**Project Overview**

* Project goal: To deploy a micro-service implementing a Java/Groovy gNMI/gRPC client to install, manipulate and delete configuration of network devices and to view the operational data. Streaming subscription has to be considered as part of the required goals.

**Technologies Required**

* Java/Groovy: A programming language that will be used to write the micro-service.
* SpringBoot: A framework that will be used to create the micro-service.
* gNMI/gRPC: A network management protocol and framework that will be used to communicate with network devices.
* Protocol Buffers: A serialization format that will be used to encode the messages exchanged between the micro-service and the network devices.
* OpenAPI: A specification that will be used to define the input and output APIs for the micro-service.

**Potential Steps to Achieve Project Goals**

1. Set up a development environment that includes Java, Groovy, and SpringBoot.
2. Use gNMI/gRPC to implement the required RPCs (Set, Subscribe, Get, Capabilities) in the micro-service.
3. Define the input and output APIs for the micro-service using OpenAPI.
4. Implement the gNMI/gRPC client in the micro-service using Java/Groovy and SpringBoot.
5. Use Protocol Buffers to encode the messages exchanged between the micro-service and the network devices.
6. Build a test gNMI server to validate the micro-service.
7. Test the gNMI primitives using the test environment to ensure that they are working as expected.
8. Deploy the micro-service to a production environment and monitor its performance to ensure that it meets the project goals.

**gNMI**

* gNMI (gRPC Network Management Interface) is a network management protocol that uses gRPC as the underlying transport mechanism.
* gNMI is designed to provide a standard, open, and programmable interface for configuring and monitoring network devices.
* gNMI uses Protocol Buffers to encode the messages exchanged between the client and the network devices.
* gNMI provides various capabilities, including configuration management, state monitoring, and streaming telemetry.

**gRPC**

* gRPC is a high-performance, open-source RPC (Remote Procedure Call) framework that enables communication between client and server applications.
* gRPC uses HTTP/2 as the underlying transport protocol, which provides several benefits, including reduced latency, increased throughput, and support for bidirectional streaming.
* gRPC uses Protocol Buffers as the default serialization format for messages exchanged between client and server applications.
* gRPC supports multiple programming languages, including Java, C++, Python, Go, and more, making it a popular choice for building micro-services and distributed systems.

**Identify the gNMI/gRPC Java/Groovy implementation**

* There are several gNMI/gRPC Java implementations available, such as OpenConfig gNMI Java, gNMI4j, and gRPC-Java.
* The choice of implementation depends on factors such as ease of use, compatibility with existing systems, and community support.

Potential steps:

1. Research available gNMI/gRPC Java implementations and evaluate them based on the project requirements.
2. Select the most suitable implementation based on the evaluation results.
3. Install and configure the chosen implementation in the development environment.
4. Test the implementation to ensure that it works as expected.

**Define input and output APIs for the microservice based on Open-API**

* OpenAPI is a specification for building APIs that allows developers to define, create, and document RESTful APIs.
* OpenAPI provides a standard way of describing input and output parameters, API operations, and error messages.

**Potential steps:**

1. Identify the endpoints and operations that the microservice needs to implement, based on the project requirements.
2. Define the input and output parameters for each operation, including data types, formats, and validation rules.
3. Use the OpenAPI specification to document the input and output parameters, operations, and error messages.
4. Generate client libraries and server stubs based on the OpenAPI specification, using tools such as Swagger Codegen or OpenAPI Generator.
5. Test the APIs using tools such as Postman or Swagger UI.