tetrate

Tetrate Service Bridge (TSB) Application Security Architecture

An overview of the security guarantees Tetrate Service Bridge provides to your applications running on the mesh and how.

Table of Contents

Executive Summary

Many enterprises and government agencies strive to achieve a zero trust architecture, recognizing that traditional perimeter-based security is no longer sufficient in today's threat landscape. The zero trust mindset acknowledges that attackers could potentially infiltrate any network. Therefore, the focus shifts to containing and mitigating attacks both spatially and temporally. To achieve this at runtime, you need to be performing at minimum five checks on every hop between components in your infrastructure:

1. Encryption in transit
2. Service authentication
3. Service to service authorization
4. End-user authentication
5. End-user to resource authorization

In addition to these runtime checks, we need the ability to monitor the system continuously to ensure policy is being enforced and to respond to changes on demand by updating policy.

Tetrate Service Bridge (TSB) uses the Istio service mesh to manage your application's traffic. As a dedicated infrastructure layer, the service mesh is an invaluable security tool for modern applications. The service mesh's sidecar intercepts all traffic in and out of your applications, where it acts as a universal policy enforcement point. This allows the service mesh - which centrally manages a fleet of your applications' sidecars - to become the modern cloud native security kernel (NIST SP 800-204B).

The sidecar is able to enforce security and traffic policies, as well as generate telemetry to allow operators to close the loop on policy changes: they can author a change, observe its effect on the runtime and make additional changes as needed - all in a real time feedback control loop. In other words, the mesh provides the capabilities to implement the runtime controls needed to achieve a zero trust posture.

Policy That TSB Can Enforce for Applications

Using the service mesh, Tetrate Service Bridge is able to provide the following capabilities to applications:

- **Encryption in transit.**
  The mesh manages and rotates short-lived application identity certificates.
- **Service-to-service authorization.**
  The mesh's identity certificates can be used to implement authorization policies at runtime, per request (rather than per connection).
- **Web application firewall (WAF), API gateway, egress and other "perimeter" policies.**
  The mesh can implement a variety of controls inside your system that we traditionally only enforce at the perimeter, for example: WAF controls for payload inspection, end-user authentication, per-user rate limiting/throttling and controlling outbound traffic to external systems.
- **Custom policies.**
  The mesh exposes a variety of extension points - including in-process in the proxy, via proxy extensions and via APIs - for using the sidecar to enforce custom policy decisions. These are used to integrate with your organization's existing policy systems.
- **Application traffic flow.**
  The mesh can provide client-side load balancing, retries, timeouts, circuit breaking and a variety of other tools for increasing system resiliency and availability.
- **Logs, traces, and metrics.**
  Detailed telemetry gives us confidence that policies are deployed and being enforced at runtime.

These policies can be enforced at both ingress and egress - implementing traditional north-south controls, as well as controls for east-west communication - to establish a modern, zero trust security stance.

How TSB Achieves Enforcement

TSB enforces the above policies at runtime in a reliable, available way by implementing a tree-shaped architecture composed of three layers:

- **Management plane.** TSB's management plane is the root of the tree, where your stakeholders - application developers, platform, security, networking and operations owners - configure and observe the mesh.
- **Control plane.** TSB delivers both a global control plane, which is responsible for cross-cluster service discovery and configuration distribution and a local control plane, which is responsible for programming an upstream, OSS Istio control plane.
- **Data plane.** Deployed as sidecars and gateways, Envoy acts as the mesh's data plane, handling the bits-and-bytes of your application traffic. Envoy can be deployed in many modes: as an ingress or egress proxy, traditional reverse load balancer, or as a sidecar. TSB programs Envoy to act as any of the three you need, by configuring the Istio control plane that manages that Envoy.

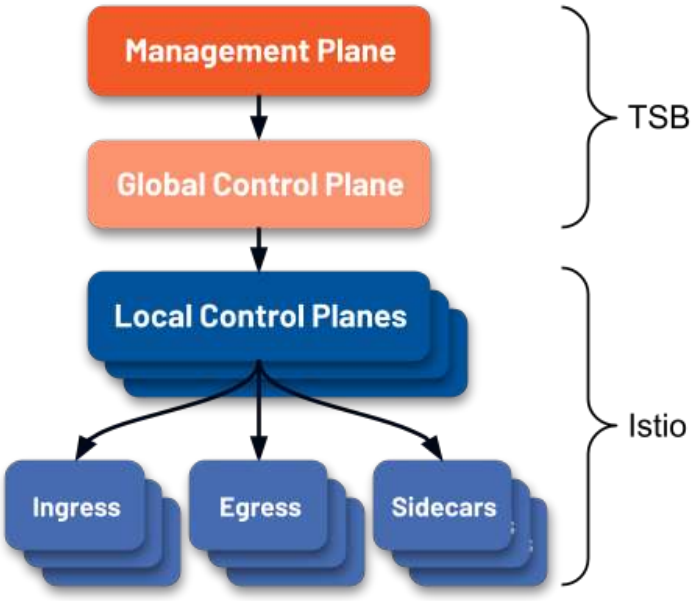

Fig 1. Tetrate Service Bridge provides a management plane and global control plane, layered on top of OSS Istio's control plane and Envoy data plane.

