Automated VM Registration for Mesh Expansion

Owner: sven@google.com, howardjohn@ **Working Group:** Environments, Networking	**Status**: WIP | In Review | **Approved** | Obsolete **Created**: 2019/11/15 **Approvers**: Environments [x], Networking [x]

TL;DR

- We can improve Mesh Expansion by automating creation of WorkloadEntries when they connect to Pilot.

Overview

As part of simplifying the onboarding of VMs to the mesh, we propose to automatically configure WorkloadEntries from Istiod. This alleviates the need for users to manually manage workload entries as instances are added/deleted, allowing them to utilize Istiod as a service discovery mechanism rather than manual action.

Related documents:

- Virtual Machine Health Checking builds upon this document, adding automated health checking
- Design: Simplified VM Configuration Generation adds a new WorkloadGroup containing a template for WorkloadEntry. This is a prerequisite.
- XDS Authorization will add authorization to verify clients do not spoof their identity. This is a prerequisite.

Goals:

- Support auto-registration of VMs in the mesh.

Non-goals

- Support health checking and status reporting of VMs (covered by another doc)

User Guide

Today, a user would:

1. create a WorkloadGroup with information about their workload

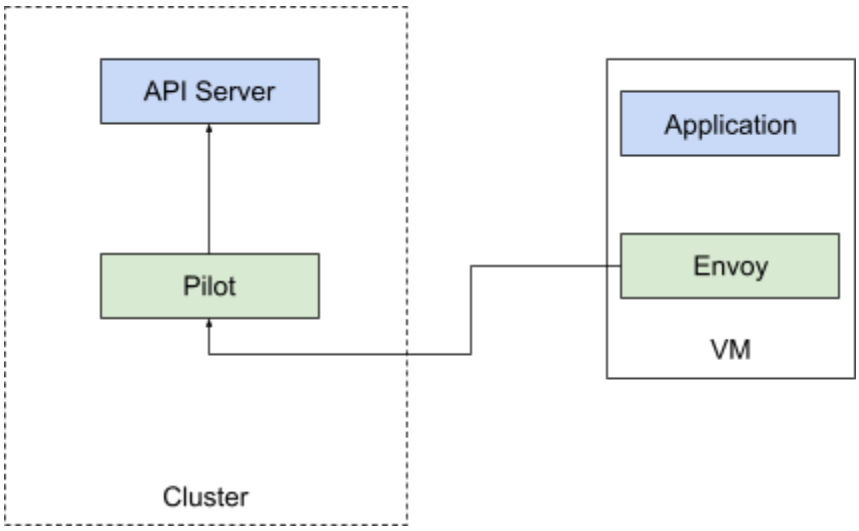
2. Setup the VM to run Istio
3. Manually create a workload entry pointing at the VM
4. When the VM is removed, they will need to manually remove the workload entry

With these changes:

1. create a WorkloadGroup with information about their workload
2. Setup the VM to run Istio

Steps 3 and 4 are configured automatically.

Design



High Level Overview

To enable auto-registration we would add code to Pilot to support creating a WorkloadEntry for each Envoy connected from a VM. On connection, we will create a WorkloadEntry. The contents of this will be derived from the XDS Node Metadata. On disconnect, this WorkloadEntry will be removed (see "Life Cycle" for details).

WorkloadEntry creation

From the node metadata, we will retrieve the backing WorkloadGroup, which we will take the ports, service account, and labels from. The IP address is also included in the metadata and will be used. These are configured by `ISTIO_META_WORKLOAD_NAME`, `ISTIO_META_NAMESPACE`, and `ISTIO_META_INSTANCE_IPS`, all of which are included as part of Design: Simplified VM Configuration Generation.

The name will be derived from the IP address.
For example:

```yaml
apiVersion: networking.istio.io/v1beta1
kind: WorkloadGroup
metadata:
  name: foo
  namespace: bar
spec:
  labels:
    app: foo
    bar: baz
  network: default
  ports:
    http: 8080
    grpc: 3550
  serviceAccount: foobar
```

Connecting with IP address 1.2.3.4

Will translate to:

```yaml
apiVersion: networking.istio.io/v1beta1
kind: WorkloadEntry
metadata:
  name: foo-1-2-3-4
  namespace: bar
  labels:
    workloadentry.istio.io/managed-by: workloadentry-controller.istio.io
    istio.io/workload-group: foo
spec:
  labels:
    app: foo
    bar: baz
  network: default
  ports:
    http: 8080
    grpc: 3550
  serviceAccount: foobar
  address: 1.2.3.4
status:
  istio.io/workloadentry-controller: istiod-554684c688-lx8zn
```

The `workloadentry.istio.io/managed-by` and `istio.io/workload-group` are inspired by EndpointSlice, which has similar labels. The first identifies this as auto registered, while the second provides a backpointer to the original workload group.

Authorization

As part of this effort, we need to ensure XDS clients cannot impersonate other workloads and hijack traffic unexpectedly. Specifically, we will check that the namespace and service account of the WorkloadGroup matches the identity of the XDS client (derived from the SPIFFE cert, which encodes both of these).

