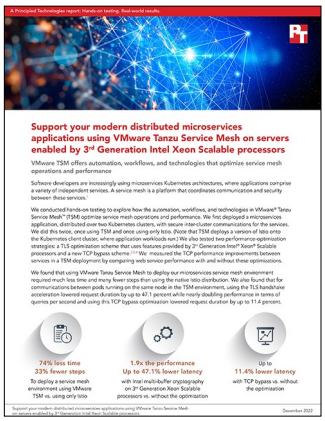

The science behind the report:

Support your modern distributed microservices applications using VMware Tanzu Service Mesh on servers enabled by 3rd Generation Intel Xeon Scalable processors

This document describes what we tested, how we tested, and what we found. To learn how these facts translate into real-world benefits, read the report Support your modern distributed applications using VMware Tanzu Service Mesh on servers enabled by 3rd Generation Intel Xeon Scalable processors.

We concluded our hands-on testing on September 5, 2022. During testing, we determined the appropriate hardware and software configurations and applied updates as they became available. The results in this report reflect configurations that we finalized on July 15, 2022 or earlier. Unavoidably, these configurations may not represent the latest versions available when this report appears.

Our results

To learn more about how we have calculated the wins in this report, go to http://facts.pt/calculating-and-highlighting-wins. Unless we state otherwise, we have followed the rules and principles we outline in that document.

Table 1: Time and steps required to deploy a microservices application on a multi-cluster service mesh.

	Istio	VMware TSM	Percentage improvement with VMware TSM
Time	0:24:27	0:06:15	74.4%
Number of steps	45	30	33.3%

Table 2: Performance impact of enabling Intel multi-buffer cryptography in TSM with Intel 3rd Generation Xeon Scalable processors.

	Without Intel multi-buffer cryptography	With Intel multi-buffer cryptography	Percentage improvement with Intel multi-buffer cryptography
99th percentile latency in milliseconds	312	165	47.1%
Queries per second (higher is better)	3,304	6,339	91.8%

Table 3: Performance impact of enabling TCP bypass in TSM at various thread counts.

Thread count	99th percentile latency with TCP bypass (lower is better)	99th percentile latency without TCP bypass (lower is better)	Percentage improvement with TCP bypass
16	2.428	2.556	5.0%
32	3.964	4.478	11.4%
64	7.873	8.677	9.2%
128	15.900	17.194	7.5%

System configuration information

Table 4: Detailed information on the systems we tested.

System configuration information	Dell PowerEdge R650
Number of servers	10
Purpose	SUT
BIOS name and version	Dell 1.6.5
Non-default BIOS settings	No hyperthreads
Operating system name and version	VMware® ESXi 7.0.3, 18905247
Date of last OS updates/patches applied	08/16/21
Power management policy	Performance
Processor	
Number of processors	2
Vendor and model	Intel® Xeon® Gold 6330 CPU
Core count (per processor)	28
Core frequency (GHz)	2.00
Stepping	6
Memory module(s)	
Total memory in system (GB)	256
Number of memory modules	8
Vendor and model	Hynix® HMAA4GR7AJR8N-XN
Size (GB)	32
Type	PC4-25600R
Speed (MHz)	3,200
Speed running in the server (MHz)	3,200
Storage controller 1	
Vendor and model	Dell HBA355i Front
Cache size (GB)	N/A
Firmware version	17.15.08.00
Driver version	N/A
Storage controller 2	
Vendor and model	BOSS-S2
Cache size (GB)	N/A
Firmware version	2.5.13.4008
Driver version	N/A

System configuration information	Dell PowerEdge R650
Local storage	
Purpose	OS and application
Controller	BOSS-S2
Number of drives	2
Vendor and model	Micron® MTFDDAV240TDU
Drive size(GB)	240
Drive information (interface, type)	SATA, SSD
Network adapter 1	
Vendor and model	Intel Ethernet 25G 2P E810-XXV
Number and type of ports	2 x 25GbE
Driver version	ice, 0.8.1-k
Network adapter 2	
Vendor and model	Broadcom® Gigabit Ethernet BCM5720
Number and type of ports	2 x 1 GbE
Driver version	tg3, 3.137
Vendor and model	Dell
Number of cooling fans	16
Vendor and model	Dell 01CW9GA04
Number of power supplies	2
Wattage of each (W)	1,400

Table 5: Detailed information on the systems we tested.

System configuration information	Dell EMC PowerSwitch S4048-ON
Firmware revision	9.14.2.11
Number and type of ports	48x 10GbE SFP+, 6x QSFP+
Number and type of ports used in test	11 x 10GbE SFP+
Non-default settings used	Jumbo frames enabled

How we tested

Note: The configuration files, e.g. to create the Kubernetes clusters, are in the section called Files. We use TSM version 5.0.7 in these tests.

Servers and auxiliary VM

We deployed 10 Dell PowerEdge R650 servers. We assigned eight of them to two four-node clusters and used the remaining two servers to support the AVI load balancer. Each of the eight cluster servers had two network connections: one 1Gbps connection for management, and one 10Gbps connection for Kubernetes traffic. The two AVI servers each had the 1Gbps management interface, and two 10Gpbs interfaces, one to each cluster network. The management network could connect to the internet.

We used an auxiliary VM for controlling Kubernetes, deploying Istio, and running the performance tests. We installed Ubuntu 20.04 on this VM, and it has one network connection to the management network, and two 10Gpbs interfaces, one to each cluster network.

