

Use Service Layer API to Bring your Controller on Cisco IOS XR Router

Bring your protocol or controller on IOS XR router to interact with the network infrastructure layer components using Service Layer API.

For example, you can bring your controller to gain control over the Routing Information Base (RIB) tables and many more use cases.

- Get to Know Service Layer API, on page 1
- Enable Service Layer, on page 4
- Write Your Service Layer Client API, on page 5
- Preprogram Backup LSPs Using Service Layer API, on page 6

Get to Know Service Layer API

Service Layer API is a model-driven API over Google-defined remote procedure call (gRPC).

gRPC enables you to bring your applications, routing protocols, controllers in a rich set of languages including C++, Python, GO, and many more.

Service Layer API is available out of the box and no extra packages required.

In IOS XR, routing protocols use RIB, the MPLS label manager, BFD, and other modules, to program the forwarding plane. You can expose these protocols through the service layer API.

Benefits

The Service Layer API gives direct access to the Network Infrastructure Layer (Service-Adaptation Layer). Therefore, you have the following advantages:

- **High Performance**: Direct access to the Network Infrastructure Layer, without going through a Network state database, results in higher performance than equivalent Management APIs.

  For example, Batch updates straight to the Label Switching Data Base (LSDB), the Routing Information Base (RIB) (over gRPC). The LSDB stores label-to-address mappings for efficient traffic routing in Label-switching routers. And, RIB contains the active and potential routes to various network destinations.

- **Flexibility**: The Service Layer API gives you the flexibility to bring your Protocol or Controller over gRPC.

- **Offload low-level tasks to IOS XR**: IOS XR infrastructure layer handles the following. Hence, you can focus on higher-layer protocols and controller logic:
  - Conflict resolution
  - Transactional notifications
  - Data plane abstraction

Components of Service Layer API

The following are the components of the Service Layer API architecture:

- **Functionality Verticals/Domains**: The verticals define the broader capability categories supported by the API. The following are the supported verticals. Each vertical supports data structure and RPCs defined in gpb
  - **Initialization**: Handles global initialization, sets up an event notification channel using GRPC streaming capabilities.

    The initialization RPCs are mandatory. Use the initialization RPCs to connect a client to the gRPC server on the router. Also, to send heartbeats and termination requests from the server to the client.
  - **IPv4, IPv6 Route (RIB)**: Handles route manipulations (add, update, delete) for a certain VRF.
  - **MPLS**: Handles allocation of label blocks and any incoming MPLS label mapping to a forwarding function.
  - **Interface**: Handles subscription of the registered clients to the interface state event notifications.
  - **IPv4, IPv6 BFD**: Manages BFD sessions, and corresponding BFD session state notifications.
  - **Policy-Based Routing (PBR)**: Manages routing decisions based on user-defined policies rather than the default routing table.
- **Protobuf Schema/Model**: Use gRPC to model the service layer API.
- **gRPC**: gRPC utilizes GPB protobuf IDL by default to convert the models into bindings in various languages (c++, python, golang, and more). The gRPC server (running on the router) and the gRPC client use the generated bindings to serialize data and encode or decode the request or response between the server and the client.
- **Service Layer gRPC clients**: Based on the business needs, the gRPC clients for service layer can exist in one of the following ways:
  - On-box (agents runnning on their own sand-boxed third-party containers)
  - Off-box (within Controllers or other open-source tools)
- **gRPC Authentication Modes:**

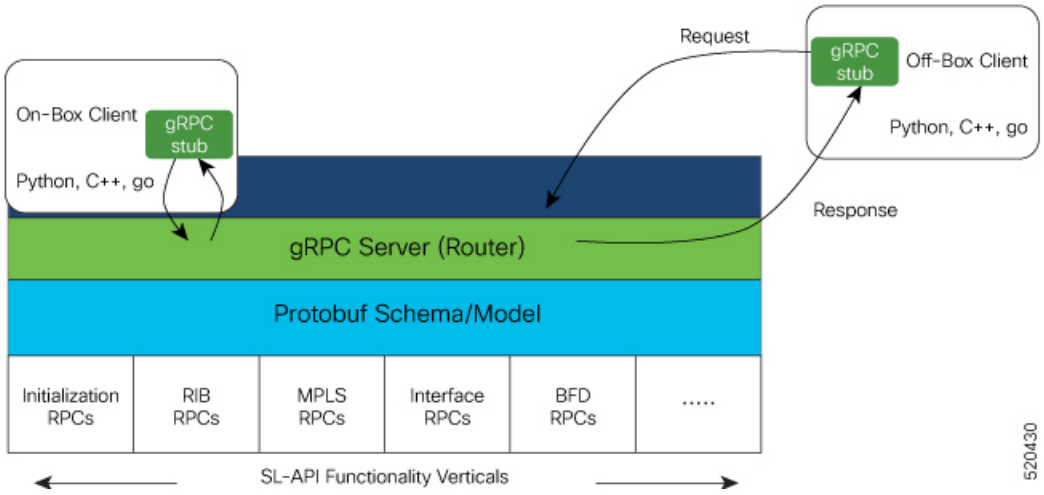
  gRPC supports the following authentication modes to secure communication between clients and servers. These authentication modes help ensure that only authorized entities can access the gRPC services, like gNOI, gRIBI, and P4RT. Upon receiving a gRPC request, the device will authenticate the user and perform various authorization checks to validate the user.

