Using Istio to Externalize Authentication and Authorization

Analysis prepared by \_\_\_\_\_

Table of Contents

[1.0.0 Goals 4](#_Toc43113534)

[1.1.0 Externalize Authentication and Authorization 4](#_Toc43113535)

[1.2.0 Support Multiple Identity Providers 4](#_Toc43113536)

[1.3.0 Service-to-Service Communication may be Restricted 4](#_Toc43113537)

[1.4.0 Public APIs Should Be Restricted for Fair-Use 4](#_Toc43113538)

[1.5.0 Support Routing to Partitioned Services by Request Header 4](#_Toc43113539)

[2.0.0 Current State of Istio 5](#_Toc43113540)

[2.1.0 How stable is Istio? 5](#_Toc43113541)

[2.2.0 Is Istio performant? 5](#_Toc43113542)

[2.3.0 Is Istio scalable? 5](#_Toc43113543)

[2.4.0 How complex is Istio? 5](#_Toc43113544)

[2.5.0 Does Istio work with AKS? 5](#_Toc43113545)

[2.6.0 Can Azure Application Gateway front Istio services in AKS? 6](#_Toc43113546)

[2.7.0 Can Azure API Management front Istio services in AKS? 6](#_Toc43113547)

[3.0.0 Architecture 6](#_Toc43113548)

[3.1.0 Azure Kubernetes Service (AKS) 6](#_Toc43113549)

[3.2.0 Azure VNETs and Subnets 6](#_Toc43113550)

[3.3.0 Namespaces 7](#_Toc43113551)

[3.4.0 Routing 7](#_Toc43113552)

[3.5.0 Design Patterns 7](#_Toc43113553)

[3.5.1 Auth at the Istio Gateway 8](#_Toc43113554)

[3.5.2 Auth at Services 10](#_Toc43113555)

[3.5.3 Single API Service 12](#_Toc43113556)

[4.0.0 Configuration 13](#_Toc43113557)

[5.0.0 Installation 14](#_Toc43113558)

[5.1.0 Generate Manifest 14](#_Toc43113559)

[5.2.0 Internal Load Balancer 15](#_Toc43113560)

[5.3.0 Peered VNET 15](#_Toc43113561)

[5.4.0 Azure Application Gateway 16](#_Toc43113562)

[5.4.1 Application Gateway Ingress Controller (AGIC) 16](#_Toc43113563)

[5.5.0 Azure API Management 17](#_Toc43113564)

[5.5.1 API – APIM Frontend 18](#_Toc43113565)

[5.5.2 Health - APIM Frontend 19](#_Toc43113566)

[5.5.3 APIM Policies 20](#_Toc43113567)

[5.6.0 Istio Gateway 21](#_Toc43113568)

[5.7.0 Sidecars 22](#_Toc43113569)

[5.8.0 Microservices 23](#_Toc43113570)

[5.9.0 Istio Components 24](#_Toc43113571)

[5.10.0 Authentication and Authorization 25](#_Toc43113572)

[6.0.0 Performance Tests 26](#_Toc43113573)

[6.1.0 Performance Testing with go-wrk 27](#_Toc43113574)

[6.2.0 Performance Testing with httperf 28](#_Toc43113575)

[6.3.0 Performance Testing from Istio 28](#_Toc43113576)

[7.0.0 Authentication 28](#_Toc43113577)

[7.1.0 User Authentication via OpenID Connect 29](#_Toc43113578)

[7.2.0 User Authorization using JWT 30](#_Toc43113579)

[7.2.1 Authorization using Claims 31](#_Toc43113580)

[7.2.2 Authorization using Multiple IDPs 32](#_Toc43113581)

[7.3.0 Anonymous Access for Health Endpoints 33](#_Toc43113582)

[7.4.0 Require Mutual-TLS 33](#_Toc43113583)

[7.5.0 Client Authorization 34](#_Toc43113584)

[7.6.0 Other Scenarios 35](#_Toc43113585)

[7.7.0 Apply to Pods 36](#_Toc43113586)

[7.8.0 Namespaces for Public Services 36](#_Toc43113587)

[7.9.0 Debugging 36](#_Toc43113588)

[8.0.0 Throttling 37](#_Toc43113589)

[9.0.0 Partitioning 37](#_Toc43113590)

[9.1.0 Route by DestinationRule 38](#_Toc43113591)

[9.2.0 Route by Service 40](#_Toc43113592)

[10.0.0 Distribution of Services across Cloud Providers 41](#_Toc43113593)

[10.1.0 Distribution using Istio 41](#_Toc43113594)

[10.2.0 Route without Istio 42](#_Toc43113595)

[11.0.0 Questions and Concerns 42](#_Toc43113596)

[11.1.0 Network hairpins in inter-service communication could make latency go very high. 42](#_Toc43113597)

[11.2.0 Should patterns like retry, circuit breaker, backpressure, etc. be done in each service or the mesh? 42](#_Toc43113598)

[11.3.0 How do we hook into Istio for telemetry, monitoring, etc.? 43](#_Toc43113599)

[A.0.0 Performance Testing Results with httperf 44](#_Toc43113600)

[A.1.0 S2S without Istio Latency Test 44](#_Toc43113601)

[A.2.0 S2S with Istio Latency Test 45](#_Toc43113602)

[A.3.0 AppGw + WAF + Istio Latency Test 46](#_Toc43113603)

[A.3.1 AppGw + WAF + Istio Latency Test (Run 1) 46](#_Toc43113604)

[A.3.2 AppGw + WAF + Istio Latency Test (Run 2) 47](#_Toc43113605)

[A.3.3 AppGw + WAF + Istio Latency Test (Run 3) 48](#_Toc43113606)

[A.3.4 AppGw + WAF + Istio Latency Test (Run 4) 49](#_Toc43113607)

[A.3.5 AppGw + WAF + Istio Latency Test (Run 5) 50](#_Toc43113608)

[A.4.0 AppGw + WAF + SSL + Istio Latency Test 51](#_Toc43113609)

[A.4.1 AppGw + WAF + SSL + Istio Latency Test (Run 1) 51](#_Toc43113610)

[A.4.2 AppGw + WAF + SSL + Istio Latency Test (Run 2) 52](#_Toc43113611)

[A.4.3 AppGw + WAF + SSL + Istio Latency Test (Run 3) 53](#_Toc43113612)

[A.4.4 AppGw + WAF + SSL + Istio Latency Test (Run 4) 54](#_Toc43113613)

[A.4.5 AppGw + WAF + SSL + Istio Latency Test (Run 5) 55](#_Toc43113614)

[A.4.6 AppGw + WAF + SSL + Istio Latency Test (Run 6) 56](#_Toc43113615)

[B.0.0 Performance Testing Results with go-wrk 57](#_Toc43113616)

[B.1.0 S2S without Istio 57](#_Toc43113617)

[B.2.0 S2S with Istio 57](#_Toc43113618)

[B.3.0 S2S with Istio + mTLS 57](#_Toc43113619)

[B.3.0 AppGw + WAF 58](#_Toc43113620)

[B.4.0 AppGw + WAF + SSL 58](#_Toc43113621)

[B.5.0 AppGw + WAF + SSL + JWT 59](#_Toc43113622)

[B.6.0 S2S with Istio + JWT 60](#_Toc43113623)

[B.7.0 AppGw + SSL 61](#_Toc43113624)

[B.8.0 S2S with Istio + mTLS + filter 61](#_Toc43113625)

[B.9.0 AppGw + WAF + SSL + APIM + JWT 62](#_Toc43113626)

# Goals

This document is evidence of a spike evaluating the effectiveness of using Istio in Azure Kubernetes Service (AKS) to meet the following requirements:

## 1.1.0 Externalize Authentication and Authorization

The solution may have hundreds of microservices written across a wide range of languages and frameworks. Externalizing the authentication and authorization into a service mesh solution (instead of the service code) allows the requirements to be met via configuration instead of code. This affords the following benefits:

* Requirements can be implemented much faster.
* Requirements can be implemented consistently – there are no differences based on language or framework.
* It is easier to review security – requires knowledge of only one platform instead of many.

## 1.2.0 Support Multiple Identity Providers

It is possible that different services in the solution might use different IDPs. It is also possible that a single service might support authentication from multiple IDPs. The following addition requirements apply:

* It is required that a ***running*** service be able to support multiple IDPs.
* The IDPs will provide a JWT identity or access token using the OpenID Connect specification.
* Different IDPs will produce tokens with different claims.

For performance sake, it is desirable if the solution could investigate the issuer of a JWT token to determine the IDP before fetching public certificates and before validating the signature. In other words, it is less desirable to simply chain multiple authentication attempts which might negatively impact performance.

## 1.3.0 Service-to-Service Communication may be Restricted

Some services may be restricted so that only certain other services are allowed to send requests to them.

## 1.4.0 Public APIs Should Be Restricted for Fair-Use

Throttling the number of requests per second to public APIs should ensure that no one user can monopolize the system.

## 1.5.0 Support Routing to Partitioned Services by Request Header

Some services are deployed with one or more instances per customer. It is desirable to address those services using the same DNS name but to supply a header that can be used to route to the correct customer Pod.

# 2.0.0 Current State of Istio

This document addresses how to meet the requirements using Istio on Azure Kubernetes Service. Before getting to the implementation on Azure, there are some general questions regarding whether Istio itself is a suitable product to meet the requirements.

## 2.1.0 How stable is Istio?

Istio’s current version is 1.6.1. There have been huge design changes with 1.4, 1.5, and 1.6 within the past 8 months. These changes could largely be categorized as simplifying the solution, which it has done, but it is fair to call Istio a work-in-progress.

In particular, authentication and authorization have changed very dramatically over the past several versions. Those features are currently in beta: <https://istio.io/about/feature-stages/#security-and-policy-enforcement>.

## 2.2.0 Is Istio performant?

From <https://istio.io/pt-br/docs/ops/deployment/performance-and-scalability/>, for version 1.6.1:

The Istio load tests mesh consists of 1000 services and 2000 sidecars with 70,000 mesh-wide requests per second. After running the tests using Istio 1.6.1, we get the following results:

* The Envoy proxy uses 0.5 vCPU and 50 MB memory per 1000 requests per second going through the proxy.
* The istio-telemetry service uses 0.6 vCPU per 1000 mesh-wide requests per second.
* Pilot uses 1 vCPU and 1.5 GB of memory.
* The Envoy proxy adds 6.3 ms to the 90th percentile latency.

Performance testing was also conducted for this assessment and results are available in this document. The Istio overhead is far less than using AppGw for WAF.

## 2.3.0 Is Istio scalable?

As shown above, Istio tests up to 70,000 requests per second across 1,000 services.

## 2.4.0 How complex is Istio?

For the most part, Istio “just works”. However, when it doesn’t it can be complex to track down why it isn’t working. There are almost 100 categories of logging and it requires a good understanding of the platform to know where to troubleshoot.

AKS is a hybrid-management model (Azure manages the masters, you manage the agents), but is very easy to deploy, use, and get support for. Istio is an open-source solution supported by the community. This adds an additional layer of complexity that you do not directly have support for.

**RECOMMENDATION**: Do not underestimate how much Istio changes the traffic patterns for your solution. Spend time getting to know the platform and troubleshooting issues.

