain scopes.

Routers, Routers Everywhere

VMs and containers are typically hosted within other workloads and must rely on their hosts to forward traffic on their behalf. This means that every node that hosts containers or VMs is effectively a router.



Additionally, service discovery in orchestration systems can make use of a virtual IP (VIP) address for abstracting and load-balancing service calls across multiple endpoints. This can be achieved with dynamic network address translation (DNAT), which obscures the real source and destination of traffic from a policy engine. A policy enforcement implementation must provide an option to evaluate policy before or after DNAT transformations in order to understand the true source and destination of any traffic.

IP Addresses Are Not Identities

Before workloads can be deployed to bare-metal or VM servers, a lengthy provisioning process is typically required, during which an operating system must be installed and configured. To amortize this impact, patches and updates are applied directly to servers as applications and operating systems evolve, thus making them long-lived and pseudo-static devices.

Much to the chagrin of administrators who've been forced to do IP renumbering exercises as a result of data center changes, the

pseudo-static nature of servers have been used to justify employing static IP addresses as identifiers in network policy frameworks. This approach is quite common for traditional firewall appliances that serve as static barriers between network segments, but it does not lend itself well to a dynamic application environment.

IP addresses change frequently in container orchestration platforms and in environments in which VMs are treated as ephemeral instances. Consequently, unless absolutely required, a policy framework should abstain from using IP addresses in policy definitions. Instead, policy constraints should be rendered in a just-in-time manner and mapped to currently assigned workload IP addresses.

Greenfields are a Figment of Your Imagination

Most organizations have invested years in embedding business processes and other intellectual property into software systems and staff is resistant to colossal changes. Plus, even if an organization's leadership wanted to go all-in on a containment strategy, the transition would never be immediate. Greenfield deployments are therefore very rare. Consequently, a policy framework should both support an organization in its current state and help in achieving future architectural goals.

To be successful, a policy framework should provide homogeneous building blocks for constraining and isolating network traffic, while offering support for heterogeneous computing and workload containment models.

A Policy Framework for Any Workload

To apply policy to heterogeneous workloads in a homogeneous way, two components are required:

- **An endpoint**—the network interface in which policy is applied to individual workloads.

- **A label**—a multidimensional mechanism for mapping policies to endpoints.

Mapping Workloads to Endpoints

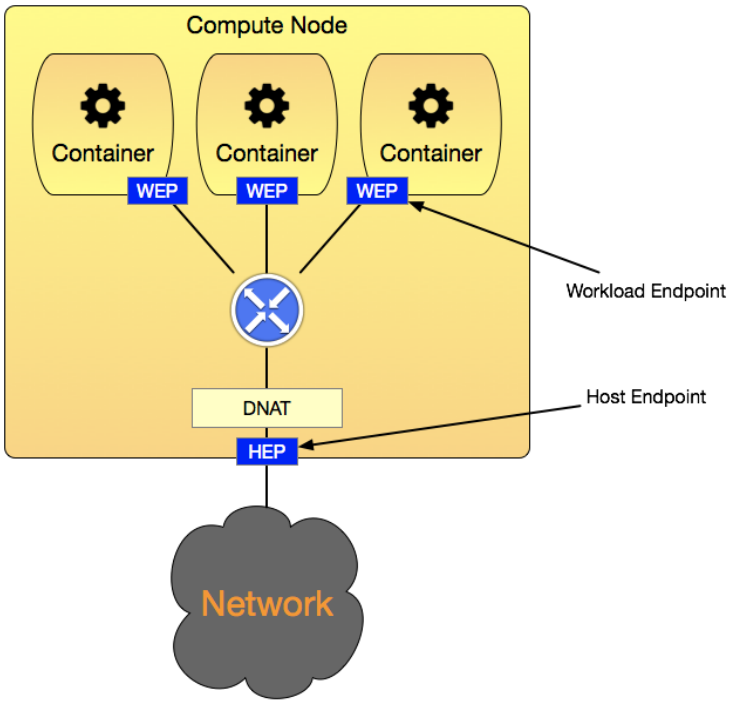
Static workloads can be created in a number of ways, including manually and automatically. In either case, a policy framework must be informed about an endpoint and instructed on how to map it to a network interface. Container workloads are created dynamically by an orchestrator. Therefore, a policy engine needs to listen to lifecycle events and dynamically map endpoints for enforcement.

