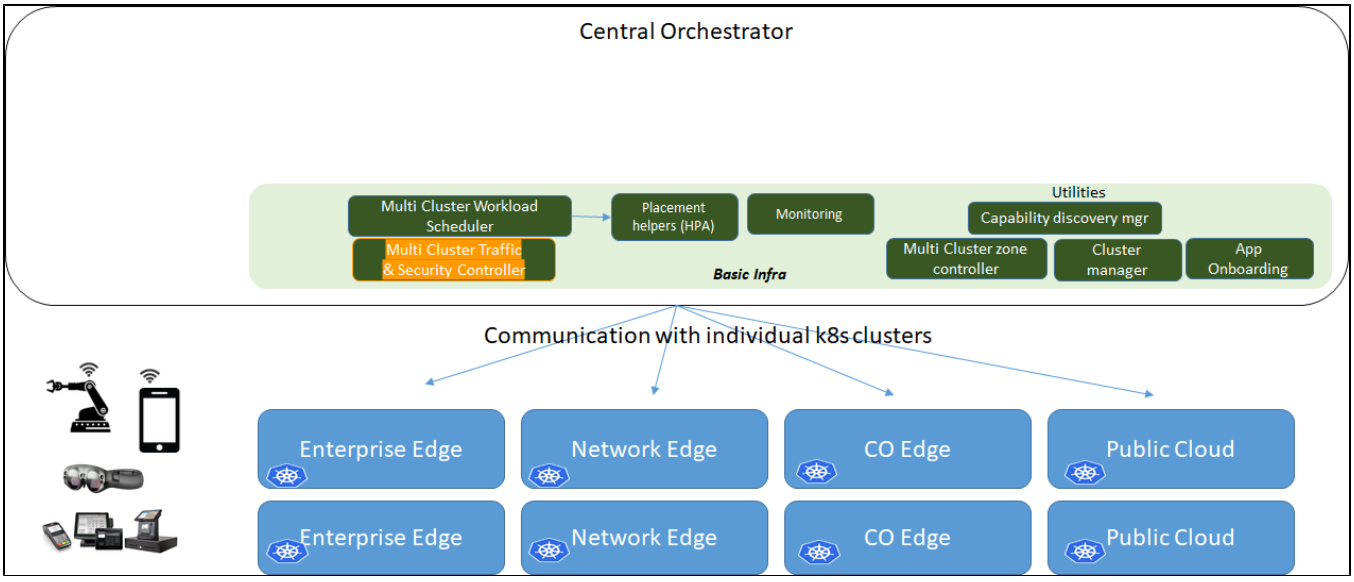
L7 Proxy Service Mesh Controller

Background

L7 Proxy Service Mesh Controller intends to provide connectivity, shape the traffic, apply policies, RBAC and provide mutual TLS for applications/microservices running across clusters (with service mesh), within the cluster and with external applications. The functionalities are subjected to the usage of underlying service mesh technology.

Design Overview



Traffic Controller Design Internals

blocked URL

Internal Implementation Details

NOTE - Current implementation will support the ISTIO service mesh technology and SD-WAN load balancer and ExternalDNS as DNS provider. The plugin architecture of the controller makes it extensible to work with any Service mesh technology and any external load balancer as well. It is also designed to configure and communicate with external DNS servers.

JIRA

Component	JIRA Items
1. REST API Interface	MULTICLOUD-913 - Getting issue details... STATUS
2. Controller Interface, Backend Process	MULTICLOUD-914 - Getting issue details... STATUS
3. Developing backend code with interfaces	MULTICLOUD-915 - Getting issue details... STATUS
4. Loadbalancer configuration (Firewall, IPVS, NAT, and other L3 connectivity)	MULTICLOUD-924 - Getting issue details... STATUS MULTICLOUD-925 - Getting issue details... STATUS MULTICLOUD-926 - Getting issue details... STATUS
5. External DNS Configuration	MULTICLOUD-909 - Getting issue details... STATUS
6. Testing	MULTICLOUD-918 - Getting issue details... STATUS
7.Documentation	MULTICLOUD-923 - Getting issue details... STATUS

Elements of Traffic Controller with ISTIO as the service mesh

1. Gateways - The inbound/outbound access for the service mesh. It is an envoy service
2. VirtualServices - To expose the service outside the service mesh
3. DestinationRule - To apply rules for the traffic flow
4. AuthorizationPolicy - Authorization for service access
5. serviceEntry - Add an external service into the mesh
6. Authentication Policy - Authenticate external communication

These are the Kubernetes resources generated per cluster. There will be multiple of these resources depending on the intent