Tetrate Service Bridge also enables management of the mesh's capabilities: it lets you manage who in your organization can change which mesh configurations affecting what physical infrastructure. In other words, it allows safe multi-tenant usage of the service mesh: your central teams can mandate policy while delegating configuration to application teams - who can't configure the system outside of the bounds they've been given. These management guardrails offer safety and agility, transforming the platform and security teams from a bottleneck into an agility booster for the organization. On top of the runtime policies that the mesh can enforce for your applications, TSB provides a variety of management capabilities to make the mesh useful in a complex enterprise:

- **Hierarchical identity and access management:** bring your existing identity provider, assign your physical infrastructure into a hierarchy, then collaboratively enforce controls across the entire infrastructure with policy defaults, bounds, and delegation.

- **Default configurations and controls:** set configuration defaults for all types of configuration on your hierarchy, with the ability for teams lower in the hierarchy to replace with only stricter policy. Enforce permissive policies in test environments but strict policies in production.
- **Visibility and operability:** view your application topology and understand how traffic flows - at the application level, across your entire infrastructure. Assess application health with mesh operational metrics and bootstrap availability measures in your organization with objective measures like ApDex.
- **Audit and traceability:** see what changed in the system when and visualize the system's runtime behavior before and after. Every event in the system is recorded and this audit log can be exported to other systems. TSB's visibility closes the loop on policy change to runtime effect.

Where TSB Fits in Your Infrastructure

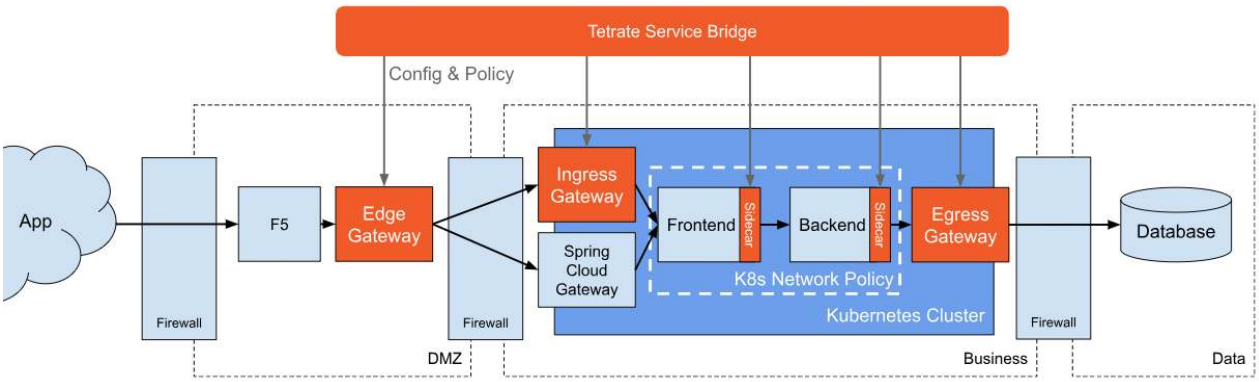

Fig 2. Tetrate Service Bridge manages a set of Istio control planes deployed across your infrastructure. In turn, those Istio instances manage a set of Envoy proxies deployed as Edge Gateways, Ingress Gateways, Sidecars, and Egress Gateways.

Tetrate Service Bridge sits atop your existing network, abstracting application traffic flow from the underlying network complexity. Applications consume each other using DNS names, while the mesh - programmed by TSB - routes application traffic and applies policies configured by your platform, security, and traffic management teams.