Configuring the eight cluster servers

1. Deploy Ubuntu 20.04 on the servers.
2. Configure the time synchronization service:

```
echo "NTP=10.40.0.1" | sudo tee -a /etc/systemd/timesyncd.conf
sudo systemctl enable systemd-timesyncd
sudo systemctl restart systemd-timesyncd
```

3. Install additional packages for the Docker runtime:

```
sudo apt install -y apt-transport-https ca-certificates curl
sudo apt install -y gnupg-agent software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_
release -cs) stable"
sudo apt update
sudo apt remove -y docker docker-engine docker.io containerd runc
sudo apt install -y docker-ce docker-ce-cli containerd.io
sudo systemctl enable docker.service  containerd.service
sudo groupadd docker
sudo usermod -aG docker $USER
newgrp docker
```

4. Configure the server for Kubernetes:

```
swapoff -a
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
echo " deb https://apt.kubernetes.io/ kubernetes-xenial main" |\
   sudo tee /etc/apt/sources.list.d/kubernetes.list
```

5. Install the components for Kubernetes 1.21.0:

```
sudo apt update
sudo apt-get install -y kubelet=1.21.0-00 kubeadm=1.21.0-00 kubectl=1.21.0-00
sudo apt-mark hold kubelet=1.21.0-00 kubeadm=1.21.0-00 kubectl=1.21.0-00
sudo kubeadm config images pull --kubernetes-version=1.21.0
```

6. Append the following host table to /etc/hosts:

```
192.168.127.101 sut01
192.168.127.103 sut03
192.168.127.105 sut05
192.168.127.107 sut07
192.168.255.102 sut02
192.168.255.104 sut04
192.168.255.106 sut06
192.168.255.108 sut08
```

7. Install kubectl 1.21.0 on the auxiliary VM and synchronized its system time to the same time source as in step 2.

Configuring the two AVI servers

1. Configure the time synchronization service:

```
echo "NTP=10.40.0.1" | sudo tee -a /etc/systemd/timesyncd.conf
sudo systemctl enable systemd-timesyncd
sudo systemctl restart systemd-timesyncd
```

2. Install additional packages for the Docker runtime:

```
sudo apt install -y apt-transport-https ca-certificates curl
sudo apt install -y gnupg-agent software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_
release -cs) stable"
sudo apt update
sudo apt remove -y docker docker-engine docker.io containerd runc
sudo apt install -y docker-ce docker-ce-cli containerd.io
sudo systemctl enable docker.service  containerd.service
sudo groupadd docker
sudo usermod -aG docker $USER
newgrp docker
```

3. Install python2 and the jinja2 package:

```
sudo apt install python2
wget https://bootstrap.pypa.io/pip/2.7/get-pip.py
sudo python2 get-pip.py
sudo pip2 install jinja2==2.3q
```

On the server that runs the AVI controller, we performed the following steps:

4. Transfer the AVI controller installation package, version 21.1.4. to this server.
5. Untar the package:

```
tar -xf docker_install-21.1.4-2p3-9009.tar.gz
cd /docker_install-21.1.4-2p3-900
```

6. Install the AVI controller:

```
sudo python2 ./avi_baremetal_setup.py
sudo systemctl start avicontroller
```

On the server that runs the AVI service engine, we performed the following steps:

7. Create a user called ptuseralb:

```
sudo useradd ptuseralb
sudo passwd ptuseralb
```

8. Add the following line to /etc/sudoers so the user has passwordless access to sudo:

```
ptuseralb ALL=(ALL) NOPASSWD: ALL
```

We logged into the AVI controller interface and performed the following configuration steps:

9. From the Administration tab, create new user credentials resource for user ptuseralb and generate a new SSH key pair. We downloaded the new public key and copied to /home/ptuseralb/.ssh/ authorized_keys on the server that runs the AVI service engine.
10. From the Infrastructure Tab, create a new cloud called istio-cloud of type Linux Server that uses SSH user ptusetlb.
11. From the Infrastructure Tab, create a new Service Engine Group called istio-group in Cloud Istio-cloud.
12. From the Infrastructure tab, create a new Service Engine in Cloud istio-cloud by editing its configuration:
    a. Set its Template Service Engine Group to Istio-group.
    b. Click add Servers and enter the IP address for the management interface for the server that will run the AVI service engine.
    c. Click Save.
13. Pause until the new service engine on the dashboard reaches a green state.
14. From the Infrastructure tab, create network NET-istio-01 in cloud Istio-cloud with DHCP enabled and with IP subnet of 10.34.220.0/24.
15. From the Infrastructure tab, create network NET-istio-01 in cloud Istio-cloud with DHCP enabled and with IP subnet of 10.34.221.0/24.
16. From the Templates tab, create an IPAM profile called istio-IPAM in Cloud istio-cloud and add two networks: NET-IPAM, and NET-istio01.
17. From the Template tab, create an IPAM profile called isto-DNS in cloud Istio-cloud, and add domain acme.example
18. Add the new IAM and DNS profiles to the Istio-cloud configuration.

Deploying the Kubernetes clusters

For our investigation in the amount of time and effort needed to deploy a microservices application, we created two four-node Kubernetes clusters on separate networks (see steps 1 through 10, below). We created routes on each node so that nodes in each cluster could reach each other's cluster network.

For the TCP bypass and TLS-acceleration performance testing, we used one four-node Kubernetes cluster (see steps 1 through 5, below).

1. Log onto the control-plane node for cluster 1, `sut01, as user ptuser`
2. Install Kubernetes on the four nodes in this cluster using the `kubeadm` parameters in file `kubeadm-config-01.yaml`.

```
sudo kubeadm init --config ./kubeadm-config-01.yaml
```

3. Copy the cluster context to the local user:

```
mkdir -p ~/.kube
sudo cp /etc/kubernetes/admin.conf ~/.kube/config
sudo chown ptuser:ptuser $HOME/ ~/.kube/config
chmod 600 ~/.kube/config
```

4. Add Antrea networking to the cluster:

```
kubectl taint node sut01 node-role.kubernetes.io/master:NoSchedule-
kubectl apply -f https://github.com/antrea-io/antrea/releases/download/v1.5.1/antrea.yml
kubectl -n kube-system rollout status ds antrea-agent -w
kubectl -n kube-system rollout status deploy antrea-controller -w
```

5. Add the three worker nodes to cluster 1:

```
join=$(kubeadm token create --print-join-command)
for i in sut{03,05,07}; do
  echo $i
  ssh $i sudo "$join"
done
```