## 2.5.0 Does Istio work with AKS?

Yes, Istio is fully functional in AKS. Section 5 will address the installation.

## 2.6.0 Can Azure Application Gateway front Istio services in AKS?

Yes, AppGw can provide SSL termination and WAF functionality in front of AKS (including Istio). It can also provide routing, but you probably want to do that at a lower level. The confusion typically lies with Application Gateway Ingress Controller (AGIC). Since both Istio and AGIC use a custom ingress controller, it is best not to use AGIC with Istio, however, you can still provision and manage AppGw without AGIC.

## 2.7.0 Can Azure API Management front Istio services in AKS?

Yes, APIM can be deployed into a private VNET in Azure. Traffic would be routed to AppGw, then to APIM, then to the internal load balancer, and then to the Istio ingress controller.

A lot of APIM features are duplicative with Istio, so if you intend to use both, there should be a mapping exercise to determine what each layer is expected to perform. For example:

* AppGw à SSL termination, WAF
* APIM à throttling, metering
* Istio à authentication, routing

# 3.0.0 Architecture

This section will not be an in-depth look microservices and/or routing architecture, but it will cover some basics as related to AKS and Istio.

## 3.1.0 Azure Kubernetes Service (AKS)

AKS will be deployed in an Azure VNET. Depending on requirements, you might choose to deploy AKS across Availability Zones or with Advanced Networking.

## 3.2.0 Azure VNETs and Subnets

If you are deploying AppGw and/or APIM, you should create an Azure VNET with multiple subnets for those services. This VNET will be peered to the AKS VNET. This deployment methodology provides a clear separation between the Azure-managed resources and the resources that are managed by customer.

For example:

* VNET: 172.16.0.0/16
* Tools Subnet: 172.16.0.0/24
* AppGw Subnet: 172.16.1.0/24
* APIM Subnet: 172.16.2.0/24

The tools subnet is not strictly required, but there are often other VMs that need to be deployed to support a full solution.

## 3.3.0 Namespaces

The Istio components will be installed in the “istio-system” namespace. You should deploy your services into other namespaces. Consider deploying separate namespaces for:

* external authenticated services
* external anonymous services
* internal services

By deploying those into separate namespaces, you can easily deploy Istio CRDs that govern the default behavior of each type of service.

## 3.4.0 Routing

A common routing pattern might have this flow:

1. Client
   1. Public IP address
2. Azure AppGw
   1. Public IP address proxied to private IP address (Azure VNET)
   2. Accepts HTTPS
   3. Provides SSL termination
   4. Provides WAF
3. Azure Standard Load Balancer in the VNET
   1. Private IP address
   2. Accepts TCP
   3. Provides load balancing
4. Istio Gateway
   1. Private IP address
   2. Accepts HTTP
   3. Might require JWT authentication (authN)
   4. Might enforce authorization policies (authZ)
   5. Provides routing to one or more public-facing services
5. Pod with Istio sidecar
   1. Private IP address
   2. Accepts HTTP
   3. Might use mTLS for client authentication
   4. Might require JWT authentication (authN)
   5. Might enforce authorization policies (authZ)

## 3.5.0 Design Patterns

This section will detail several common patterns for auth and routing. However, before getting to that, there are a few important concepts:

* SSL termination is done on the AppGw for performance reasons, therefore is no encryption in the internal Azure VNET. This should be considered a safe network and should have Network Security Group rules and other protections.
* There is no reason to do authentication for services not externally exposed unless there is a specific requirement to do so (some of which will be outlined below). You should valid routing rules to ensure that traffic can only reach the external services, not the internal ones.

### 3.5.1 Auth at the Istio Gateway

If you have a requirement to expose all external services the same way, you could apply your authN/Z rules directly to the Pod supporting the Istio Gateway. This could quickly involve a lot of CRDs scoped to different paths, but at least they would all be applied at the same place.

It is important to note that the principal created from a JWT authentication is ***not*** passed down to the service. As such, the service cannot impose any additional authZ rules (such as validation of claims); all rules will have to applied at the gateway.

It is possible to forward the JWT token (the header will simply be included in the proxied request) and/or to output the JWT payload (the content will be base64 encoded and passed as the specified header). Since the service won’t be verifying the JWT, the payload is all you need and it is smaller. You can see an example here:

apiVersion: security.istio.io/v1beta1

kind: RequestAuthentication

metadata:

name: gateway-jwt-auth

namespace: istio-system

spec:

selector:

matchLabels:

app: istio-ingressgateway

jwtRules:

- issuer: "https://login.microsoftonline.com/a...a/v2.0"

jwksUri: "https://login.microsoftonline.com/common/discovery/v2.0/keys"

forwardOriginalToken: TRUE

outputPayloadToHeader: "x-payload"

The service would read the payload header, decode it, and use those claims to determine the user’s identity and roles within the service. It can trust the payload provided you do not allow any external access to this Pod except via the gateway.

Anytime you are adding a header in your flow that your other services will depend on, you should remove that header at the gateway to ensure that someone doesn’t pass in that header. You can do that like so:

apiVersion: networking.istio.io/v1alpha3

kind: VirtualService

metadata:

name: gateway-virtual-service

namespace: istio-system

spec:

hosts:

- "\*"

gateways:

- istio-gateway

http:

- route:

- destination:

host: service-c.default.svc.cluster.local

subset: v1

headers:

request:

remove: [ "x-payload", "x-forwarded-client-cert" ]

If the service accepts traffic initiated from other services (i.e. the traffic did not come through the gateway), it will not have a payload header. If you allow (PERMISSIVE, UNSET) or enforce (STRICT) mTLS, traffic from the client will include an “x-forwarded-client-cert” header which can be interrogated to determine the source of the traffic. If this is important, you should use STRICT mTLS (so you can trust the header was legitimately added) and you should add an AuthorizationPolicy to restrict access to the service to certain clients and/or paths. If you do lock down your service in this way, you will need to add the ingress gateway as a client as well. By default, the principal is “cluster.local/ns/istio-system/sa/istio-ingressgateway-service-account” (and it will also have an “x-forwarded-client-cert” header).

An example “x-forwarded-client-cert” header looks like this:

By=spiffe://cluster.local/ns/default/sa/default;Hash=1983e8e8bf38a7f9ad3cea1bf1d801e4e9ea404f426dada86613c11f58838c49;Subject="";URI=spiffe://cluster.local/ns/default/sa/client'

Consider the case where a user makes a request to an API service which in turn makes calls to other services. If the user’s identity is important in those subsequent calls, each calling service will need to read the payload header and pass it along with each call. Again, it may be important to restrict which services are allowed to talk to one another to ensure trust.

The flow might look like this:

1. Client
   1. Public IP address
2. Azure AppGw
   1. Public IP address proxied to private IP address (Azure VNET)
   2. Accepts HTTPS
   3. Provides SSL termination
   4. Provides WAF
3. Azure Standard Load Balancer in the VNET
   1. Private IP address
   2. Accepts TCP
   3. Provides load balancing
4. Istio Gateway
   1. Private IP address
   2. Accepts HTTP
   3. Strips x-payload header (if there was one)
   4. Requires JWT authentication (authN)
   5. Enforces authorization policies (authZ)
   6. Adds x-payload header
5. service-api
   1. Private IP address
   2. Accepts HTTP
   3. Requires mTLS and only accepts traffic from Istio Gateway
   4. Decodes x-payload header, uses claims to decide the user is authorized to do requested operation
   5. Adds x-payload header to service-a call
6. service-a
   1. Private IP address
   2. Accepts HTTP
   3. Requires mTLS and only accepts traffic from service-api
   4. Decodes x-payload header, uses claims to decide the user is authorized to do requested operation

### 3.5.2 Auth at Services

In this pattern, you will apply all authN/Z rules to the individual services accepting external traffic and not on the gateway. The gateway will route traffic to the appropriate services, but then each will determine its own requirements for access. You can of course, use a namespace to apply the same rules across everything in the same namespace and/or use a selector to apply the same rules across some number of services.

This pattern allows for each service to have its own rules. This is particularly helpful if each service has a lot of paths that may need different rules. In addition, because the principal is now available at the service, you can have rules based on claims.

You should remove headers as shown in section 3.5.1 as well.

In all calls below this first service, the same pattern as described in 3.5.1 is used – the payload must be passed and interrogated.

The flow might look like this:

1. Client
   1. Public IP address
2. Azure AppGw
   1. Public IP address proxied to private IP address (Azure VNET)
   2. Accepts HTTPS
   3. Provides SSL termination
   4. Provides WAF
3. Azure Standard Load Balancer in the VNET
   1. Private IP address
   2. Accepts TCP
   3. Provides load balancing
4. Istio Gateway
   1. Private IP address
   2. Accepts HTTP
   3. Strips x-payload header (if there was one)
5. service-api
   1. Private IP address
   2. Accepts HTTP
   3. Requires mTLS and only accepts traffic from Istio Gateway
   4. Requires JWT authentication (authN)
   5. Enforces authorization policies (authZ)
   6. Adds x-payload header
   7. Adds x-payload header to service-a call
6. service-a
   1. Private IP address
   2. Accepts HTTP
   3. Requires mTLS and only accepts traffic from service-api
   4. Decodes x-payload header, uses claims to decide the user is authorized to do requested operation

### 3.5.3 Single API Service

This pattern is the same as either 3.5.1 or 3.5.2 except that all external traffic is terminating at a single API service. It doesn’t matter whether the authN/Z rules are applied to the gateway or the service because there is only a single instance of each. The single API service acts as a single point of ingress to all other microservices.

This model is effectively an API management service, so you could consider a solution like that for this layer as well. There are advantages and disadvantages to this approach:

Advantages

* This allows your security reviews have 2 different classifications: (a) a high-bar for security on the API service, but (b) a lower-bar for security on all other services since they are never exposed outside the internal network.
* Often when there are a large number of developers on a project, many of them don’t understand the full routing and security models. While those coding the API service must still understand everything, those coding other services do not.
* The backend microservices can change interfaces, versions, etc. all without impacting the API that you share with the world.
* When you have dozens or hundreds of microservices, you almost always require mash-ups from multiple services to answer a request. This can be done in the API service without your clients having to know about all the component services.
* Some clients might require different protocols to consume data (gRPC, OData, etc.), the API can consume other services in their native protocol and then present the data in one or more protocols.
* It is possible design choice that authorization validation could happen here instead of the other services to reduce complexity.

Disadvantages

* There is an additional hop which adds latency.
* This is additional code to write and support.
* Every time you add a service that needs external exposure, you must also modify your API service.

# 4.0.0 Configuration

The test configuration includes:

* Azure VNET with Subnets
* Azure Kubernetes Service (AKS)
  + v1.15.11
  + 3x DS2 v2
  + VMSS
  + RBAC
  + Basic Networking (kubenet) was used, though Advanced Networking (azure) was also tested successfully
  + No Availability Zones
  + Standard Load Balancer SKU
* Azure Container Registry
  + Standard SKU
* Azure Application Gateway
  + WAF v2
  + 2 Scale Units
* Azure API Management
  + Premium Tier (required to deploy in a VNET)
  + 1 Scale Unit
* Azure Private DNS Zone
* Istio
  + v1.6.0

# 5.0.0 Installation

Microsoft documentation details the following guidelines in the installation guide for Istio with AKS: <https://docs.microsoft.com/en-us/azure/aks/servicemesh-istio-install?pivots=client-operating-system-linux>.

