**Configure health probes for pods.**

Overview

Implementing [health probes](https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/) in your application is a great way for Kubernetes to automate some tasks to improve availability in the event of an error.

Because OSM reconfigures application Pods to redirect all incoming and outgoing network traffic through the proxy sidecar, httpGet and tcpSocket health probes invoked by the kubelet will fail due to the lack of any mTLS context required by the proxy.

For health probes to continue to work as expected from within the mesh, OSM adds configuration to expose the probe endpoint via the proxy and rewrites the probe definitions for new Pods to refer to the proxy-exposed endpoint. All of the functionality of the original probe is still used, OSM simply fronts it with the proxy so the kubelet can communicate with it.

For HTTP probes, the following table shows the modified path and port for each probe type.

| **Probe** | **Path** | **Port** |
| --- | --- | --- |
| Liveness | /osm-liveness-probe | 15901 |
| Readiness | /osm-readiness-probe | 15902 |
| Startup | /osm-startup-probe | 15903 |

HTTPS and tcpSocket probes will have their ports modified the same way as HTTP probes. For HTTPS probes, the path is left unchanged.

Only predefined httpGet and tcpSocket probes are modified. If a probe is undefined, one will not be added in its place. exec probes (including those using grpc\_health\_probe) are never modified and will continue to function as expected as long as the command does not require network access outside of localhost.

Examples

The following examples show how OSM handles health probes for Pods in a mesh.

HTTP

Consider a Pod spec defining a container with the following livenessProbe:

livenessProbe:

httpGet:

path: /liveness

port: 14001

scheme: HTTP

When the Pod is created, OSM will modify the probe to be the following:

livenessProbe:

httpGet:

path: /osm-liveness-probe

port: 15901

scheme: HTTP

The Pod’s proxy will contain the following Envoy configuration.

An Envoy cluster which maps to the original probe port 14001:

{

"cluster": {

"@type": "type.googleapis.com/envoy.config.cluster.v3.Cluster",

"name": "liveness\_cluster",

"type": "STATIC",

"connect\_timeout": "1s",

"load\_assignment": {

"cluster\_name": "liveness\_cluster",

"endpoints": [

{

"lb\_endpoints": [

{

"endpoint": {

"address": {

"socket\_address": {

"address": "0.0.0.0",

"port\_value": 14001

}

}

}

}

]

}

]

}

},

"last\_updated": "2021-03-29T21:02:59.086Z"

}

A listener for the new proxy-exposed HTTP endpoint at /osm-liveness-probe on port 15901 mapping to the cluster above:

{

"listener": {

"@type": "type.googleapis.com/envoy.config.listener.v3.Listener",

"name": "liveness\_listener",

"address": {

"socket\_address": {

"address": "0.0.0.0",

"port\_value": 15901

}

},

"filter\_chains": [

{

"filters": [

{

"name": "envoy.filters.network.http\_connection\_manager",

"typed\_config": {

"@type": "type.googleapis.com/envoy.extensions.filters.network.http\_connection\_manager.v3.HttpConnectionManager",

"stat\_prefix": "health\_probes\_http",

"route\_config": {

"name": "local\_route",

"virtual\_hosts": [

{

"name": "local\_service",

"domains": [

"\*"

],

"routes": [

{

"match": {

"prefix": "/osm-liveness-probe"

},

"route": {

"cluster": "liveness\_cluster",

"prefix\_rewrite": "/liveness"

}

}

]

}

]

},

"http\_filters": [...],

"access\_log": [...]

}

}

]

}

]

},

"last\_updated": "2021-03-29T21:02:59.092Z"

}

HTTPS

Consider a Pod spec defining a container with the following livenessProbe:

livenessProbe:

httpGet:

path: /liveness

port: 14001

scheme: HTTPS

When the Pod is created, OSM will modify the probe to be the following:

livenessProbe:

httpGet:

path: /liveness

port: 15901

scheme: HTTPS

The Pod’s proxy will contain the following Envoy configuration.

An Envoy cluster which maps to the original probe port 14001:

{

"cluster": {

"@type": "type.googleapis.com/envoy.config.cluster.v3.Cluster",

"name": "liveness\_cluster",

"type": "STATIC",

"connect\_timeout": "1s",

"load\_assignment": {

"cluster\_name": "liveness\_cluster",

"endpoints": [

{

"lb\_endpoints": [

{

"endpoint": {

"address": {

"socket\_address": {

"address": "0.0.0.0",

"port\_value": 14001

}

}

}

}

]

}

]

}

},

"last\_updated": "2021-03-29T21:02:59.086Z"

}

A listener for the new proxy-exposed TCP endpoint on port 15901 mapping to the cluster above:

{

"listener": {

"@type": "type.googleapis.com/envoy.config.listener.v3.Listener",

"name": "liveness\_listener",

"address": {

"socket\_address": {

"address": "0.0.0.0",

"port\_value": 15901

}

},

"filter\_chains": [

{

"filters": [

{

"name": "envoy.filters.network.tcp\_proxy",

"typed\_config": {

"@type": "type.googleapis.com/envoy.extensions.filters.network.tcp\_proxy.v3.TcpProxy",

"stat\_prefix": "health\_probes",

"cluster": "liveness\_cluster",

"access\_log": [...]

