WHITE PAPER–AUGUST 2020

Deploying Antrea for Kubernetes Networking

Table of contents

Overview

Antrea is designed to be Kubernetes-centric and Kubernetes-native. It focuses on and is optimized for networking and security of a Kubernetes cluster. Its implementation leverages Kubernetes and Kubernetes-native solutions as much as possible.

Antrea leverages Open vSwitch (OVS) as the networking data plane. Open vSwitch is a high-performance, programmable virtual switch that supports Linux and Windows. Open vSwitch enables Antrea to implement Kubernetes NetworkPolicies in a high-performance and efficient manner. Thanks to the programmable characteristic of Open vSwitch, Antrea is able to implement an extensive set of networking and security features and services on top of Open vSwitch.

Some information in this white paper, in particular when it comes to the Antrea Agent, is specific to running Antrea on Linux nodes. For information about how to run Antrea on Windows nodes, please refer to *windows-design.md*.

Components

In a Kubernetes cluster, Antrea creates a deployment that runs the Antrea Controller, and a DaemonSet that includes two containers to run the Antrea Agent and OVS daemons respectively, on every node. The DaemonSet also includes an init container that installs the container network interface (CNI) plug-in, antrea-cni, on the node and ensures that the OVS kernel module is loaded and chained with the portmap CNI plug-in. The Antrea Controller, Agent, OVS daemons and antrea-cni bits are included in a single Docker image. Antrea also has a command-line tool, called antctl, and an *Octant* UI plug-in.

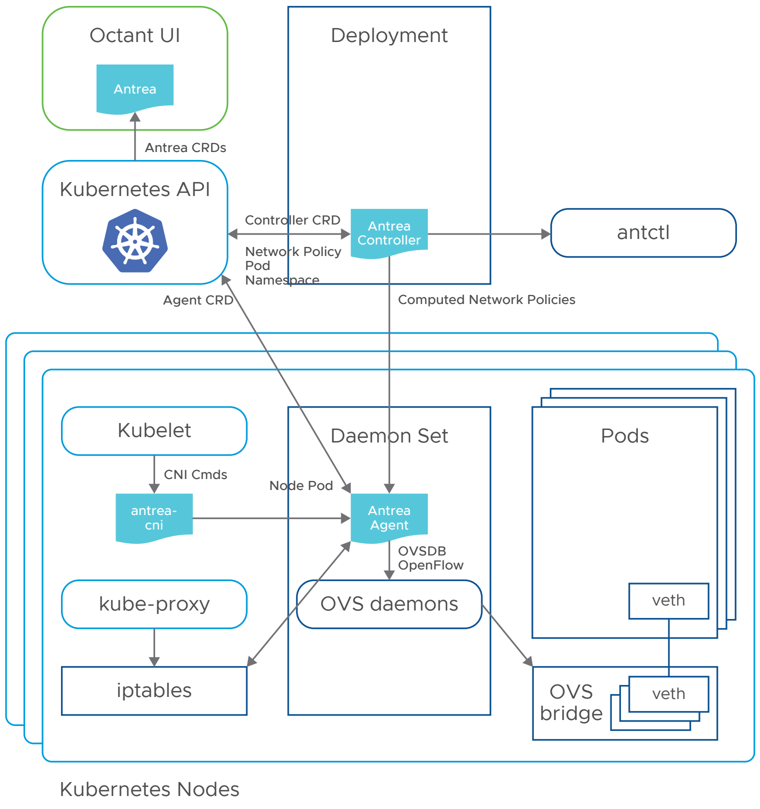

FIGURE 1: Antrea architecture and the interactions between the Antrea Controller and the Kubernetes API.

Antrea Controller

The Antrea Controller watches NetworkPolicy, pod and namespace resources from the Kubernetes API, computes NetworkPolicies and distributes the computed policies to all Antrea Agents. Currently, the Antrea Controller supports only a single replica and mainly exists for NetworkPolicy implementation. If you only care about connectivity between pods, but not NetworkPolicy support, you may choose not to deploy the Antrea Controller at all. However, in the future, Antrea might support more features that require the Antrea Controller.

