EMCO Architecture and Design

Background

Edge Multi-Cluster Orchestrator (EMCO), is a geo-distributed application orchestrator for Kubernetes*. EMCO operates at a higher level than Kubernetes and interacts with multiple edge servers and clouds that are running Kubernetes. EMCO's main objective is to automate the deployment of applications and services across multiple clusters. It acts as a central orchestrator that can manage edge services and network functions across geographically distributed edge clusters from different third parties.

Increasingly we see a requirement for deploying 'composite applications' in multiple geographical locations. Some of the catalysts for this change are:

- Latency - requirements for new low latency application use cases such as AR/VR. Ultra low latency response is needed in IIOT for example. This requires supporting some of the application functionality on edges closer to the user.
- Bandwidth - processing data on edges avoids the costs associated with transporting the data to clouds for processing.
- Context/Promixity - running some part of the application on edge servers near the user when they require local context
- Privacy/Legal - some data may have a legal requirement to remain in a geographic location.

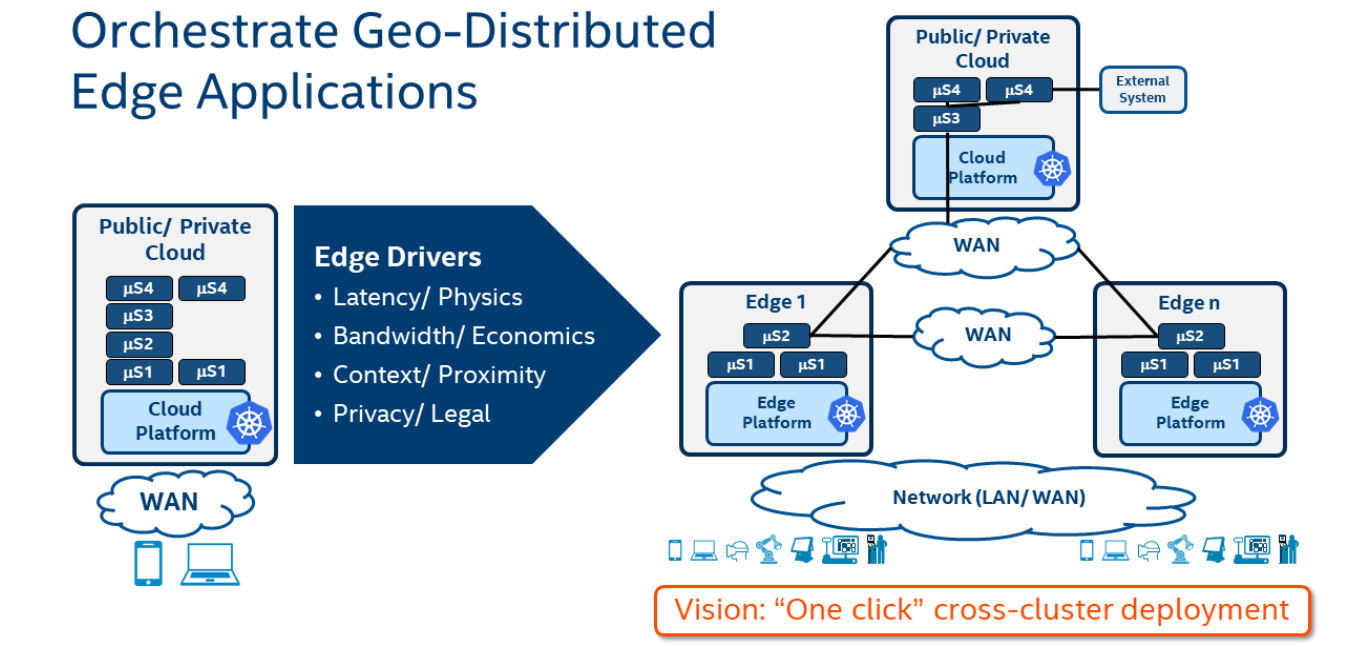

Figure 1 - Orchestrate GeoDistributed Edge Applications

NOTE: A 'composite application' is a combination of multiple applications with each application packaged as a Helm Chart. Based on the deployment intent, various applications of the composite application get deployed at various locations, and get replicated in multiple locations.

The life cycle management of composite applications is complex. Instantiation, updates and terminations of the composite application across multiple K8s clusters (edges and clouds), monitoring the status of the composite application deployment, Day 2 operations (modification of the deployment intent, upgrades, etc..) are a few examples of these complex operations.

For a given use case, the number of K8s clusters (edges or clouds) could be in tens of thousands. The number of composite applications that need to be managed could be in the hundreds. The number of applications in a composite application could be in the tens. The number of micro-services in each application of the composite application can be in the tens. Moreover, there can be multiple deployments of the same composite application for different purposes. To reduce this complexity, all of these operations can be automated. A Multi-Edge and Multi-Cloud distributed application orchestrator enables one-click deployment of the composite applications and makes a one simple dashboard to know the status of the composite application deployment at any time possible.

Compared with other multiple-cluster orchestration, EMCO focuses on the following functionalities:

- Enrolling multiple geographically distributed clusters.
- Orchestrating composite applications (composed of multiple individual applications) across different clusters.
- Deploying edge services and network functions to different nodes spread across different clusters.
- Monitoring the health of the deployed edge services and network functions across different clusters.
- Orchestrating edge services and network functions with deployment intents based on compute, acceleration, and storage requirements.
- Supporting multiple tenants from different enterprises while ensuring confidentiality and full isolation between the tenants.

EMCO Terminology and Concepts

Term	Description
Cluster Provider	The cluster provider is the entity that owns and registers clustesrs with EMCO.
Cluster	Clusters are registered with EMCO. Access to a given cluster may be via kubeconfig or via one of the supported gitOps methods.
Project	The project provides a means of grouping collections of applications. Multiple applications may exist for any project. Projects allow for defining applications with different tenants - i.e. a given tenant is associated with one or more projects that are distinct from another tenants projects.
Logical Cloud	The logical cloud is an EMCO concept that groups a number of clusters together and provides a common set of namespace, permissions and quotas. Logical clouds are defined under projects and are the target deployment destination of EMCO Composite Applications. Placement scheduling will ultimately determine which cluster(s) in a logical cloud any specific component of a composite application will deployed to.

