appviewx

Architecture Guide

Version: 2022.1.0

Copyright AppViewX, Inc.

Copyright © 2022 AppViewX, Inc. All Rights Reserved.

This document may not be copied, disclosed, transferred, or modified without the prior written consent of AppViewX, Inc. While all content is believed to be correct at the time of publication, it is provided as general-purpose information. The content is subject to change without notice and is provided "as is" and with no expressed or implied warranties whatsoever, including, but not limited to, a warranty for accuracy made by AppViewX. The software described in this document is provided under written license only, contains valuable trade secrets and proprietary information, and is protected by the copyright laws of the United States and other countries. Unauthorized use of software or its documentation can result in civil damages and criminal prosecution.

Trademarks

The trademarks, logos, and service marks displayed in this manual are the property of AppViewX or other third parties. Users are not permitted to use these marks without the prior written consent of AppViewX or such third party which may own the mark.

External Reference Links

This product includes software developed by the CentOS Project (www.centos.org).

This product includes software developed by Red Hat, Inc. (www.redhat.com).

This product includes software developed by VMware, Inc. (www.vmware.com).

All other trademarks mentioned in this document are the property of their respective owners.

Contact Information

AppViewX, Inc.

222 Broadway, FL 19

New York, NY 10038

Email: info@appviewx.com

Web: www.appviewx.com

Contents

Preface......iv

Revision History......iv

About this Guide iv

Audience......iv

Text Conventions......iv

Chapter 1. Architecture...... 5

Chapter 2. Features and Benefits...... 6

Chapter 3. Architecture Diagram......7

Chapter 4. Caching...... 9

Chapter 5. Container Networking Architecture Review...... 10

Communication between Pods in the same node......10

Communication between Pods in a single DC......11

Communication between Pods in two DCs......12

Service Mesh - Discoverability and Identity Management......12

PKI......14

Ingress Proxy......15

Deployment Considerations......15

Firewall Rules...... 18

Load Balancer Configuration...... 19

Chapter 6. Security Policy Enforcement...... 21

Chapter 7. Glossary......22

Copyright © 2022 AppViewX, Inc. All Rights Reserved.

Preface

Revision History

Revision	Description	Date
1.0	Initial release of document for Release 2022.1.0	June 2022

About this Guide

This guide outlines the steps for AppViewX's architecture, which is designed around the concept of a Zero Trust Network

Audience

This guide is intended for network engineers, service engineers, system administrators, and infrastructure engineers.

Text Conventions

The following text conventions are used in this document:

Convention	Description
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`codeblock`	Indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Chapter 1: Architecture

AppViewX's architecture is designed around the concept of a Zero Trust Network. To achieve this, we need to implement four key points.

1. Allowed network connections are subject to enforcement; all connections are denied unless explicitly allowed.
2. Each endpoint should be identifiable. The IP Address and port numbers do not serve as identities.
3. All traffic must be encrypted to prevent the leak of sensitive information. If encryption is not an option, at least authenticity of data should be established.
4. Comprised nodes must not be able to circumvent security policies.

This is provided for by a combination of 3 open-source tools:

1. Kubernetes - manages and maintains containerized applications/microservices
2. Calico - for network policy enforcement
3. Istio - for service discoverability at L4-L7 layers, identity management and encryption

Copyright © 2022 AppViewX, Inc. All Rights Reserved.