6. Log onto the control-plane node for cluster 2, `sut02, as user ptuser`
7. Install Kubernetes on the four nodes in this cluster using the `kubeadm` parameters in file `kubeadm-config-02.yaml`.

```
sudo kubeadm init --config ./kubeadm-config-02.yaml
```

8. Copy the cluster context to the local user:

```
mkdir -p ~/.kube
sudo cp /etc/kubernetes/admin.conf ~/.kube/config
sudo chown ptuser:ptuser $HOME/ ~/.kube/config
chmod 600 ~/.kube/config
```

9. Add Antrea networking to the cluster:

```
kubectl taint node sut02 node-role.kubernetes.io/master:NoSchedule-
kubectl apply -f https://github.com/antrea-io/antrea/releases/download/v1.5.1/antrea.yml
kubectl -n kube-system rollout status ds antrea-agent -w
kubectl -n kube-system rollout status deploy antrea-controller -w
```

10. Add the three worker nodes to the cluster:

```
join=$(kubeadm token create --print-join-command)
for i in sut{03,05,07}; do
  echo $i
  ssh $i sudo "$join"
done
```

Deploying the AVI Kubernetes Operator (AKO)

We performed the following steps on each cluster's control-plane node (sut01 or sut03).

1. Install helm version 3.9.0:

```
wget https://get.helm.sh/helm-v3.9.0-linux-amd64.tar.gz
tar -xf helm-v3.9.0-linux-amd64.tar.gz
sudo install linux-amd64/helm /usr/local/bin/helm
```

2. Add the AKO helm repository:

```
helm repo add ako https://avinetworks.github.io/avi-helm-charts/charts/stable/ako
```

3. Install the AKO operator:

   - For cluster 1,

```
helm install ako/ako --generate-name -f values-ako01.yaml \
  --namespace=avi-system --create-namespace
```

   - For cluster 2,

```
helm install ako/ako --generate-name -f values-ako02.yaml \
  --namespace=avi-system --create-namespace
```

Testing ease of deploying a multi-cluster microservices application

Performing prerequisite tasks common to both environments

We installed Kubernetes 1.21 on both four-node clusters, and users and containers in the clusters can communicate to each other's pod networks. Each cluster has a separate console terminal from which we can execute shell, kubectl, or itioctl commands. In addition, it is convenient to have a utility host for deploying Istio to both clusters.

Prepare the Google Online Boutique microservices demo application on both two-cluster mesh before installing TSM or Istio mesh. The application won't be operative until after the mesh is deployed.

1. Download version 0.3.9 of the microservices demo from https://github.com/GoogleCloudPlatform/microservices-demo/releases, and untar the archive.
2. From this single-cluster application, divide its microservices between the two cluster as follows:
    a. cluster 1: adservice, checkoutservice, currencyservice, emailservice, paymentservice, shippingservice
    b. cluster 2: frontend, loadgenerator, productcatalogservice, recommendationservice, redis-cart
3. From the file `microservices-demo-0.3.9/release/kubernetes-manifests.yaml`, extract and separate its deployments into files `cluster1-deployment.yaml`, and `cluster2-deployment.yaml`.
4. From this file, extract and separate its services into files `cluster1-services.yaml`, and `cluster2-services.yaml`.

Installing and configuring Istio service mesh

Installing Istio mesh on the clusters and deploying the Online Boutique application

1. Log onto the deployment host.
2. Download Istio version 1.12.0:

```
curl -L https://istio.io/downloadIstio | ISTIO_VERSION=1.12.0 TARGET_ARCH=x86_64 sh -
```

3. Copy cluster 1's Kubernetes config file to this host:

```
scp sut01:.kube/config config-01
```

4. Copy cluster 2's Kubernetes config file to this host:

```
scp sut02:.kube/config config-02
```

5. Modify the default usernames in cluster 1's config file:

```
sed -i 's/\<kubernetes-admin\>/admin1/g' config-01
```

6. Modify the default usernames in cluster 2's config file:

```
sed -i 's/\<kubernetes-admin\>/admin2/g' config-02
```

7. Create the default location for the local Kubernetes config file:

```
mkdir ~/.kube
```

8. Combine the config files into one:

```
KUBECONFIG=~/config-01:~/config-02 kubectl config view --flatten > ~/.kube/config
```

9. If the appropriate Kubernetes context variables have not been set, create them:

```
export C01=admin1@cluster1
export C02=admin2@cluster2
```

10. Navigate to the directory where you downloaded Istio:

```
cd ~/istio-1.12.0/
```

11. Create the certificate directory:

```
mkdir certs
```

12. Navigate to the certificate directory:

```
cd certs
```

13. Create the root certificate:

```
make -f ../tools/certs/Makefile.selfsigned.mk root-ca
```

14. Create cluster 1's certificate:

```
make -f ../tools/certs/Makefile.selfsigned.mk cluster1-cacerts
```

15. Create cluster 2's certificate:

```
make -f ../tools/certs/Makefile.selfsigned.mk cluster2-cacerts
```

16. Create the `istio-system` namespace in the first cluster:

```
kubectl --context=$C01 create ns istio-system
```

17. Create the `istio-system` namespace in the second cluster:

```
kubectl --context=$C02 create ns istio-system
```

18. Add your created secrets to the first Istio cluster:

```
kubectl --context=$C01 -n istio-system create secret generic cacerts \
  --from-file=cluster1/ca-cert.pem \
  --from-file=cluster1/ca-key.pem \
  --from-file=cluster1/root-cert.pem \
  --from-file=cluster1/cert-chain.pem
```

19. Install Istio in the first Istio cluster:

```
istioctl --context=$C01 install -y
```

20. Allow Istio sidecar injection in the application's namespace of the first cluster:

```
kubectl --context=$C01 create ns pinnacle
kubectl --context=$C01 label namespace pinnacle istio-injection=enabled
```

21. Add your created secrets to the second Istio cluster:

```
kubectl --context=$C02 -n istio-system create secret generic cacerts \
  --from-file=cluster2/ca-cert.pem \
  --from-file=cluster2/ca-key.pem \
  --from-file=cluster2/root-cert.pem \
  --from-file=cluster2/cert-chain.pem
```