Composite Application	The composite application is a combination of multiple applications that work together. Each application in the composite application is a Helm chart. Through the use of placement intents, EMCO allows different applications in the composite application to be deployed (and replicated as necessary) to different sets of clusters. For example, a composite application composed of a user facing application along with a database application could be deployed by EMCO such that the user facing application is deployed to many edge clusters while the database application is deployed on a centralized cluster.
Deployment Intents	The deployment intents are the various placement and action intents that have been defined to control the deployment of an EMCO composite application. Placement and action intents are defined and handled by various EMCO controllers. Different composite application deployments may use differents sets of placement and action intents. EMCO does not expect the Helm charts which are onboarded and comprise a composite application to be edited or modified. Instead, any customizations or additional resources required to ensure the composite application operates correctly when deployed are defined by the set of deployment intents.
Placement Intents	Placement intents are used to identify to which clusters the resources of each application of a composite application are to be deployed. Each composite application is required to define a set of generic placement intents. Additional placement intents, for more specialized purposes such as hardware capabilities, latency, etc., may be defined as well. The additional placement intents will refine the results of the generic placement.
Action Intents	Action intents are provided by special action controllers and are used to perform specific customizations to the EMCO composite application before it is deployed. After EMCO has processed the placement intents for a specific composite application deployment, the action intents are processed. The action intents may do anything from customizing existing application resources to adding new resources that need to be deployed. EMCO provides many action controllers and additional controllers may be easily written and used with EMCO.
EMCO Database	The EMCO data base (currently MongoDB) is used primarily to store the resources onboarded by the EMCO REST APIs (e.g. composite applications, cluster data, all the various deployment intents).
EMCO AppContext	The EMCO AppContext is a data store (currently `etcd`) that is used by EMCO to prepare and manage the resources that will be deployed to edge clusters. When EMCO is preparing a composite application for deployment by processing the various deployment intents, the results are prepared in the AppContext. Once all placement and action intents have been handled, the AppContext contains the set of resources that need to be applied to the set of target clusters. The EMCO `rsync` ('resource synchronizer and status collector') microservice then deploys the resources to the edge clusters. The AppContext is also used to collect status information back from the edge clusters as detected by `rsync`.

EMCO Architecture

The following diagram depicts a high level overview of the EMCO architecture.

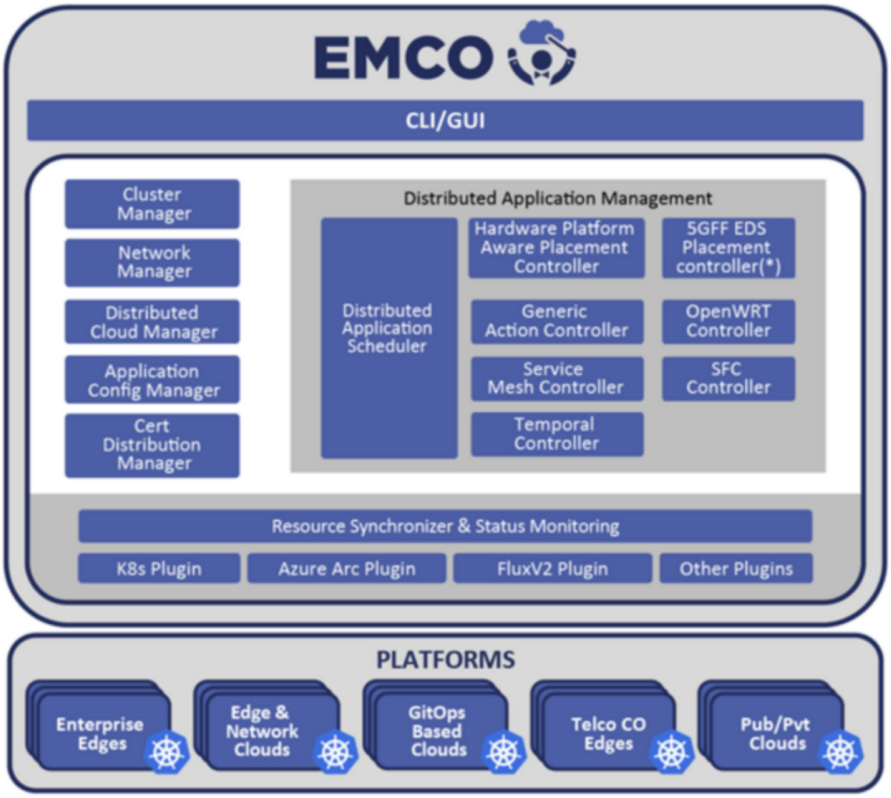

Figure 2 - EMCO Architecture

- Cluster Registration Controller registers clusters by cluster owners.
- Distributed Application Scheduler provide simplified and extensible placement.
- Network Configuration Management handles creation and management of virtual and provider networks.
- Hardware Platform Aware Controller enables scheduling with auto-discovery of platform features/ capabilities.
- Distributed Cloud Manager presents a single logical cloud from multiple edges.
- Secure Mesh Controller auto-configures both service mesh (ISTIO) and security policy (NAT, firewall).
- Secure WAN Controller automates secure overlays across edge groups.
- Resource Synchronizer manages instantiation of resources to clusters.
- Monitoring covers distributed applications.

In addition to the above components, EMCO provides a number of other controllers. All of the EMCO microservice will be summarized below. Additional information about some of the EMCO controller designs can be found by looking at the documents available in the [EMCO docs] folder. Additionally, EMCO provides many example use cases in the [EMCO Examples] folder and most of these examples provide `Readme` documents that describe how the specific example use case works - i.e. takes advantage of some subset of EMCO features and controllers.

Cluster Registration - `clm`

The `clm` microservice exposes a RESTful API. Users can register cluster providers and their associated clusters through these APIs. After preparing edge clusters and cloud clusters, which can be any Kubernetes cluster, users can onboard those clusters to EMCO by creating a cluster provider and then adding clusters to the cluster provider. After cluster providers are created, the KubeConfig files of the edge and cloud clusters should be provided to EMCO as part of the multi-part POST call to the Cluster API. Alternatively, EMCO supports clusters that are accessed via gitOps. In this case, appropriate gitOps credential and repository information is provided to the cluster registration. See [EMCO gitOps Support] for more information.

Additionally, after a cluster is created, labels and key value pairs can be added to the cluster via the EMCO API. Clusters can be specified by label when preparing placement intents.
> **NOTE**: The cluster provider is someone who owns clusters and registers them to EMCO. If an Enterprise has clusters, for example from AWS, then the cluster provider for those clusters from AWS is still considered as from that Enterprise. AWS is not the provider. Here, the provider is someone who owns clusters and registers them to EMCO. Since AWS does not register their clusters to EMCO, AWS is not considered the cluster provider in this context.