Chapter 2: Features and Benefits

• Microservices architecture

• Promotes modularity

• Horizontally and independently scalable

• Highly resilient

• Quicker release cycles

• Lesser time for Go Live

• Enterprise grade security

• Near zero downtime during upgrades

• In-house loadbalancing

• Rate throttling

• Multi-tenancy

• Plug and play deployment

• Centralized role based access control

• Independently testable components

• End-to-End transaction monitoring

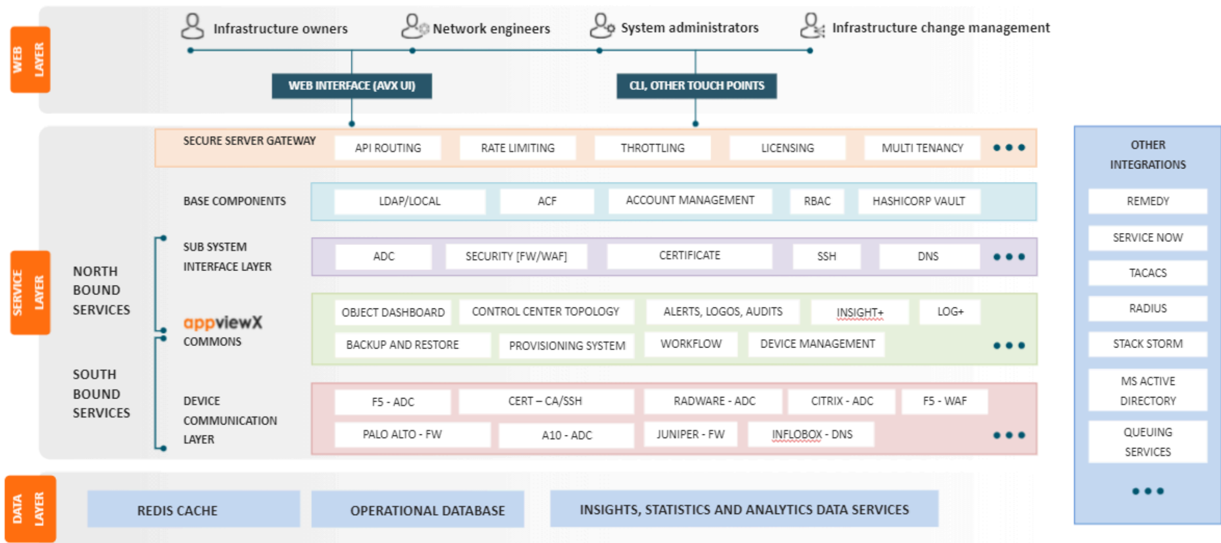
• REST based communication

• Out-of-the-box Swagger documentation

Copyright © 2022 AppViewX, Inc. All Rights Reserved.

Chapter 3: Architecture Diagram

Architecture - Explained



In the diagram:

- **Presentation/ Web Layer** - houses the AppViewX user interface related files and interacts with the service layer
- **Service Layer** - contains the Northbound & Southbound services that can be further classified into:
  - **Business Layer**:
    - Houses AppViewX specific business logic
    - Interacts with the Data layer for persisting the input data
  - **Device Communication Layer**:
    - Low code
    - Stateless layer
    - Routes communication to the respective vendor through APIs or SSH
    - Houses vendor specific business logic
- **Data Layer**:

Copyright © 2022 AppViewX, Inc. All Rights Reserved.

- Houses data persistence and retrieval logic
- Redis caching is available

Copyright © 2022 AppViewX, Inc. All Rights Reserved.

Chapter 4: Caching

This section provides information on the caching feature introduced in this release. The caching feature is deployed through a Redis server.