The Antrea Controller leverages the *Kubernetes apiserver library* to implement the communication channel to Antrea Agents. Each Antrea Agent connects to the Controller API server and watches the computed NetworkPolicy objects. The Controller also exposes a REST API for antctl on the same HTTP endpoint. See more information about the Controller API server implementation in the *Controller API server section*.

Controller API server

The Antrea Controller leverages the Kubernetes apiserver library to implement its own API server. The API server implementation is customized and optimized for publishing the computed NetworkPolicies to Agents:

- The API server keeps all the states in in-memory caches and does not require a datastore to persist the data.
- It sends the NetworkPolicy objects to only those nodes that need to apply the NetworkPolicies locally. A node receives a NetworkPolicy only if the NetworkPolicy is applied to at least one pod on the node.
- It supports sending incremental updates to the NetworkPolicy objects to Agents.
- Messages between the Controller and the Agent are serialized using the Protobuf format for reduced size and higher efficiency.

The Antrea Controller API server also leverages Kubernetes services for:

- Service discovery
- Authentication and authorization

The Controller API endpoint is exposed through a Kubernetes ClusterIP type service. The Antrea Agent gets the service's ClusterIP from the service environment variable and connects to the Controller API server using the ClusterIP. The Controller API server delegates authentication and authorization to the Kubernetes API—the Antrea Agent uses a Kubernetes ServiceAccount token to authenticate to the Controller, and the Controller API server validates the token and whether the ServiceAccount is authorized for the API request with the Kubernetes API.

The Antrea Controller also exposes a REST API for antctl using the API server HTTP endpoint. It leverages *Kubernetes API aggregation* to enable antctl to reach the Antrea Controller API through the Kubernetes API. antctl connects and authenticates to the Kubernetes API, which will proxy the antctl API requests to the Antrea Controller. In this way, antctl can be executed on any machine that can reach the Kubernetes API, and it can leverage the kubectl configuration (kubeconfig file) to discover the Kubernetes API and authentication information. See the *antctl section*.

Antrea Agent

The Antrea Agent manages the OVS bridge and pod interfaces, and implements pod networking with OVS on every Kubernetes node.

The Antrea Agent exposes a gRPC service (CNI service), which is invoked by the antrea-cni binary to perform CNI operations. For each new pod to be created on the node, after getting the CNI ADD call from antrea-cni, the Agent creates the pod's network interface, allocates an IP address, connects the interface to the OVS bridge, and installs the necessary flows in OVS. To learn more about the OVS flows, read the *OVS pipeline doc*.

The Antrea Agent includes two Kubernetes controllers:

• The node controller watches the Kubernetes API server for new nodes and creates an OVS (Geneve/VXLAN/GRE/STT) tunnel to each remote node.

• The NetworkPolicy controller watches the computed NetworkPolicies from the Antrea Controller API and installs OVS flows to implement the NetworkPolicies for the local pods.

The Antrea Agent also exposes a REST API on a local HTTP endpoint for antctl.

OVS daemons

The two OVS daemons—ovsds-server and ovs-vswitched—run in a separate container, called antrea-ovs, of the Antrea Agent DaemonSet.

antrea-cni

antrea-cni is the *CNI* plug-in binary of Antrea. It is executed by kubelet for each CNI command. It is a simple gRPC client that issues an RPC to the Antrea Agent for each CNI command. The Agent performs the actual work (sets up networking for the pod) and returns the result or an error to antrea-cni.

antctl

antctl is a command-line tool for Antrea. At the moment, it can show basic runtime information for the Antrea Controller and the Antrea Agent for debugging purposes.

When accessing the Controller, antctl invokes the Controller API to query the required information. As previously described, antctl can reach the Controller API through the Kubernetes API, and have the Kubernetes API authenticate, authorize and proxy the API requests to the Controller. antctl also can be executed through kubectl as a kubectl plug-in.

When accessing the Agent, antctl connects to the Agent's local REST endpoint and can only be executed locally in the Agent's container.

Octant UI plug-in

Antrea also implements an Octant plug-in, which can show the Controller and Agent's health and basic runtime information in the Octant UI. The Octant plug-in gets the Controller and Agent's information from the AntreaControllerInfo and AntreaAgentInfo Custom Resource Definition (CRD) in the Kubernetes API. A CRD is created by the Antrea Controller and each Antrea Agent to populate their health and runtime information.