A screenshot of a cell phone

Description automatically generated

I started with v1.4.0, however, there were a number of reasons, I chose instead to replace it with v1.6.0:

* v1.6.0 includes a simplified infrastructure (Istiod replaces Pilot, Citadel, Galley, and the sidecar injector).
* v1.5.0+ includes the PeerAuthentication and RequestAuthentication which are a significant change to how authentication is handled, and is a much improved design.
* v1.5.0+ includes deny semantics and exclusion matching for AuthorizationPolicy.
* v1.5.0+ deprecates policies in favor of newer functionality.
* v1.4.4+ includes an upgrade option which will make management of the solution much easier.

I installed Istio v1.6.0 using the customizable install with a default policy as described here: <https://istio.io/docs/setup/install/istioctl/>.

## 5.1.0 Generate Manifest

If using GitOps (<https://www.gitops.tech/>) or a similar deployment methodology, it might be useful to have a complete manifest for the deployment of resources to Kubernetes.

To generate a manifest you can run a command like so (generates “full-manifest.yaml”):

“istioctl manifest generate -f manifests/profiles/default.yaml -f patch.yaml > full-manifest.yaml”

Unfortunately, it does not seem to add the namespace for “istio-system”, so that needs to be added to the top of the file, like so:

apiVersion: v1

kind: Namespace

metadata:

name: istio-system

---

To apply the manifest, simply run a kubectl apply, like so:

“kubectl apply -f ./full-manifest.yaml”

To verify the installation was successful, you can run:

“istioctl verify-install -f manifests/profiles/default.yaml -f patch.yaml”

## 5.2.0 Internal Load Balancer

In order to deploy using an internal Azure load balancer (i.e. no public IP address), you must provide an annotation during the installation. You can create a patch file like the following (I called it “patch.yaml”):

apiVersion: install.istio.io/v1alpha1

kind: IstioOperator

spec:

values:

gateways:

istio-ingressgateway:

serviceAnnotations:

service.beta.kubernetes.io/azure-load-balancer-internal: "true"

You can install the default profile with the patch by running:

“istioctl install -f manifests/profiles/default.yaml -f patch.yaml”

In the next section, VNET peering is used. While using a Basic Load Balancer over VNET peering in the same region is supported, it did not work during testing. Using a Standard Load Balancer is recommended anyway because other AKS features (such as multiple node pools) requires this SKU.

If AKS is deployed via command line, like this…

“az aks create -g pelasne-internal -n pelasne-internal --location eastus --enable-vmss --load-balancer-sku Standard”

…a Standard Load Balancer with a public IP is deployed for outbound traffic (desired) but no inbound traffic (also desired). When Istio is installed with the above annotation, a new Standard Load Balancer is created with a private IP that load balances to the AKS VMSS (desired).

## 5.3.0 Peered VNET

Before adding AppGw and/or APIM, a VNET should be created with multiple subnets. This VNET will be peered to the AKS VNET. This design is discussed in Section 3.2.0.

A subnet should be created for:

* Tools
* AppGw
* APIM

It is important that the VNET address space not overlap with the AKS VNET address space. After creating the VNET with all subnets, it should be peered to the AKS VNET. Virtual Network Access should be enabled at least from the gateway VNET, but Forwarding is not required from either side.

The Tools subnet is not strictly required, but there are often additional VMs that need to be created to support a large solution and this provides a space for them. In addition, if you are having trouble with unexpected traffic routing behaviors, it is often advantageous to deploy a VM to this VNET for testing.

## 5.4.0 Azure Application Gateway

There are a number of ways to deploy an AppGw in front of the solution. The gateway can provide SSL termination, layer-7 load balancing, and routing rules, however, all those things can be provided by Istio. The feature that AppGw can provide that Istio cannot is Web Application Firewall (WAF).

The deployment used for this evaluation looked like this:

* AppGw had a public IP to accept internet traffic.
* AppGw was deployed in a VNET/Subnet (see Sections 5.3.0 and 3.2.0).
  + Only AppGw instances can be deployed into this Subnet.
* AppGw backend points to then next layer, which could be either…
  + AKS Internal Load Balancer IP
  + API Management (HTTP protocol, using DNS name)
* AppGw listener was deployed to listen only on port 443 and included an SSL certificate.
* AppGw HTTP settings contact the load balancer over port 80.
  + When directing to APIM, the “Override with new host name” must be set to “yes”. The “Pick host name from backend target” should also be set provided a valid DNS name was used in the backend setting.
* A custom health probe was created to check the /status/health endpoint.
  + “/status” was published as a separate API in APIM with an operation of “/health”.
  + In Istio “/health” was excluded from all authorization.

### 5.4.1 Application Gateway Ingress Controller (AGIC)

Generally if there is a requirement for an AppGw in front of AKS without Istio, AGIC is a good recommendation. Unfortunately, both AppGw and Istio require their own custom ingress controller so this is not practical. AppGw can still be used, just not AGIC.

One big advantage of AGIC is that routing can be declared in YAML alongside the resources in AKS that require those routes (instead of deploying routing via ARM in Azure). When using Istio, routing will be handled by the Istio VirtualService applied to the Gateway via YAML anyway, so this benefit goes away.

## 5.5.0 Azure API Management

There are a number of ways to deploy APIM in front of the solution. A lot of traffic management features are available in both APIM and Istio, so you should clearly outline what features will be implemented at each layer. If you are publishing APIs for use by 3rd parties, APIM has a comprehensive feature set around securing, monitoring, throttling, etc. your APIs and that might be a reason to include both.

The deployment used for this evaluation looked like this:

* APIM was deployed in a VNET/Subnet (see Sections 5.3.0 and 3.2.0).
  + The “Internal” option was selected.
  + Only APIM can be deployed into this Subnet.
  + This provides a public IP for control plane traffic.
  + This provides a private IP for all runtime traffic and the developer portal.
* A Private Azure DNS Zone was created for “azure-api.net”.
  + This was linked to the VNET.
  + DNS entries were created for APIM per <https://docs.microsoft.com/en-us/azure/api-management/api-management-using-with-internal-vnet#access-on-default-host-names>.
* In General Settings, the URL scheme was set to HTTP as the APIM was on an internal only network and SSL termination happed at the AppGw.
* The backend HTTP endpoint was set to the IP address of the AKS Internal Load Balancer IP.
* APIs were defined in APIM.
* A “/health” API was created so the AppGw could query for an operational service.
  + No subscription key was required for this API.

### 5.5.1 API – APIM Frontend

This section shows the frontend configuration from APIM for the API service.

{

"openapi": "3.0.1",

"info": {

"title": "API",

"description": "",

"version": "1.0"

},

"servers": [

{

"url": "http://pelasne.azure-api.net"

}

],

"paths": {

"/stuff": {

"get": {

"summary": "GetStuff",

"description": "GetStuff",

"operationId": "getstuff",

"responses": {

"200": {

"description": "null"

}

}

}

}

},

"components": {

"securitySchemes": {

"apiKeyHeader": {

"type": "apiKey",

"name": "Ocp-Apim-Subscription-Key",

"in": "header"

},

"apiKeyQuery": {

"type": "apiKey",

"name": "subscription-key",

"in": "query"

}

}

},

"security": [

{

"apiKeyHeader": []

},

{

"apiKeyQuery": []

}

]

}

### 5.5.2 Health - APIM Frontend

This section shows the frontend configuration from APIM for the health service.

Note that while it shows security schemes for validating the subscription key, that is turned off in the General Settings. Alternatively, you could issue a subscription key for checking health and hard-code that on the AppGw for a specific health route.

{

"openapi": "3.0.1",

"info": {

"title": "Status",

"description": "",

"version": "1.0"

},

"servers": [

{

"url": "http://pelasne.azure-api.net/status"

}

],

"paths": {

"/health": {

"get": {

"summary": "Health",

"description": "Health",

"operationId": "health",

"responses": {

"200": {

"description": "null"

}

}

}

}

},

"components": {

"securitySchemes": {

"apiKeyHeader": {

"type": "apiKey",

"name": "Ocp-Apim-Subscription-Key",

"in": "header"

},

"apiKeyQuery": {

"type": "apiKey",

"name": "subscription-key",

"in": "query"

}

}

},

"security": [

{

"apiKeyHeader": []

},

{

"apiKeyQuery": []

}

]

}

### 5.5.3 APIM Policies

This section shows the policy configuration from APIM for all operations for both API and health. There was no specific policies defined for any of the operations.

<policies>

<inbound>

<base />

<set-backend-service base-url="http://10.240.0.7" />

</inbound>

<backend>

<base />

</backend>

<outbound>

<base />

</outbound>

<on-error>

<base />

</on-error>

</policies>

## 5.6.0 Istio Gateway

Here is an example of the Gateway and its associated VirtualService:

apiVersion: networking.istio.io/v1alpha3

kind: Gateway

metadata:

name: istio-gateway

namespace: istio-system

spec:

selector:

istio: ingressgateway

servers:

- port:

name: http

number: 80

protocol: HTTP

hosts:

- "\*"

---

apiVersion: networking.istio.io/v1alpha3

kind: VirtualService

metadata:

name: gateway-virtual-service

namespace: istio-system

spec:

hosts:

- "\*"

gateways:

- istio-gateway

http:

- route:

- destination:

host: service-a.default.svc.cluster.local

subset: v1

There are a few important points on the gateway implementation:

* The VirtualService attached to a Gateway (as with all DestinationRules) must include a destination that is a Kubernetes Service (also called a “real” destination in the documentation). If you do not, the Gateway will report an HTTP 502.
* Note that the Gateway is in the istio-system namespace and so the destination VirtualService must be identified with a fully qualified name (or at least servicename.namespace).

## 5.7.0 Sidecars

By default, sidecars can be automatically injected to a namespace by applying a label (this example shows applying to the default namespace): “kubectl label namespace default e=enabled”.

You could also apply this via YAML, like so:

apiVersion: v1

kind: Namespace

metadata:

name: default

labels:

istio-injection: "enabled"

More information can be found here: <https://istio.io/docs/setup/additional-setup/sidecar-injection/>.