API

RESTful North API (with examples)

Types	Intent APIs	Functionality
1. outbound service communication	/v2/projects/{project-name}/composite-apps/blue-app/{version}/traffic-intent-set/inbound-intents/	Define outbound traffic for a service
2. inbound service communication	v2/projects/{project-name}/composite-apps/blue-app/{version}/traffic-intent-set/outbound-intents/	Define Inbound service for a service
3. Compound service communication	/v2/projects/{project-name}/composite-apps/blue-app/{version}/traffic-intent-set/compound-intents/{compund-intent-name}/inbound-intents/	Define a virtual path for connecting to multiple services

```
URL: /v2/projects/{project-name}/composite-apps/{composite-app-name}/{version}/traffic-intent-set
POST BODY:
{
        "name": "john",
        "description": "Traffic intent groups"
    "set":[
      {
         "inbound":"abc"
      },
      {
         "outbound":"abc"
      }
   ]
}
```

1. Inbound access

POST

POST

```
URL: /v2/projects/{project-name}/composite-apps/blue-app/{version}/traffic-intent-set/inbound-intents/

POST BODY:
{
        "metadata": {
        "name": "<>" // unique name for each intent
    "description": "connectivity intent for inbound communication"
        "userdata1": <>,
        "userdata2": <>
        }

        "spec": { // update the memory allocation for each field as per OpenAPI standards
        "application": "<app1>",
        "servicename": "httpbin" //actual name of the client service - {istioobject - serviceEntry of client's
cluster}
        "externalName": "httpbin.k8s.com"
        "traffic-weeight": "" // Default is "". Used for redirecting traffic percentage when compound API is
called
        "protocol": "HTTP",
        "headless": "false", // default is false. Option "True" will make sure all the instances of the
headless service will have access to the client service
        "mutualTLS": "MUTUAL", // default is simple. Option MUTUAL will enforce mtls {istioobject -
destinationRule}
        "port" : "80", // port on which service is exposed as through servicemesh, not the port it is actually
running on
        "serviceMesh": "istio", // get it from cluster record
        "sidecar-proxy": "yes", // The features (mTLS, LB, Circuit breaking) are not available to services
without istio-proxy. Only inbound routing is possible.
        // Traffic management fields below are valid only if the sidecar-proxy is set to "yes"
        "traffic-management-info" : {
        // Traffic configuration - Loadbalancing is applicable per service. The traffic to this service is
distrbuted amongst the pods under it.
        "loadbalancingType": "ConsistenHash", // "Simple" and "consistentHash" are the two modes - {istioobject
- destinationRule}
        "loadBalancerMode": "httpCookie" // Modes for consistentHash - "httpHeaderName", "httpCookie",
"useSourceIP", "minimumRingSize", Modes for simple - "LEAST_CONN", "ROUND_ROBIN",         "RANDOM",
"PASSTHROUGH" // choices of the mode must be explicit - {istioobject - destinationRule}
        "httpCookie": "user1" // Name of the cookie to maitain sticky sessions - {istioobject - destinationRule}

        // Circuit Breaking
        "maxConnections": 10 //connection pool for tcp and http traffic - {istioobject - destinationRule}
        "concurrenthttp2Requests": 1000 // concurent http2 requests which can be allowed - {istioobject -
destinationRule}
        "httpRequestPerConnection": 100 // number of http requests per connection. Valid only for http traffic
- {istioobject - destinationRule}
        "consecutiveErrors": 8 // Default is 5.  Number of consecutive error before the host is removed -
{istioobject - destinationRule}
        "baseEjectionTime" : 15 // Default is 5, - {istioobject - destinationRule}
        "intervalSweep": 5m, //time limit before the removed hosts are added back to the load balancing pool. -
{istioobject - destinationRule}
        }

        // credentials for mTLS.
        "Servicecertificate" : "" // Present actual certificate here.
        "ServicePrivateKey" : "" // Present actual private key here.
        "caCertificate" : "" // present the trusted certificate to verify the client connection, Required only
when mtls mode is MUTUAL
        }
}

RETURN STATUS: 201
RETURN BODY:
{
  "name": "<name>"
  "Message": "inbound service created"
}
```