Pod networking

Pod interface configuration and IP address management (IPAM)

On every node, the Antrea Agent creates an OVS bridge (named br-int by default), and creates a veth pair for each pod, with one end being in the pod's network namespace and the other connected to the OVS bridge. On the OVS bridge, the Antrea Agent also creates an internal port—antrea-gw0 by default—to be the gateway of the node's subnet, and a tunnel port—antrea-tun0—for creating overlay tunnels to other nodes.

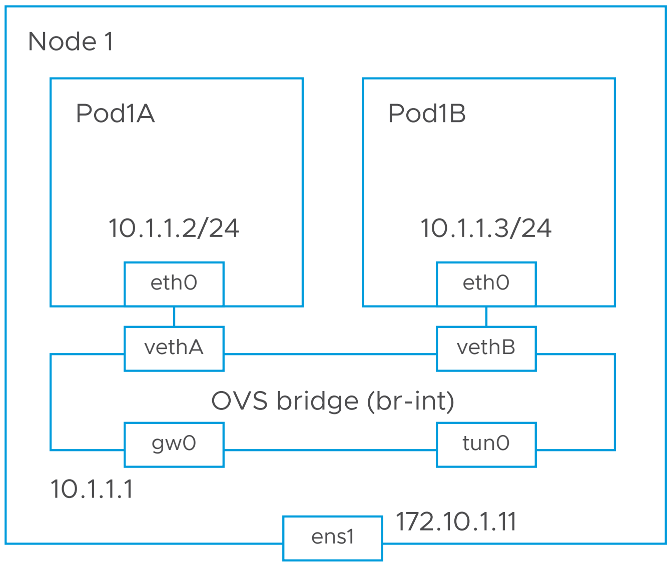

FIGURE 2: Each pod on the node receives a unique IP address.

Each node is assigned a single subnet, and all pods on the node get an IP from the subnet. Antrea leverages Kubernetes' NodeIPAMController for the node subnet allocation, which sets the podCIDR field of the Kubernetes node spec to the allocated subnet. The Antrea Agent retrieves the subnets of nodes from the podCIDR field. It reserves the first IP of the local node's subnet to be the gateway IP and assigns it to the antrea-gw0 port, and invokes the *host-local IPAM plug-in* to allocate IPs from the subnet to all local pods. A local pod is assigned an IP when the CNI ADD command is received for that pod.

Traffic walk

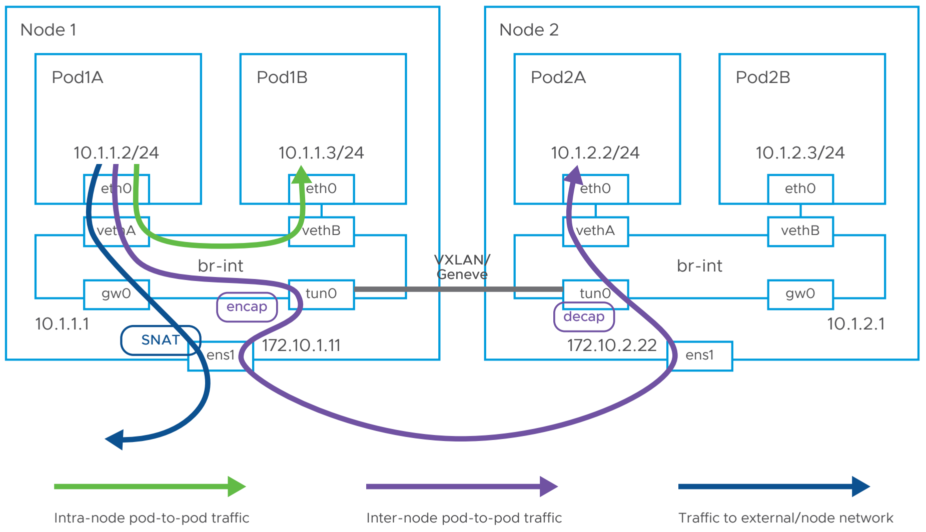

FIGURE 3: Types of connectivity in Kubernetes networking.