## 5.8.0 Microservices

Here is an example of a deployed microservice:

apiVersion: apps/v1

kind: Deployment

metadata:

labels:

app: service-b

name: service-b

namespace: default

spec:

replicas: 3

selector:

matchLabels:

app: service-b

template:

metadata:

labels:

app: service-b

version: v1

name: service-b

spec:

containers:

- name: service-b

image: pelasne.azurecr.io/http-receiver:1.0.0

env:

- name: FORMAT

value: "text"

- name: RESPONSE\_CODE

value: "200"

---

apiVersion: v1

kind: Service

metadata:

name: service-b

spec:

ports:

- name: http

port: 80

selector:

app: service-b

## 5.9.0 Istio Components

Here is an example of a VirtualService and DestinationRule:

apiVersion: networking.istio.io/v1beta1

kind: DestinationRule

metadata:

name: service-b

spec:

host: service-b

trafficPolicy:

loadBalancer:

simple: ROUND\_ROBIN

subsets:

- name: v1

labels:

version: v1

---

apiVersion: networking.istio.io/v1beta1

kind: VirtualService

metadata:

name: service-b

spec:

hosts:

- service-b

http:

- route:

- destination:

host: service-b

subset: v1

## 5.10.0 Authentication and Authorization

Here is an example of the components for authentication and authorization. Note that the following conditions are allowed (and everything else denied):

* Path is equal to “/health” (the health service).
* Service principal was created from a JWT authentication

apiVersion: security.istio.io/v1beta1

kind: RequestAuthentication

metadata:

name: jwt-auth

namespace: default

spec:

selector:

matchLabels:

app: service-b

jwtRules:

- issuer: "https://login.microsoftonline.com/a...a/v2.0"

jwksUri: "https://login.microsoftonline.com/common/discovery/v2.0/keys"

- issuer: "https://accounts.google.com"

jwksUri: "https://www.googleapis.com/oauth2/v3/certs"

- issuer: "accounts.google.com"

jwksUri: "https://www.googleapis.com/oauth2/v3/certs"

---

apiVersion: security.istio.io/v1beta1

kind: AuthorizationPolicy

metadata:

name: allow-healthz

namespace: default

spec:

selector:

matchLabels:

app: service-b

action: ALLOW

rules:

- to:

- operation:

paths: [ "/health" ]

---

apiVersion: security.istio.io/v1beta1

kind: AuthorizationPolicy

metadata:

name: allow-if-principal

namespace: default

spec:

action: ALLOW

selector:

matchLabels:

app: service-b

rules:

- from:

- source:

requestPrincipals: [ "\*" ]

# 6.0.0 Performance Tests

The performance tests are shown below. Test results are shown from go-wrk, httperf, and Istio’s own testing. The testing contacted a microservice that simply returned an HTTP 200 response with no body.

My interpretation of the results is:

* Istio has a negligible impact on performance.
* Istio with mTLS has a negligible impact on performance.
* AppGw has a large impact on performance.
* AppGw introduces a much wider variance on performance.
* APIM has a large impact on performance.
* APIM introduces a much wider variance on performance.

This testing included AppGw and APIM to get a realistic vision of the performance, but this was not a spike on how to optimize those features or layers. It is possible with spikes on those technologies that different SKUs, configuration values, etc. might improve or at least stabilize performance.

“S2S with Istio + JWT” was tested, not because that is a realistic scenario, but to understand the impact of authorization separate from AppGw (~10 ms). It is possible that there is also additional WAF rules that are impacted by including an Authorization Bearer token.

“AppGw + SSL” (without WAF) was tested, again not because it should be run that way, but to understand the impact of WAF separate from the other gateway features. Unfortunately, the results of the testing were so varied that I don’t think we can draw a good conclusion.

## 6.1.0 Performance Testing with go-wrk

I used go-wrk (<https://github.com/tsliwowicz/go-wrk>) to do performance testing with following parameters:

* 100 sessions
* 60 seconds
* Timeout of 5 seconds (though this doesn’t seem to be enforced)

Full test results are available in Appendix B.

## 6.2.0 Performance Testing with httperf

I attempted to do some testing with httperf with the following parameters:

* 1000 sessions, adding 20 per second
* 100 requests per second
* 5-10 second timeout

This should yield 100,000 requests (~667 per second), however, as soon as the Application Gateway was introduced, it could not keep up with that request rate, instead averaging around 60,000 total requests.

Unfortunately, once SSL was introduced, it became impossible to get results with this parameter set. I got one run that was successful and all other failed prematurely. These test results are incomplete.

Full test results are available in Appendix A.

## 6.3.0 Performance Testing from Istio

Istio has some comprehensive benchmarks here:

<https://istio.io/blog/2019/performance-best-practices/#istio-1-2-performance>

# 7.0.0 Authentication

Istio can be used for a number of authentication and authorization scenarios. This section will cover a number of scenarios and the configuration required to implement them.

## 7.1.0 User Authentication via OpenID Connect

APIs may be exposed to the internet from your application but require that a user login before their functionality is available. Most popular Identity Providers (IdP) allow for an OpenID Connect authentication (OIDC). This document will not describe how OIDC works, as there is plenty of documentation available already.

To implement JWT authentication in Istio, you simply need to create a RequestAuthentication CRD like so:

apiVersion: security.istio.io/v1beta1

kind: RequestAuthentication

metadata:

name: jwt-auth

namespace: default

spec:

selector:

matchLabels:

app: service-b

jwtRules:

- issuer: "https://login.microsoftonline.com/a...a/v2.0"

jwksUri: "https://login.microsoftonline.com/common/discovery/v2.0/keys"

- issuer: "https://accounts.google.com"

jwksUri: "https://www.googleapis.com/oauth2/v3/certs"

- issuer: "accounts.google.com"

jwksUri: "https://www.googleapis.com/oauth2/v3/certs"

You should reference the Istio documentation for a complete list of capabilities (<https://istio.io/docs/reference/config/security/jwt/>), but a few things I will call out in this sample:

* The CRD must be deployed into the same namespace as the resources it will be applied to.
* The selector can be omitted to apply to all Pods in the namespace.
* The selector could be modified to apply to any group of Pods by label.
* Each jwtRule must include an issuer. The issuer in the JWT is checked before the certificates are fetched/cached/used. This design allows for any number of IDPs to be supported live in the environment with effectively no impact on performance.
* Note that since Google has 2 different issuers, so you must include them as if they were separate IDPs.

## 7.2.0 User Authorization using JWT

Authentication (authN) involves taking some identifying information, verifying that it is valid, and producing a principal based on that identifying information. It does not enforce that the principal identifies any classification of user or even that a principal was able to be created at all (the user may not have provided identifying information, the information couldn’t be verified, etc.). Defining what claims the principal should contain is the job of authorization (authZ).

If you simply need to know that a user is authenticated, you can use a policy like this that ensures that the VirtualService “service-b” in namespace “default” is authenticated (has a request principal). You could remove the selector to apply this to all Pods with an Istio sidecar in the namespace.

The “action: ALLOW” specification designates that under these conditions, the traffic will be allowed, but denied under other conditions.

apiVersion: security.istio.io/v1beta1

kind: AuthorizationPolicy

metadata:

name: allow-if-authenticated

namespace: default

spec:

action: ALLOW

selector:

matchLabels:

app: service-b

rules:

- from:

- source:

requestPrincipals: [ "\*" ]

You can read more here: <https://istio.io/docs/tasks/security/authorization/authz-jwt/> and here: <https://istio.io/docs/reference/config/security/authorization-policy/>.

### 7.2.1 Authorization using Claims

While the above sample showed the acceptance of any principal with no inspection of claims, it might be desired to check claims, like version, roles, etc. This example shows a claim “ver” that must be “2.0” or the authorization will fail:

apiVersion: security.istio.io/v1beta1

kind: AuthorizationPolicy

metadata:

name: gateway-allow-if-principal

namespace: istio-system

spec:

action: ALLOW

selector:

matchLabels:

app: istio-ingressgateway

rules:

- from:

- source:

requestPrincipals: [ "\*" ]

when:

- key: request.auth.claims[ver]

values: ["2.0"]

See the architecture section of this document for why this might not be as commonly used as you might expect.

### 7.2.2 Authorization using Multiple IDPs

When a single service supports multiple Identity Providers which have different claims in their tokens, using multiple AuthorizationPolicy CRDs might help. For instance, if IDP-A uses the claim “role” equal to “admin” for an administrative user, whereas IDP-B uses the claim “roles” equal to “administrator” for the same, you might deploy 2 policies like this:

apiVersion: security.istio.io/v1beta1

kind: AuthorizationPolicy

metadata:

name: from-idp-a

namespace: istio-system

spec:

action: ALLOW

selector:

matchLabels:

app: service-a

rules:

- when:

- key: request.auth.claims[iss]

values: ["a.idp.com"]

- key: request.auth.claims[role]

values: ["admin"]

---

apiVersion: security.istio.io/v1beta1

kind: AuthorizationPolicy

metadata:

name: from-idp-b

namespace: istio-system

spec:

action: ALLOW

selector:

matchLabels:

app: service-a

rules:

- when:

- key: request.auth.claims[iss]

values: ["b.idp.com"]

- key: request.auth.claims[roles]

values: ["administrator"]

Since there are ALLOW policies, if a request matches either policy, it will be allowed. If a request matches neither, it will be denied.

## 7.3.0 Anonymous Access for Health Endpoints

Another common scenario is wanted health endpoints to be exempt from authentication requirements. This can often be a requirement because a gateway needs to determine if an endpoint is operational before sending traffic to it.

As an example, to allow all paths of “/health” in the namespace without authentication but to deny everything else, you could do something like this:

apiVersion: security.istio.io/v1beta1

kind: AuthorizationPolicy

metadata:

name: allow-healthz

namespace: default

spec:

action: ALLOW

rules:

- to:

- operation:

paths: [ "/health" ]

Keep in mind that this will deny literally everything else, so you would likely need to implement other AuthorizationPolicy CRDs that addressed other scenarios you want to allow.

## 7.4.0 Require Mutual-TLS

With v1.6.0+ Istio has mutual-TLS enabled by default, however, until you apply a PeerAuthentication CRD to require mTLS it will not be enforced.

To require mTLS to all Pods with an Istio sidecar in a namespace, you could implement something like this:

apiVersion: security.istio.io/v1beta1

kind: PeerAuthentication

metadata:

name: all-mtls

namespace: default

spec:

mtls:

mode: STRICT

Of course, like the other CRDs, you can apply this selectively.

You can read more here: <https://istio.io/docs/reference/config/security/peer_authentication/>.

## 7.5.0 Client Authorization

You might have a scenario where only certain microservices are allowed to access other microservices.

First, you must implement mTLS between those services.

Next, you must identify a particular service, for example, this microservice is identified via the ServiceAccount of “client”:

apiVersion: v1

kind: ServiceAccount

metadata:

name: client

---

apiVersion: apps/v1

kind: Deployment

metadata:

labels:

app: client

name: client

namespace: default

spec:

replicas: 1

selector:

matchLabels:

app: client

template:

metadata:

labels:

app: client

name: client

spec:

serviceAccountName: client

containers:

- name: client

image: ubuntu

command: ["/bin/bash", "-ec", "while :; do echo '.'; sleep 5 ; done"]

Then you can apply a policy, similar to this one that ensures only “service-a” or “client” can access “service-c”:

apiVersion: security.istio.io/v1beta1

kind: AuthorizationPolicy

metadata:

name: allow-if-service-or-client

namespace: default

spec:

action: ALLOW

selector:

matchLabels:

app: service-c

rules:

- from:

- source:

principals: [ "cluster.local/ns/default/sa/service-a", "cluster.local/ns/default/sa/client" ]

It is not necessarily required that every service have its own identity. If you had a classification of services (like external APIs), you could consider all of them using the same identity so that other services could accept that whole classification of services as allowed.

You can read more here: <https://istio.io/docs/reference/config/security/authorization-policy/>.

## 7.6.0 Other Scenarios

There are lots of other scenarios that could be implemented by referring to the Istio documentation, including but not limited to:

* Restriction by IP address
* Restriction by ports
* Restriction by HTTP methods
* Restriction by HTTP paths

You can read more here: <https://istio.io/docs/reference/config/security/authorization-policy/>.