Distributed Application Scheduler - `orchestrator`

The distributed application scheduler microservice, known as the `orchestrator`, functionality includes:
- Project Management which provides multi-tenancy in the application from a user perspective.
- Composite App Management manages composite apps that are collections of Helm Charts, one per application.
- Composite Profile Management manages composite profiles that are collections of profiles, one per application.
- Deployment Intent Group Management manages Intents for composite applications.
- Controller Registration manages placement and action controller registration, priorities etc.
- Status Notifier framework allows user to get on-demand status updates or notifications on status updates.
- Scheduler for composite application deployments:
- Placement Controllers: Generic Placement Controller.
- Action Controllers.

Orchestrator Lifecycle Operations

The Distributed Application Scheduler supports operations on a deployment intent group resource to instantiate the associated composite application with any placement and action intents performed by the registered placement and action controllers. The basic flow of lifecycle operations on a deployment intent group after all the supporting resources have been created via the APIs are:

Operation	Description
approve	marks that the deployment intent group has been approved and is ready for instantiation.
instantiate	the Distributed Application Scheduler prepares the application resources for deployment, and applies placement and action intents before invoking the Resource Synchronizer to deploy them to the intended remote clusters. In some cases, if a remote cluster is intermittently unreachable, the instantiate operation will retry the instantiate operation for that cluster until it succeeds.
status	(may be invoked at any step) provides information on the status of the deployment intent group.
terminate	terminates the application resources of an instantiated application from all of the clusters to which it was deployed. The terminate operation will cause the instantiate operation to complete (i.e. fail), before the termination operation is performed.
stop	In some cases, if the remote cluster is intermittently unreachable, the Resource Synchronizer will continue retrying an instantiate or terminate operation. The stop operation can be used to force the retry operation to stop, and the instantiate or terminate operation will complete (with a failed status). In the case of terminate, this allows the deployment intent group resource to be deleted via the API, since deletion is prevented until a deployment intent group resource has reached a completed terminate operation status.
update	After a deployment intent group has been instantiated, it may be updated (applications may be modified and/or various deployment intents may be added or modified). The update operation is used to deploy the changes to the running / deployed composite application.
rollback	After a deployment intent group has been instantiated and updated one or more times, it is possible to rollback the composite application to a previously deployed version.

Refer to [EMCO Resource Lifecycle Operations] for more details.

Placement and Action Controllers in EMCO

This section illustrates some key aspects of the EMCO controller architecture. Depending on the needs of a composite application, intents that handle specific operations for application resources (e.g. addition, modification, etc.) can be created via the APIs provided by the corresponding controller API. The following diagram shows the sequence of interactions to register controllers with EMCO.

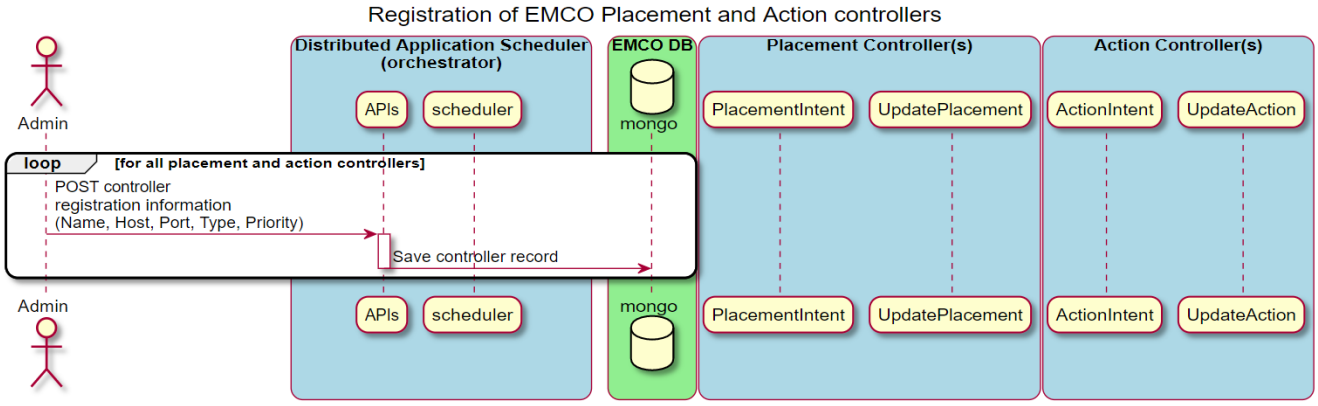

Figure 3 - Register placement and action controllers with EMCO

This diagram illustrates the sequence of operations taken to prepare a Deployment Intent Group that utilizes some intents supported by controllers. The desired set of controllers and associated intents are included in the definition of a Deployment Intent Group to satisfy the requirements of a specific deployed instance of a composite application.