GET

GET

```
URL: /v2/projects/{project-name}/composite-apps/blue-app/{version}/traffic-intent-set/inbound-intents/<name>

RETURN STATUS: 201
RETURN BODY:
{
        "metadata": {
        "name": "<>" // unique name for each intent
    "description": "connectivity intent for stateless micro-service to stateless micro-service communication"
        "userdata1": <>,
        "userdata2": <>
        }

        "spec": { // update the memory allocation for each field as per OpenAPI standards
        "application": "<app1>",
        "servicename": "<>" //actual name of the client service - {istioobject - serviceEntry of client's
cluster}
        "externalName": "<>" // prefix to expose this service outside the cluster
        "protocol": "", // supported protocols are HTTP, TCP, UDP and HTTP2
        "headless": "", // default is false. Option "True" will make sure all the instances of the headless
service will have access to the client service
        "mutualTLS": "", // default is simple. Option MUTUAL will enforce mtls {istioobject - destinationRule}
        "port" : "80", // port on which service is exposed as through servicemesh, not the port it is actually
running on
        "serviceMesh": "istio", // get it from cluster record
        "sidecar-proxy": "yes", // The features (mTLS, LB, Circuit breaking) are not available to services
without istio-proxy. Only inbound routing is possible.

        / Traffic management fields below are valid only if the sidecar-proxy is set to "yes"
        "traffic-management-info" : {
                // Traffic configuration - Loadbalancing is applicable per service. The traffic to this service
is distrbuted amongst the pods under it.
                "loadbalancingType": "", // "Simple" and "consistentHash" are the two modes - {istioobject -
destinationRule}
                "loadBalancerMode": "" // Modes for consistentHash - "httpHeaderName", "httpCookie",
"useSourceIP", "minimumRingSize", Modes for simple - "LEAST_CONN", "ROUND_ROBIN",         "RANDOM",
"PASSTHROUGH" // choices of the mode must be explicit - {istioobject - destinationRule}
                "httpCookie": "user1" // Name of the cookie to maitain sticky sessions - {istioobject -
destinationRule}

                // Circuit Breaking
                "maxConnections": "" //connection pool for tcp and http traffic - {istioobject -
destinationRule}
                "concurrenthttp2Requests": "" // concurent http2 requests which can be allowed - {istioobject -
destinationRule}
                "httpRequestPerConnection": "" // number of http requests per connection. Valid only for http
traffic - {istioobject - destinationRule}
                "consecutiveErrors": "" // Default is 5.  Number of consecutive error before the host is
removed - {istioobject - destinationRule}
                "baseEjectionTime" : "" // Default is 5, - {istioobject - destinationRule}
                "intervalSweep": '', //time limit before the removed hosts are added back to the load balancing
pool. - {istioobject - destinationRule}
                }

        // credentials for mTLS.
        "Servicecertificate" : "" // Present actual certificate here.
        "ServicePrivateKey" : "" // Present actual private key here.
        "caCertificate" : "" // present the trusted certificate to verify the client connection, Required only
when mtls mode is MUTUAL

        // Access Control
        "namespaces": [] // Workloads from this namespaces can access the inbound service - {istioobject -
authorizationPolicy}
        "serviceAccountAccess" : {[ "SaDetails": ["ACTION": "URI"]} // {istioobject - authorizationPolicy, will
be applied for the inbound service}

        }
}
```

DELETE

DELETE

```
DELETE
URL: /v2/projects/{project-name}/composite-apps/blue-app/{version}/traffic-intent-set/inbound-intents/<name>

RETURN STATUS: 204
```

2. Outbound access

POST -

POST

```
URL: /v2/projects/{project-name}/composite-apps/{composite-app-name}/{version}/traffic-group-intent/outbound-
intents/
POST BODY:
{
        "metadata": {
        "name": "<name>" // unique name for each intent
    "description": "connectivity intent add client communication"
        "application": "<app1>",
        "userdata1": <>,
        "userdata2": <>
        }

        spec: {
                "clientServiceName": "<>", // Name of the client service
                "type": "", // options are istio, k8s and external
                "inboundServiceName": "<>"
                "headless": "false", // default is false. Option "True" will generate the required configs for
all the instances of headless service
        }
}

RETURN STATUS: 201
RETURN BODY:
{
  "name": "<name>"
  "Message": "Client created"
}
```

3. Compound Service access

POST

```
URL: /v2/projects/{project-name}/composite-apps/blue-app/{version}/traffic-intent-set/compound-intents/

POST BODY:
{
        "metadata": {
        "name": "<>" // unique name for each intent
    "description": "connectivity intent for inbound communication"
        "userdata1": <>,
        "userdata2": <>
        }

        "spec": {
        "application": "<app1>",
        "externalPrefix": "/canary"

        }
}

RETURN STATUS: 201
RETURN BODY:
{
  "name": "<name>"
  "Message": "inbound service created"
}
```

Note - After the compound intent is created, Call the inbound services under it and make sure you assign the weightage to each service under it. As shown in the below example

