**Checklist for debugging Istio networking configuration**

Whenever you run into configuration issues, you can use this set of steps to walk through and resolve the issue. In the first section, we are checking if the configuration is valid. If the configuration is valid, the next step is to look at how the runtime is handling the configuration, and for this, you need the basic understanding of Envoy configuration.

**Configuration**

**1. Is the configuration valid?**

Istio CLI has a command called validate we can use to validate the YAML configuration. The most common problems with YAML are indentation and array notation related issues.

To validate a configuration, pass in the YAML file to the validate command like this:

$ istioctl validate -f myresource.yaml

validation succeed

If the resource is invalid, the CLI will give us a detailed error. For example, if we misspelled a field name:

unknown field "worloadSelector" in v1alpha3.ServiceEntry

Another command we can use is istioctl analyze. Using this command, we can detect potential issues with the Istio configuration. We can either run it against a local set of configuration files or a live cluster. Also, look for any warnings or errors coming from istiod.

Here’s a sample output from the command that catches a typo in the destinations host name:

$ istioctl analyze

Error [IST0101] (VirtualService customers.default) Referenced host not found: "cusomers.default.svc.cluster.local"

Error [IST0101] (VirtualService customers.default) Referenced host+subset in destinationrule not found: "cusomers.default.svc.cluster.local+v1"

Error: Analyzers found issues when analyzing namespace: default.

See https://istio.io/docs/reference/config/analysis for more information about causes and resolutions.

**2. Is the naming correct? Are resources in the right namespace?**

Nearly all Istio resources are namespace scoped. Make sure they’re in the same namespace as the service you’re working on. Having Istio resources in the same namespace is especially important because *selectors* are namespaced as well.

A common misconfiguration is to publish VirtualService in the application’s namespace (default for example) and then use istio: ingressgateway selector to bind to the ingress gateway deployment in the istio-system namespace. This only works if your VirtualService is also in the istio-system namespace.

Similarly, don’t deploy a Sidecar resource in the istio-system namespace that references a VirtualService from the application namespace. Instead, deploy a set of Envoy gateways per application that needs ingress.

**3. Are the resource selectors correct?**

Verify the pods in your deployment have the right labels set. As mentioned in the previous step, resource selectors are bound to the namespace the resource is published in.

At this point we should be reasonably confident the configuration is correct. The next steps involve looking more into how the runtime system is handling the configuration.

**Runtime**

An experimental feature in Istio CLI can provide the information to help us understand the configuration impacting a Pod or a Service. Here’s an example of running the describe command against a Pod that has a typo in the host name:

$ istioctl x describe pod customers-v1-64455cd4c6-xvjzm.default

Pod: customers-v1-64455cd4c6-xvjzm

Pod Ports: 3000 (svc), 15090 (istio-proxy)

--------------------

Service: customers

Port: http 80/HTTP targets pod port 3000

DestinationRule: customers for "customers.default.svc.cluster.local"

Matching subsets: v1

No Traffic Policy

VirtualService: customers

WARNING: No destinations match pod subsets (checked 1 HTTP routes)

Route to cusomers.default.svc.cluster.local

**1. Did Envoy accept (ACK) the configuration?**

You can use the istioctl proxy-status command to check the status and see if Envoy accepted the configuration. We are expecting the status of everything to be set to SYNCHED. Any other value might indicate an error, and you should check Pilot’s logs.

$ istioctl proxy-status

NAME CDS LDS EDS RDS ISTIOD VERSION

customers-v1... SYNCED SYNCED SYNCED SYNCED istiod-67b4c76c6-8lwxf 1.9.0

customers-v1... SYNCED SYNCED SYNCED SYNCED istiod-67b4c76c6-8lwxf 1.9.0

istio-egress... SYNCED SYNCED SYNCED NOT SENT istiod-67b4c76c6-8lwxf 1.9.0

istio-ingress... SYNCED SYNCED SYNCED SYNCED istiod-67b4c76c6-8lwxf 1.9.0

web-frontend-... SYNCED SYNCED SYNCED SYNCED istiod-67b4c76c6-8lwxf 1.9.0

The list shows all proxies connected to a Pilot instance. If there’s a proxy missing from the list, it means it’s not connected to the Pilot, and it’s not receiving any configuration. If any of the proxies is marked STALE, there might be networking issues, or we need to scale the Pilot.

If Envoy accepted the configuration, yet we are still seeing issues, we need to make sure the configuration is manifested as expected in Envoy.

