IoT Communication APIs

Internet of Things , Lecture-6

Rahul Shandilya

REST-based Communication APIs

Representational State Transfer (REST) is a set of architectural principles by which you can design web services and web APIs that focus on a system's resources and how resource states are addressed and transferred. REST APIs follow the request-response communication model. The REST architectural constraints apply to the components, connectors, and data elements, within a distributed hypermedia systems.

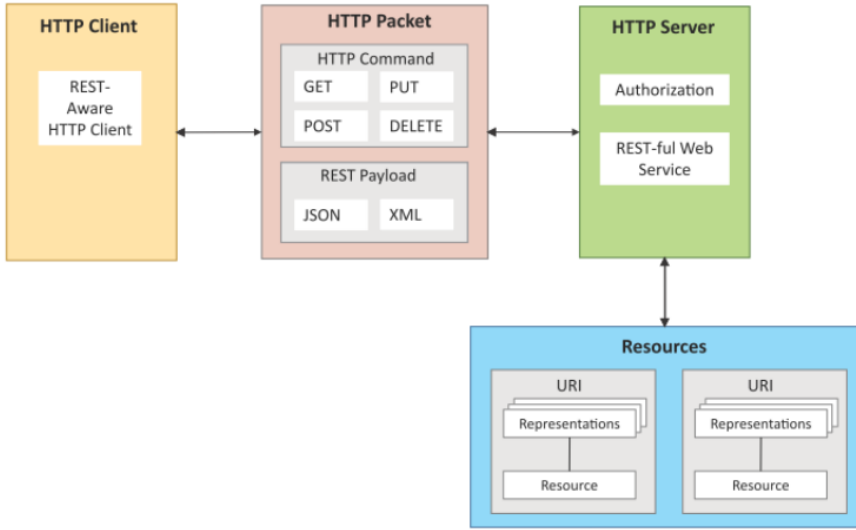


Figure: REST based API

REST Constraints

The REST architectural constrains are as follows:

- **Client-Server:** The principle behind the client-server constraint is the separation of concerns. For example, clients should not be concerned with the storage of data which is a concern of the server. Similarly, the server should not be concerned about the user interface, which is a concern of the client. Separation allows client and server to be independently developed and updated.

REST Constraints

The REST architectural constrains are as follows:

- **Client-Server:** The principle behind the client-server constraint is the separation of concerns. For example, clients should not be concerned with the storage of data which is a concern of the server. Similarly, the server should not be concerned about the user interface, which is a concern of the client. Separation allows client and server to be independently developed and updated.
- **Stateless:** Each request from client to server must contain all the information necessary to understand the request, and cannot take advantage of any stored context on the server. The session state is kept entirely on the client.

REST Constraints

The REST architectural constrains are as follows:

- **Client-Server:** The principle behind the client-server constraint is the separation of concerns. For example, clients should not be concerned with the storage of data which is a concern of the server. Similarly, the server should not be concerned about the user interface, which is a concern of the client. Separation allows client and server to be independently developed and updated.
- **Stateless:** Each request from client to server must contain all the information necessary to understand the request, and cannot take advantage of any stored context on the server. The session state is kept entirely on the client.
- **Cache-able:** Cache constraint requires that the data within a response to a request be implicitly or explicitly labeled as cache-able or non-cache-able. If a response is cache-able, then a client cache is given the right to reuse that response data for later, equivalent requests. Caching can partially or completely eliminate some interactions and improve efficiency and scalability.

- **Layered System:** Layered system constraint, constrains the behavior of components such that each component cannot see beyond the immediate layer with which they are interacting. For example, a client cannot tell whether it is connected directly to the end server, or to an intermediary along the way. System scalability can be improved by allowing intermediaries to respond to requests instead of the end server, without the client having to do anything different.

- **Layered System:** Layered system constraint, constrains the behavior of components such that each component cannot see beyond the immediate layer with which they are interacting. For example, a client cannot tell whether it is connected directly to the end server, or to an intermediary along the way. System scalability can be improved by allowing intermediaries to respond to requests instead of the end server, without the client having to do anything different.
- **Uniform Interface:** Uniform Interface constraint requires that the method of communication between a client and a server must be uniform. Resources are identified in the requests (by URIs in web based systems) and are themselves separate from the representations of the resources that are returned to the client. When a client holds a representation of a resource it has all the information required to update or delete the resource (provided the client has required permissions). Each message includes enough information to describe how to process the message.