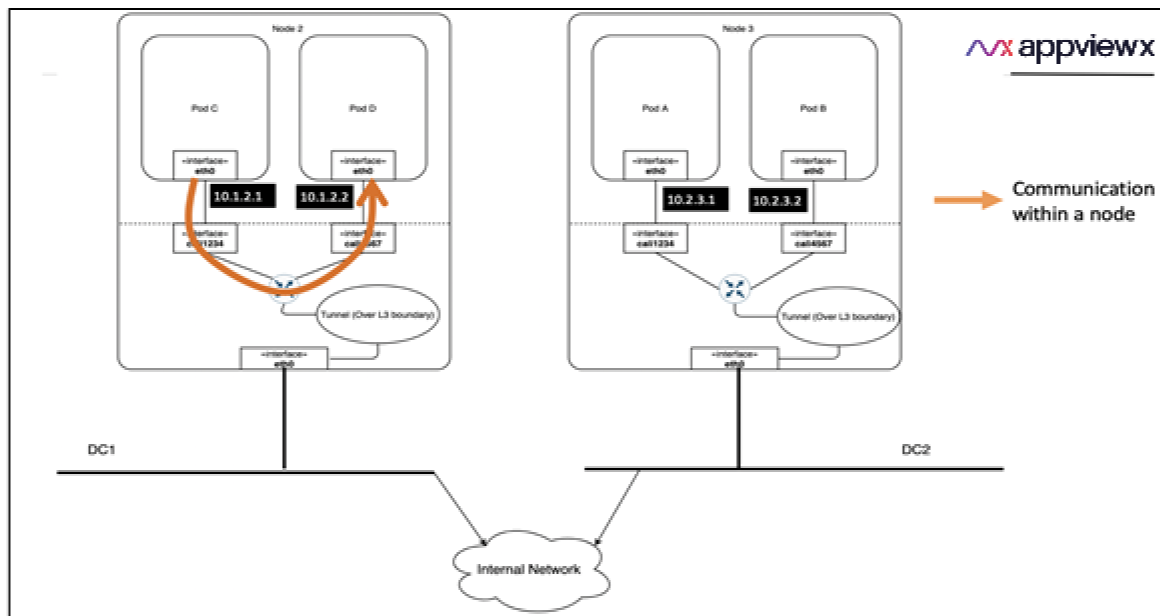
1. Redis Caching - provides the following:
    a. DB Caching
        i. uses the primary key to access the document
        ii. is used for frequent and slow queries
    b. API Caching
    c. Eviction policies
2. Redis Pub-Sub is available for inter-services notifications that are currently implemented using websockets between ADC Notification subsystem and WebApp.
3. Caching for Instructions/Commands for each type of Agent/device - Hash data structure is suggested
    a. By Command (Delete) or
    b. By Device Type or
    c. By Device Make
4. Configurations and decisions related to each functionality.

Copyright © 2022 AppViewX, Inc. All Rights Reserved.

Chapter 5: Container Networking Architecture Review

- Communication between Pods in the same node
- Communication between Pods in a single DC
- Communication between Pods in two DCs
- Service Mesh - Discoverability and Identity Management

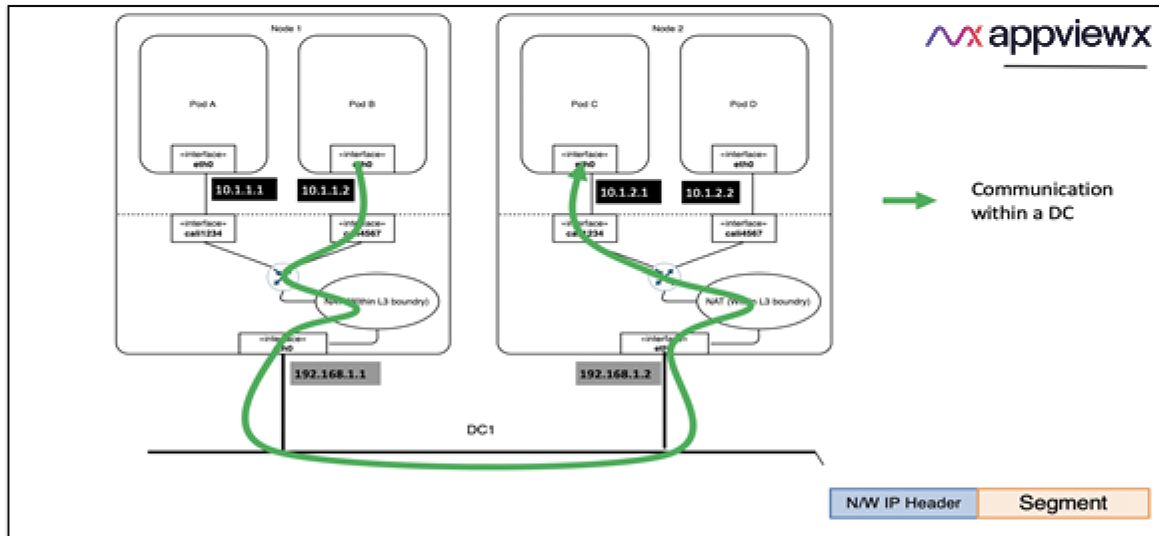
Communication between Pods in the same node



Two pods, within the same node, are in the same subnet. This communication never needs to leave the node. Therefore, the Pods do a simple ARP to query and setup communication.