**2. Did the configuration appear as expected in Envoy?**

We can use the proxy-config command to retrieve the information about a specific Envoy instance. Refer to the table below for different proxy configurations we can retrieve.

| **Command** | **Description** |
| --- | --- |
| istioctl proxy-config cluster [POD] -n [NAMESPACE] | Retrieves cluster configuration |
| istioctl proxy-config bootstrap [POD] -n [NAMESPACE] | Retrieves bootstrap configuration |
| istioctl proxy-config listener [POD] -n [NAMESPACE] | Retrieves listener configuration |
| istioctl proxy-config route [POD] -n [NAMESPACE] | Retrieves route configuration |
| istioctl proxy-config endpoints [POD] -n [NAMESPACE] | Retrieves endpoints configuration |

The command collects the data from Envoy’s admin endpoint (mostly /config\_dump), and it contains a lot of useful information.

Also, refer back to the figure showing the mapping between Envoy and Istio resources. For example, many VirtualService rules will manifest as Envoy routes, whereas DestinationRules and ServiceEntries manifest as Clusters.

DestinationRules will not appear in the configuration unless a ServiceEntry for their host exists first.

Let’s take the customers VirtualService as an example:

apiVersion: networking.istio.io/v1alpha3

kind: VirtualService

metadata:

name: customers

spec:

hosts:

- 'customers.default.svc.cluster.local'

http:

- route:

- destination:

host: customers.default.svc.cluster.local

port:

number: 80

subset: v1

weight: 80

- destination:

host: customers.default.svc.cluster.local

port:

number: 80

subset: v2

weight: 20

timeout: 5s

If you run the command istioctl proxy-config routes [POD] -o json, you will see how the weighted destinations and timeout manifest themselves in the configuration:

..

{

"name": "80",

"virtualHosts": [

{

"name": "customers.default.svc.cluster.local:80",

"domains": [

"customers.default.svc.cluster.local",

...

],

"routes": [

{

"match": {

"prefix": "/"

},

"route": {

"weightedClusters": {

"clusters": [

{

"name": "outbound|80|v1|customers.default.svc.cluster.local",

"weight": 80

},

{

"name": "outbound|80|v2|customers.default.svc.cluster.local",

"weight": 20

}

]

},

"timeout": "5s",

...

When you’re evaluating VirtualServices, you’re looking for hostnames to be present in the Envoy configuration like you wrote them (e.g. customers.default.svc.cluster.local) and for routes to be present (see the 80-20 traffic split in the output). You could also use the previous example and trace the calls through listeners, routes, and clusters (and endpoints).

Envoy filters will manifest where you tell Istio to put them (the applyTo field in the EnvoyFilter resource). Typically, a bad filter will manifest as Envoy rejecting the configuration (i.e., not showing the SYNCED state). In that case, you need to check the Istiod logs for errors.

**3. Are there any errors in Istiod (Pilot)?**

The fastest way to see the errors from the Pilot is to follow the logs (use the --follow flag), and then apply the configuration. Here’s an example of an error from the Pilot that happened due to a typo in the inline code of the filter:

2020-11-20T21:49:16.017487Z warn ads ADS:LDS: ACK ERROR sidecar~10.120.1.8~web-frontend-58d497b6f8-lwqkg.default~default.svc.cluster.local-4 Internal:Error adding/updating listener(s) virtualInbound: script load error: [string "fction envoy\_on\_response(response\_handle)..."]:1: '=' expected near 'envoy\_on\_response'

If the configuration didn’t appear in Envoy at all (Envoy did not ACK it), or it’s an EnvoyFilter configuration, the configuration is likely invalid. Istio cannot syntactically validate the configuration inside of an EnvoyFilter. Another issue might be that the filter is located in the wrong spot in Envoy’s configuration.

In either case, Envoy will reject the configuration as invalid, and Pilot will log the error. You can generally search for the name of your resource to find the error.

Here, you’ll have to use judgment to determine if it’s an error in the configuration you wrote or a bug in Pilot resulting in it producing an invalid configuration.

**Inspecting Envoy logs**

To inspect the logs from the Envoy proxies, we can use the kubectl logs command:

kubectl logs PODNAME -c istio-proxy -n NAMESPACE

To understand the access log format and the response flags, we can refer to the [Envoy Access logging](https://www.envoyproxy.io/docs/envoy/latest/configuration/observability/access_log/usage#config-access-log-format-response-flags).

Most common response flags: - NR: No route configured, check DestinationRule or VirtualService - UO: Upstream overflow with circuit breaking. Check the circuit breaker configuration in DestinationRule - UF: Failed to connect upstream, check the mTLS configuration if using Istio authentication - UH: no healthy upstream hosts

**Configure istiod logging**

We can use the ControlZ dashboard to configure the stack trace level and log levels through the Logging Scopes menu.

To open the dashboard, run:

istioctl dashboard controlz $(kubectl -n istio-system get pods -l app=istiod -o jsonpath='{.items[0].metadata.name}').istio-system

Once the dashboard opens, click the **Logging Scopes** option and adjust the log level and stack trace levels.

**COMPLETE & CONTINUE**