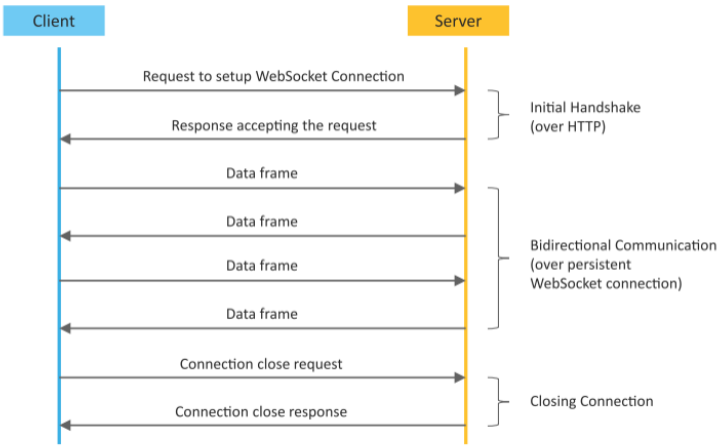
- **Layered System:** Layered system constraint, constrains the behavior of components such that each component cannot see beyond the immediate layer with which they are interacting. For example, a client cannot tell whether it is connected directly to the end server, or to an intermediary along the way. System scalability can be improved by allowing intermediaries to respond to requests instead of the end server, without the client having to do anything different.
- **Uniform Interface:** Uniform Interface constraint requires that the method of communication between a client and a server must be uniform. Resources are identified in the requests (by URIs in web based systems) and are themselves separate from the representations of the resources that are returned to the client. When a client holds a representation of a resource it has all the information required to update or delete the resource (provided the client has required permissions). Each message includes enough information to describe how to process the message.
- **Code on demand:** Servers can provide executable code or scripts for clients to execute in their context. This constraint is the only one that is optional.

WebSocket-based Communication APIs

The WebSocket API is an advanced technology that makes it possible to open a two-way interactive communication session between the user's browser and a server. With this API, you can send messages to a server and receive event-driven responses without having to poll the server for a reply.

WebSocket-based Communication APIs

The WebSocket API is an advanced technology that makes it possible to open a two-way interactive communication session between the user's browser and a server. With this API, you can send messages to a server and receive event-driven responses without having to poll the server for a reply.



Feature of Websocket API

- WebSocket APIs allow bi-directional, full duplex communication between clients and servers. WebSocket APIs follow the exclusive pair communication model.

Feature of Websocket API

- WebSocket APIs allow bi-directional, full duplex communication between clients and servers. WebSocket APIs follow the exclusive pair communication model.
- Unlike request-response APIs such as REST, the WebSocket APIs allow full duplex communication and do not require a new connection to be setup for each message to be sent.

Feature of Websocket API

- WebSocket APIs allow bi-directional, full duplex communication between clients and servers. WebSocket APIs follow the exclusive pair communication model.
- Unlike request-response APIs such as REST, the WebSocket APIs allow full duplex communication and do not require a new connection to be setup for each message to be sent.
- WebSocket communication begins with a connection setup request sent by the client to the server. This request (called a WebSocket handshake) is sent over HTTP and the server interprets it as an upgrade request.

Feature of Websocket API

- WebSocket APIs allow bi-directional, full duplex communication between clients and servers. WebSocket APIs follow the exclusive pair communication model.
- Unlike request-response APIs such as REST, the WebSocket APIs allow full duplex communication and do not require a new connection to be setup for each message to be sent.
- WebSocket communication begins with a connection setup request sent by the client to the server. This request (called a WebSocket handshake) is sent over HTTP and the server interprets it as an upgrade request.
- If the server supports WebSocket protocol, the server responds to the WebSocket handshake response. After the connection is setup, the client and server can send data/messages to each other in full-duplex mode.

Feature of Websocket API

- WebSocket APIs allow bi-directional, full duplex communication between clients and servers. WebSocket APIs follow the exclusive pair communication model.
- Unlike request-response APIs such as REST, the WebSocket APIs allow full duplex communication and do not require a new connection to be setup for each message to be sent.
- WebSocket communication begins with a connection setup request sent by the client to the server. This request (called a WebSocket handshake) is sent over HTTP and the server interprets it as an upgrade request.
- If the server supports WebSocket protocol, the server responds to the WebSocket handshake response. After the connection is setup, the client and server can send data/messages to each other in full-duplex mode.
- WebSocket APIs reduce the network traffic and latency as there is no overhead for connection setup and termination requests for each message. WebSocket is suitable for IoT applications that have low latency or high throughput requirements.