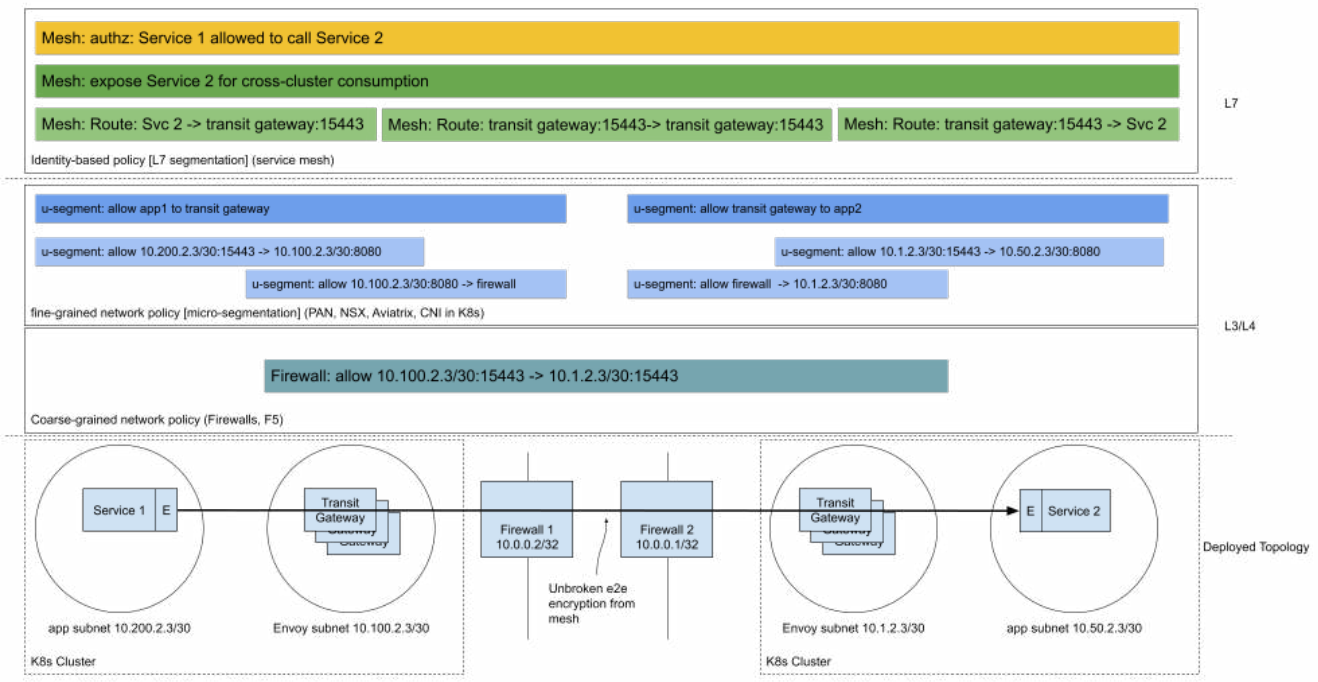

Fig 3. To facilitate communication between applications across a DMZ – e.g. from Cloud to an on-prem application – you need policy at several layers. TSB's policy sits at L7, cutting across the infrastructure, while lower policy layers need many policies in each part of the infrastructure to facilitate communication.

From a policy perspective, Tetrate Service Bridge sits at Layer 7, allowing you to author access policy in terms of service identities rather than IP subnets or network labeling. This is powerful in a few ways:

- **Write once, enforce everywhere.** It means policy can span environments and topologies - we can write a policy once and enforce it everywhere, rather than bespoke policies per environment.
- **Human readable primitives.** The policy we write uses human-understandable primitives ("service A can call service B") not network oriented primitives ("10.1.2.3/30 is allowed to call 10.100.2.3/30 on port 8080"). This context is critical - in our experience, the lack of context for rules is a key reason for lack of agility around traditional network policy today.
- **Contextual intent codified in a single policy.** There is a single policy, not a set of policies that need to be pieced together to understand intent. A human can read a policy like "the frontend service is allowed to call 'GET /foo' on the backend service" and understand the access the policy intends to convey - even if, for example, the frontend is deployed in the cloud and the backend on prem. It's significantly harder to read a set of network peering and firewall rules allowing communication across the DMZ for a set of subnets and understand the access the policy intends to convey. In turn, this means it's harder to write the wrong policy and easier for a human to understand when a policy isn't correct.

Practical Example: Simplifying Perimeter Network Policy

Tetrate Service Bridge provides opportunities to simplify traditional network-oriented security policies - especially in cases like bridging application topologies that span across on-prem and cloud.

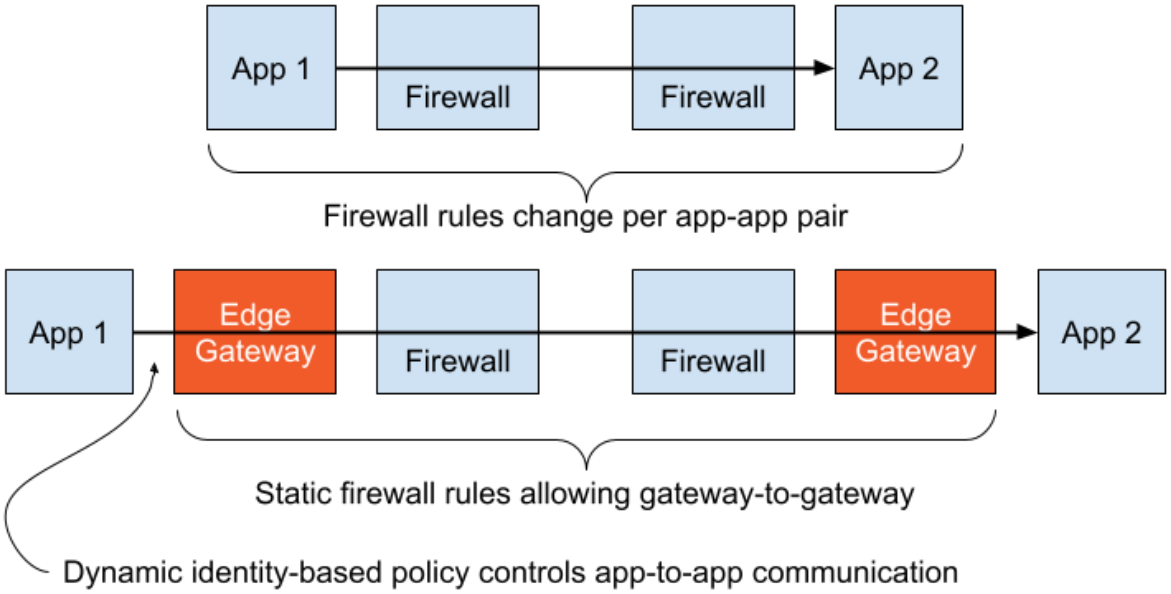

Fig 4. Comparison of connectivity across firewalls for traditional app-to-app communication, compared to using TSB's Edge Gateways to bridge the connection and enable static network policy.