POST

```
URL: /v2/projects/{project-name}/composite-apps/blue-app/{version}/traffic-intent-set/compound-intents/
{compoungd-intent-name}/inbound-intents/

POST BODY:
{
        "metadata": {
        "name": "<>" // unique name for each intent
    "description": "connectivity intent for inbound communication"
        "userdata1": <>,
        "userdata2": <>
        }

        "spec": { // update the memory allocation for each field as per OpenAPI standards
        "application": "<app1>",
        "servicename": "httpbin" //actual name of the client service - {istioobject - serviceEntry of client's
cluster}
        "externalName": "httpbin.k8s.com"
        "traffic-weight": "50" // Default is "". Used for redirecting traffic percentage when compound API is
called
        "protocol": "HTTP",
        "headless": "false", // default is false. Option "True" will make sure all the instances of the
headless service will have access to the client service
        "mutualTLS": "MUTUAL", // default is simple. Option MUTUAL will enforce mtls {istioobject -
destinationRule}
        "port" : "80", // port on which service is exposed as through servicemesh, not the port it is actually
running on
        "serviceMesh": "istio", // get it from cluster record
        "sidecar-proxy": "yes", // The features (mTLS, LB, Circuit breaking) are not available to services
without istio-proxy. Only inbound routing is possible.
        // Traffic management fields below are valid only if the sidecar-proxy is set to "yes"
        "traffic-management-info" : {
        // Traffic configuration - Loadbalancing is applicable per service. The traffic to this service is
distrbuted amongst the pods under it.
        "loadbalancingType": "ConsistenHash", // "Simple" and "consistentHash" are the two modes - {istioobject
- destinationRule}
        "loadBalancerMode": "httpCookie" // Modes for consistentHash - "httpHeaderName", "httpCookie",
"useSourceIP", "minimumRingSize", Modes for simple - "LEAST_CONN", "ROUND_ROBIN",         "RANDOM",
"PASSTHROUGH" // choices of the mode must be explicit - {istioobject - destinationRule}
        "httpCookie": "user1" // Name of the cookie to maitain sticky sessions - {istioobject - destinationRule}

        // Circuit Breaking
        "maxConnections": 10 //connection pool for tcp and http traffic - {istioobject - destinationRule}
        "concurrenthttp2Requests": 1000 // concurent http2 requests which can be allowed - {istioobject -
destinationRule}
        "httpRequestPerConnection": 100 // number of http requests per connection. Valid only for http traffic
- {istioobject - destinationRule}
        "consecutiveErrors": 8 // Default is 5.  Number of consecutive error before the host is removed -
{istioobject - destinationRule}
        "baseEjectionTime" : 15 // Default is 5, - {istioobject - destinationRule}
        "intervalSweep": 5m, //time limit before the removed hosts are added back to the load balancing pool. -
{istioobject - destinationRule}
        }

        // credentials for mTLS.
        "Servicecertificate" : "" // Present actual certificate here.
        "ServicePrivateKey" : "" // Present actual private key here.
        "caCertificate" : "" // present the trusted certificate to verify the client connection, Required only
when mtls mode is MUTUAL
        }
}

RETURN STATUS: 201
RETURN BODY:
{
  "name": "<name>"
  "Message": "inbound service created"
}
```

Scenarios supported for the current release

Nature of application	Page link	comments
HTTP	**HTTP**	
HTTPS	HTTPS	
TCP	TCP	

Development

1. go API library - https://github.com/gorilla/mux
2. backend - mongo - https://github.com/onap/multicloud-k8s/tree/master/src/k8splugin/internal/db - Reference
3. intent to config conversion - use go templates and admiral? https://github.com/istio-ecosystem/admiral
4. writing the config to etcd - WIP
5. Unit tests and Integration test - go tests

External DNS - Design and intent API

See here: External DNS provider update design and intent API

External application communication intents

Considering DNS resolution, No DNS resolution (IP addresses), Egress proxies of the Service Mesh, Third-party egress proxy

User facing communication intents

Considering Multiple DNS Servers

Considering multiple user-facing entities

Considering RBAC/ABAC

Internal Design details

Guidelines that need to keep in mind

- Support for metrics that can be retrieved by Prometheus
- Support for Jaeger distributed tracing by including open tracing libraries around HTTP calls.
- Support for logging that is understood by fluentd
- Mutual exclusion of database operations (keeping internal modules accessing database records simultaneously and also by replication entities of the scheduler micro-service).
- Resilience - ensure that the information returned by controllers is not lost as the synchronization of resources to remote edge clouds can take hours or even days when the edge is not up and running and possibility of restart of scheduler micro service in the meantime.
- Concurrency - Support multiple operations at a time and even synchronizing resources in various edge clouds in parallel.
- Performance - Avoiding file system operations as much as possible.