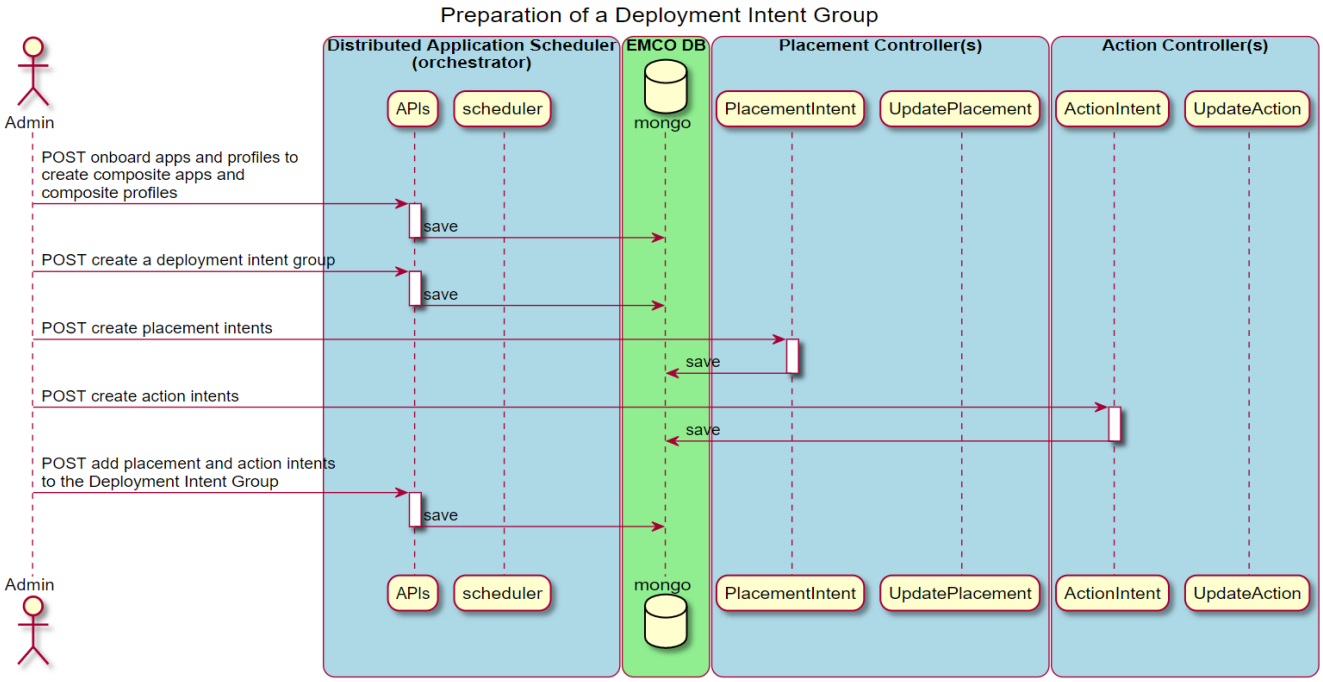

Figure 4 - Create a Deployment Intent Group

When the Deployment Intent Group is instantiated, the identified set of controllers are invoked in order to perform their specific operations.

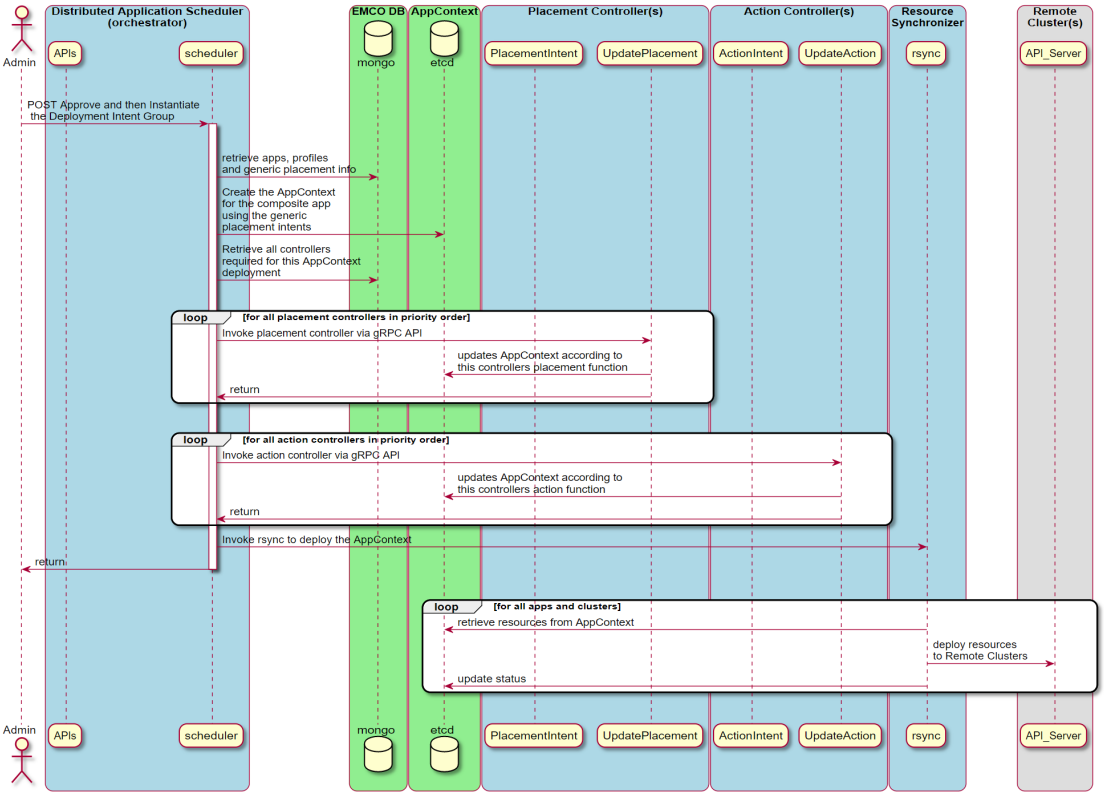

Figure 5 - Instantiate a Deployment Intent Group

In the initial release of EMCO, a built-in generic placement controller is provided in the `orchestrator`. HPA Placement Controller is an example of a Placement Controller. Some action controllers provided with EMCO are the OVN Action, Traffic, and Generic Action controllers.

Orchestrator Placement Controllers

EMCO currently provides the following placement controllers:

Generic Placement Controller - `orchestrator`

- The generic placement controller is built into the orchestrator and is always invoked as the first placement controller.

Hardware Platform Aware Placement Controller - `hpa-plc`

This placement controller provides intents that can be used to ensure apps are placed on clusters which support specified platform or hardware capabilities.

Orchestrator Action Controllers

EMCO currently provides the following action controllers:

OVN Action controller - `ovnaction`

This action controller provides intents that can be used to specify that specific workloads in an application (say the Pod template of a Deployment resource), needs to be attached to an additional network interface in the edge cluster. On deployment, this action controller will annotate the Pod template with the appropriate annotations to connect the Pod to the identified network in the edge cluster.
`ovnaction` supports specifying interfaces which attach to networks created by the Network Configuration Management microservice.

Generic Action Controller - `gac`

The Generic Action Controller provides intents which can be used to create new resources or customize resources associated with an app in a deployment intent group. Changes can be applied to every cluster in a deployment intent group, or to specific clusters. Customizations to existing resources can be handled via JSON Patching or Kubernetes Strategic Merge.

Hardware Platform Aware Action Controller - `hpa-ac`

This placement controller provides intents that can be used to ensure apps are configured to request the identified hardware capabilities.