## 7.7.0 Apply to Pods

RequestAuthentication (authN) and AuthorizationPolicy (authZ) CRDs can be applied to:

* Pods with Istio sidecars that are running your APIs. For example, your traffic might flow like this…  
  AppGw 🡪 Load Balancer 🡪 Gateway/Pod 🡪 VirtualService 🡪 Service 🡪 Pod  
  …and you could apply the CRDs to the destination Pod.
* The Pod implementing the Gateway. For example, your traffic might flow like this…  
  AppGw 🡪 Load Balancer 🡪 Gateway/Pod 🡪 VirtualService 🡪 Service 🡪 Pod  
  …and you could apply the CRDs to the Gateway Pod so that all downstream Pods require authentication. The selector is “istio-ingressgateway” by default. You can query for it by running “kubectl get pod -n istio-system”. To apply the CRD its namespace needs to be “istio-system”.  
    
  You can read more here: <https://istio.io/docs/tasks/security/authorization/authz-ingress/>.

## 7.8.0 Namespaces for Public Services

If you desire that new services have authN/Z restrictions applied automatically, you might consider creating one or more namespaces for public services. Deploying a CRD with a namespace but no selector will ensure that all new Pods created in that namespace get the restrictions.

## 7.9.0 Debugging

It can be difficult to determine what principals are coming into a service and why the access might be denied. Istio provides a very exhaustive list of logging categories. You can enable the RBAC debugging by running something similar to:

kubectl exec $(kubectl get pods -l app=service-c -o jsonpath='{.items[0].metadata.name}') -c istio-proxy -- pilot-agent request POST 'logging?rbac=debug'

You can make some requests and then view the logs by running something similar to:

kubectl logs service-c-867f8d7c67-4jlk7 istio-proxy --tail 200

# 8.0.0 Throttling

To meet the requirement for public APIs to support developer-level throttling to ensure fair-user (Section 1.4.0), it is recommended to implement Azure API Management. This is an out-of-the-box feature of APIM and more information can be found here: <https://docs.microsoft.com/en-us/azure/api-management/api-management-sample-flexible-throttling>.

# 9.0.0 Partitioning

To meet the requirement for instances of a service to be partitioned by customer (Section 1.5.0), Istio can be configured to support traffic routing by header.

The first step to support this scenario is to define 1 Pod per customer for the service – if there are 3 customers for service-A, then there will be 3 Pods for service-A, each running the same container image but labeled uniquely for each customer.

For instance, here is a Pod for “customer-data” for “acme”:

apiVersion: apps/v1

kind: Deployment

metadata:

labels:

app: customer-data-acme

name: customer-data-acme

namespace: default

spec:

replicas: 3

selector:

matchLabels:

app: customer-data-acme

template:

metadata:

labels:

app: customer-data-acme

version: v1

type: customer-data

partition: acme

name: customer-data-acme

spec:

containers:

- name: customer-data

image: pelasne.azurecr.io/customer-data:1.0.0

Here is a second Pod for “customer-data” for “fabrikam”:

apiVersion: apps/v1

kind: Deployment

metadata:

labels:

app: customer-data-fabrikam

name: customer-data-fabrikam

namespace: default

spec:

replicas: 3

selector:

matchLabels:

app: customer-data-fabrikam

template:

metadata:

labels:

app: customer-data-fabrikam

version: v1

type: customer-data

partition: fabrikam

name: customer-data-fabrikam

spec:

containers:

- name: customer-data

image: pelasne.azurecr.io/customer-data:1.0.0

Notice, they each share the label for the “type” of service, but have a unique label for “partition”.

From here, there are 2 possible ways to handle routing, use either method (9.1.0 or 9.2.0), but not both.

## 9.1.0 Route by DestinationRule

In this method, you define a single Service that points to all “customer-data” services using the “type” label:

apiVersion: v1

kind: Service

metadata:

name: customer-data

spec:

ports:

- port: 80

selector:

type: customer-data

Then you create a DestinationRule with a subset for each customer:

apiVersion: networking.istio.io/v1alpha3

kind: DestinationRule

metadata:

name: customer-data-dest

spec:

host: customer-data

trafficPolicy:

loadBalancer:

simple: ROUND\_ROBIN

subsets:

- name: acme

labels:

partition: acme

- name: fabrikam

labels:

partition: fabrikam

Finally, you create a VirtualService that looks for a header called “x-partition” and uses that information to select the appropriate subset:

apiVersion: networking.istio.io/v1alpha3

kind: VirtualService

metadata:

name: customer-data

spec:

hosts:

- customer-data

http:

- match:

- headers:

x-partition:

exact: acme

route:

- destination:

host: customer-data

subset: acme

- match:

- headers:

x-partition:

exact: fabrikam

route:

- destination:

host: customer-data

subset: fabrikam

You can then direct to the “acme customer data” service by adding the appropriate “x-partition” header:

curl http://customer-data -i -H "x-partition: acme"

Or you can direct to the “fabrikam customer data” service by adding the appropriate “x-partition” header:

curl http://customer-data -i -H "x-partition: fabrikam"

## 9.2.0 Route by Service

In this method, you define a separate Service for each customer, like so (notice the selector can still be by type and partition):

apiVersion: v1

kind: Service

metadata:

name: customer-data-acme

spec:

ports:

- port: 80

selector:

type: customer-data

partition: acme

---

apiVersion: v1

kind: Service

metadata:

name: customer-data-fabrikam

spec:

ports:

- port: 80

selector:

type: customer-data

partition: fabrikam

Then you need to define a Service endpoint that you can call that acts as the redirector:

apiVersion: v1

kind: Service

metadata:

name: customer-data

spec:

ports:

- port: 80

Finally, you define a VirtualService that routes to each different service endpoint:

apiVersion: networking.istio.io/v1alpha3

kind: VirtualService

metadata:

name: customer-data

spec:

hosts:

- customer-data

http:

- match:

- headers:

x-partition:

exact: acme

route:

- destination:

host: customer-data-acme

- match:

- headers:

x-partition:

exact: fabrikam

route:

- destination:

host: customer-data-fabrikam

You can then direct to the “acme customer data” service by adding the appropriate “x-partition” header:

curl http://customer-data -i -H "x-partition: acme"

Or you can direct to the “fabrikam customer data” service by adding the appropriate “x-partition” header:

curl http://customer-data -i -H "x-partition: fabrikam"

# 10.0.0 Distribution of Services across Cloud Providers

It was mentioned that some solutions may have service distributed across multiple cloud providers. There would need to be specific requirements and then a specific routing plan could be determined, however, there are some general statements that can be made regarding distribution across cloud providers.

## 10.1.0 Distribution using Istio

Istio supports a wide range of deployment options (<https://istio.io/latest/docs/ops/deployment/deployment-models/>) including the capability to deploy across multiple Kubernetes clusters which may be in different networks and even different regions.

If all services across all clouds are deployed via Istio, one or more control planes can be deployed that allows for service discovery regardless of the cloud deployment. This will typically require bridging the networks together via VPN or using Istio gateways with public IP addresses.

## 10.2.0 Route without Istio

Pods in AKS can generally initiate outbound connections to services over the internet. An Istio Egress Gateway, an egress load balancer rule in AKS, Network Security Groups applied to the Azure VNET, etc. might all contribute to how the egress traffic is handled.

The Pods in AKS will need some network connectivity to the other cloud services. This will typically require bridging the networks together via VPN or using public IP addresses for the services.

Azure Traffic Manager provides a location-aware DNS load balancing solution. Azure Front Door provides a location-aware layer-7 reverse proxy with many of the same features as AppGw but deployed as a global service across the Microsoft edge network. It is probable that one or both of these services would assist with this scenario.

# 11.0.0 Questions and Concerns

The following questions and concerns prompted this evaluation, so I will attempt to address those here.

## 11.1.0 Network hairpins in inter-service communication could make latency go very high.

While it is important not to introduce too many extra layers, performance testing has shown that the reverse proxy features of Istio has a negligible impact on latency even under heavy load.

JWT authentication does add ~10 ms to the latency. I would recommend only validating the JWT for external services (those exposed outside the internal network). The JWT can be passed from service to service and claims in it used without validation (trusting it was validated at the edge). One additional control that might help is using mutual-TLS authentication to ensure that the services communicating with one another are authorized to do so (and thereby adding to the assurance that the JWT authentication was already validated).

## 11.2.0 Should patterns like retry, circuit breaker, backpressure, etc. be done in each service or the mesh?

Using a product like Istio that provides the features outside of the code has a number of advantages, including:

* Services can be written using different languages and frameworks. Istio provides features like timeout, retry, circuit breaker, authentication, fault injection, routing, etc. uniformly across all those languages and frameworks without coding it multiple times.
* Istio moves those features from code to configuration, significantly reducing the time to implement and test, while removing the requirement to code-review.
* Istio is an open-source project with many contributors. You can get the value of all the code they have implemented and tested for free.

Implementing these features in your code can also offer some advantages:

* You have complete control over the configuration and implementation of the features allowing you to match your requirements exactly.
* You can implement features that are not supported in the Istio platform. For example, Istio does not support any kind of backpressure management, but you could implement that in your own code.

Generally, if a reliable, supported platform offers the features you need, you should leverage it and only code the exceptional cases.

## 11.3.0 How do we hook into Istio for telemetry, monitoring, etc.?

* You can query metrics from Prometheus: <https://istio.io/docs/tasks/observability/metrics/querying-metrics/>.
* You can visualize metrics with Grafana: <https://istio.io/docs/tasks/observability/metrics/using-istio-dashboard/>.
* You can visualize your service mesh using Kiali: <https://istio.io/docs/tasks/observability/kiali/>.
* You can output access logs: <https://istio.io/docs/tasks/observability/logs/access-log/>.
* You can do distributed tracing using Jaeger, Zipkin, or Lightstep: <https://istio.io/docs/tasks/observability/distributed-tracing/overview/>.