Modules (Description, internal structures etc..)

Service Mesh Config:

Main Function: the module is invoked by traffic controller after traffic controller receives intents from external world, and parses requests from traffic controller and extracts some key information to assemble a new yaml file for creating instances of inbound services and clients based on istio.

Main Operations:

1. create/destroy inbound services (API: Add Inbound service)
2. create/destroy client services (API: Add Clients)
3. create/remove security details for client services (API: Add Security details for clients)
4. create/destroy ServiceEntry for inbound services used by clients
5. create/destroy DestinationRules for both inbound and client services
6. create/destroy VirtualService for client services
7. create/destroy AuthorizationPolicy for inbound services used by clients

The key information includes but not limited:

- client name
- inbound service name
- protocol: http/https/tcp
- TLS options: no/simple/mutual
- port

The interface between SM config and Traffic Controller: maybe via gPRC, and APIs are TBD

Traffic Controller

Main Function: it acts as main controlling loop/daemon, and receives the request in a form of REST from external modules e.g. orchestrator. Then it parses these requests and figures out the exact purposes which these requests want to express e.g. service creation, DNS update or workload adjust. Afterwards, it invoke corresponding components like SM config, DNS updater, to fulfill these requirements by creating and configuring related uServices based on the various mechanisms of istio.

Main steps:

0. Traffic controller need to be registered in orchestrtor by calling the APIs provided by orchestrtor

1. Orchestrtor starts to instantiate the traffic controller

2. Traffic controller finds the config files about various plugins like SM config, Loadbalancers and DNS updater from some certain locations, and then instantiate these plugins. Here, these plugins may be defined as istio VirtualService and their associated yaml files should be provided beforehand.

3. Traffic controller need to have some health-check about the instances of these plugins and make sure they are up and running well(some heath-check criteria also need to be defined).

4. Traffic controller may need to notify orchestrator that it, including the plugins, is ready to serve (which API provided by orchestrator should be invoked?).

5. At this moment, orchestrator can start to monitor and manage the life-cycle of traffic controller. And the way/APIs of monitor and manage need to be clarified. (Is HA is required for traffic controller?)

6. Users/admin are allowed to send their request to create uServices or access the running uServices directly via REST, like the inbound/client services creation. After traffic controller convert the intents to service description, the generated yml files which will be used by istio to create uSevices should be given to workload scheduler/placement helper to place and instantiate these uService on edge cloud clusters. Namely, traffic controller need to inform the workload scheduler/placement helper that there are some uServices to be placed and instantiated on edge cloud clusters.

7. Traffic controller need to call DNS module to expose the domain names of services to external world, after it is aware of these uServices have been instantiated on edge cloud clusters. (how is traffic controller aware of the accomplishment of of uServices instantiation?)

8. Traffic controller may need to manage the lifecycle of uServices (or done by some modules within orchestrator?) by a way e.g. detecting the heartbeat from various uServices periodically e.g. one check per 10 seconds.

9. Considering the HA, traffic controller should instantiate at least 2 of those plugins, and should be able to monitor the health of those instances of plugins. when any of instances is down, traffic controller can restart/recreate one for it again.

LoadBalancer (aka GSLB/LB controller?)

GSLB(Geo-replicated Services LB/LB controller) is used to balance incoming load across multiple istio-ingress-gateways.
It shall be able to be aware of the run-time load of various working uService instances which are distributed on different edge clouds or the general load level of each kind of uService in each edge cloud by interacting with actual LB(e.g. metallb, or other module except metallb) running on one edge cloud. Metallb is responsible for the load balance jobs between instances of uService within an edge cloud, while GSLB(controller) shall be in charge of guiding the traffic load to multiple edge clouds. GSLB(LB controller) should be aware of the public IPs (achieved by using metallb? or SDWAN?) of all edge clouds.

GSLB(LB controller) get requests from external users, and it evaluates the load level of the uServices that the user want to access on various edge clouds, and then choose one certain edge cloud on which the targeted(or a set of) uService instance is running, next GSLB returns the domain name of the chosen uService instance to users, so users can utilize this new domain name to access their expected uService. These steps mentioned above imply that those domain names associated with different uServices distributed on edge clouds should point to the IP address of center cloud on which GSLB is running first, after GSLB figure out to which edge cloud the users' request should be forwarded, the real domain name of uServices is given back to users by GSLB.

....

Sequence flows

Test cases