  The following table lists the authentication type and configuration requirements:

Table 1: Types of Authentication with Configuration

Type	Authentication Method	Authorization Method	Configuration Requirement	Requirement From Client
Metadata with TLS	username, password	username	**grpc**	username, password, and CA
Metadata without TLS	username, password	username	**grpc no-tls**	username, password
Metadata with Mutual TLS	username, password	username	**grpc tls-mutual**	username, password, client certificate, client key, and CA
Certificate based Authentication	client certificate's common name field	username from client certificate's common name field	**grpc tls-mutual** and **grpc certificate authentication**	client certificate, client key, and CA

Figure 1: Components of Service Layer API



Bring your controller

To bring your controller on IOS XR, first, enable the service layer on the router and then write your Service Layer Client API.

1. Enable Service Layer, on page 4
2. Write Your Service Layer Client API.

Enable Service Layer

Procedure

Step 1 Enable the Service Layer.

Example:

```
Router#configure
Router(config)#grpc
Router(config-grpc)#port 57777
Router(config-grpc)#service-layer
Router(config-grpc)#no-tls
Router(config-grpc)#commit
```

The default port value for gNMI service port is 9339. You can set gNMI service port value from 57344 to 57999. Whereas, the default port value for gRIBI service port is default 9340. You can set gRIBI service port value from 57344 to 57999.

Step 2 Verify if the Service Layer is operational:

Example:

```
Router#show running-config grpc
Mon Nov  4 04:19:14.044 UTC
grpc
 port 57777
 no-tls
 service-layer
 !
!
```

Step 3 Verify the gRPC state.

Example:

```
Router#show service-layer state
Mon Feb 24 04:18:40.055 UTC
------------service layer state----------
config on:                   YES
standby connected :          NO
idt done:                    NO
blocked on ndt:              NO
connected to RIB for IPv4:   YES
connected to RIB for IPv6:   YES
Initialization state:        estab sync
pending requests:            0
BFD Connection:              UP
MPLS Connection:             UP
Interface Connection:        UP
Objects accepted:            NO
interface registered:        NO
bfd registered for IPv4:     NO
bfd registered for IPv6:     NO
```

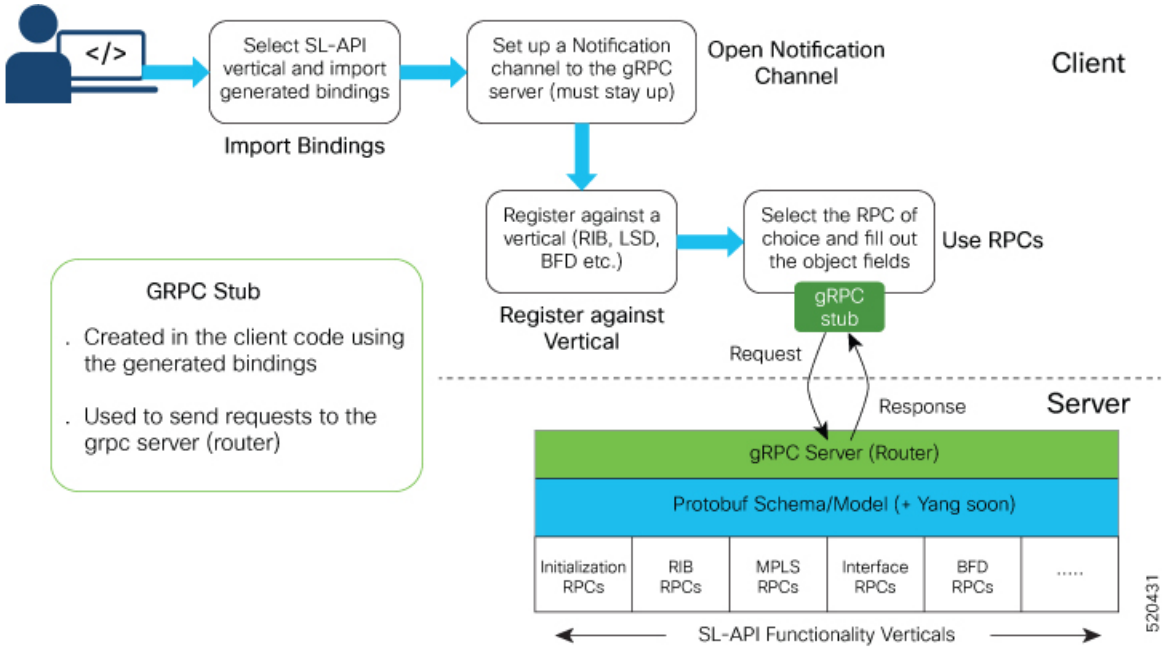
Write Your Service Layer Client API

You can write a Service Layer API based on your business needs. Follow these steps to write a Service Layer API client for a particular functionality vertical.