# A.0.0 Performance Testing Results with httperf

## A.1.0 S2S without Istio Latency Test

httperf --hog --server=service-a --wsess=1000,100,1 --rate 20 --timeout 5

httperf --hog --timeout=5 --client=0/1 --server=service-a --port=80 --uri=/ --rate=20 --send-buffer=4096 --recv-buffer=16384 --wsess=1000,100,1.000

httperf: warning: open file limit > FD\_SETSIZE; limiting max. # of open files to FD\_SETSIZE

Maximum connect burst length: 1

Total: connections 1000 requests 100000 replies 100000 test-duration 149.236 s

Connection rate: 6.7 conn/s (149.2 ms/conn, <=1000 concurrent connections)

Connection time [ms]: min 99266.1 avg 99450.5 max 99797.1 median 99440.5 stddev 84.3

Connection time [ms]: connect 0.0

Connection length [replies/conn]: 100.000

Request rate: 670.1 req/s (1.5 ms/req)

Request size [B]: 62.0

Reply rate [replies/s]: min 60.0 avg 688.0 max 1000.1 stddev 320.2 (29 samples)

Reply time [ms]: response 4.0 transfer 0.0

Reply size [B]: header 147.0 content 0.0 footer 0.0 (total 147.0)

Reply status: 1xx=0 2xx=100000 3xx=0 4xx=0 5xx=0

CPU time [s]: user 48.37 system 93.82 (user 32.4% system 62.9% total 95.3%)

Net I/O: 136.8 KB/s (1.1\*10^6 bps)

Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0

Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

Session rate [sess/s]: min 0.00 avg 6.70 max 20.40 stddev 9.34 (1000/1000)

Session: avg 1.00 connections/session

Session lifetime [s]: 99.5

Session failtime [s]: 0.0

Session length histogram: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1000

## A.2.0 S2S with Istio Latency Test

httperf --hog --server=service-a-istio --wsess=1000,100,1 --rate 20 --timeout 5

httperf --hog --timeout=5 --client=0/1 --server=service-a-istio --port=80 --uri=/ --rate=20 --send-buffer=4096 --recv-buffer=16384 --wsess=1000,100,1.000

httperf: warning: open file limit > FD\_SETSIZE; limiting max. # of open files to FD\_SETSIZE

Maximum connect burst length: 1

Total: connections 1000 requests 100000 replies 100000 test-duration 149.488 s

Connection rate: 6.7 conn/s (149.5 ms/conn, <=1000 concurrent connections)

Connection time [ms]: min 99326.0 avg 99576.2 max 100056.2 median 99551.5 stddev 133.9

Connection time [ms]: connect 0.0

Connection length [replies/conn]: 100.000

Request rate: 669.0 req/s (1.5 ms/req)

Request size [B]: 68.0

Reply rate [replies/s]: min 60.0 avg 688.1 max 999.5 stddev 318.8 (29 samples)

Reply time [ms]: response 5.3 transfer 0.0

Reply size [B]: header 147.0 content 0.0 footer 0.0 (total 147.0)

Reply status: 1xx=0 2xx=100000 3xx=0 4xx=0 5xx=0

CPU time [s]: user 48.84 system 93.18 (user 32.7% system 62.3% total 95.0%)

Net I/O: 140.5 KB/s (1.2\*10^6 bps)

Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0

Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

Session rate [sess/s]: min 0.00 avg 6.69 max 20.20 stddev 9.37 (1000/1000)

Session: avg 1.00 connections/session

Session lifetime [s]: 99.6

Session failtime [s]: 0.0

Session length histogram: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1000

## A.3.0 AppGw + WAF + Istio Latency Test

### A.3.1 AppGw + WAF + Istio Latency Test (Run 1)

httperf --hog --server=istio.plasne.com --wsess=1000,100,1 --rate 20 --timeout 5

httperf --hog --timeout=5 --client=0/1 --server=istio.plasne.com --port=80 --uri=/ --rate=20 --send-buffer=4096 --recv-buffer=16384 --wsess=1000,100,1.000

httperf: warning: open file limit > FD\_SETSIZE; limiting max. # of open files to FD\_SETSIZE

Maximum connect burst length: 1

Total: connections 1000 requests 58930 replies 58407 test-duration 157.617 s

Connection rate: 6.3 conn/s (157.6 ms/conn, <=969 concurrent connections)

Connection time [ms]: min 6017.6 avg 65134.5 max 113461.7 median 48785.5 stddev 40203.7

Connection time [ms]: connect 0.0

Connection length [replies/conn]: 58.997

Request rate: 373.9 req/s (2.7 ms/req)

Request size [B]: 69.0

Reply rate [replies/s]: min 47.6 avg 376.8 max 651.2 stddev 159.6 (31 samples)

Reply time [ms]: response 67.8 transfer 0.0

Reply size [B]: header 148.0 content 0.0 footer 0.0 (total 148.0)

Reply status: 1xx=0 2xx=58407 3xx=0 4xx=0 5xx=0

CPU time [s]: user 32.28 system 123.83 (user 20.5% system 78.6% total 99.0%)

Net I/O: 78.8 KB/s (0.6\*10^6 bps)

Errors: total 523 client-timo 523 socket-timo 0 connrefused 0 connreset 0

Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

Session rate [sess/s]: min 0.00 avg 3.03 max 14.20 stddev 4.59 (477/1000)

Session: avg 1.00 connections/session

Session lifetime [s]: 105.6

Session failtime [s]: 27.0

Session length histogram: 10 16 18 11 13 14 19 8 11 19 7 15 9 16 9 9 14 9 14 15 14 12 7 11 13 9 12 13 15 11 12 13 8 6 14 13 14 12 10 3 14 7 10 11 2 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 477

### A.3.2 AppGw + WAF + Istio Latency Test (Run 2)

httperf --hog --server=istio.plasne.com --wsess=1000,100,1 --rate 20 --timeout 5

httperf --hog --timeout=5 --client=0/1 --server=istio.plasne.com --port=80 --uri=/ --rate=20 --send-buffer=4096 --recv-buffer=16384 --wsess=1000,100,1.000

httperf: warning: open file limit > FD\_SETSIZE; limiting max. # of open files to FD\_SETSIZE

Maximum connect burst length: 1

Total: connections 1000 requests 55721 replies 55132 test-duration 154.521 s

Connection rate: 6.5 conn/s (154.5 ms/conn, <=1000 concurrent connections)

Connection time [ms]: min 6023.6 avg 60382.4 max 110420.8 median 47686.5 stddev 37433.4

Connection time [ms]: connect 0.0

Connection length [replies/conn]: 55.353

Request rate: 360.6 req/s (2.8 ms/req)

Request size [B]: 69.0

Reply rate [replies/s]: min 49.8 avg 367.2 max 785.1 stddev 188.3 (30 samples)

Reply time [ms]: response 44.8 transfer 0.0

Reply size [B]: header 148.0 content 0.0 footer 0.0 (total 148.0)

Reply status: 1xx=0 2xx=55132 3xx=0 4xx=0 5xx=0

CPU time [s]: user 34.91 system 117.66 (user 22.6% system 76.1% total 98.7%)

Net I/O: 75.9 KB/s (0.6\*10^6 bps)

Errors: total 589 client-timo 589 socket-timo 0 connrefused 0 connreset 0

Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

Session rate [sess/s]: min 0.00 avg 2.66 max 9.00 stddev 3.82 (411/1000)

Session: avg 1.00 connections/session

Session lifetime [s]: 103.1

Session failtime [s]: 30.2

Session length histogram: 4 10 20 13 12 14 12 10 6 12 10 17 19 14 12 11 10 11 12 12 17 13 11 12 11 11 12 11 9 16 13 13 9 18 13 12 8 9 10 15 14 11 15 12 12 14 13 6 4 1 1 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 411

### A.3.3 AppGw + WAF + Istio Latency Test (Run 3)

httperf --hog --server=istio.plasne.com --wsess=1000,100,1 --rate 20 --timeout 5

httperf --hog --timeout=5 --client=0/1 --server=istio.plasne.com --port=80 --uri=/ --rate=20 --send-buffer=4096 --recv-buffer=16384 --wsess=1000,100,1.000

httperf: warning: open file limit > FD\_SETSIZE; limiting max. # of open files to FD\_SETSIZE

Maximum connect burst length: 1

Total: connections 1000 requests 69295 replies 68878 test-duration 163.535 s

Connection rate: 6.1 conn/s (163.5 ms/conn, <=1000 concurrent connections)

Connection time [ms]: min 6018.6 avg 76664.1 max 121046.5 median 0.0 stddev 38922.8

Connection time [ms]: connect 0.0

Connection length [replies/conn]: 69.155

Request rate: 423.7 req/s (2.4 ms/req)

Request size [B]: 69.0

Reply rate [replies/s]: min 22.8 avg 430.5 max 790.3 stddev 203.2 (32 samples)

Reply time [ms]: response 86.6 transfer 0.0

Reply size [B]: header 148.0 content 0.0 footer 0.0 (total 148.0)

Reply status: 1xx=0 2xx=68498 3xx=0 4xx=0 5xx=380

CPU time [s]: user 31.06 system 129.69 (user 19.0% system 79.3% total 98.3%)

Net I/O: 89.7 KB/s (0.7\*10^6 bps)

Errors: total 417 client-timo 417 socket-timo 0 connrefused 0 connreset 0

Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

Session rate [sess/s]: min 0.00 avg 3.56 max 13.40 stddev 5.27 (583/1000)

Session: avg 1.00 connections/session

Session lifetime [s]: 108.2

Session failtime [s]: 31.9

Session length histogram: 4 6 8 5 11 14 10 12 6 11 6 4 6 8 8 7 6 4 11 11 9 9 11 6 10 9 11 7 11 5 10 5 6 10 8 13 9 8 10 5 6 10 10 7 13 8 10 6 2 1 2 0 0 1 1 1 1 1 0 2 0 0 0 0 0 1 1 0 0 2 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 583

### A.3.4 AppGw + WAF + Istio Latency Test (Run 4)

httperf --hog --server=istio.plasne.com --wsess=1000,100,1 --rate 20 --timeout 10

httperf --hog --timeout=10 --client=0/1 --server=istio.plasne.com --port=80 --uri=/ --rate=20 --send-buffer=4096 --recv-buffer=16384 --wsess=1000,100,1.000

httperf: warning: open file limit > FD\_SETSIZE; limiting max. # of open files to FD\_SETSIZE

Maximum connect burst length: 1

Total: connections 1000 requests 75950 replies 75592 test-duration 188.470 s

Connection rate: 5.3 conn/s (188.5 ms/conn, <=1000 concurrent connections)

Connection time [ms]: min 11042.3 avg 96717.7 max 143020.9 median 0.0 stddev 37155.1

Connection time [ms]: connect 0.0

Connection length [replies/conn]: 75.743

Request rate: 403.0 req/s (2.5 ms/req)

Request size [B]: 69.0

Reply rate [replies/s]: min 5.0 avg 408.6 max 770.9 stddev 205.4 (37 samples)

Reply time [ms]: response 237.9 transfer 0.0

Reply size [B]: header 148.0 content 0.0 footer 0.0 (total 148.0)

Reply status: 1xx=0 2xx=75592 3xx=0 4xx=0 5xx=0

CPU time [s]: user 29.35 system 157.14 (user 15.6% system 83.4% total 98.9%)

Net I/O: 85.2 KB/s (0.7\*10^6 bps)

Errors: total 358 client-timo 358 socket-timo 0 connrefused 0 connreset 0

Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

Session rate [sess/s]: min 0.00 avg 3.41 max 16.60 stddev 5.23 (642/1000)

Session: avg 1.00 connections/session

Session lifetime [s]: 123.1

Session failtime [s]: 48.9

Session length histogram: 2 1 0 1 1 3 3 5 1 4 5 6 8 6 8 7 5 10 7 5 7 9 10 4 10 10 4 4 9 6 6 5 9 8 5 8 7 9 12 6 8 6 7 8 6 6 7 4 10 5 9 11 6 4 6 4 5 4 3 2 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 642

### A.3.5 AppGw + WAF + Istio Latency Test (Run 5)

httperf --hog --server=istio.plasne.com --wsess=1000,100,1 --rate 20 --timeout 10

httperf --hog --timeout=10 --client=0/1 --server=istio.plasne.com --port=80 --uri=/ --rate=20 --send-buffer=4096 --recv-buffer=16384 --wsess=1000,100,1.000

httperf: warning: open file limit > FD\_SETSIZE; limiting max. # of open files to FD\_SETSIZE

Maximum connect burst length: 1

Total: connections 1000 requests 72376 replies 72009 test-duration 189.349 s

Connection rate: 5.3 conn/s (189.3 ms/conn, <=1000 concurrent connections)