Even with the push for zero trust architectures, given the understanding of regulators today, network oriented policy will not disappear soon. Therefore, TSB's policy fits into a defense in depth strategy where lower level network controls can be loosened but not entirely removed, while being augmented by more expressive TSB L7 policy.

For example, TSB can deploy Envoy instances that act as "transit gateways," dramatically simplify underlying network policy management by enabling teams to consume cloud services on prem and as well as enabling applications in the cloud to call back to the data center. Rather than a set of firewall rules per application on prem that needs to consume a service in the cloud (or vice versa), we can create a single static set of rules allowing communication between pools of Envoy proxies - what we call Edge Gateways in TSB. Then, we can manage identity-oriented policies to control which applications can communicate over those edge gateways.

System Layers

Tetrate Service Bridge is made up of three layers:

- **Management plane.** TSB's management plane is the root of the tree, where your stakeholders - application developers, platform, security, networking and operations owners - configure and observe the mesh.
- **Control plane.** TSB delivers both a global control plane, which is responsible for cross-cluster service discovery and configuration distribution and a local control plane, which is responsible for programming an upstream, OSS Istio control plane.
- **Data plane.** Deployed as sidecars and gateways, Envoy acts as the mesh's data plane, handling the bits-and-bytes of your application traffic. Envoy can be deployed in many modes: as an ingress or egress proxy, traditional reverse load balancer, or as a sidecar. TSB programs Envoy to act as any of the three you need, by configuring the Istio control plane that manages that Envoy.

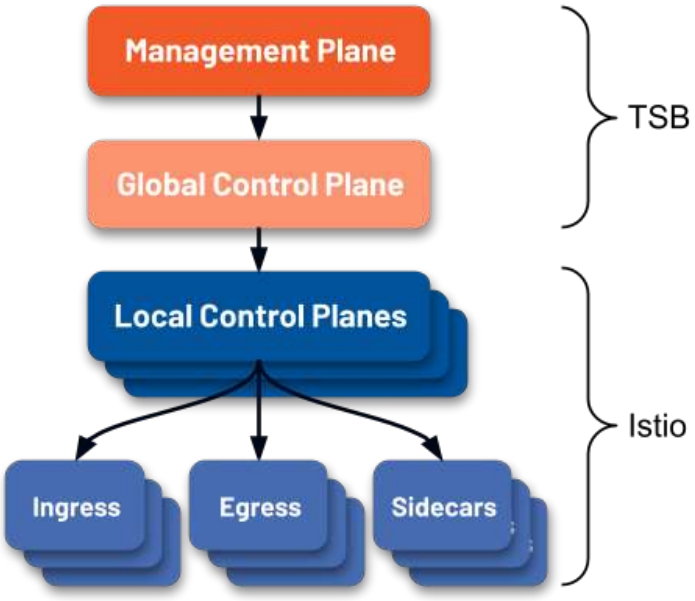

Fig 5. Tetrate Service Bridge provides a management plane and global control plane, layered on top of OSS Istio's control plane and Envoy data plane.

This structure ensures that the service mesh scales and performs well from small to large deployments and at any level of traffic. Further, by persisting configuration beside each local control plane and programming Istio to minimize updates needed for cross-cluster communication, Tetrate Service Bridge maintains a highly available and reliable system.

The CNCF Security TAG's Security Whitepaper defines the variety Cloud Native Layers that make up an application: the tooling to build, deploy and manage application lifecycle; the environment and runtime the application is executing; the compute and storage resources the application consumes; and the access layer that controls how the application is composed and consumed.