22. Install Istio in the second Istio cluster:

```
istioctl --context=$C02 install -y
```

23. Allow Istio sidecar injection in the application's namespace of the second cluster:

```
kubectl --context=$C02 create ns pinnacle
kubectl --context=$C02 label namespace pinnacle istio-injection=enabled
```

Installing Multi-Primary on different networks

1. Navigate to the Istio directory:

```
cd ~/istio-1.12.0/
```

2. Add a label to the first cluster's networking:

```
kubectl --context=$C01 label namespace istio-system topology.istio.io/network=network1
```

3. Install an Istio operator into the first network:

```
cat <<EOF | istioctl --context=$C01 install -y -f -
apiVersion: install.istio.io/v1alpha1
kind: IstioOperator
spec:
  values:
    global:
      meshID: mesh1
      multiCluster:
        clusterName: cluster1
      network: network1
EOF
```

4. Use the included Istio script to create an East-West gateway for the first cluster using the previously created Istio operator:

```
samples/multicluster/gen-eastwest-gateway.sh --network network1 |\
  istioctl --context=$C01 install -y -f -
```

5. Verify the East-West gateway is running before continuing:

```
kubectl --context=$C01 get svc istio-eastwestgateway -n istio-system
```

6. Enable traffic for the East-West gateway:

```
kubectl --context=$C01 apply -n istio-system -f samples/multicluster/expose-services.yaml
```

7. Add a label to the second cluster's networking:

```
kubectl --context=$C02 label namespace istio-system topology.istio.io/network=network2
```

8. Install an Istio operator into the second network:

```
cat <<EOF | istioctl --context=$C02 install -y -f -
apiVersion: install.istio.io/v1alpha1
kind: IstioOperator
spec:
  values:
    global:
      meshID: mesh1
      multiCluster:
        clusterName: cluster2
      network: network2
EOF
```

9. Use the included Istio script to create an East-West gateway for the second cluster using the previously created Istio operator:

```
samples/multicluster/gen-eastwest-gateway.sh --network network2 |\
  istioctl --context=$C02 install -y -f -
```

10. Verify the East-West gateway is running before continuing:

```
kubectl --context=$C02 get svc istio-eastwestgateway -n istio-system
```

11. Enable traffic for the East-West gateway:

```
kubectl --context=$C02 apply -n istio-system -f samples/multicluster/expose-services.yaml
```

12. Create a secret for the first cluster's gateway and add it to the second cluster:

```
istioctl --context=$C01 x create-remote-secret --name=cluster1 |\
  kubectl --context=$C02 apply -f -
```

13. Create a secret for the second cluster's gateway and add it to the first cluster:

```
istioctl --context=$C02 x create-remote-secret --name=cluster2 |\
  kubectl --context=$C01 apply -f -
```

Making application specific-modifications to Istio's default settings

1. Apply the following Istio security settings to each cluster to ensure mTLS is used between all service endpoints and gateways:

```
cat << EOF1 | kubectl --context $C01 apply -f -
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: mtls-pinnacle
spec:
  mtls:
    mode: STRICT
apiVersion: networking.istio.io/v1beta1
---
kind: DestinationRule
metadata:
  name: mtls-pinnacle-rule
spec:
  host: *
  exportTo:
    - pinnacle
  trafficPolicy:
    tls:
      mode: ISTIO_MUTUAL
EOF1
cat << EOF2 | kubectl --context $C02 apply -f -
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: mtls-pinnacle
spec:
  mtls:
    mode: STRICT
apiVersion: networking.istio.io/v1beta1
---
kind: DestinationRule
metadata:
  name: mtls-pinnacle-rule
spec:
  host: *
  exportTo:
    - pinnacle
  trafficPolicy:
    tls:
      mode: ISTIO_MUTUAL
EOF2
```

Deploying the Online Boutique application to the mesh

1. Deploy the cluster 1 microservices:

```
kubectl --context $C01 -n pinnacle apply -f cluster1-deployment.yaml -f cluster1-services.yaml
```

2. Create place-holder services on cluster 1 from cluster 2's services:

```
kubectl --context $C01 -n pinnacle apply -f cluster1-deployment.yaml -f cluster1-services.yaml
```

3. Deploy the cluster 2 microservices:

```
kubectl --context $C01 -n pinnacle apply -f cluster2-services.yaml
```

4. Create place-holder services on cluster 2 from cluster 1's services:

```
kubectl --context $C02 -n pinnacle apply -f cluster1-services.yaml
```

5. Create the application's Istio gateway and virtual service on cluster 2 using the file `microservices-demo-0.3.9/release/istio-defaults.yaml`.
6. `Change the name of the virtual service's host to frontend.pinnacle`

```
sed -i 's/frontend.default/frontend.pinnacle/' istio-defaults.yaml
kubectl --context $C02 apply -f istio-defaults.yaml -n pinnacle
```

Verifying the Online Boutique application works in the two-cluster Istio service mesh.

1. Determine the internet address for the application, which is the external address of cluster 2's ingress gateway.

```
kubectl --context $C02 get svc -n istio-system | grep ingress
```

2. Open a web browser to the application's internet address, and add products to the sales cart, etc.

Installing and configuring VMware TSM

Installing TSM

This installation assumes that the user has already signed up for TSM, and the user is onboarding fresh Kubernetes clusters into TSM. The user has console access to each cluster, and the ability to use `kubectl` to apply changes to each cluster.

We will deploy the Online boutique microservices demo to the `acme namespace`. So, from the console for each cluster, create this namespace; e.g., `kubectl create ns acme`.

1. Log in to the Tanzu Service Mesh website.
2. In the TSM home page, click Open Onboarding Panel.
3. In the Onboard Clusters page, enter your first cluster name and click Generate Security Token.
4. Copy the registration YAML command from the generated field.
5. In a console for cluster 1, paste the command into the console, and press Enter.
6. Copy the security token creation command from the second generated field.
7. In a console for cluster 1, paste the command into the console, and press Enter.
8. Click Install Tanzu Service Mesh.
9. Click Onboard Another Cluster.
10. In the Onboard Clusters page, enter your second cluster name and click Generate Security Token.
11. Copy the registration YAML command from the generated field.
12. In a console for cluster 2, paste the command into the console, and press Enter.
13. Copy the security token creation command from the second generated field.
14. In a console for cluster 2, paste the command into the console, and press Enter.
15. Click Install Tanzu Service Mesh.