Copyright © 2022 AppViewX, Inc. All Rights Reserved.

Communication between Pods in a single DC

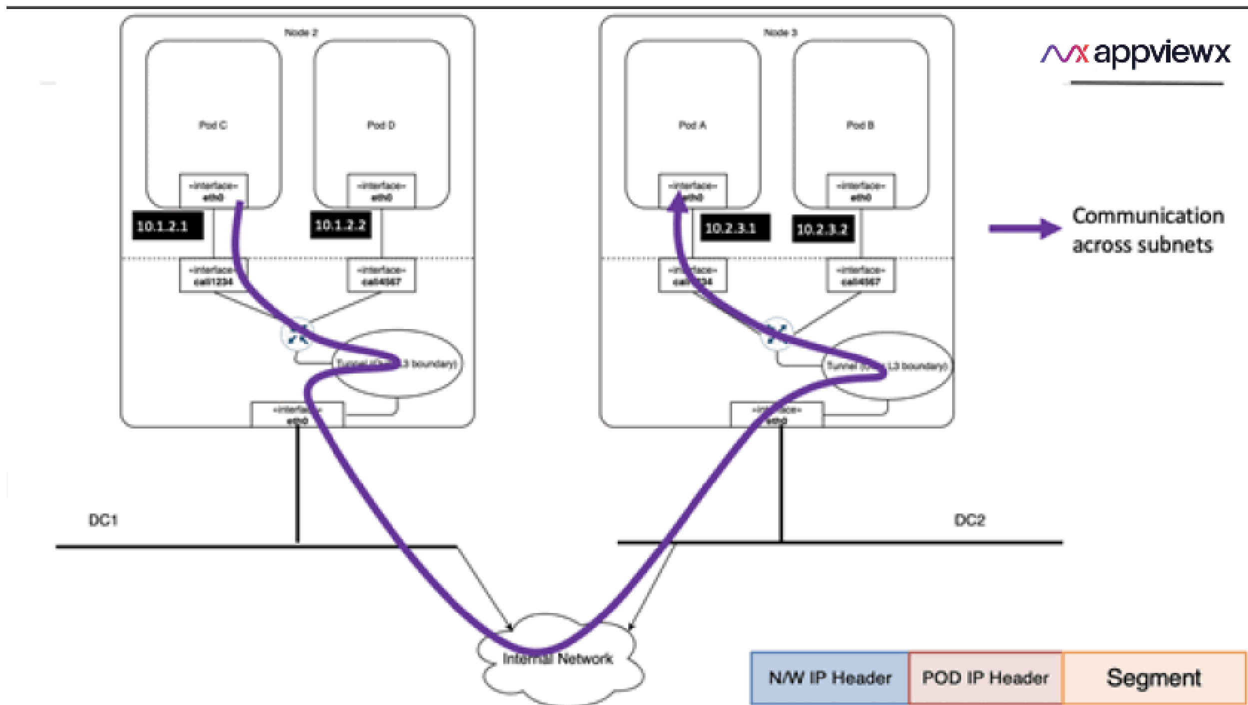


When two pods in two different nodes which are hosted in the same Datacenter try to communicate, we translate PODs IPs to the eth0 IP of the network segment. The receiving pod de-NATs this communication and passes to the respective service destination that has been called by the source POD.

In this case, we do not use the overlay and IP-IP to encapsulate. We will be enabling IP-IP tunnel for all communication even within the same PODs. NAT would be disabled. The flow would be similar as described in the Communication between Pods in two DCs section.

Copyright © 2022 AppViewX, Inc. All Rights Reserved.

Communication between Pods in two DCs



Two PODs, hosted in two different Datacenters communicate, we will use the IP-IP tunnel (Overlay network). The actual IP packets between the pods are encapsulated inside the IP header of the node-to-node communication. These packets will have two IP headers. The first IP header would be the N/W IP header and the second would the tunnel IP header.