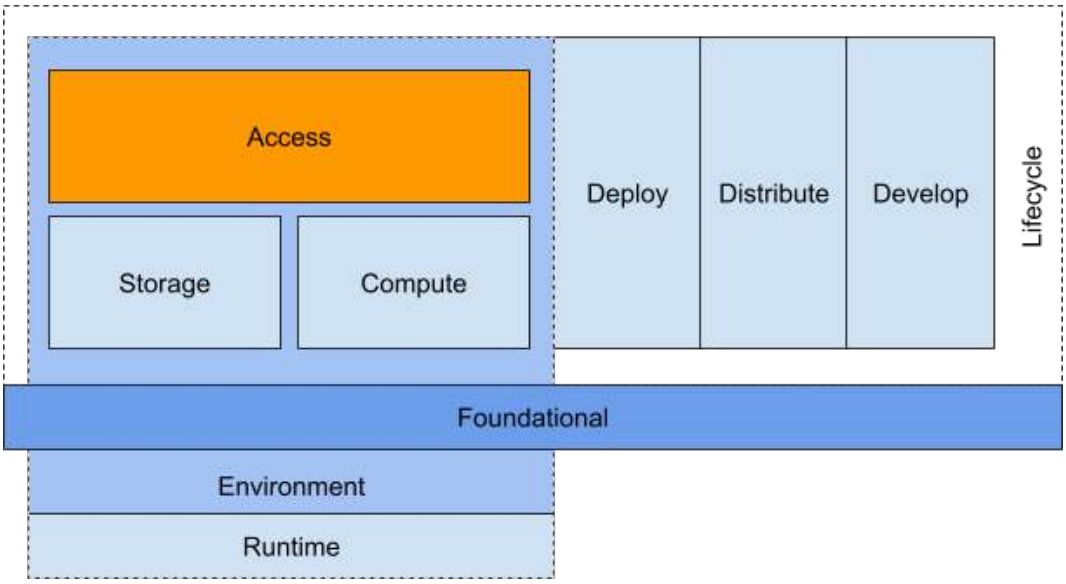

Fig 6. The Cloud Native Layers, Cloud Native Security Whitepaper, the CNCF Security TAG. The service mesh helps implement the Access Layer.

The service mesh sits atop foundational services like the network and hardware/cloud provider load balancers to facilitate application connectivity: it implements the access layer in the CNCF's architecture. Tetrate Service Bridge, as a management plane on top of a set of Istio control plane instances, does not directly make runtime decisions but sits to the side as part of the Deploy, Distribute, Develop triad for managing and configuring applications' behavior.

Runtime

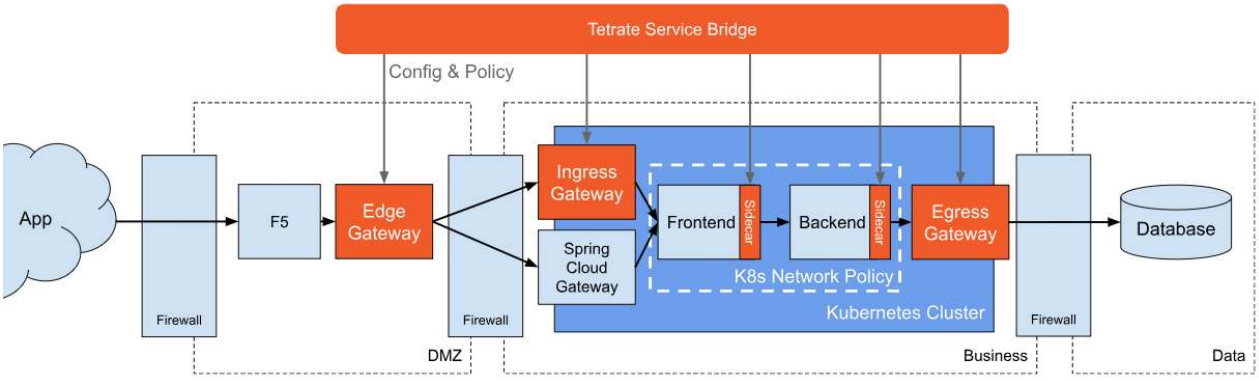

Fig 7. User traffic from an application (e.g. mobile app or website) traversing the DMZ to reach a typical three-tiered application. Tetrate Service Bridge can be used to apply policy in many locations, which are highlighted in orange: at an Edge Gateway, Ingress Gateway, Sidecars, and Egress Gateway.

To act as the access layer, the service mesh deploys proxies that act as policy enforcement points at a variety of locations in the infrastructure:

- As a sidecar, beside each application instance intercepting all traffic into and out of the application, handling “east-west” internal communication between services in your infrastructure. This is the primary use case we think of when we think of the service mesh.
- As an ingress gateway, controlling how applications in your cluster are exposed outside: managing what names, certificates, ports, protocols, and application endpoints are served to the world outside your cluster. Think of this as the service mesh managing a traditional reverse proxy similar to Spring Cloud Gateway, NGINX, or HAProxy.
- As an egress gateway, controlling how applications in your cluster communicate with the outside world. This can be used for traditional egress filtering and logging like a Squid proxy, but can also implement identity based policy for what is allowed to call out, as well as perform credential exchange - presenting a set of credentials (like an mTLS certificate for a partner API) on behalf of the application so the application doesn’t need to handle them (e.g. the application does not need to handle communicating via mTLS with the partner API). Think of it as a next generation, identity aware Squid proxy.
- As an edge gateway, accepting external traffic before the ingress gateway and performing fine-grained load balancing across clusters or sites. It’s used to terminate external traffic, enable infrastructure level failover, blue-green cluster deploys and facilitate ingress-gateway-per-team deployments without requiring each of those teams to have publicly routable ingress gateways. Think of it as a modern software based local traffic manager like F5, that can apply per-request policy rather than per-connection.