Creating the Global Namespace

1. In the TSM home page, click New Global Namespace.
2. In the General Details page, enter boutique for the namespace and acme.demo for the domain and click Next.
3. In Namespace Mapping, select cluster1 and pinnacle, then click Add Mapping Rule.
4. Select cluster2 and pinnacle, then click Next.
5. Leave External Services blank and click Next.
6. Leave Public Services blank and click Next.
7. In GSLB & Resiliency, leave defaults and click Next.
8. In Configuration Summary, click Finish to begin the Global Namespace creation. The dashboard will begin showing traffic between the two clusters immediately after.

Making application-specific modifications to TSM default settings

No changes are required.

Deploying the Online Boutique application to the mesh

1. Change the domain for the host name in the deployments to acme.demo:

```
sed -i 's/\(value: [a-z]*\):\([1-9[0-9]*\)$/\1.acme.demo:\2/' \
  cluster1-deployments_services.yaml cluster2-deployments_services.yaml
```

2. Deploy the cluster 1 microservices:

```
# in cluster 1's control terminal
kubectl -n pinnacle apply -f cluster1-deployment.yaml -f cluster1-services.yaml
```

3. Deploy the cluster 2 microservices:

```
# in cluster 2's control terminal
kubectl -n pinnacle apply -f cluster2-deployment.yaml -f cluster2-services.yaml
```

4. Create the application's Istio gateway and virtual service on cluster 2 using the file `microservices-demo-0.3.9/release/istio-defaults.yaml`.
5. Change the name of the virtual service's host to `frontend.pinnacle`

```
sed -i 's/frontend.default/frontend.pinnacle/' istio-defaults.yaml
kubectl --context $C02 apply -f istio-defaults.yaml -n pinnacle
```

Verifying the Online Boutique application works in TSM with two clusters

1. Determine the internet address for the application, which is the external address of cluster 2's ingress gateway. Use the console for cluster 2:

```
kubectl get svc -n istio-system | grep ingress
```

2. Open a web browser to the application's internet address, and add products to the sales cart, etc.

TLS-acceleration testing

We deleted the Kubernetes clusters and recreated one four-node cluster. We then deployed TSM as in section Installing TSM, above. We also installed and used `istioctl version 1.12.2`.

The TLS-acceleration optimization, at the time of this testing, was not integrated with TSM. Accordingly, one must modify the default TSM configuration to add container images with Intel multi-buffer cryptography libraries, remove two Envoy filters that interfere with these images, disable Istio CNI as there are no Intel images for it at present, and increase the CPU and memory allocations to the containers that will run Intel multi-buffer cryptography.

For this tests, the pods associated with the load generator, web server, and the virtual service all run on three different nodes.

1. Log onto the utility VM.
2. Delete the two Istio Envoy filters that interfere with Intel Istio binaries. All runs will occur without them.

```
kubectl -n istio-system delete envoyfilters istio-gateway istio-telemetry
```

3. Create cryptographic certificates for the microservices gateway:

```
mkdir -p certificates
openssl req -x509 -newkey rsa:2048 -nodes -subj /CN=localhost \
  -keyout certificates/key.pem -out certificates/cert.pem
chmod +r certificates/key.pem
kubectl create -n istio-system secret tls istio-gateway-secret \
  --key=certificates/key.pem --cert=certificates/cert.pem
```

4. Perform additional Kubernetes configuration for the test:

```
kubectl create ns fortio
kubectl create -n istio-system configmap k6-script-configmap \
  --from-file=k6.js=k6-no-connection-reuse-csv.js
```

5. Make additional configuration changes to the TSM configuration for the testing:
   a. Obtain the default TSM configuration:

```
istioctl manifest generate > tsm-default.yaml
```

   b. TLS-acceleration testing only: Patch the default file with the contents of `tsm-cryptomb.txt` to create the file `tsm-cryptomb.yaml`, and apply that configuration to the default TSM mesh:

```
cp tsm_default.yaml tsm-cryptomb.yaml
patch tsm-cryptomb.yaml tsm-cryptomb.txt
istioctl install -f tsm-cryptomb.yaml -y
```

   c. Baseline testing only: Patch the default file with the contents of tsm-baseline.txt to create the file tsm-baseline.yaml, and apply that configuration to the default TSM mesh:

```
cp tsm_default.yaml tsm-baseline.yaml
patch tsm-baseline.yaml tsm-baseline.txt
istioctl install -f tsm-baseline.yaml -y
```

6. Run the performance test:
   a. Start the Fortio server, and establish service to it:

```
kubectl apply -f fortio-server.yaml -f istio-gw.yaml -f istio-vs.yaml
```

   b. Start the web-client:

```
kubectl apply -f k6-loadgenerator-job.yaml
```

   c. Wait 240 seconds and gather the run's results:

```
kubectl logs -n istio-system -l app=loadgenerator
```

Testing the performance impact of TCP bypass

We deleted the Kubernetes clusters and recreated one four-node cluster. We then deployed TSM as in section Installing TSM, above.

For this test, the pods associated with the load generator, web server, and the virtual service all run on the same node.