- **Import Bindings**: After generating the bindings, import the binding in your code.
- **Open Notification Channel:** Utilize the initialization functionality vertical to create a notification channel to register the client to the gRPC server running on the router.
- **Register against Vertical**: Register for a functionality vertical to utilize an RPC using the registration RPC before making calls. The system rejects any calls without prior registration.
- **Use RPCs**: Once registered against a vertical, select the RPC of your choice. Then complete the object fields in the gRPC stub.

To know more about creating a Service Layer API, see. Cisco IOS-XR Service Layer.

Figure 2: Service Layer API Workflow



Note Removing VRF or interface configurations referenced by SL-API objects is not supported and can impact traffic. Ensure Service Layer API clients reroute traffic and update routing before making such changes.

To know more about using gRPC protocol, see Use gRPC Protocol to Define Network Operations with Data Models Chapter in Programmability Configuration Guide.

Preprogram Backup LSPs Using Service Layer API

Table 2: Feature History Table

Feature Name	Release Information	Feature Description
Preprogram Backup LSPs Using Service Layer API	Release 24.2.1	Introduced in this release on: NCS 5500 fixed port routers; NCS 5500 modular routers (NCS 5500 line cards). This feature extends the Service Layer API, allowing the controller to preprogram backup Label Switched Paths (LSPs) in the hardware. When the *Path Priority* flag indicates a transition from the backup LSP to the primary LSP, the controller switches the traffic to the backup LSP.

With this feature, the primary LSP failure is detected through a controller-defined mechanism. Upon detecting a failure, the controller switches the primary LSP to backup in a down state and promotes the backup LSP to primary using the provided API parameters.

You can use the Service Layer API to preprogram LSPs as either primary or backup paths by using the *Path Priority* attribute. You can group LSPs with the *set-ID* attribute and determine their operational status as active or inactive using the *Path State* attribute. To ensure seamless traffic management, you can monitor the status of the LSPs using the controller. If traffic needs to be rerouted to the backup LSP, you can modify the priority of the preconfigured backup LSP to primary through the controller, thus allowing the backup path to take over the traffic load. The primary LSP then acts as the backup with its *Path State* set as down to retain the preprogram state. For more information about Service Layer API, see Github - Service Layer API and https://xrdocs.io/cisco-service-layer/.

Verify the Preprogramed Backup Paths

Use the **show service-layer mpls** command to verify the backup programming state for an LSP. For a given path, you can view path priority, and path set ID.

In the following command output, the Next-Hop Label Forwarding Entry 1 (**NHLFE 1**) is the primary LSP as the **path priority** is primary and the LSP state is up. **NHLFE 2** is the backup LSP as the **path priority** is backup and it belongs to the set ID 1. The status of the backup LSP is up.

```
Router#show service-layer mpls label 24000 exp default
Tue Jun 11 04:58:03.154 UTC
vrf name: mpls-default,  vrf state: eof,
vrf magic: valid,  purge timer: 600 seconds,  vrf flags: eof,

local label: 24000, update priority: high, magic: valid, flags: elsp, EXP: default,
    nhlfe: 1, magic: valid,
        ref count: 1, protected bitmap: 0x0, path id: 0, backup path id: 0,
        flags: path priority: primary, path setid: 0, path up
        path protection flags: 0, next hop: 10.10.10.2, load metric: 32,
```

```
    label action: 1,
    remote address:
    remote labels: 34000,
    interface name: Bundle-Ether1,

nhlfe: 2, magic: valid,
    ref count: 1, protected bitmap: 0x0, path id: 0, backup path id: 0,
    flags: path priority: backup, path setid: 1, path up
    path protection flags: 0, next hop: 10.10.10.3, load metric: 1,
    label action: 1,
    remote address:
    remote labels: 44000,
    interface name: Bundle-Ether2,

nhlfe: 3, magic: valid,
    ref count: 1, protected bitmap: 0x0, path id: 0, backup path id: 0,
    flags: path priority: backup, path setid: 1, path up
    path protection flags: 0, next hop: 10.10.10.8, load metric: 31,
    label action: 1,
    remote address:
    remote labels: 44000,
    interface name: Bundle-Ether3
```

The following table describes the possible values for the path attributes:

Attribute	Possible Values
Path Priority	Primary or Backup
set-ID	0–3
Path State	Up or Down