Note:

The pods within the Kubernetes cluster communicate to and from a configurable set of IP addresses assigned to every pod that gets spinned up. The IP addresses that a pod can use is configurable and steps to configure the same is mentioned in the Configuring POD/Service IP CIDR section.

Service Mesh - Discoverability and Identity Management

Kubernetes is great at managing PODs and containers. It will detect, manage and spin up containers when they misbehave at layer 3. However, Kubernetes doesn’t understand the concept of a service. A service is at L4-L7 of the network stack. For example, let us say that a POD is up but continuously throwing an http 503 error. Kubernetes doesn't understand this http error code to detect that the service is

Copyright © 2022 AppViewX, Inc. All Rights Reserved.

unavailable and that the container cannot service requests. If a POD goes down, then Kubernetes detects and replenishes with a new POD of the same type.

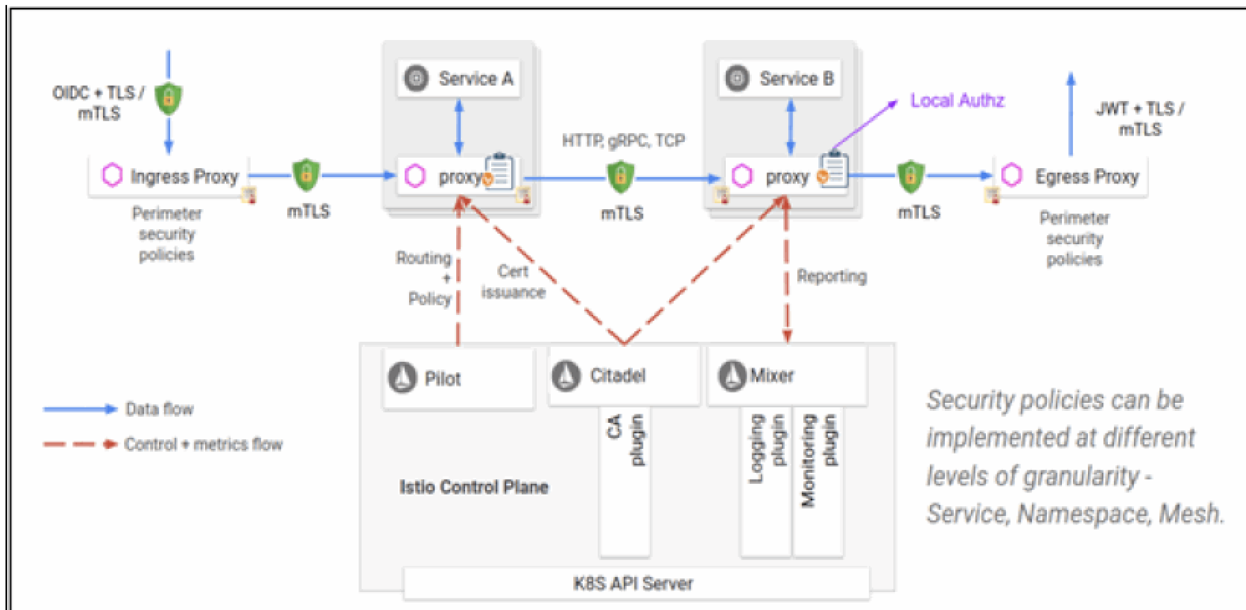
The containers have to be managed based on the health of the service it provides and not just whether the IP/Port is responding. To get over this problem, the concept of Service Mesh has been introduced. The Service mesh itself is implemented using ISTIO.

AppViewX uses ISTIO to solve two specific challenges.

1. Identity of Pod - In a zero trust network, each pod/container needs to be uniquely identifiable; this has to be over and beyond IP Address/Port. Due to the inherent nature of containers, the IP address of a pod keeps changing.
2. Encryption of traffic - all inter-pod traffic needs to be encrypted to prevent eavesdropping and stop a man-in-the-middle scenario.

For simplicity, we will assume that all containers are running on the overlay network and don't have a view of the physical network. In this section we will consider the networking L1-L3 is taken care of by Kubernetes as described in the section above.