1. Log onto the utility VM.
2. Install the Fortio client and webserver:
    a. Install go version 18.2:

```
wget https://go.dev/dl/go1.18.2.linux-amd64.tar.gz
sudo tar -C /usr/local -xf go1.18.2.linux-amd64.tar.gz
export PATH=$PATH:/usr/local/go/bin
```

    b. Compile Fortio version 1.25.0:

```
go install fortio.org/fortio@v1.25.0
```

3. Prepare the python environment for Fortio test-harness:
    a. Download the service-mesh benchmark-harness (performed on 18 August 2022).

```
git clone https://github.com/istio/tools.git
cd tools/perf/benchmark
```

    b. Prepare benchmark environment

```
export NAMESPACE=twopods-istio
export INTERCEPTION_MODE=REDIRECT
export ISTIO_INJECT=true
export LOAD_GEN_TYPE=fortio
export DNS_DOMAIN=v104.qualistio.org
./setup_test.sh
pipenv --three
pipenv shell
pipenv install
```

Running the performance test without TCP bypass

1. We performed steps 1 and 2 10 times, and identified the median run from the last three runs.
2. Perform the test:

```
python runner/runner.py --conn 8 --qps 100  --duration 240
```

3. Gather result metrics:

```
kubectl -n twopods-istio port-forward svc/fortioclient 9076:9076 &
export FORTIO_CLIENT_URL=http://localhost:9076
python ./runner/fortio.py $FORTIO_CLIENT_URL --csv \
StartTime,ActualDuration,Labels,NumThreads,ActualQPS,\
p50,p90,p99,p999,\
cpu_mili_avg_istio_proxy_fortioclient,\
cpu_mili_avg_istio_proxy_fortioserver,\
cpu_mili_avg_istio_proxy_istio-ingressgateway,\
mem_Mi_avg_istio_proxy_fortioclient,\
mem_Mi_avg_istio_proxy_fortioserver,\
mem_Mi_avg_istio_proxy_istio-ingressgateway
pkill kubectl
```

Running the performance test with TCP bypass

1. We performed steps 3 and 4 10 times, and identified the median run from the last three runs.
2. Clone the following repository (performed on 15 August 2022):

```
git clone https://github.com/intel/istio-tcpip-bypass.git
```

3. Load BPF bypass via a daemonset:

```
kubectl apply -f bypass-tcpip-daemonset.yaml
```

4. Perform the test:

```
python runner/runner.py --conn 8 --qps 100  --duration 240tn
```

5. Gather result metrics:

```
kubectl -n twopods-istio port-forward svc/fortioclient 9076:9076 &
export FORTIO_CLIENT_URL=http://localhost:9076
python ./runner/fortio.py $FORTIO_CLIENT_URL --csv \
StartTime,ActualDuration,Labels,NumThreads,ActualQPS,\
p50,p90,p99,p999,
cpu_mili_avg_istio_proxy_fortioclient,\
cpu_mili_avg_istio_proxy_fortioserver,
cpu_mili_avg_istio_proxy_istio-ingressgateway,\
mem_Mi_avg_istio_proxy_fortioclient,
mem_Mi_avg_istio_proxy_fortioserver,\
mem_Mi_avg_istio_proxy_istio-ingressgateway
```

Files

Kubeadm configuration parameters for cluster 1

Name: `kubeadm-config-01.yaml`

```
apiVersion: kubeadm.k8s.io/v1beta2
bootstrapTokens:
- groups:
  - system:bootstrappers:kubeadm:default-node-token
  token: abcdef.0123456789abcdef
  ttl: 24h0m0s
  usages:
  - signing
  - authentication
kind: InitConfiguration
localAPIEndpoint:
  advertiseAddress: 192.168.127.101
  bindPort: 6443
nodeRegistration:
  criSocket: /var/run/dockershim.sock
  name: sut01
  taints: null
---
apiServer:
  timeoutForControlPlane: 4m0s
apiVersion: kubeadm.k8s.io/v1beta2
certificatesDir: /etc/kubernetes/pki
clusterName: cluster2
controllerManager: {}
dns:
  type: CoreDNS
etcd:
  local:
    dataDir: /var/lib/etcd
imageRepository: k8s.gcr.io
kind: ClusterConfiguration
kubernetesVersion: 1.21.0
networking:
  dnsDomain: cluster.local
  podSubnet: 192.168.0.0/17
  serviceSubnet: 172.24.0.0/16
scheduler: {}
---
kind: KubeletConfiguration
apiVersion: kubelet.config.k8s.io/v1beta1
cgroupDriver: systemd
```

Kubeadm configuration parameters for cluster 2

Name: `kubeadm-config-02.yaml`

```
apiVersion: kubeadm.k8s.io/v1beta2
bootstrapTokens:
- groups:
  - system:bootstrappers:kubeadm:default-node-token
  token: abcdef.0123456789abcdef
  ttl: 24h0m0s
  usages:
  - signing
  - authentication
kind: InitConfiguration
localAPIEndpoint:
  advertiseAddress: 192.168.255.102
  bindPort: 6443
nodeRegistration:
  criSocket: /var/run/dockershim.sock
  name: sut02
  taints: null
---
apiServer:
  timeoutForControlPlane: 4m0s
apiVersion: kubeadm.k8s.io/v1beta2
certificatesDir: /etc/kubernetes/pki
clusterName: cluster2
controllerManager: {}
dns:
  type: CoreDNS
etcd:
  local:
    dataDir: /var/lib/etcd
imageRepository: k8s.gcr.io
kind: ClusterConfiguration
kubernetesVersion: 1.21.0
networking:
  dnsDomain: cluster.local
  podSubnet: 192.168.128.0/17
  serviceSubnet: 172.25.0.0/16
scheduler: {}
---
kind: KubeletConfiguration
apiVersion: kubelet.config.k8s.io/v1beta1
cgroupDriver: systemd
```