SFC controller - `sfc`

The Service Function Chaining action controller provides intents that can be used to create a Service Function Chain from a number of the apps in the deployment intent group. This is a capability supported by the Nodus CNI. A network-chaining CR will be created and added to the deployment intent group.
Refer to [SFC Overview] and [SFC Example] for more more information about the `sfc` and `sfcclient` action controllers.

SFC Client controller - `sfcclient`

The SFC Client controller is used to attach components of a composite application to an SFC that has been deployed on the edge cluster(s). This controller will ensure that the appropriate Pod templates are labelled such that the Nodus CNI on the edge cluster will connect them to the identified SFC upon deployment.

Distributed Traffic Controller - `dtc`

The Distributed Traffic Controller provides intents which are used to control the network traffic between apps in the composite application, both within a given cluster as well as between clusters. This covers the range from network policy to the configuration of service mesh resources. The `dtc` supports a sub-controller architecture, where specific sub-controllers specialize in specific functional areas.
NOTE:For network policy to work, the edge cluster must have network policy support using a CNI such as calico.

Istio Traffic Subcontroller - `its` (a DTC sub-controller)

The `its` controller handles DTC intents and ensure the configuration of Istio resources (e.g. Service Entries, Virtual Serives, Routes, etc.) to ensure that composite application workloads are configured to use mTLS, etc. both inter and intra cluster.

Network Policy Subcontroller - `nps` (a DTC sub-controller)

The `nps` controller handles DTC intents to ensure that Kuberenetes NetworkPolicy is applied appropriately in the destination clusters.

Service Discovery Subcontroller - `sds` (a DTC sub-controller)

The `sds` controller is used to monitor the status of a deployed composite application and find the IP address of identified services once these have been deployed in the edge clusters.

SDEWAN Controller - `swc` (a DTC sub-controller)

The `swc` is used to ensure appropriate SDEWAN resources are configured in the edge clusters to allow communication between clusters via the SDEWAN CNF. Refer to [SDEWAN Example] for more information.

Network Configuration Management - `ncm`

The `ncm` microservice consists of:
- Provider Network Management to create provider networks.
- Virtual Network Management to create dynamic virtual networks.
- Controller Registration to manage network plugin controllers, priorities etc.
- Status Notifier framework allows users to get on-demand status updates or notifications on status updates.
- Scheduler with Built in Controller - Nodus (aka OVN-for-K8s-NFV Plugin) CNI Controller.

NCM Lifecycle Operations

The Network Configuration Management microservice supports operations on the network intents of a cluster resource to instantiate the associated provider and virtual networks that have been defined via the API for the cluster. The basic lifecycle operations flow on a cluster, after the supporting network resources have been created via the APIs, are:

Operation	Description
apply	the Network Configuration Management microservice prepares the network resources and invokes the Resource Synchronizer to deploy them to the designated cluster.
status	(may be invoked at any step) provides information on the status of the cluster networks.
terminate	terminates the network resources from the cluster to which they were deployed. In some cases, if a remote cluster is intermittently unreachable, the Resource Synchronizer may still retry the instantiate operation for that cluster. The terminate operation will cause the instantiate operation to complete (i.e. fail), before the termination operation is performed.
stop	In some cases, if the remote cluster is intermittently unreachable, the Resource Synchronizer will continue retrying an instantiate or terminate operation. The stop operation can be used to force the retry operation to stop, and the instantate or terminate operation will be completed (with a failed status). In the case of terminate, this allows the deployment intent group resource to be deleted via the API, since deletion is prevented until a deployment intent group resource has reached a completed terminate operation status.

Distributed Cloud Manager - `dcm`

The Distributed Cloud Manager (DCM) provides the Logical Cloud abstraction and effectively completes the concept of "multi-cloud". One Logical Cloud is a grouping of one or many clusters, each with their own control plane, specific configurations and geo-location, which get partitioned for a particular EMCO project. This partitioning is made via the creation of distinct, isolated namespaces in each of the Kubernetes clusters that thus make up the Logical Cloud.

A Logical Cloud is the overall target of a Deployment Intent Group and is a mandatory parameter (the specific applications under it further refine what gets run and in which location). A Logical Cloud must be explicitly created and instantiated before a Deployment Intent Group can be instantiated.

Due to the close relationship with Clusters, which are provided by Cluster Registration (clm), it is important to understand the mapping between the two. A Logical Cloud groups many Clusters together but a Cluster may also be grouped by multiple Logical Clouds, effectively turning the cluster multi-tenant. The partitioning/multi-tenancy of a particular Cluster, via the different Logical Clouds, is done today at the namespace level: different Logical Clouds access different namespace names, and the name is consistent across the multiple clusters of the Logical Cloud.

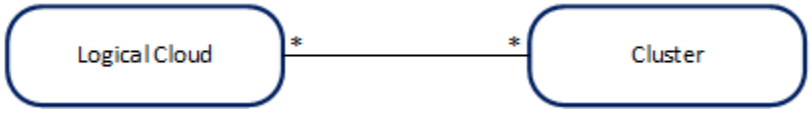

Figure 6 - Mapping between Logical Clouds and Clusters

Standard Logical Clouds

Logical Clouds were introduced to group and partition clusters in a multi-tenant way and across boundaries, improving flexibility and scalability. A Standard Logical Cloud is the default type of Logical Cloud providing just that much. When projects request a Logical Cloud to be created, they provide what permissions are available, resource quotas and clusters that compose it. The Distributed Cloud Manager, alongside the Resource Synchronizer, sets up all the clusters accordingly, with the necessary credentials, namespace/resources, and finally generating the kubeconfig files used to authenticate and reach each of those clusters in the context of the Logical Cloud.

Admin Logical Clouds

In some use cases, and in the administrative domains where it makes sense, a project may want to access raw, unmodified, administrator-level clusters. For such cases, no namespaces need to be created and no new users need to be created or authenticated in the API. To solve this, the Distributed Cloud Manager introduces Admin Logical Clouds, which offer the same consistent interface as Standard Logical Clouds to the Distributed Application Scheduler. Being of type Admin means this is a Logical Cloud at the Administrator level. As such, no changes will be made to the clusters themselves. Instead, the only operation that takes place is the reuse of credentials already provided via the Cluster Registration API for the clusters assigned to the Logical Cloud (instead of generating new credentials, namespace/resources and kubeconfig files.)

Privileged Logical Clouds

This type of Logical Cloud provides most of the capabilities that an Admin Logical Cloud provides but at the user-level like a Standard Logical Cloud. New namespaces are created, with new user and kubeconfig files. However, the EMCO project can now request an enhanced set of permissions/privileges, including targeting cluster-wide Kubernetes resources.