Endpoint Types and Policy Rules

It is important to note that workloads hosting guest containers require special treatment so they can forward traffic. It turns out that simply distinguishing between container and non-container workloads is sufficient for implementing policy consistently in a heterogeneous workload environment.

Every workload will then contain one of these endpoint types:

- **Host endpoint**—assigned to non-containerized or orchestrated ephemeral workloads
- **Workload endpoint**—assigned to dynamically created containerized or orchestrated endpoints.



Since traffic bound for a workload endpoint transits a host endpoint, the policy enforcement engine must not allow policies defined for a host endpoint to override or interfere with workload endpoints unless specifically requested by an administrator. For host endpoints, policies can specify whether they should be applied before or after the DNAT transformation.

Finally, in most platforms, a container workload can bind directly to the network interface of the underlying host, instead of using its own network interface. In this case, policies would apply to the host as a host endpoint.

Selectively Applying Policy

To achieve modular policy assignment, labels are used to associate policies with endpoints. For workload endpoints, labels can be derived from orchestrator metadata. However, host endpoints are established

separately from the container orchestrator and must be assigned labels at creation.

By using similar labels across both types of endpoints, administrators can establish a unified policy framework for governing heterogeneous workload containment models.

Example: Kubernetes

Kubernetes, a popular container orchestrator, illustrates the utility of using labels and endpoints to apply policies.

Protecting the Control Plane

Kubernetes orchestrates the deployment of container workloads across a cluster of nodes using multiple microservices that comprise a control plane. Several control plane components, such as the kube-apiserver and kubelet, are bound to host interfaces.

These control plane components may be installed on nodes on which container workloads have also been scheduled. Every host containing control plane components can be designated as a host endpoint and policies can be deployed that protect these sensitive elements.

Kubernetes has supported network policies for several releases. As a result, users can specify policy using an API that they are already familiar with. In some cases, a network policy may require features that are not supported by the Kubernetes resource. When this happens, policies can be applied to the workload endpoints using inherited labels that are native to the orchestrator. By inheriting these labels, native policies can be applied consistently to dynamic workload endpoints and statically configured host endpoints alike.

Since both host endpoints and workload endpoints share a common policy framework and enforcement engine, container workloads can interact with control plane components and vice versa.

Protecting Services Exposed as Node Ports

Kubernetes can also map host ports to internal clusters of recognized virtual IP addresses (VIPs). To achieve this mapping, inbound traffic destined for a specific host port is first forwarded to the VIP, which is then dynamically transformed to a real workload endpoint.

A policy that protects access to VIPs or the host can include a directive to the enforcement engine that specifies whether it should be applied before or after DNAT translation.

Conclusion

Endpoint abstraction and labels enable network security policies to be applied consistently across heterogeneous workload models. This approach also simplifies policy coordination across all network-connected computing assets. The granularity and interoperability of policies defined in this framework can help organizations move closer to a zero-trust network security model in which all allowed network access is explicitly defined. This label-based approach underpins the security, control, and simplicity found in Tigera Secure. With this solution, organizations can establish secure application connectivity across multi-cloud and legacy environments.

If you are interested in understanding how to achieve secure application connectivity for your heterogeneous workloads, please contact us.

About Tigera

Tigera delivers solutions for secure application connectivity for the cloud native world. Tigera technology is used by the world's largest enterprises and public cloud providers to power connectivity for application development and deployment and to address the connectivity and security challenges that arise in at-scale production. Tigera Secure meets enterprise needs for zero trust network security, multi-cloud and legacy environment support, organizational control and compliance, and operational simplicity. Tigera Secure builds on leading open source projects Kubernetes, Calico, and Istio, which Tigera engineers help maintain and contribute to as active members of the cloud native community.

tigera.io

email: contact@tigera.io

phone: +1.415.612.9546

Tigera, Inc. 58 Maiden Lane, Fifth Floor, San Francisco CA 94018 USA