The contents of this section are from the documents in the istio website. This is a summary of the content in the below website: https://istio.io/v1.3/docs/concepts/security/



There are 4 key components of ISTIO. In the AppViewX scheme of things, Pilot and Mixer isn't used for Security Policy enforcement. Calico, mentioned in the above section, takes care of policy enforcement.

Copyright © 2022 AppViewX, Inc. All Rights Reserved.

- Citadel for key and certificate management
- Sidecar and perimeter proxies (Envoy) to implement secure communication between clients and servers. Each pod has an envoy proxy side-car implemented. The certificate/identity is implemented within this proxy. The TLS encryption/decryption is also implemented here.
  - Dynamic service discovery and Load balancing
  - TLS termination
  - Circuit breakers and Health checks
  - Staged rollouts with %-based traffic split Fault injection Rich metrics
- Pilot to distribute authentication policies and secure naming information to the proxies
- Mixer to manage authorization and auditing

- PKI
- Ingress Proxy
- Deployment Considerations
- Firewall Rules
- Load Balancer Configuration

PKI

The Istio PKI is built on top of Istio Citadel and securely provisions strong identities to every workload. Istio uses X.509 certificates to carry the identities in SPIFFE format. The PKI also automates the key and certificate rotation at scale.

In the Kubernetes scenario, the certificate key provisioning mechanisms are done in the following manner:

1. Citadel watches the Kubernetes api server, creates a SPIFFE certificate and key pair for each of the existing and new service accounts. Citadel stores the certificate and key pairs as Kubernetes secrets.
2. When you create a pod, Kubernetes mounts the certificate and key pair to the pod according to its service account via Kubernetes secret volume.
3. Citadel watches the lifetime of each certificate, and automatically rotates the certificates by rewriting the Kubernetes secrets.
4. Pilot generates the secure naming information, which defines what service account or accounts can run a certain service. Pilot then passes the secure naming information to the sidecar Envoy.

Copyright © 2022 AppViewX, Inc. All Rights Reserved.

Ingress Proxy

All incoming traffic into this application will be through Ingress Proxy (also known as ISTIO gateway). This proxy has the intelligence to identify the route traffic according to the HTTPS URL/URI of the request. This is also implemented using the Envoy Proxy.

Deployment Considerations

In this section, we talk about the deployment considerations of AppViewX 2021.1.0 in a 2 DC 7 node setup as an example.

In AppViewX, each plugin is managed as a single pod i.e. a single pod will have only one container. In the deployment plan (refer section ‘Production Environment’) below we will have 10 different types of containers i.e. 10 types of pods (some functioning as primary/secondary or client/server). We will be spinning multiple instances of each pod depending on the requirements of load.

We can control which pods spins-up where and in how many numbers depending on pod affinity and pod anti-affinity which allow you to specify rules about how pods should be placed relative to other pods. The rules are defined using custom labels on nodes and label selectors specified in pods. Pod affinity/anti-affinity allows a pod to specify an affinity (or anti-affinity) towards a group of pods it can be placed with. The node does not have control over the placement. We have configured Affinity settings to be DC specific and not pod-specific, inter-node specific.

- Pod affinity - tells the scheduler to locate a new pod on the same DC as other pods if the label selector on the new pod matches the label on the current pod.
- Pod anti-affinity - will prevent the scheduler from locating a new pod on the same node as pods with the same labels if the label selector on the new pod matches the label on the current pod.

You can find more information on affinity and anti-affinity here:

https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/

A specific example of anti-affinity is that we don't want two platform DBs to be running on the same node. We configure the labels such that there is only one platform_database pods in a node in a DC.

Some of the deployment and design considerations are:

- Kube master should be deployed as an isolated node - Best practice.
- No pods should be deployed in Kube master if the Kube master has 4GB RAM/4 Core cpu - Best practice.
- Node running mongo db primary has to be sparsely loaded

Copyright © 2022 AppViewX, Inc. All Rights Reserved.