}

}

]

}

]

},

"last\_updated": "2021-04-07T15:09:22.704Z"

}

tcpSocket

Consider a Pod spec defining a container with the following livenessProbe:

livenessProbe:

tcpSocket:

port: 14001

When the Pod is created, OSM will modify the probe to be the following:

livenessProbe:

tcpSocket:

port: 15901

The Pod’s proxy will contain the following Envoy configuration.

An Envoy cluster which maps to the original probe port 14001:

{

"cluster": {

"@type": "type.googleapis.com/envoy.config.cluster.v3.Cluster",

"name": "liveness\_cluster",

"type": "STATIC",

"connect\_timeout": "1s",

"load\_assignment": {

"cluster\_name": "liveness\_cluster",

"endpoints": [

{

"lb\_endpoints": [

{

"endpoint": {

"address": {

"socket\_address": {

"address": "0.0.0.0",

"port\_value": 14001

}

}

}

}

]

}

]

}

},

"last\_updated": "2021-03-29T21:02:59.086Z"

}

A listener for the new proxy-exposed TCP endpoint on port 15901 mapping to the cluster above:

{

"listener": {

"@type": "type.googleapis.com/envoy.config.listener.v3.Listener",

"name": "liveness\_listener",

"address": {

"socket\_address": {

"address": "0.0.0.0",

"port\_value": 15901

}

},

"filter\_chains": [

{

"filters": [

{

"name": "envoy.filters.network.tcp\_proxy",

"typed\_config": {

"@type": "type.googleapis.com/envoy.extensions.filters.network.tcp\_proxy.v3.TcpProxy",

"stat\_prefix": "health\_probes",

"cluster": "liveness\_cluster",

"access\_log": [...]

}

}

]

}

]

},

"last\_updated": "2021-04-07T15:09:22.704Z"

}

How to Verify Health of Pods in the Mesh

Kubernetes will automatically poll the health endpoints of Pods configured with startup, liveness, and readiness probes.

When a startup probe fails, Kubernetes will generate an Event (visible by kubectl describe pod <pod name>) and restart the Pod. The kubectl describe output may look like this:

...

Events:

Type Reason Age From Message

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

Normal Scheduled 17s default-scheduler Successfully assigned bookstore/bookstore-v1-699c79b9dc-5g8zn to osm-control-plane

Normal Pulled 16s kubelet Successfully pulled image "openservicemesh/init:v0.10.0" in 26.5835ms

Normal Created 16s kubelet Created container osm-init

Normal Started 16s kubelet Started container osm-init

Normal Pulling 16s kubelet Pulling image "openservicemesh/init:v0.10.0"

Normal Pulling 15s kubelet Pulling image "envoyproxy/envoy-alpine:v1.17.2"

Normal Pulling 15s kubelet Pulling image "openservicemesh/bookstore:v0.10.0"

Normal Pulled 15s kubelet Successfully pulled image "openservicemesh/bookstore:v0.10.0" in 319.9863ms

Normal Started 15s kubelet Started container bookstore-v1

Normal Created 15s kubelet Created container bookstore-v1

Normal Pulled 14s kubelet Successfully pulled image "envoyproxy/envoy-alpine:v1.17.2" in 755.2666ms

Normal Created 14s kubelet Created container envoy

Normal Started 14s kubelet Started container envoy

Warning Unhealthy 13s kubelet Startup probe failed: Get "http://10.244.0.23:15903/osm-startup-probe": dial tcp 10.244.0.23:15903: connect: connection refused

Warning Unhealthy 3s (x2 over 8s) kubelet Startup probe failed: HTTP probe failed with statuscode: 503

When a liveness probe fails, Kubernetes will generate an Event (visible by kubectl describe pod <pod name>) and restart the Pod. The kubectl describe output may look like this:

...