Connection time [ms]: min 11082.4 avg 92809.4 max 145298.7 median 0.0 stddev 44631.8

Connection time [ms]: connect 0.0

Connection length [replies/conn]: 72.226

Request rate: 382.2 req/s (2.6 ms/req)

Request size [B]: 69.0

Reply rate [replies/s]: min 9.0 avg 389.2 max 713.1 stddev 179.1 (37 samples)

Reply time [ms]: response 242.8 transfer 0.0

Reply size [B]: header 148.0 content 0.0 footer 0.0 (total 148.0)

Reply status: 1xx=0 2xx=72009 3xx=0 4xx=0 5xx=0

CPU time [s]: user 30.96 system 157.35 (user 16.4% system 83.1% total 99.5%)

Net I/O: 80.8 KB/s (0.7\*10^6 bps)

Errors: total 367 client-timo 367 socket-timo 0 connrefused 0 connreset 0

Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

Session rate [sess/s]: min 0.00 avg 3.34 max 13.80 stddev 4.93 (633/1000)

Session: avg 1.00 connections/session

Session lifetime [s]: 125.2

Session failtime [s]: 36.3

Session length histogram: 3 9 12 10 7 5 10 9 8 14 7 9 4 5 7 6 6 9 6 11 9 5 11 8 8 9 9 5 11 5 8 5 10 5 5 9 14 9 10 8 5 4 6 10 0 0 0 0 4 1 2 1 1 0 1 0 1 0 0 0 0 0 1 0 0 0 0 1 0 0 1 0 1 2 1 2 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 633

## A.4.0 AppGw + WAF + SSL + Istio Latency Test

I wanted to get more runs with 1000 sessions, but unfortunately, I was never able to get another one without: “httperf: failed to connect to SSL server (err=-1, reason=5)”.

### A.4.1 AppGw + WAF + SSL + Istio Latency Test (Run 1)

httperf --hog --server=istio.plasne.com --wsess=1000,100,1 --rate 20 --timeout 5 --ssl

httperf --hog --timeout=5 --client=0/1 --server=istio.plasne.com --port=443 --uri=/ --rate=20 --send-buffer=4096 --recv-buffer=16384 --ssl --wsess=1000,100,1.000

httperf: warning: open file limit > FD\_SETSIZE; limiting max. # of open files to FD\_SETSIZE

Maximum connect burst length: 1

Total: connections 1000 requests 61848 replies 61345 test-duration 157.946 s

Connection rate: 6.3 conn/s (157.9 ms/conn, <=1000 concurrent connections)

Connection time [ms]: min 6042.6 avg 66820.0 max 112570.1 median 56533.5 stddev 38813.1

Connection time [ms]: connect 29.9

Connection length [replies/conn]: 61.530

Request rate: 391.6 req/s (2.6 ms/req)

Request size [B]: 69.0

Reply rate [replies/s]: min 21.4 avg 395.8 max 796.1 stddev 192.3 (31 samples)

Reply time [ms]: response 52.4 transfer 0.0

Reply size [B]: header 176.0 content 0.0 footer 0.0 (total 176.0)

Reply status: 1xx=0 2xx=61345 3xx=0 4xx=0 5xx=0

CPU time [s]: user 35.37 system 121.80 (user 22.4% system 77.1% total 99.5%)

Net I/O: 93.5 KB/s (0.8\*10^6 bps)

Errors: total 503 client-timo 503 socket-timo 0 connrefused 0 connreset 0

Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

Session rate [sess/s]: min 0.00 avg 3.15 max 11.40 stddev 4.63 (497/1000)

Session: avg 1.00 connections/session

Session lifetime [s]: 104.3

Session failtime [s]: 29.4

Session length histogram: 3 17 14 16 12 11 16 7 10 7 13 11 13 5 12 10 8 13 12 10 9 9 12 9 8 11 10 8 9 12 12 8 7 9 9 9 12 7 13 11 10 9 15 11 14 13 10 4 1 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 497

### A.4.2 AppGw + WAF + SSL + Istio Latency Test (Run 2)

httperf --hog --server=istio.plasne.com --wsess=100,100,1 --rate 20 --timeout 30 --ssl

httperf --hog --timeout=30 --client=0/1 --server=istio.plasne.com --port=443 --uri=/ --rate=20 --send-buffer=4096 --recv-buffer=16384 --ssl --wsess=100,100,1.000

httperf: warning: open file limit > FD\_SETSIZE; limiting max. # of open files to FD\_SETSIZE

Maximum connect burst length: 1

Total: connections 100 requests 10000 replies 10000 test-duration 105.642 s

Connection rate: 0.9 conn/s (1056.4 ms/conn, <=100 concurrent connections)

Connection time [ms]: min 100479.6 avg 100635.0 max 100981.9 median 0.0 stddev 90.1

Connection time [ms]: connect 20.0

Connection length [replies/conn]: 100.000

Request rate: 94.7 req/s (10.6 ms/req)

Request size [B]: 69.0

Reply rate [replies/s]: min 59.2 avg 95.1 max 99.6 stddev 10.4 (21 samples)

Reply time [ms]: response 15.7 transfer 0.0

Reply size [B]: header 176.0 content 0.0 footer 0.0 (total 176.0)

Reply status: 1xx=0 2xx=10000 3xx=0 4xx=0 5xx=0

CPU time [s]: user 45.34 system 60.28 (user 42.9% system 57.1% total 100.0%)

Net I/O: 22.7 KB/s (0.2\*10^6 bps)

Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0

Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

Session rate [sess/s]: min 0.00 avg 0.95 max 17.40 stddev 3.80 (100/100)

Session: avg 1.00 connections/session

Session lifetime [s]: 100.6

Session failtime [s]: 0.0

Session length histogram: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 100

### A.4.3 AppGw + WAF + SSL + Istio Latency Test (Run 3)

httperf --hog --server=istio.plasne.com --wsess=200,100,1 --rate 20 --timeout 10 --ssl

httperf --hog --timeout=10 --client=0/1 --server=istio.plasne.com --port=443 --uri=/ --rate=20 --send-buffer=4096 --recv-buffer=16384 --ssl --wsess=200,100,1.000

httperf: warning: open file limit > FD\_SETSIZE; limiting max. # of open files to FD\_SETSIZE

Maximum connect burst length: 1

Total: connections 200 requests 20000 replies 20000 test-duration 111.055 s

Connection rate: 1.8 conn/s (555.3 ms/conn, <=200 concurrent connections)

Connection time [ms]: min 100452.2 avg 100870.3 max 101406.5 median 0.0 stddev 229.5

Connection time [ms]: connect 20.9

Connection length [replies/conn]: 100.000

Request rate: 180.1 req/s (5.6 ms/req)

Request size [B]: 69.0

Reply rate [replies/s]: min 59.4 avg 181.6 max 198.2 stddev 38.5 (22 samples)

Reply time [ms]: response 18.1 transfer 0.0

Reply size [B]: header 176.0 content 0.0 footer 0.0 (total 176.0)

Reply status: 1xx=0 2xx=20000 3xx=0 4xx=0 5xx=0

CPU time [s]: user 41.82 system 69.21 (user 37.7% system 62.3% total 100.0%)

Net I/O: 43.3 KB/s (0.4\*10^6 bps)

Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0

Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

Session rate [sess/s]: min 0.00 avg 1.80 max 19.00 stddev 5.31 (200/200)

Session: avg 1.00 connections/session

Session lifetime [s]: 100.9

Session failtime [s]: 0.0

Session length histogram: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 200

### A.4.4 AppGw + WAF + SSL + Istio Latency Test (Run 4)

httperf --hog --server=istio.plasne.com --wsess=300,100,1 --rate 20 --timeout 10 --ssl

httperf --hog --timeout=10 --client=0/1 --server=istio.plasne.com --port=443 --uri=/ --rate=20 --send-buffer=4096 --recv-buffer=16384 --ssl --wsess=300,100,1.000

httperf: warning: open file limit > FD\_SETSIZE; limiting max. # of open files to FD\_SETSIZE

Maximum connect burst length: 1

Total: connections 300 requests 30000 replies 30000 test-duration 116.386 s

Connection rate: 2.6 conn/s (388.0 ms/conn, <=300 concurrent connections)

Connection time [ms]: min 100706.0 avg 101003.7 max 101785.9 median 0.0 stddev 154.2

Connection time [ms]: connect 20.9

Connection length [replies/conn]: 100.000

Request rate: 257.8 req/s (3.9 ms/req)

Request size [B]: 69.0

Reply rate [replies/s]: min 59.2 avg 260.7 max 298.2 stddev 71.0 (23 samples)

Reply time [ms]: response 19.4 transfer 0.0

Reply size [B]: header 176.0 content 0.0 footer 0.0 (total 176.0)

Reply status: 1xx=0 2xx=30000 3xx=0 4xx=0 5xx=0

CPU time [s]: user 39.67 system 76.59 (user 34.1% system 65.8% total 99.9%)

Net I/O: 61.9 KB/s (0.5\*10^6 bps)

Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0

Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

Session rate [sess/s]: min 0.00 avg 2.58 max 20.40 stddev 6.45 (300/300)

Session: avg 1.00 connections/session

Session lifetime [s]: 101.0

Session failtime [s]: 0.0

Session length histogram: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 300

### A.4.5 AppGw + WAF + SSL + Istio Latency Test (Run 5)

httperf --hog --server=istio.plasne.com --wsess=400,100,1 --rate 20 --timeout 30 --ssl

httperf --hog --timeout=30 --client=0/1 --server=istio.plasne.com --port=443 --uri=/ --rate=20 --send-buffer=4096 --recv-buffer=16384 --ssl --wsess=400,100,1.000

httperf: warning: open file limit > FD\_SETSIZE; limiting max. # of open files to FD\_SETSIZE

Maximum connect burst length: 1

Total: connections 400 requests 40000 replies 40000 test-duration 121.182 s

Connection rate: 3.3 conn/s (303.0 ms/conn, <=400 concurrent connections)

Connection time [ms]: min 100773.4 avg 101110.5 max 101571.7 median 0.0 stddev 132.5

Connection time [ms]: connect 20.5

Connection length [replies/conn]: 100.000

Request rate: 330.1 req/s (3.0 ms/req)

Request size [B]: 69.0

Reply rate [replies/s]: min 59.4 avg 333.2 max 397.0 stddev 106.3 (24 samples)

Reply time [ms]: response 20.5 transfer 0.0

Reply size [B]: header 176.0 content 0.0 footer 0.0 (total 176.0)

Reply status: 1xx=0 2xx=40000 3xx=0 4xx=0 5xx=0

CPU time [s]: user 33.40 system 87.70 (user 27.6% system 72.4% total 99.9%)

Net I/O: 79.3 KB/s (0.6\*10^6 bps)

Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0

Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

Session rate [sess/s]: min 0.00 avg 3.30 max 20.60 stddev 7.24 (400/400)

Session: avg 1.00 connections/session

Session lifetime [s]: 101.1

Session failtime [s]: 0.0

Session length histogram: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 400