For every remote node, the Antrea Agent adds an OVS flow to send the traffic to that node through the appropriate tunnel. The flow matches the packet's destination IP against each node's subnet:

- Intra-node traffic – Packets between two local pods will be forwarded by the OVS bridge directly.
- Inter-node traffic – Packets to a pod on another node will be first forwarded to the antrea-tun0 port, encapsulated and sent to the destination node through the tunnel. Then, they will be decapsulated, injected through the antrea-tun0 port to the OVS bridge and, finally, forwarded to the destination pod.
- Pod to external traffic – Packets sent to an external IP or the node's network will be forwarded to the antrea-gw0 port (as it is the gateway of the local pod subnet), routed (based on routes configured on the node) to the appropriate network interface of the node (e.g., a physical network interface for a bare-metal node), and sent out to the node network from there. The Antrea Agent creates an iptables (MASQUERADE) rule to perform SNAT on the packets from pods, so their source IP will be rewritten to the node's IP before going out.

ClusterIP service

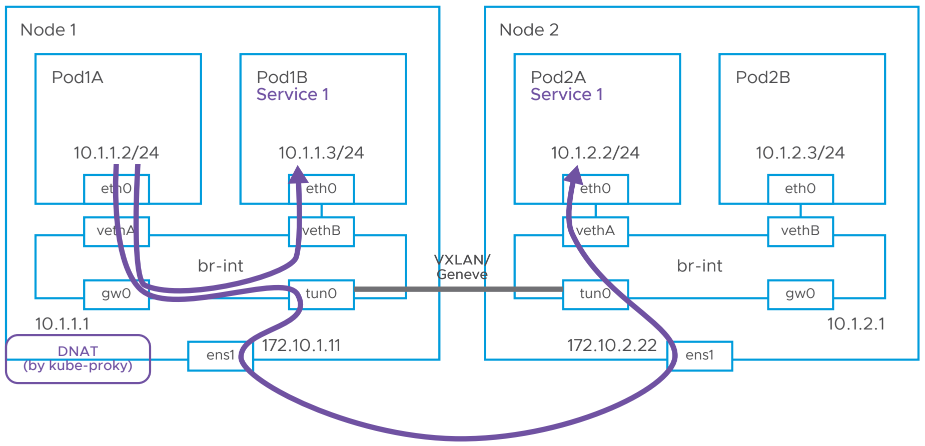

FIGURE 4: Connecting applications to services in Kubernetes.

At the moment, Antrea leverages kube-proxy to serve traffic for ClusterIP and NodePort type services. The packets from a pod to a service's ClusterIP will be forwarded through the antrea-gw0 port, then kube-proxy will select one service back-end pod to be the connection's destination and DNAT the packets to the pod's IP and port. If the destination pod is on the local node, the packets will be forwarded to the pod directly. If it is on another node, the packets will be sent to that node via the tunnel.

kube-proxy can be used in any supported mode: user-space iptables or IPVS. See the *Kubernetes Service documentation* for more details.

In the future, Antrea will also implement a ClusterIP service with OVS and make it a configurable option. This should help to achieve better performance than kube-proxy when kube-proxy is used in the user-space or iptables modes.

NetworkPolicy

An important design choice Antrea took regarding the NetworkPolicy implementation is centralized policy computation. The Antrea Controller watches NetworkPolicy, pod and namespace resources from the Kubernetes API. It processes podSelectors, namespaceSelectors and ipBlocks as follows:

- PodSelectors directly under the NetworkPolicy spec (which define the pods to which the NetworkPolicy is applied) will be translated to member pods.
- Selectors (podSelectors and namespaceSelectors) and ipBlocks in rules (which define the ingress and egress traffic allowed by this policy) will be mapped to pod IP addresses/IP address ranges.

The Antrea Controller also computes which nodes need to receive a NetworkPolicy. Each Antrea Agent receives only the computed policies that affect pods running locally on its node, and directly uses the IP addresses computed by the Controller to create OVS flows, enforcing the specified NetworkPolicies.