Policy

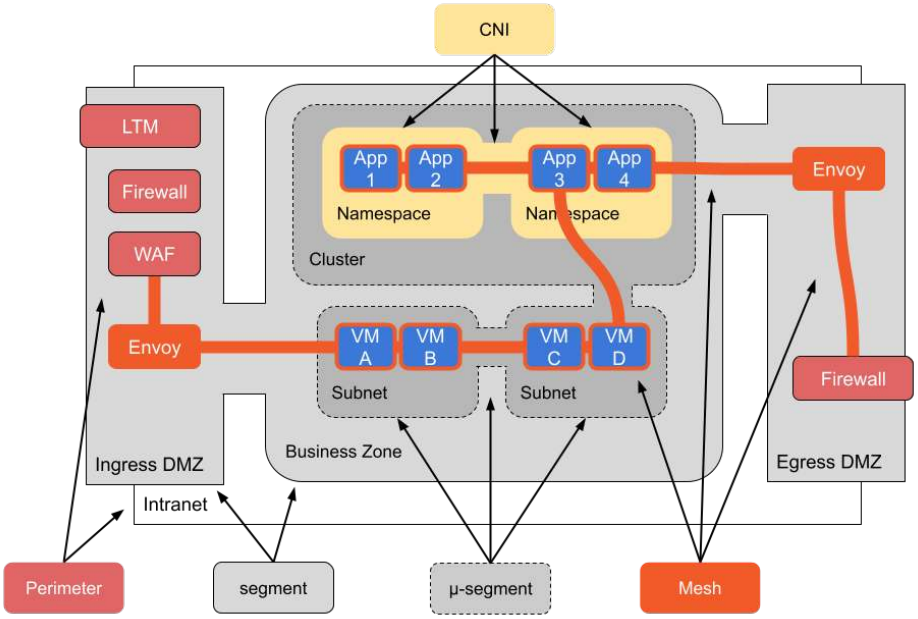

Fig 8. Tetrate Service Bridge's policy sits on top of your existing network policies – whether that's perimeter controls like firewalls and WAFs, (micro-)segmentation like NSX or VPCs, or cloud native controls like CNI. By implementing portable, identity oriented controls with TSB, you can loosen controls at lower layers, gaining agility.

Finally, from a policy lens TSB sits at the application layer. It provides short lived cryptographically authenticatable runtime identities for services (implementing SPIFFE), and allows you to author policy about service access using those identities. This means we're writing policy in terms of things developers can readily reason about - services - not subnets or network labels. This means:

- The policy we write uses human-understandable primitives, not network oriented primitives. This context is critical - in our experience, the lack of context for rules is a key reason for lack of agility around traditional network policy today.
- Policy spans environments. We can write it once and apply it globally. We don't need to specialize per data center or region or subnet or anything else.
- There is a single policy, not a set of policies that need to be pieced together to understand intent. This means less opportunity for error in the first place and it's easier to correct when there is a mistake or stale policy.

This also means that TSB sits on top of the network policy you already have - whether it's coarse grained perimeter controls or fine grained microsegmentation via NSX or CNIs. To help understand how TSB's policies layer on top of existing network policies and allow you to loosen network oriented controls in exchange for identity based controls enforced by the mesh, we find it's easiest to work through a real world example.

Practical Example: Bridging Cloud and Data Centers

An incredibly common pattern we see with customers is a deployment across a set of on-prem data centers and one or more public clouds. All traffic across environments must traverse a "network demilitarized zone" (network DMZ) - usually implemented with firewalls. Changing these firewall rules tends to take days to weeks. Meanwhile, the move to cloud and increased usage of SaaS means that firewall updates need to happen more and more frequently. We see these perimeter policy updates slowing application developer agility.

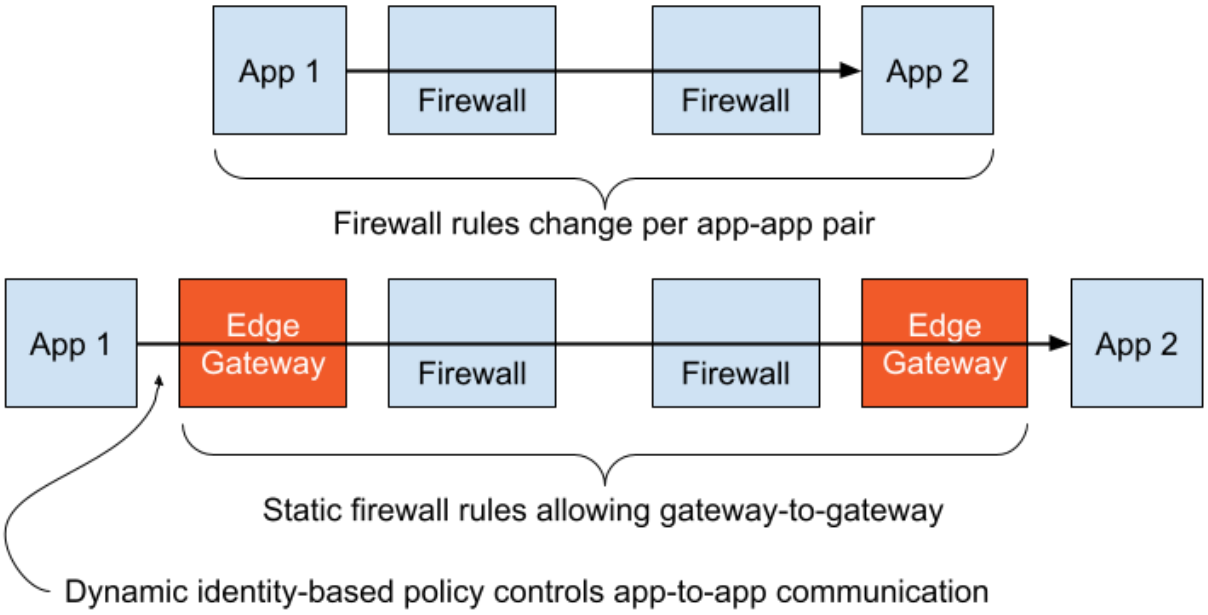

Fig 9.