DCM Lifecycle Operations

Prerequisites to using Logical Clouds:
* With the project-less Cluster Registration API (`clm`), create the cluster providers, clusters, and optionally cluster labels.
* With the Distributed Application Scheduler API (`orchestrator`), create a project which acts as a tenant in EMCO.

The basic flow of lifecycle operations to get a Logical Cloud up and running via the Distributed Cloud Manager API is:
* Create a Logical Cloud specifying the following attributes:
- Level: For Standard/Privileged Logical Clouds, set to 1. For Admin Logical Clouds, set to 0.
- (*for Standard/Privileged only*) Namespace name - the namespace to use in all of the Clusters of the Logical Cloud.
- (*for Standard/Privileged only*) User name - the name of the user that will be authenticating to the Kubernetes APIs to access the namespaces created.
* (*for Standard/Privileged only*) User permissions - permissions that the specified user will have in the specified namespace, in all of the clusters.
* (*for Standard/Privileged only*) Create resource quotas and assign them to the Logical Cloud. This specifies what quotas/limits the user will face in the Logical Cloud, for each of the Clusters.
* Assign the Clusters previously created with the project-less Cluster Registration API to the newly-created Logical Cloud.
* Instantiate the Logical Cloud. All of the clusters assigned to the Logical Cloud are automatically set up to join the Logical Cloud. Once this operation is complete, the Distributed Application Scheduler's lifecycle operations can be followed to deploy applications on top of the Logical Cloud.

Apart from the creation/instantiation of Logical Clouds, the following operations are also available:
* Terminate a Logical Cloud - this removes all of the Logical Cloud related resources from all of the respective Clusters.
* Delete a Logical Cloud - this eliminates all traces of the Logical Cloud in EMCO.

CA Certificate Distribution - `ca-certs`

The `ca-certs` controller provides intents which may be used to distribute Intermediate CA Certificates with common root CA's to sets of clusters. These intents can be used to either set the default CA Certificate of Istio in the edge clusters or two set up multiple CA Certificates in edge clusters, differentiated by namespace. This feature is a useful preparatory step that allows the `dtc` controller to configure mTLS connections between workloads on different clusters since the clusters will share a common CA root certificate.

Refer to [CA Cert Distribution Example] and [CA Cert Distribution Design] for more information.

Temporal Workflow Manager - `workflowmgr`

EMCO provides integration with the Temporal Workflow engine. Refer to [Temporal Workflows in EMCO] for more information.

Resource Synchronization and Monitoring - `rsync`

This microservice is the one which deploys the resources in edge/cloud clusters. The 'AppContext' created by various microservices is used by this microservice. It takes care of retrying, in case the remote clusters are temporarily unreachable.
The `orchestrator` and other controllers like `ncm`, `ca-certs` and `dcm` communicate with `rsync` using gRPC.

An internal plugin architecture is supported in `rsync` to allow support for communicating to edge clusters with a variety of methods. Currently communcations to clusters can be done via a `kubectl`-like method with `kubeconfig` files, or via a number of gitOps mechanisms. Each cluster that is onboarded to EMCO will include information appropriate for the access method that `rsync` will use to communicate with that cluster.

Rsync supports following gRPC methods:
- Instantiation
- Termination
- Update
- Read

Each gRPC request includes the AppContext on which the operation is requested.
AppContextID is copied in the active area in etcd, needed for Restart.

Rsync maintains a queue per AppContext to collect operations requested on that
AppContext. Each element in AppContext Queue stores AppContextElement.

Each AppContextElement consists of:
- Rsync event: Identifies the event, which can be instantiate, terminate, update etc.
- UCID: Contains the unique appContextID.
- Status: Reports whether status is pending or done.

One main thread runs per AppContext and picks up the next element from the AppContext queue in the “pending” state. If it is not running, the main thread is started.
The Status Flag is used by the orchestrator or other controllers to discover the current status of the AppContext. This will change as more events are processed.
If an event is not valid for the current state (for example terminate before instantiate) that event is skipped, and status marked as error in the queue. Rsync will then move on to the next event. Every event is completed before picking up the next event. One exception is terminate while waiting for cluster to be ready, see below.

The main thread for a AppContext starts a stop thread. Then the main thread picks the next pending event from the queue, decides on whether to apply, delete or read resources based on the event. It starts one thread per App and per cluster to handle the operation required. Within a cluster resources are handled in order as specified in the AppContext. Dependency between applications and resources will be added later.

Instantiation

Instantiate events follow this basic flow as outlined above.

Termination

If the terminate event is received while there are other events pending, only the current event being processed will be completed.
All other events will be skipped. For example, if the queue has items -> `instantiate`, `update` and rsync is processing instantiate and when the terminate event is added, then instantiation will be completed but update will be skipped. If the current event is waiting for cluster ready at the time terminate is received (and terminate is a valid state change), that wait is stopped, and that cluster will fail the current event (for example instantiation, update, etc).

Use of Stop flag in Rsync

Stop Event is different from other events. It is not a gRPC call. This event is communicated to Rsync using a AppContext Stop Flag. Rsync runs a thread per active AppContext that checks this flag every second to see if this flag is set. If set, all waiting threads are signaled to exit. Any installation/termination that is currently happening will finish. This ensures a way to stop waiting for cluster ready if user wants to do so. If Stop flag is set no other processing is possible on the AppContext.

Update

The orchestrator creates a new AppContext which includes all the desired resources after update/migrate. Rsync update gRPC is called with two AppContexts. The first is the Initial AppContext which must be in the instantiated state. The second is the Update AppContext. The Update happens in two phases.

The first phase is the Instantiation phase. This event takes two AppContexts: The first is the Update AppContext and the seconds is the Initial AppContext. The difference between the two AppContexts is captured. All resources that need to be modified are marked with skip=false. All new apps, clusters that are added will be marked as skip=false. A byte level comparison between the resources in the two AppContexts is made. If differences are found, then that resource is applied.

At the end of the Instantiation phase a new event called UpdateDeleteEvent is enqueued in the AppContext Queue. This is an internal event which is treated the same as other events. This event also takes two AppContexts: The first is the Initial AppContext and the seconds is the Update AppContext. This division ensures that any new event that comes for the Update AppContext will also be honored in the order received. Again, a difference between the two AppContexts is captured. All resources marked with skip=false are deleted including apps, clusters, and individual resources. At this stage nothing is deleted if the number of Initial and Update AppContext apps, clusters, and resources match.