The following are the major benefits of a centralized computation approach:

- Only one Antrea Controller instance needs to receive and process all NetworkPolicy, pod and namespace updates, and compute podSelectors and namespaceSelectors. This has a much lower overall cost compared to watching these updates and performing the same complex policy computation on all nodes.
- It could enable scale-out of Controllers with multiple Controllers working together on the NetworkPolicy computation, each one being responsible for a subset of NetworkPolicies (although, Antrea currently supports only a single Controller instance).
- The Antrea Controller is the single source of NetworkPolicy computation. It is much easier to achieve consistency among nodes and easier to debug the NetworkPolicy implementation.

As previously described, the Antrea Controller leverages the Kubernetes apiserver library to build the API and communication channel to Agents.

IPsec encryption

Antrea supports encrypting GRE tunnel traffic with the IPsec encapsulating security payload (ESP). The IPsec implementation leverages *OVS IPsec* and uses *strongSwan* as the IKE daemon.

To enable IPsec, an extra container—antrea-ovs-ipsec—must be added to the Antrea Agent DaemonSet, which runs the ovs-monitor-ipsec and strongSwan daemons. Antrea now only supports using a pre-shared key (PSK) for IKE authentication, and the PSK string must be passed to the Antrea Agent using an environment variable—ANTREA_IPSEC_PSK. The PSK string can be specified in the *Antrea IPsec deployment yaml*, which creates a Kubernetes Secret to save the PSK value and populates it to the ANTREA_IPSEC_PSK environment variable of the Antrea Agent container.

When IPsec is enabled, the Antrea Agent will create a separate GRE tunnel port on the OVS bridge for each remote node, and will write the PSK string and the remote node IP address to two OVS interface options of the tunnel interface. Then, ovs-monitor-ipsec can detect the tunnel and create IPsec security policies with PSK for the remote node, and strongSwan can create the IPsec security associations based on the security policies. These additional tunnel ports are not used to send traffic to a remote node—the tunnel traffic is still output to the default tunnel port (antrea-tun0) with OVS-flow-based tunneling. However, the traffic from a remote node will be received from the node's IPsec tunnel port.

FURTHER READING

Visit *antrea.io* for more information about the project and to read the latest blogs on Antrea.

Go to the *Antrea repo on GitHub* to get the source code and learn more about contributing to the project.

Hybrid, NoEncap, NetworkPolicyOnly TrafficEncapModes

Besides the default encap mode, which always creates overlay tunnels among nodes and encapsulates inter-node pod traffic, Antrea also supports other TrafficEncapModes, including Hybrid, NoEncap and NetworkPolicyOnly modes:

- **Hybrid** – When two nodes are in two different subnets, pod traffic between the two nodes is encapsulated. When the two nodes are in the same subnet, pod traffic between them is not encapsulated. Instead, the traffic is routed from one node to another. The Antrea Agent adds routes on the node to enable the routing within the same node subnet. For every remote node in the same subnet as the local node, the Agent adds a static route entry that uses the remote node IP as the next hop of its pod subnet. Hybrid mode requires the node network to allow packets with pod IPs to be sent out from the node's NICs.
- **NoEncap** – Pod traffic is never encapsulated. Antrea just assumes the node network can handle routing of pod traffic across nodes. Typically, this is achieved by the Kubernetes cloud provider implementation, which adds routes for pod subnets to the node network routers. The Antrea Agent still creates static routes on each node for remote nodes in the same subnet, which is an optimization that routes pod traffic directly to the destination node without going through the extra hop of the node network router. The Antrea Agent also creates the iptables (MASQUERADE) rule for SNAT of pod-to-external traffic. *Antrea supports Google Kubernetes Engine* with NoEncap mode.
- **NetworkPolicyOnly** – Inter-node pod traffic is neither tunneled nor routed by Antrea. Antrea just implements NetworkPolicies for pod traffic but relies on another cloud CNI and cloud network to implement pod IPAM and cross-node traffic forwarding. Refer to the *NetworkPolicyOnly mode design doc* for more information. *Antrea for the Azure Kubernetes Service engine* and *Antrea support for the Amazon Elastic Kubernetes Service* work in NetworkPolicyOnly mode.