A common pattern we implement with Tetrate Service Bridge is to deploy a set of edge gateways in each environment: in each cloud and in on prem data centers. We create a static set of firewall rules allowing communication between these edge gateways. Using the service mesh, we effectively "tunnel" application communication between environments via these edge gateways - transparently to the application. The edge gateways enforce access policies based on application identity to control which applications can communicate over the "tunnel". We say "tunnel" throughout (in quotes) because the connection does not look like a traditional VPN-based tunneling system (though this pattern can be deployed directly on top of a VPN-based connection between sites).

Let's look at the set of policies in play at each layer in the stack when we implement the connectivity pattern we describe above:

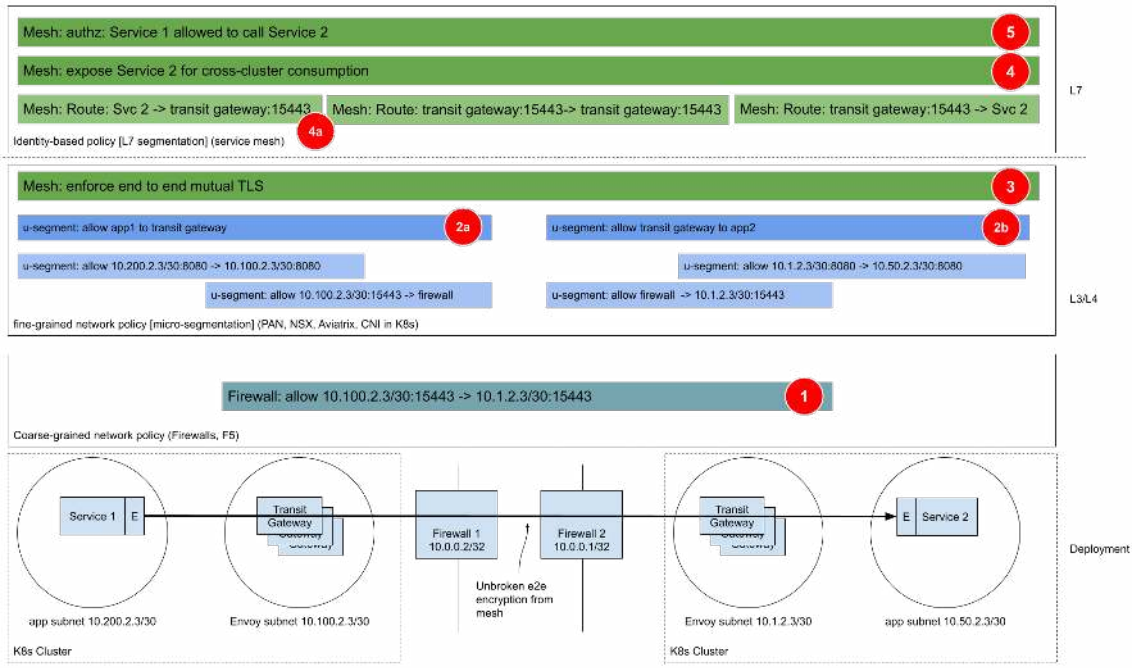

Fig 10. Transit GW policy example

First, let's understand the setup in more depth. At the bottom part our diagram:

- Our topology is two Kubernetes clusters deployed in separate networks. Each network has an inbound and outbound firewall.
- Applications are deployed into subnets in each cluster. Additionally, there's an edge gateway deployed in each cluster, in a separate subnet from the application.
- Each application is deployed with a sidecar.
- We want to enable the application Service 1 in one cluster (which we'll call the local cluster) to communicate with the application Service 2 in the other cluster (which we'll call the remote cluster).

Now, let's look at the policy, working from network-oriented controls at Layer 3 and Layer 4 up to application controls at Layer 7.

Perimeter Controls

First, we have traditional perimeter controls implemented via a coarse-grained firewall: policy allowing communication from the edge gateway in the local cluster to the edge gateway in the remote cluster. This is labeled (1) in Fig 10. That policy looks something like (using the addresses from Fig x):

```
Allow 10.100.2.3/30 to 10.1.2.3/30
via TCP from source port 15443 to destination port 1544
```

Fig 11.

Normally we'd need to create a set of rules like this every time there was a set of applications that needed to communicate (e.g. for a new microservice in the cloud to call an existing database on prem). Here, we have a static set of rules that we configure once when the edge gateways are deployed.