Update also waits for the clusters to be ready just like other operations.

Rsync restart logic

Whenever rsync restarts, it restores those AppContextIDs which got cancelled during the processing phase when the rsync was restarted. Any AppContextID which is currently being processed by the rsync is called "active AppContextID". Whenever rsync starts handling an AppContextID, it enqueues it to the AppContextQueue and also records the active context in the "activecontext" area of `etcd`. For example, when we look into etcd, we could see a record similar to:

`/activecontext/99999999888/->99999999888`

This says that there exists an active AppContextID of 99999999888

Once the processing completes, the rsync calls for the deletion of the active AppContextID.

Whenever rsync starts or restarts, it checks for active AppContextIDs in the "activecontext" area of the etcd using the prefix key: `/activecontext`.
If it finds an active AppContextID, it creates the AppContextData and starts the main thread for handling the pending AppContextIDs which were recorded as active contextIDs.

Rsync state machine

Event	Valid Starting Current State	Desired State	Current State	Error State
InstantiateEvent	Created, Terminated, TerminationFailed, Instantiated, Instantiating, InstantiationFailed, Update, Updating, UpdateFailed	Instantiated	Instantiating	InstantiateFailed
TerminateEvent	Terminating, TerminationFailed, Instantiated, Instantiating, InstantiationFailed, Updated, Updating, UpdateFailed	Terminated	Terminating	TerminationFailed
UpdateEvent	Created, Updated	Updated	Updating	UpdateFailed
UpdateDeleteEvent	Instantiated	Instantiated	Instantiating	InstantiateFailed

Status Monitoring and Queries in EMCO

When a resource like a Deployment Intent Group is instantiated, status information about both the deployment and the deployed resources in the cluster are collected and made available for query by the API. The following diagram illustrates the key components involved. For more information about status queries see [EMCO Resource Lifecycle and Status Operations].

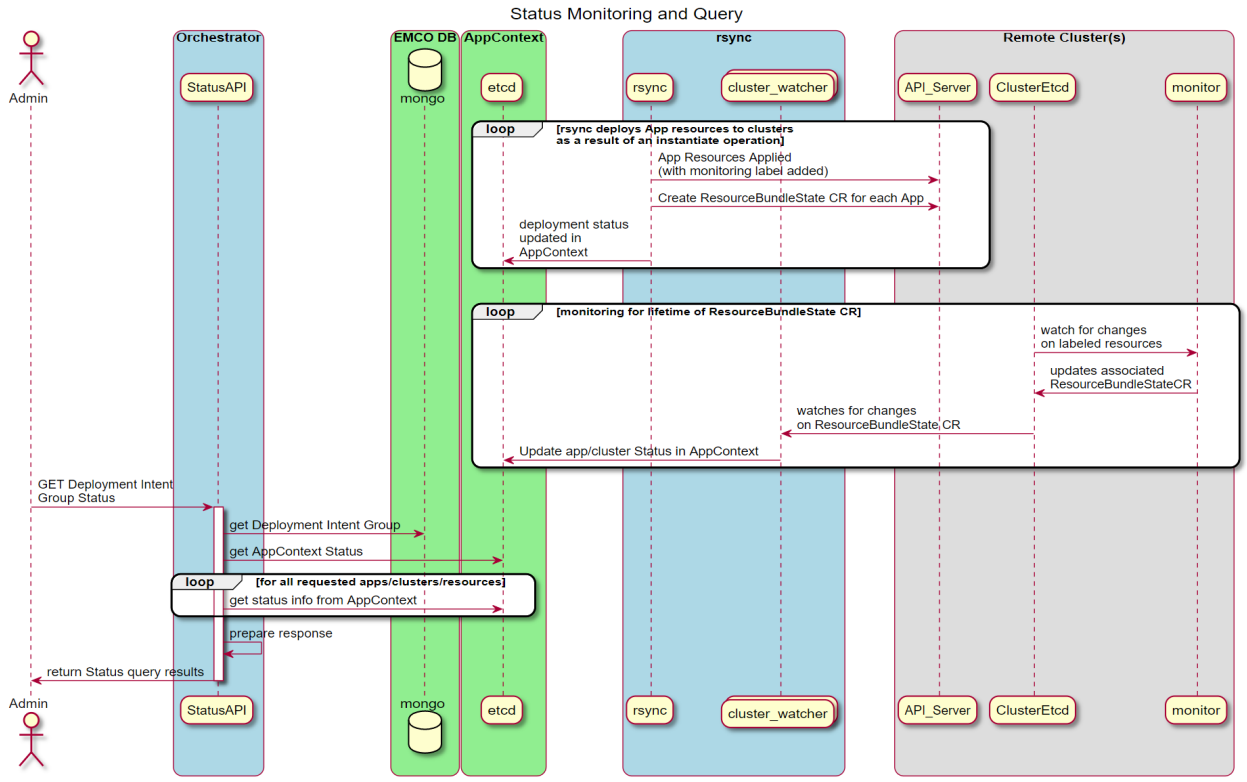

Figure 9 - Status Monitoring and Query Sequence

EMCO API

For user interaction, EMCO provides a [RESTful API]. Apart from that, EMCO also provides a CLI. For detailed usage, refer to [EMCO CLI]

NOTE: The EMCO RESTful API is the foundation for the other interaction facilities like the EMCO CLI, EMCO GUI and other orchestrators.

EMCO Authentication and Authorization

EMCO uses Istio* and other open source solutions to provide a Multi-tenancy solution leveraging Istio Authorization and Authentication frameworks. This is achieved without adding any logic to EMCO microservices.
- Authentication and Authorization for EMCO users is done at the Istio Ingress Gateway, where all the traffic enters the cluster.

- Istio along with authservice (an Istio ecosystem project) enables request-level authentication with JSON Web Token (JWT) validation. Authservice is an entity that works alongside Envoy proxy. It is used to work with external IAM systems (OAUTH2). Many Enterprises have their own OAUTH2 server for authenticating users and providing roles.

- Authservice and Istio can be configured to talk to multiple OAUTH2 servers. Using this capability EMCO can support multiple tenants.

- Using Istio AuthorizationPolicy access for different EMCO resources can be controlled based on roles defined for the users.

The following figure shows various EMCO services running in a cluster with Istio.