- Node running mongo db secondary has to have some spare juice to accommodate for failure of mongo db primary.
- The order of deployment of components should strive to keep the node's resource utilization in a distributed fashion.
- For HA of Kube master, 3 masters are recommended and spread across datacenters
- For HA of mongo DB, 1 primary, 3 secondaries and 1 arbiter are recommended and spread across datacenters in a 2 DC environment
- Web and Gateway URLs are available through the ISTIO Ingress Proxy IP. ISTIO Ingress Proxy runs in Kube worker nodes and is configurable in the configuration file

Copyright © 2022 AppViewX, Inc. All Rights Reserved.

Recommended Deployment Configurations	7 Node 2 DC Environment									
	DC 1				DC 2			Pod affinity / Anti Affinity	Auto scale	Resource Utilization
	Node(s)				Node(s)					
	1	2	3	4	1	2	3			
Kube Master										Light
avx_platform_database (Pr)								Anti-Affinity	No	Heavy
avx_platform_database (Sec)								Anti-Affinity	No	Medium
avx_platform_database (Arbiter)								Affinity	No	Light
Ingress gateway								Affinity	No	Light
avx_platform_consul(server)								Affinity	No	Light
avx_platform_consul(client)								Affinity	No	Light
avx_platform_vault								Affinity	No	Light
avx_config_server								Affinity	Yes	Light
avx_platform_core								Affinity	Yes	Medium
avx_platform_queue								Affinity	No	Medium
avx_subsystems								Affinity	Yes	Heavy
avx_subsystems_sync								Affinity	Yes	Heavy
avx_vendor_cert_network_discovery								Affinity	No	Heavy
avx_vendors								Affinity	No	Light
avx_platform_web								Affinity	No	Light
avx_platform_gateway								Affinity	No	Light

Legend

- L = 4G RAM, 4vCPU, 100G HDD (Kube Master)
- H = 32G RAM, 8vCPU, 1000G HDD (Kube Worker)
- Light = Less that 1GB of memory requirement

Copyright © 2022 AppViewX, Inc. All Rights Reserved.

- Medium = Between 1 to 4 GB of memory requirement
- Heavy = Greater than 4 GB memory requirement and high CPU cycles

Firewall Rules

The following section details about the firewall rules required to be applied within and outside the Kube cluster to access the application.

No.	Source		Destination		Protocol Used	TCP/ UDP	Type of Information Communicated
	IP	Port	IP	Port			
1	All Nodes	Any	All Nodes	22	SSH	TCP	Required for AppViewX installation and prerequisite checks
2	All Nodes	Any	All Nodes	179	BGP	TCP	To establish a common routing table for the overlay network
3	All Nodes	Any	All Nodes	6443	HTTPS	TCP	Kubernetes API server for communication between Kubernetes master and worker nodes
4	All Nodes	Any	All Nodes	10250	HTTPS	TCP	Used by Kubelet Agent which expose Rest endpoints for the Kubernetes API Server
5	All Nodes	Any	All Nodes	4243	HTTP	TCP	Required during installation and scaling up. Used to load docker images when spinning up a new container. Triggered by the node where the install process is started.No sensitive data is being transferred through this port
6	F5 (SNAT)	Any	ISTIO Ingress Proxy IP (Kube Worker)	31443	HTTPS	TCP	To access AppVIewX web UI
7	F5 (SNAT)	Any	ISTIO Ingress	30190	HTTPS	TCP	To access the AppVIewX management console

Copyright © 2022 AppViewX, Inc. All Rights Reserved.

			Proxy IP (Kube Worker)				
8	All Nodes	-	All Nodes	-	IP-IP IP Protocol 4	NA	Overlay network established with IP-IP tunnels. Information over this tunnel is encrypted using mTLS.
9	Master	Any	Master	2379	HTTPS	TCP	Will be required for etcd server communication in a multi master setup
10	Master	Any	Master	2380	HTTPS	TCP	Will be required for etcd server communication in a multi master setup
11	All Nods	Any	All Nodes	9100	HTTP	TCP	Required for monitoring the node metrics.