### A.4.6 AppGw + WAF + SSL + Istio Latency Test (Run 6)

httperf --hog --server=istio.plasne.com --wsess=400,100,1 --rate 5 --timeout 30 --ssl

httperf --hog --timeout=30 --client=0/1 --server=istio.plasne.com --port=443 --uri=/ --rate=5 --send-buffer=4096 --recv-buffer=16384 --ssl --wsess=400,100,1.000

httperf: warning: open file limit > FD\_SETSIZE; limiting max. # of open files to FD\_SETSIZE

Maximum connect burst length: 1

Total: connections 400 requests 40000 replies 40000 test-duration 181.407 s

Connection rate: 2.2 conn/s (453.5 ms/conn, <=400 concurrent connections)

Connection time [ms]: min 100709.9 avg 101334.3 max 102286.1 median 0.0 stddev 343.4

Connection time [ms]: connect 21.0

Connection length [replies/conn]: 100.000

Request rate: 220.5 req/s (4.5 ms/req)

Request size [B]: 69.0

Reply rate [replies/s]: min 15.0 avg 222.2 max 396.4 stddev 122.4 (36 samples)

Reply time [ms]: response 22.7 transfer 0.0

Reply size [B]: header 176.0 content 0.0 footer 0.0 (total 176.0)

Reply status: 1xx=0 2xx=39999 3xx=0 4xx=0 5xx=1

CPU time [s]: user 60.11 system 121.24 (user 33.1% system 66.8% total 100.0%)

Net I/O: 53.0 KB/s (0.4\*10^6 bps)

Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0

Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

Session rate [sess/s]: min 0.00 avg 2.20 max 5.60 stddev 2.52 (400/400)

Session: avg 1.00 connections/session

Session lifetime [s]: 101.3

Session failtime [s]: 0.0

Session length histogram: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 400

# B.0.0 Performance Testing Results with go-wrk

## B.1.0 S2S without Istio

167358 requests in 59.554063046s, 20.79MB read

Requests/sec: 2810.19

Transfer/sec: 357.45KB

Avg Req Time: 35.584831ms

Fastest Request: 642.792µs

Slowest Request: 420.824702ms

Number of Errors: 0

## B.2.0 S2S with Istio

170104 requests in 59.589597639s, 21.15MB read

Requests/sec: 2854.59

Transfer/sec: 363.46KB

Avg Req Time: 35.031273ms

Fastest Request: 626.092µs

Slowest Request: 995.741497ms

Number of Errors: 0

## B.3.0 S2S with Istio + mTLS

Avg = 35.84 ms (35.93+35.98+35.62) / 3

165662 requests in 59.519877163s, 20.59MB read

Requests/sec: 2783.31

Transfer/sec: 354.21KB

Avg Req Time: 35.928503ms

Fastest Request: 640.292µs

Slowest Request: 367.000694ms

Number of Errors: 0

165607 requests in 59.582670199s, 20.59MB read

Requests/sec: 2779.45

Transfer/sec: 353.82KB

Avg Req Time: 35.978352ms

Fastest Request: 497.495µs

Slowest Request: 389.496031ms

Number of Errors: 0

167330 requests in 59.606389271s, 20.81MB read

Requests/sec: 2807.25

Transfer/sec: 357.44KB

Avg Req Time: 35.622057ms

Fastest Request: 909.19µs

Slowest Request: 364.123816ms

Number of Errors: 0

## B.3.0 AppGw + WAF

Avg = 100.51 ms (117.39+78.25+105.89) / 3

39853 requests in 46.785126434s, 4.99MB read

Requests/sec: 851.83

Transfer/sec: 109.21KB

Avg Req Time: 117.394239ms

Fastest Request: 12.230156ms

Slowest Request: 4.977796804s

Number of Errors: 265

57279 requests in 44.823456628s, 7.17MB read

Requests/sec: 1277.88

Transfer/sec: 163.69KB

Avg Req Time: 78.254607ms

Fastest Request: 12.326155ms

Slowest Request: 1.006098789s

Number of Errors: 1521

42197 requests in 44.681074036s, 5.29MB read

Requests/sec: 944.40

Transfer/sec: 121.19KB

Avg Req Time: 105.886849ms

Fastest Request: 12.495453ms

Slowest Request: 1.005864979s

Number of Errors: 2482

## B.4.0 AppGw + WAF + SSL

Avg = 88.01 ms (70.16+52.32+94.46+51.44+171.67) / 5

53947 requests in 37.851558972s, 8.22MB read

Requests/sec: 1425.23

Transfer/sec: 222.44KB

Avg Req Time: 70.164344ms

Fastest Request: 11.632262ms

Slowest Request: 8.758816686s

Number of Errors: 3827

34338 requests in 17.965331952s, 5.23MB read

Requests/sec: 1911.35

Transfer/sec: 298.30KB

Avg Req Time: 52.319098ms

Fastest Request: 11.482265ms

Slowest Request: 7.322949178s

Number of Errors: 2830

47684 requests in 45.041717938s, 7.27MB read

Requests/sec: 1058.66

Transfer/sec: 165.22KB

Avg Req Time: 94.458765ms

Fastest Request: 11.472165ms

Slowest Request: 31.496125922s

Number of Errors: 3634

43577 requests in 22.416016035s, 6.64MB read

Requests/sec: 1944.01

Transfer/sec: 303.45KB

Avg Req Time: 51.440016ms

Fastest Request: 11.278067ms

Slowest Request: 8.84537606s

Number of Errors: 5953

30247 requests in 51.926498792s, 4.61MB read

Requests/sec: 582.50

Transfer/sec: 90.90KB

Avg Req Time: 171.674872ms

Fastest Request: 11.638963ms

Slowest Request: 1m5.616053118s

Number of Errors: 1002

## B.5.0 AppGw + WAF + SSL + JWT

Avg = 105.34 ms (70.82+72.92+159.89+154.28+68.78) / 5

20573 requests in 14.569287922s, 3.14MB read

Requests/sec: 1412.08

Transfer/sec: 220.60KB

Avg Req Time: 70.817517ms

Fastest Request: 12.009664ms

Slowest Request: 9.739184491s

Number of Errors: 2767

38792 requests in 28.286411408s, 5.92MB read

Requests/sec: 1371.40

Transfer/sec: 214.17KB

Avg Req Time: 72.918156ms

Fastest Request: 12.359658ms

Slowest Request: 8.940374757s

Number of Errors: 5684

36937 requests in 59.058738404s, 5.63MB read

Requests/sec: 625.43

Transfer/sec: 97.64KB

Avg Req Time: 159.890457ms

Fastest Request: 12.516755ms

Slowest Request: 1m5.452850223s

Number of Errors: 1311

34976 requests in 53.960707743s, 5.33MB read

Requests/sec: 648.18

Transfer/sec: 101.23KB

Avg Req Time: 154.279242ms

Fastest Request: 12.570354ms

Slowest Request: 1m4.994385826s

Number of Errors: 6991

48364 requests in 33.26408364s, 7.38MB read

Requests/sec: 1453.94

Transfer/sec: 227.08KB

Avg Req Time: 68.778603ms

Fastest Request: 12.617453ms

Slowest Request: 9.747027908s

Number of Errors: 6645

## B.6.0 S2S with Istio + JWT

Avg = 46.05 ms (45.90+46.14+46.10) / 3

130387 requests in 59.853460554s, 16.25MB read

Requests/sec: 2178.44

Transfer/sec: 277.94KB

Avg Req Time: 45.904469ms

Fastest Request: 1.275285ms

Slowest Request: 342.867055ms

Number of Errors: 0

129663 requests in 59.820750512s, 16.32MB read

Requests/sec: 2167.53

Transfer/sec: 279.31KB

Avg Req Time: 46.135559ms

Fastest Request: 1.182686ms

Slowest Request: 404.23863ms

Number of Errors: 0

129789 requests in 59.838100496s, 16.17MB read

Requests/sec: 2169.00

Transfer/sec: 276.74KB

Avg Req Time: 46.104138ms

Fastest Request: 1.286085ms

Slowest Request: 361.74783ms

Number of Errors: 0

## B.7.0 AppGw + SSL

Avg = 88.82 ms (97.16+42.70+126.59) / 3

46251 requests in 44.935444286s, 7.06MB read

Requests/sec: 1029.28

Transfer/sec: 160.79KB

Avg Req Time: 97.155616ms

Fastest Request: 10.513484ms

Slowest Request: 1m4.676454019s

Number of Errors: 4517

43301 requests in 18.490838867s, 6.60MB read

Requests/sec: 2341.75

Transfer/sec: 365.77KB

Avg Req Time: 42.703029ms

Fastest Request: 10.78958ms

Slowest Request: 9.620292494s

Number of Errors: 2372

33351 requests in 42.218851069s, 5.08MB read

Requests/sec: 789.96

Transfer/sec: 123.33KB

Avg Req Time: 126.58946ms

Fastest Request: 10.459083ms

Slowest Request: 1m4.533510117s

Number of Errors: 4154

## B.8.0 S2S with Istio + mTLS + filter

Avg = 35.86 ms (36.44+34.70+36.45) / 3

163395 requests in 59.533110029s, 20.33MB read

Requests/sec: 2744.61

Transfer/sec: 349.71KB

Avg Req Time: 36.435086ms

Fastest Request: 942.989µs

Slowest Request: 341.693275ms

Number of Errors: 0

171692 requests in 59.582925294s, 21.36MB read

Requests/sec: 2881.56

Transfer/sec: 367.16KB

Avg Req Time: 34.703378ms

Fastest Request: 962.489µs

Slowest Request: 364.361014ms

Number of Errors: 3

163453 requests in 59.579058447s, 20.34MB read

Requests/sec: 2743.46

Transfer/sec: 349.54KB

Avg Req Time: 36.450269ms

Fastest Request: 956.489µs

Slowest Request: 389.756621ms

Number of Errors: 4

## B.9.0 AppGw + WAF + SSL + APIM + JWT

Avg = 179.71 ms (129.72+182.07+201.75+201.91+183.12) / 5

./go-wrk -c 100 -d 60 -T 5000 -H "Authorization: Bearer $TOKEN" https://internal.plasne.com/stuff?subscription-key=4...c

46284 requests in 1m0.037870476s, 6.13MB read

Requests/sec: 770.91

Transfer/sec: 104.54KB

Avg Req Time: 129.716252ms

Fastest Request: 23.10103ms

Slowest Request: 830.418178ms

Number of Errors: 0

32976 requests in 1m0.038872846s, 4.37MB read

Requests/sec: 549.24

Transfer/sec: 74.53KB

Avg Req Time: 182.068391ms

Fastest Request: 6.529245ms

Slowest Request: 597.533623ms

Number of Errors: 0

29754 requests in 1m0.029268477s, 3.94MB read

Requests/sec: 495.66

Transfer/sec: 67.27KB

Avg Req Time: 201.751927ms

Fastest Request: 6.729062ms

Slowest Request: 629.645007ms

Number of Errors: 0

29732 requests in 1m0.031605423s, 3.94MB read

Requests/sec: 495.27

Transfer/sec: 67.21KB

Avg Req Time: 201.909072ms

Fastest Request: 6.354131ms

Slowest Request: 558.45706ms

Number of Errors: 0

32803 requests in 1m0.067767045s, 4.35MB read

Requests/sec: 546.10

Transfer/sec: 74.11KB

Avg Req Time: 183.116687ms

Fastest Request: 6.480142ms

Slowest Request: 507.476402ms

Number of Errors: 0