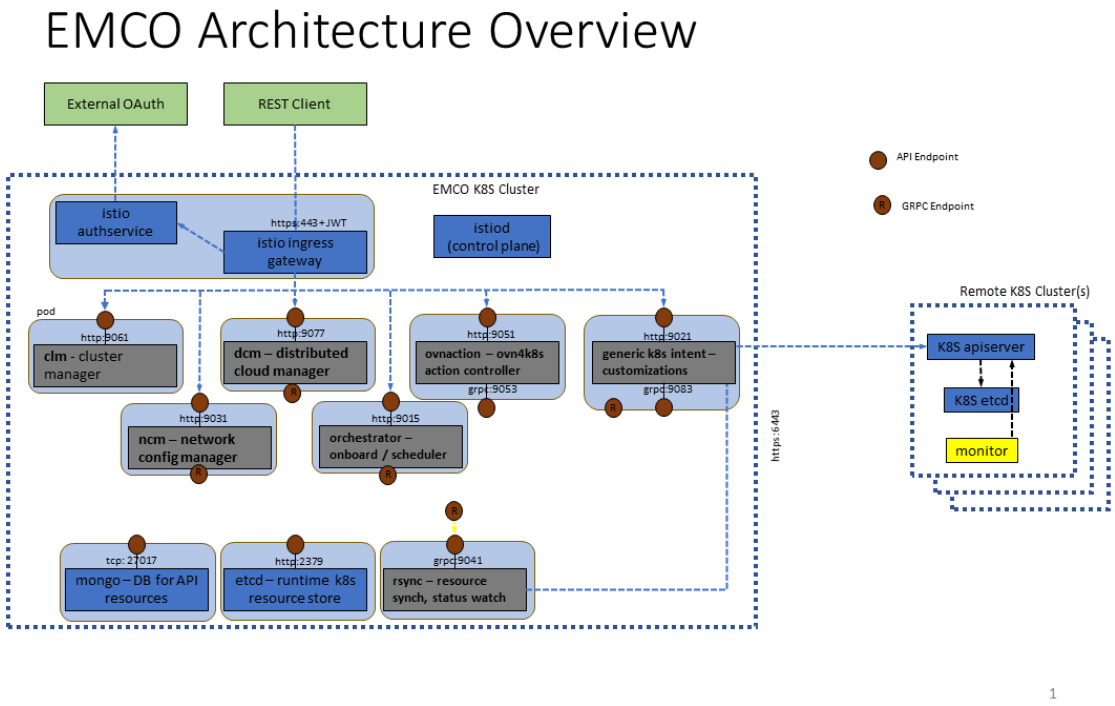

Figure 7 - EMCO setup with Istio and Authservice

The following figure shows the authentication flow with EMCO, Istio and Authservice

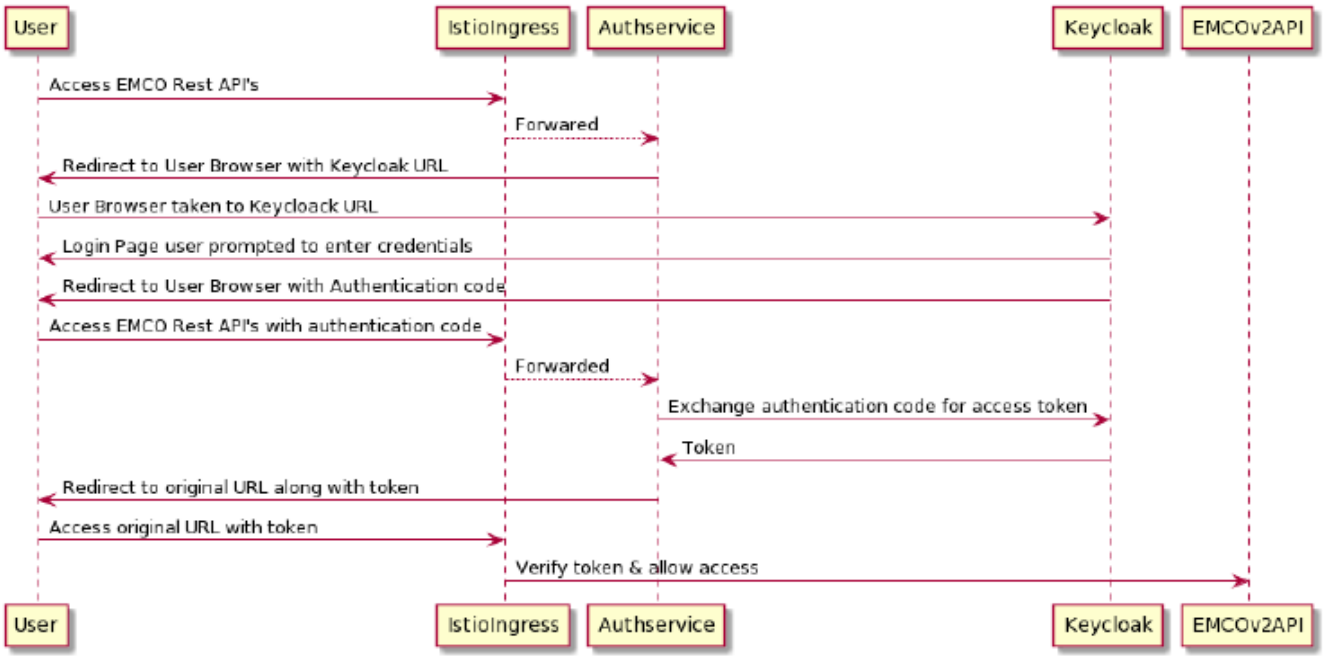

Figure 8 - EMCO Authenication with external OATH2 Server

Detailed steps for configuring EMCO with Istio can be found in the [EMCO Integrity and Access Management] documentation.

Steps for EMCO Authentication and Authorization Setup:
- Install and Configure Keycloak Server to be used in the setup. This server runs outside the EMCO cluster.
- Create a new realm, add users and roles to Keycloak.
- Install Istio in the Kubernetes cluster where EMCO is running.
- Enable Sidecar Injection in EMCO namesapce.
- Install EMCO in EMCO namespace (with Istio sidecars.)
- Configure Istio Ingress gateway resources for EMCO Services.
- Configure Istio Ingress gateway to enable running along with Authservice.
- Apply EnvoyFilter for Authservice.
- Apply Authentication and Authorization Policies.