REST APIs
REST is an architecture style for designing web service applications.

REST API
Northbound API
works on top of the HTTP protocol

defines a set of functions developers can use to perform requests and receive responses through HTTP.

The token is used to authenticate you to the restful API service. Restful API does not support authorization. An API can be considered RESTful if it has the following features.

1. Client/Server: The client handles the front end, and the server handles the back end. Either can be replaced independently of the other.

2. Stateless: No client data is stored on the server between requests. The session state is stored on the client.

3. Cacheable: Clients can cache responses to improve performance.

REST APIs and HTTP: The creators of REST-based APIs often choose HTTP because HTTP's logic matches some of the concepts defined more generally for REST APIs. HTTP uses the same principles as REST it operates with a client/server model; it uses a stateless operational model; and it includes headers that clearly mark objects as cacheable or not cacheable. It also includes verbs - words that dictate the desired action for a pair HTTP Request and Reply – which matches how applications like to work.

Software CRUD Actions and HTTP Verbs

The software industry uses a memorable acronym - CRUD - for the four primary actions performed by an application. Those actions are

1. **Create**: Allows the client to create some new instances of variables and data structures at the server and initialize their values as kept at the server

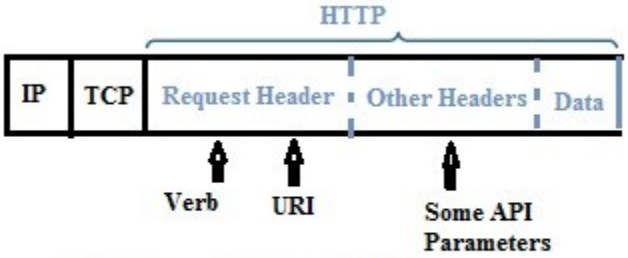
2. **Read**: Allows the client to retrieve (read) the current value of variables that exist at the server, storing a copy of the variables, structures, and values at the client.

3. **Update**: Allows the client to change (update) the value of variables that exist at the server

4. **Delete**: Allows the client to delete from the server different instances of data variables

For instance, if using the northbound REST API of a DNA controller, “Cisco Software-Defined Access (SDA),” you might want to create something new, like a new security policy. From a programming perspective, the security policy exists as a related set of configuration settings on the DNA controller, internally represented by variables. To do that, a REST client application would use a create action, using the DNA Center RESTful API, that created variables on the DNA Controller via the DNA Center REST API. The concept of creating new configuration at the controller is performed via the API using a create action per the CRUD generic acronym. Other examples of CRUD actions include a check of the status of that new configuration (a read action), an update to change some specific setting in the new configuration (an update action), or an action to remove the security policy definition completely (a delete action).

HTTP uses verbs that mirror CRUD actions. HTTP defines the concept of an HTTP request and reply, with the client sending a request and with the server answering back with a reply. Each request/reply lists an action verb in the HTTP request header, which defines the HTTP action. The HTTP messages also include a URI, which identifies the resource being manipulated for this request. As always, the HTTP message is carried in IP and TCP, with headers and data, as represented in below fig.



HTTP Verb and URI in an HTTP Request Header

NETCONF

The NETCONF protocol enables the device to expose an entire formal Application Programming Interface (API). Applications can use this straightforward API to send and receive full and partial configuration data sets.

NETCONF protocol, according to IETF RFC 6241, is a simple mechanism wherein:

A simple network device can be managed
Configuration data information can be retrieved
New configuration data can be manipulated and uploaded

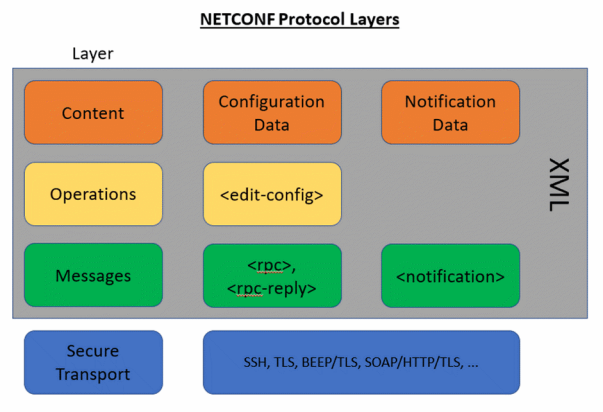
NETCONF
Netconf uses XML
functions as a device configuration mechanism
a vast improvement from SNMP wherein it only monitors, polls, and notifies whenever there is a fault in the network devices.

NETCONF uses **remote procedure calls (RPC)** model wherein the client inputs an RPC in XML and forwards it to a server utilizing a secure, connection-oriented session.

The server provides an RPC reply in XML.
Both the request and response contents are fully described in XML DTDs and/or XML schemas, which allows both sides to understand the syntax limitations that occur on the exchange. The protocol messages are also in XML and the mandatory transport protocol for NETCONF is through the Secure Shell Transport Layer Protocol (SSH).

NETCONF Protocol Layers
NETCONF is conceptually divided into four layers, as shown below.

- Secure Transport – enables the client and server to communicate with each other. NETCONF can be overlaid on top of any transport protocol that provides a set of basic requirements.
- Messages – gives a simple and independent transport framing mechanism for encoding RPCs and notifications.
- Operations – operations layer defines a set of base protocol operations called on RPC methods together with XML-encoded parameters.
- Content – has a set of managed objects, such as configuration data, status data, and statistics information.



YANG

YANG is a data modeling language for NETCONF as per RFC 6020. It is used to model configuration and state data manipulated by NETCONF, NETCONF RPCs, and notifications. YANG is utilized to model the operations and content layers of NETCONF.

YANG Data Models

The model can be presented in multiple formats based on the need during that specific instance. Below are some options:

- HTML/JavaScript
- YANG Language
- Clear text
- XML
- JSON

RESTCONF
RFC 8040 based on HTTP
used for configuring data defined in YANG version 1 or 1.1 using the datastore concepts defined in the Network Configuration Protocol (NETCONF).

RESTCONF uses HTTP methods to provide Create, Read, Update, Delete (CRUD) operations on a conceptual datastore comprising YANG-defined data, which is compatible with a server that administers NETCONF datastores.