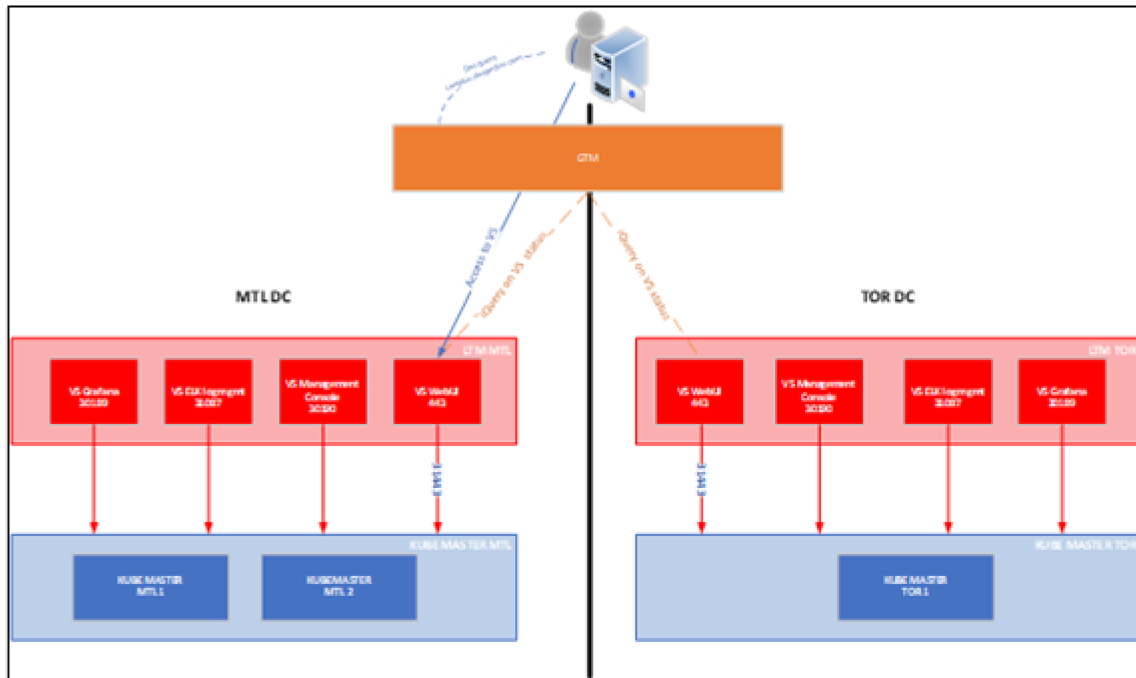
Note:

- IPs required - The system will require 1 IP per node.
- The externally exposed services will all use the nodes IP address to communicate within the network.
- Port 22 is used for administration of the node for example to log into the Linux CLI. Need SSH access the nodes to other nodes.
- We would need an external Load Balancer to distribute user/API traffic to all Kube master nodes. We can open firewall ports depending on the network setup.

Load Balancer Configuration

HLD:

Copyright © 2022 AppViewX, Inc. All Rights Reserved.



DNS and GTM:

- AppViewX will be deployed in two separate data centers in active-standby mode from the point of view of serving client's request.
- Only one DNS name is to be provided to represent all user/clients facing services (ex: prod.appviewx.com).
- GTM will monitor the state of the WebUI (443) virtual server as an indicator of health at both sites.

- GTM will respond with secondary data center virtual server IP only in the event where the primary data center VS is down (no healthy pool members).

Copyright © 2022 AppViewX, Inc. All Rights Reserved.

Chapter 6: Security Policy Enforcement

The security policy enforcement is enforced via calico-node in each node. Specific traffic flows are explicitly allowed in the calico-nodes. Every other traffic is denied by default. As mentioned previously, the route-policy across these nodes are maintained by the BGP routing protocol running on port 179.

Copyright © 2022 AppViewX, Inc. All Rights Reserved.

Chapter 7: Glossary

This section describes common terms and their abbreviations used in this guide.

Term/Abbreviation	Meaning/Expansion
BGP	Border Gateway Protocol
NAT	Network Address Translation
MTU	Maximum Transmission Unit
VM	Virtual Machine
TLS	Transport Layer Security
SPIFFE	Secure Production Identity Framework For Everyone
HTTPS	Hypertext Transfer Protocol Secure

Copyright © 2022 AppViewX, Inc. All Rights Reserved.