AKO configuration parameters for cluster 1

Name: `values-ako01.yaml`

```
replicaCount: 1
image:
  repository: projects.registry.vmware.com/ako/ako
  pullPolicy: IfNotPresent
AKOSettings:
  primaryInstance: true
  enableEvents: 'true'
  logLevel: WARN
  fullSyncFrequency: '1800'
  apiServerPort: 8080
  deleteConfig: 'false'
  disableStaticRouteSync: 'false'.
  clusterName: 'cluster1'
  cniPlugin: 'antrea'
  enableEVH: false
  layer7Only: false.
  namespaceSelector:
    labelKey: ''
    labelValue: ''
  servicesAPI: 'true'
  vipPerNamespace:
NetworkSettings:
  nodeNetworkList: []
  enableRHI: false
  nsxtT1LR: ''
  bgpPeerLabels: []
  vipNetworkList:
   - networkName: 'NET-istio01'
     cidr: '10.216.221.0/24'
L7Settings:
  defaultIngController: 'true'
  noPGForSNI: false
  serviceType: NodePort
  shardVSSize: LARGE
  passthroughShardSize: SMALL
  enableMCI: 'false'
L4Settings:
  defaultDomain: ''
  autoFQDN: default
ControllerSettings:
  serviceEngineGroupName: 'istio-Group'
  controllerVersion: '21.1.4'
  cloudName: 'istio-Cloud'
  controllerHost: '10.216.216.109'
  tenantName: 'admin'
nodePortSelector:
  key: 'istio-avi'
  value: 'true'
resources:
  limits:
    cpu: 350m
    memory: 400Mi
  requests:
    cpu: 200m
    memory: 300Mi
podSecurityContext: {}
rbac:
  pspEnable: false
avicredentials:
  username: 'admin'
  password: 'Password1!'
  authtoken:
  certificateAuthorityData:
persistentVolumeClaim: ''
mountPath: /log
logFile: avi.log
```

AKO configuration parameters for cluster 2

Name: `values-ako02.yaml`

```
replicaCount: 1
image:
  repository: projects.registry.vmware.com/ako/ako
  pullPolicy: IfNotPresent
AKOSettings:
  primaryInstance: true
  enableEvents: 'true'
  logLevel: WARN
  fullSyncFrequency: '1800'
  apiServerPort: 8080
  deleteConfig: 'false'
  disableStaticRouteSync: 'false'.
  clusterName: 'cluster2'
  cniPlugin: 'antrea'
  enableEVH: false
  layer7Only: false.
  namespaceSelector:
    labelKey: ''
    labelValue: ''
  servicesAPI: 'true'
  vipPerNamespace:
NetworkSettings:
  nodeNetworkList: []
  enableRHI: false
  nsxtT1LR: ''
  bgpPeerLabels: []
  vipNetworkList:
   - networkName: 'NET-istio01'
     cidr: '10.216.221.0/24'
L7Settings:
  defaultIngController: 'true'
  noPGForSNI: false
  serviceType: NodePort
  shardVSSize: LARGE
  passthroughShardSize: SMALL
  enableMCI: 'false'
L4Settings:
  defaultDomain: ''
  autoFQDN: default
ControllerSettings:
  serviceEngineGroupName: 'istio-Group'
  controllerVersion: '21.1.4'
  cloudName: 'istio-Cloud'
  controllerHost: '10.216.216.109'
  tenantName: 'admin'
nodePortSelector:
  key: 'istio-avi'
  value: 'true'
resources:
  limits:
    cpu: 350m
    memory: 400Mi
  requests:
    cpu: 200m
    memory: 300Mi
podSecurityContext: {}
rbac:
  pspEnable: false
avicredentials:
  username: 'admin'
  password: 'Password1!'
  authtoken:
  certificateAuthorityData:
persistentVolumeClaim: ''
mountPath: /log
logFile: avi.log
```

Fortio deployment for the TLS-acceleration testing

Name: `fortio-server.yaml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: server
  namespace: fortio
spec:
  selector:
    matchLabels:
      app.kubernetes.io/name: server
  template:
    metadata:
      labels:
        app.kubernetes.io/name: server
    spec:
      containers:
      - image: fortio/fortio:1.27.0
        imagePullPolicy: Always
        name: fortio
        ports:
        - containerPort: 8078
          name: tcp-echo
          protocol: TCP
        - containerPort: 8079
          name: grpc-ping
          protocol: TCP
        - containerPort: 8080
          name: http
          protocol: TCP
        - containerPort: 8081
          name: http-redirector
          protocol: TCP
        resources:
          requests:
            cpu: 2000m
            memory: 1024Mi
          limits:
            cpu: 2000m
            memory: 1024Mi
---
apiVersion: v1
kind: Service
metadata:
  name: fortio
  namespace: fortio
spec:
  selector:
    app.kubernetes.io/name: server
  type: ClusterIP
  ports:
  - name: tcp-echo
    port: 8078
    protocol: TCP
    targetPort: 8078
  - name: grpc-ping
    port: 8079
          name: grpc-ping
          protocol: TCP
        - containerPort: 8080
          name: http
          protocol: TCP
        - containerPort: 8081
          name: http-redirector
          protocol: TCP
        resources:
          requests:
            cpu: 2000m
            memory: 1024Mi
          limits:
```

```
            cpu: 2000m
            memory: 1024Mi
---
apiVersion: v1
kind: Service
metadata:
  name: fortio
  namespace: fortio
spec:
  selector:
    app.kubernetes.io/name: server
  type: ClusterIP
  ports:
  - name: tcp-echo
    port: 8078
    protocol: TCP
    targetPort: 8078
  - name: grpc-ping
    port: 8079
    protocol: TCP
    targetPort: 8079
  - name: http
    port: 8080
    protocol: TCP
    targetPort: 8080
  - name: http-redirector
    port: 8081
    protocol: TCP
    targetPort: 8081
```

Fortio-K6 ingress gateway for the TLS-acceleration testing

Name: `istio-gw.yaml`

```
apiVersion: networking.istio.io/v1beta1
kind: Gateway
metadata:
  name: istio-gateway
  namespace: istio-system
spec:
  selector:
    istio: ingressgateway
  servers:
  - hosts:
    - "*"
    port:
      name: https
      number: 443
      protocol: HTTPS
    tls:
      credentialName: istio-gateway-secret
      mode: SIMPLE
  - hosts:
    - "*"
    port:
      name: http
      number: 80
      protocol: HTTP
```

Fortio virtual service for the TLS-acceleration testing

Name: `istio-vs.yaml`