Note that in our example we're deploying a set of edge gateways in the application cluster - this is to simplify the example so we don't need to deal with complex networking policy from application to edge gateway to demonstrate the example. However, in a production deployment the edge gateways are typically a shared deployment we set up once for the organization.

Microsegmentation

Next, we have more fine-grained network policy - what we'd call microsegmentation. That's the policy that defines the subnets for the applications and the edge gateways and configures communication between them. The tightest set of policies we can define at this layer mean: "Service 1 is allowed to call the Edge Gateway and the Edge Gateway is allowed to call the outbound firewall. Additionally, the remote Inbound firewall is allowed to forward traffic to the remote Edge Gateway and the remote Edge Gateway is allowed to forward traffic." That's at least four rules across two sites to allow our applications to communicate, labeled (2a) and (2b) in Fig x, which look something like:

```
In local data center:

   Allow 10.200.2.3/30 to 10.100.2.3/30
   via TCP from source port 8080 to destination port 8080
   Allow 10.100.2.3/30 to 10.0.0.2/32
   via TCP from source port 15443 to destination port 15443

In remote data center:

   Allow 10.0.0.1/32 to 10.1.2.3/30
   via TCP from source port 15443 to destination port 15443
   Allow 10.1.2.3/30 to 10.50.2.3/30
   via TCP from source port 8080 to destination port 8080
```

Fig 12.

Two of these rules - the edge gateway to firewall and firewall to edge gateway are static. When we want to allow communication between a new set of applications, we'll need to update one to two of the other rules: the first to allow the new client application to communicate with the edge gateway and a second to allow the remote edge gateway to communicate with the server. Of course, with so many more rules we lean on automation - traditional software-defined networks like NSX or modern ones like Aviatrix, or cloud native ones like CNI to help us manage these microsegments.

It's harder to read this policy and understand its intent than the text description we gave of the policy that references the applications' names. In turn, that makes it harder to manage - for example, can I delete an "outdated" rule without causing an outage? This lower understandability leads to lower confidence that a policy is correct - this is a root cause for policy updates at this layer being slow.

Mesh Controls

Finally, we have three policies at the mesh layer. One is static, one is authored once by the service owner, and the last is updated every time a new client wants to consume a service.

Static Policy for mutual TLS. First we have a static policy that we create as part of installing the mesh: require mutual TLS between applications. This is policy (3) in Fig x. That policy looks something like:

```
authenticationSettings:
 trafficMode: REQUIRED
```

Fig 13. A TSB SecuritySettings which sets mutual TLS to required. Applying this configuration at TSB's root Organization enforces mTLS across all applications in the mesh.

Service Owner policy to expose Service. Second, we have a policy managed by the service owner exposing their service for consumption across clusters. This is policy (4) in Fig x:

```
TODO
```

Fig 14.

Note that there are sub-policies (labeled (4a) in Fig x) that are required to implement the high level policy. Tetrate Service Bridge manages the lifecycle of those sub-policies automatically based on the lifecycle of the main policy. Ultimately, those sub-policies manifest as Istio VirtualServices, DestinationRules and AuthorizationPolicies in the local and remote clusters.

Per-client Authorization Policy. Finally, we have a service level authorization policy that the service owner manages which is updated every time there's a new client:

```
  authorization:
     mode: CUSTOM
     serviceAccounts:
     # specific service account
     - "frontend/web-frontend-serviceaccount"
     # any service account in the namespace
     - "ns2/*"
+    # our new caller
+    - "some-client/new-client-serviceaccount"
```

Fig 15. Existing authorization policy that we update to include a new client caller.

Enabling A Virtuous Cycle of Continuous Security Posture Improvement

One of the most important takeaways here is that the policy we author via TSB - the application level policy - is much easier for a human to understand. Understandability leads to confidence that the policy is correct, which in turn gives us confidence to update the policy more quickly. Then TSB lets us close the loop: we can observe the traffic in our system in real time as the policy is deployed to ensure it's enforcing what we intend and update our policy again quickly if it's not.

Using TSB to implement this edge gateway connectivity model allows us to unblock the organizational bottleneck caused by maintaining a perimeter security posture. By loosening lower level network oriented policy in exchange for tighter, higher level identity based policy we can enable confidence in policy update and enhance policy maintainability. The net result is unlocking developer agility by making it easy to consume services across environments as well as consume SaaS offerings, while moving your organization to a better policy footing with tighter controls and more understandable policy.

About Tetrate

Rooted in open source, Tetrate was founded to solve the application networking and security challenges created by modern computing so enterprises can innovate with speed and safety in hybrid and multi-cloud environments. As applications evolve into collections of decentralized microservices, monitoring and managing the network communications and security among those myriad services becomes challenging. This is why some of the largest financial institutions, governments and other enterprises rely on Tetrate to deliver modern application networking and security on a foundation of Zero Trust. **Find out more at www.tetrate.io**

Tetrate Academy

Accelerate your service mesh journey with expertly curated, hands-on training courses from the co-creators of open source Istio and Envoy. Private training for enterprise customers available upon request. **Learn more at academy.tetrate.io**