Events:

Type Reason Age From Message

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

Normal Scheduled 59s default-scheduler Successfully assigned bookstore/bookstore-v1-746977967c-jqjt4 to osm-control-plane

Normal Pulling 58s kubelet Pulling image "openservicemesh/init:v0.10.0"

Normal Created 58s kubelet Created container osm-init

Normal Started 58s kubelet Started container osm-init

Normal Pulled 58s kubelet Successfully pulled image "openservicemesh/init:v0.10.0" in 23.415ms

Normal Pulled 57s kubelet Successfully pulled image "envoyproxy/envoy-alpine:v1.17.2" in 678.1391ms

Normal Pulled 57s kubelet Successfully pulled image "openservicemesh/bookstore:v0.10.0" in 230.3681ms

Normal Created 57s kubelet Created container envoy

Normal Pulling 57s kubelet Pulling image "envoyproxy/envoy-alpine:v1.17.2"

Normal Started 56s kubelet Started container envoy

Normal Pulled 44s kubelet Successfully pulled image "openservicemesh/bookstore:v0.10.0" in 20.6731ms

Normal Created 44s (x2 over 57s) kubelet Created container bookstore-v1

Normal Started 43s (x2 over 57s) kubelet Started container bookstore-v1

Normal Pulling 32s (x3 over 58s) kubelet Pulling image "openservicemesh/bookstore:v0.10.0"

Warning Unhealthy 32s (x6 over 50s) kubelet Liveness probe failed: HTTP probe failed with statuscode: 503

Normal Killing 32s (x2 over 44s) kubelet Container bookstore-v1 failed liveness probe, will be restarted

When a readiness probe fails, Kubernetes will generate an Event (visible with kubectl describe pod <pod name>) and ensure no traffic destined for Services the Pod may be backing is routed to the unhealthy Pod. The kubectl describe output for a Pod with a failing readiness probe may look like this:

...

Events:

Type Reason Age From Message

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

Normal Scheduled 32s default-scheduler Successfully assigned bookstore/bookstore-v1-5848999cb6-hp6qg to osm-control-plane

Normal Pulling 31s kubelet Pulling image "openservicemesh/init:v0.10.0"

Normal Pulled 31s kubelet Successfully pulled image "openservicemesh/init:v0.10.0" in 19.8726ms

Normal Created 31s kubelet Created container osm-init

Normal Started 31s kubelet Started container osm-init

Normal Created 30s kubelet Created container bookstore-v1

Normal Pulled 30s kubelet Successfully pulled image "openservicemesh/bookstore:v0.10.0" in 314.3628ms

Normal Pulling 30s kubelet Pulling image "openservicemesh/bookstore:v0.10.0"

Normal Started 30s kubelet Started container bookstore-v1

Normal Pulling 30s kubelet Pulling image "envoyproxy/envoy-alpine:v1.17.2"

Normal Pulled 29s kubelet Successfully pulled image "envoyproxy/envoy-alpine:v1.17.2" in 739.3931ms

Normal Created 29s kubelet Created container envoy

Normal Started 29s kubelet Started container envoy

Warning Unhealthy 0s (x3 over 20s) kubelet Readiness probe failed: HTTP probe failed with statuscode: 503

The Pod’s status will also indicate that it is not ready which is shown in its kubectl get pod output. For example:

NAME READY STATUS RESTARTS AGE

bookstore-v1-5848999cb6-hp6qg 1/2 Running 0 85s

The Pods’ health probes may also be invoked manually by forwarding the Pod’s necessary port and using curl or any other HTTP client to issue requests. For example, to verify the liveness probe for the bookstore-v1 demo Pod, forward port 15901:

kubectl port-forward -n bookstore deployment/bookstore-v1 15901

Then, in a separate terminal instance, curl may be used to check the endpoint. The following example shows a healthy bookstore-v1:

$ curl -i localhost:15901/osm-liveness-probe

HTTP/1.1 200 OK

date: Wed, 31 Mar 2021 16:00:01 GMT

content-length: 1396

content-type: text/html; charset=utf-8

x-envoy-upstream-service-time: 1

server: envoy

<!doctype html>

<html itemscope="" itemtype="http://schema.org/WebPage" lang="en">

...

</html>

Known issues

* [#2207](https://github.com/openservicemesh/osm/issues/2207)

Troubleshooting

If any health probes are consistently failing, perform the following steps to identify the root cause:

1. Verify httpGet and tcpSocket probes on Pods in the mesh have been modified.

Startup, liveness, and readiness httpGet and tcpSocket probes must be modified by OSM in order to continue to function while in a mesh. Ports must be modified to 15901, 15902, and 15903 for liveness, readiness, and startup probes, respectively. Only HTTP (not HTTPS) probes will have paths modified in addition to be /osm-liveness-probe, /osm-readiness-probe, or /osm-startup-probe.

Also, verify the Pod’s Envoy configuration contains a listener for the modified endpoint.

See the [examples above](https://release-v0-10.docs.openservicemesh.io/docs/guides/health_checks/health_probes/#examples) for more details.

1. Determine if Kubernetes encountered any other errors while scheduling or starting the Pod.

Look for any errors that may have recently occurred with kubectl describe of the unhealthy Pod. Resolve any errors and verify the Pod’s health again.

1. Determine if the Pod encountered a runtime error.

Look for any errors that may have occurred after the container started by inspecting its logs with kubectl logs. Resolve any errors and verify the Pod’s health again.