Rollout Plan

Auto Registration will be enabled as a feature flag in Pilot. Additionally, a new XDS node metadata will be added to opt in on a per workload basis. This allows both gradual rollouts, and allows users to manage certain WorkloadEntries, or all WorkloadEntries, manually.

Lifecycle (WIP)

For prior art, we can examine the Kubernetes EndpointSlice controller, which handles similar concerns. However, this implementation relies on joining two persisted objects(Pod and Service), whereas in our case we have a persisted object (Service) and transient information (XDS connection state). As a result, the EndpointSlice implementation does not suffer from many of these concerns.

In the happy case, the lifecycle is simple: on connection we create a WorkloadEntry, and on disconnect we remove it. However, there are many important cases off this path to consider.

1. Istio-agent disconnects, and reconnects to the same instance
2. Istio-agent disconnects, and reconnects to a different instance
3. Istio-agent disconnects, and does not reconnect
4. Istiod shutdown gracefully
5. Istiod shutdown non-gracefully (crash)
6. Istio-agent disconnects, then Istiod exits
7. Istiod exits, istio-agent never reconnects to another instance

To handle these cases, I propose the following lifecycle:

1. On connection:
    a. Lookup WorkloadEntry by a stable name defined in "WorkloadEntry creation".
        i. If it exists: update status to indicated the connected pod, and time of connection
        ii. If it does not exist: create a new workload entry, as defined in "WorkloadEntry creation"
        iii. If it is in our removal queue (see "On disconnect"), remove it
2. On disconnect:

    a. Update WorkloadEntry status to remove `istio.io/workloadentry-controller`, and add `istio.io/workloadentry-disconnect-time`.
    b. Add to a queue for removal. After GRACE_PERIOD_SECONDS, we will check the workload entry. If there is no `istio.io/workloadentry-controller` set, we will delete it entirely from the api-server. If there is a controller set, do nothing - it has reconnected to another instance.
3. Periodically, every GRACE_PERIOD_SECONDS * 10:
    a. Istiod will query all workload entries (note: this is from in memory cache, not an api-server call) for entries with `istio.io/workloadentry-controller` unset and `istio.io/workloadentry-disconnect-time` more than GRACE_PERIOD_SECONDS old.

Why this works:

- Cases 1, 2, and 3 will all result in the `istio.io/workloadentry-controller` annotation being removed. In case 1, Istiod will remove it from its removal queue on reconnect. In case 2, Istiod will attempt to remove it, but see the `istio.io/workloadentry-controller` has now been set, and skip. In case 3, it will be removed after GRACE_PERIOD_SECONDS in the queue
- Case 4, 5: in this case, no modification of the WorkloadEntry occurs. However, when it reconnects to another instance, it will now be tracked by that instance
- Case 6: This is managed by the periodic checking done by each instance
- Case 7: This is **not** currently covered.

Network Resilience

There may be cases where there are widespread connectivity issues between pods and Istiod. This would lead to a large chunk of VMs being removed at once. This is not a problem Kubernetes faces with health checks; if kubelet is disconnected from the api-server, health status cannot be reported so endpoints will retain their previous state.

We do not currently have any way to identify this situation from an aggressive scale down of VMs. As a result, the initial implementation will not have any built in mitigations for this issue.

This is an area we should consider addressing prior to promoting to beta/stable.

Alternatives

1) Do nothing and ask users to create WorkloadEntry themselves.
    a) Con: Users have to keep the VMs in sync with the API Server.
2) Do this through PilotAgent directly to the API Server
    a) Con: Requires access to the API Server (credentials),
    b) Cons: issues around health checking; if a VM suddenly dies we need a way to clean things up

Test Plan: AutoRegistration

As part of this effort, we will move the existing VM tests to auto registration, rather than manual WorkloadEntry creation. This covers the end to end happy cases. In addition:

Positive Tests:

- Autoregister same IP on different networks (create multiple instances)
- VM removed if it exits cleanly.
- Multiple pilot replicas, each VM connected to a single pilot
- Multiple pilot replicas, each has a connection asserting the same IP

Negative Tests:

- Autoregister same IP on same network (VM crashes and comes back up)
- VM garbage collected if VM crashes
- VM garbage collected if pilot crashes

Performance/Scale Tests:

- Pilot can handle autoregistration of appropriate number of WorkloadEntry instances.
- Pilot gracefully handles high churn of WorkloadEntry instances.
- Comparison of Istiod/api-server resource utilization of pods vs workload entry for large scale/churn

Appendix

@howardjohn various thoughts

We need to ensure that WorkloadEntries are stable against:

- Transient XDS disconnect/reconnect (to same or different instance)
- Pilot shut down (graceful and not graceful)
- Likely can use similar techniques used in k8s configmap leader election
    - Can we do something smart with ownerRef? make the owning pilot the owner ref so if its removed it will get GCed?
        - On graceful shutdown, remove ourself as the owner and add a label "check-for-deletion: timestamp". Other instances periodically poll for this label and if its been on an object for more than X seconds delete it
        - On forceful shutdown, there are two possibilities:
            - 1. K8s GCs the workload entry
            - 2. We reconnect to a new Istiod before GC, and swap the ownerRef
                - Make sure there is no race here...