```
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: fortio
  namespace: fortio
spec:
  gateways:
  - istio-system/istio-gateway
  hosts:
  - "*"
  http:
  - name: fortio
    match:
      - uri:
          prefix: "/echo"
    route:
    - destination:
        host: fortio
        port:
          number: 8080
```

K6 configuration for TLS-acceleration testing

Name: `k6-no-connection-reuse-csv.js`

```
import http from "k6/http";
import { check } from 'k6';
export const options = {
  discardResponseBodies: true,
  insecureSkipTLSVerify: true,
  noConnectionReuse: true,
  noVUConnectionReuse: true,
  summaryTimeUnit: 'ms',
  summaryTrendStats: ['avg', 'min', 'med', 'max', 'p(95)', 'p(99)', 'count']
};
export default function() {
  var r = http.get(__ENV.ADDRESS);
  check(r, {'status is 200': (r) => r.status === 200});
};
export function handleSummary(data) {
  var csv = '\nEnd-of-test summary:\n\nmetric_name,avg,count,fails,min,med,max,p(95),p(99),pass
es,rate,value\n';
  const iteration_duration = data.metrics["iteration_duration"].values;
  csv += 'iteration_duration,${iteration_duration.avg},,,${iteration_duration.min},${iteration_duration.
med},${iteration_duration.max},${iteration_duration["p(95)"]},${iteration_duration["p(99)"]}\n';
  const http_req_duration = data.metrics["http_req_duration{expected_response:true}"].values;
  csv += 'http_req_duration,${http_req_duration.avg},,,${http_req_duration.min},${http_req_duration.
med},${http_req_duration.max},${http_req_duration["p(95)"]},${http_req_duration["p(99)"]}\n';
  const http_req_tls_handshaking = data.metrics["http_req_tls_handshaking"].values;
  csv += 'http_req_tls_handshaking,${http_req_tls_handshaking.avg},,,${http_req_tls_handshaking.
min},${http_req_tls_handshaking.med},${http_req_tls_handshaking.max},${http_req_tls_
handshaking["p(95)"]},${http_req_tls_handshaking["p(99)"]}\n';
  const avg = http_req_duration.avg + http_req_tls_handshaking.avg;
  const p99 = http_req_duration["p(99)"] + http_req_tls_handshaking["p(99)"];
  csv += 'http_req_tls,${avg},,,,,,,${p99}\n';
  const vus = data.metrics.vus.values;
  csv += 'vus,,,,${vus.min},,${vus.max},,,,,${vus.value}\n';
  const http_reqs = data.metrics.http_reqs.values;
  csv += 'http_reqs,,${http_reqs.count},,,,,,,,${http_reqs.rate}\n';
  const checks = data.metrics.checks.values;
  csv += 'checks,,,${checks.fails},,,,,,${checks.passes},${checks.rate}\n';
  csv += 'time_unit,,,,,,,,,,ms\n';
  return {'stdout': csv};
};
```

Patch file for changes to TSM's configuration for testing with TLS-acceleration

Name: `tsm-cryptomb.txt.txt`

```
--- tsm-default.yaml    2022-09-14 19:40:23.599972070 +0000
+++ tsm-cryptomb.yaml   2022-09-14 20:01:28.483684478 +0000
@@ -10,7 +10,7 @@
     base:
       enabled: true
     cni:
-      enabled: true
+      enabled: false
       namespace: kube-system
     egressGateways:
     - enabled: true
@@ -33,7 +33,21 @@
     - enabled: true
       k8s:
         hpaSpec:
+          maxReplicas: 2
           minReplicas: 2
+        overlays:
+        - kind: Deployment
+          name: istio-ingressgateway
+          patches:
+          - path: spec.template.spec.containers.[name:istio-proxy].args.[-1]
+            value: --concurrency=2
+        resources:
+          limits:
+            cpu: 2000m
+            memory: 4096Mi
+          requests:
+            cpu: 2000m
+            memory: 4096Mi
         replicaCount: 2
         service:
           externalTrafficPolicy: Local
@@ -62,10 +76,12 @@
           value: "true"
         - name: ISTIO_GATEWAY_STRIP_HOST_PORT
           value: "true"
+        - name: PRIVATE_KEY_PROVIDER
+          value: cryptomb
         hpaSpec:
           minReplicas: 2
         replicaCount: 2
-  hub: public.ecr.aws/v6x6b8s5/vmwareallspark
+  hub: docker.io/intel
   meshConfig:
     accessLogFile: ""
     defaultConfig:
@@ -77,7 +93,7 @@
     outboundTrafficPolicy:
       mode: ALLOW_ANY
   profile: default
-  tag: 1.12.2-release-tsm-advance-v1-distroless
+  tag: 1.12.9-intel.3
   values:
     base:
       enableCRDTemplates: false
@@ -219,6 +235,14 @@
       traceSampling: 1
     sidecarInjectorWebhook:
       rewriteAppHTTPProbe: true
+      templates:
+        cryptomb: |
+          spec:
+            containers:
+            - name: istio-proxy
+              env:
+              - name: PRIVATE_KEY_PROVIDER
+                value: cryptomb # needed for proxy-agent
     telemetry:
       enabled: true
       v2:
```

Patch file for changes to TSM's configuration for testing without TLS-acceleration

Name: tsm-baseline.txt.txt

```
--- tsm-default.yaml    2022-09-14 19:40:23.599972070 +0000
+++ tsm-baseline.yaml   2022-09-14 19:46:52.095893205 +0000
@@ -33,7 +33,21 @@
     - enabled: true
       k8s:
         hpaSpec:
+          maxReplicas: 2
           minReplicas: 2
+        overlays:
+        - kind: Deployment
+          name: istio-ingressgateway
+          patches:
+          - path: spec.template.spec.containers.[name:istio-proxy].args.[-1]
+            value: --concurrency=2
+        resources:
+          limits:
+            cpu: 2000m
+            memory: 4096Mi
+          requests:
+            cpu: 2000m
+            memory: 4096Mi
         replicaCount: 2
         service:
           externalTrafficPolicy: Local
```

Read the report at https://facts.pt/wN2ch6q ▶

This project was commissioned by VMware.

Principled Technologies®

Facts matter.®