gNMI
gRPC Network Management Interface

Samuel Ribeiro

Fall 2017 - Faucet Conference

Google

Why gNMI? - And what about Openflow?

CLI is not Programmable.

- lack of transaction management;
- no structured error handling;
- ever changing structure and syntax of commands;

gNMI vs Openflow

- Openflow -> Forwarding Plane
  - Packet A goes to X
- gNMI -> Platform
  - Configuration
  - Hardware/Software
  - Environmental/Power

gNMI decomposed

- gRPC - transport
  - high performance RPC framework that can run in any environment

- gNMI - action
  - Get/Set/Subscribe/Capabilities (Service definition with a proto file)

- Tree-structured data - properties
  - OpenConfig - YANG data models

gRPC - what is it?

Client -----(HTTP/2)----> Server

(insert TCP port number here)

The HTTP/2 session can be:

- Authenticated
- Encrypted
- Compressed
- Multiplexing
- Bidirectional

- Client calls procedures in Server;
- Uses **Protocol Buffers** to serialize data;
- **Protocol Buffers** - like XML but:
  - 3x-10x smaller
  - faster
  - simpler

www.grpc.io

gRPC - how is it defined?

The set of actions that are allowed between Client and Server is defined by a Service Definition, which is also a Protocol Buffer:

```
service Greeter {
  rpc SayHello (HelloRequest) returns (HelloReply);
  rpc ForeverHello (stream HelloRequest) returns (stream HelloReply);
}

message HelloRequest {
  string name = 1;
}

message HelloReply {
  string message = 1;
}
```

C++
C#
Go
Java
Node.js
Objective-C
PHP
Python
Ruby

gNMI - defined

```
service gNMI {
  rpc Capabilities(CapabilityRequest) returns (CapabilityResponse);
  rpc Get(GetRequest) returns (GetResponse);
  rpc Set(SetRequest) returns (SetResponse);
  rpc Subscribe(stream SubscribeRequest) returns (stream SubscribeResponse);
}
```

- Server is named Target.
- Target always authenticates Client.
- Client always authenticates Target.
- Session is always encrypted.

Client ---------> Target

User Switch

Collector Server

OpenConfig
YANG data models

- YANG
  - data modeling language

- OpenConfig - (**www.openconfig.net**)
  - authoring guidelines for modeling with YANG
  - real use case driven reasoning
  - vendor neutral

```
<...>
grouping openflow-agent-config {
    description
      "Openflow agent config";

    <...>

    leaf backoff-interval {
      type uint32;
      units seconds;
      description
        "Openflow agent connection backoff interval.";
    }

    leaf inactivity-probe {
      type uint32;
      units seconds;
      description
        "Openflow agent inactivity probe period.";
    }

    <...>
  }
<...>
```

OpenConfig data structure

```
module: openconfig-system
    <...>
    +--rw system
       |  <...>
       +--rw openflow:openflow
          |  <...>
          +--rw openflow:agent
             +--rw openflow:config
             |  +--rw openflow:backoff-interval?   uint32
             |  +--rw openflow:max-backoff?        uint32
             |  +--rw openflow:inactivity-probe?   uint32
             |  <...>
             +--ro openflow:state
                +--ro openflow:backoff-interval?   uint32
                +--ro openflow:max-backoff?        uint32
                +--ro openflow:inactivity-probe?   Uint32
                <...>
```

```
# gnmi_get ... \
    -xpath "/system/openflow/agent/state/backoff-interval" \
    -xpath "/system/openflow/agent/state/max-backoff" \
    -xpath "/system/openflow/controllers/*"
```

gNMI SET - (delete, replace & update)

```
message SetRequest {
  <...>
  repeated Path delete = 2;
  repeated Update replace = 3;
  repeated Update update = 4;
}
```

- SET is Transactional
- State must not change until all of it is accepted;

```
# gnmi_set ... \
    -update "/:@set.json"

# cat set.json
{
  "system": {
    "openflow": {
      "agent": {
        "config": {
          "inactivity-probe": 15,
          "max-backoff": 12
        }
      }
    }
  }
}
```

Config (rw) vs State (ro)

- gNMI operations are Transactional.
  - So why Config vs State?
- **OpenConfig**
  - had to consider asynchronous systems where configuration changes to the system may not be reflected immediately;
- In gNMI:
  - STATE == CONFIG

```
module: openconfig-system
    |  <...>
    +--rw system
       |  <...>
       +--rw openflow:openflow
          |  <...>
          +--rw openflow:agent
             +--rw openflow:config
             |  +--rw openflow:backoff-interval?
             |  +--rw openflow:max-backoff?
             |  +--rw openflow:inactivity-probe?
             |  <...>
             +--ro openflow:state
                +--ro openflow:backoff-interval?
                +--ro openflow:max-backoff?
                +--ro openflow:inactivity-probe?
                <...>
```

Encoding

gNMI defines:

```
enum Encoding {
  JSON = 0;  <-----(rfc7159)- OKish
  BYTES = 1;
  PROTO = 2;
  ASCII = 3;
  JSON_IETF = 4; <-(rfc7951)- Prefered (made for YANG)
}
```

Certificates

In gNMI the sessions are authenticated and encrypted.

- Must use Certificates.
- Client authenticates Target (including validating the hostname).
- Target authenticates Client.

Client <----------> Target

Client Private Key	Target Private Key
Client certificate (signed by CA)	Target certificate (signed by CA)
CA certificate	CA certificate

Credentials

- username/password can be added to the session METADATA
  - HTTP/2
  - Session is encrypted

- Role Based Access Control
  - do we really need it to be done by the platform?

Subscribe - (streaming telemetry)

```
service gNMI {
  <...>
  rpc Subscribe(stream SubscribeRequest) returns (stream SubscribeResponse);
}
```

Use the same OpenConfig models to subscribe to paths.

- Subscription modes:
  - STREAM - sends value on change
  - ONCE - closes channel after sending one value
  - POLL - actively polls for the value

Capabilities

- Fetches Target Capabilities

```
service gNMI {
  rpc Capabilities(CapabilityRequest) returns (CapabilityResponse);
  <...>
}

message CapabilityResponse {
  repeated ModelData supported_models = 1;    // Supported schema models.
  repeated Encoding supported_encodings = 2;  // Supported encodings.
  string gNMI_version = 3;                    // Supported gNMI version.
}
```

Work in Progress

- OpenConfig
  - Openflow model
    - controller to be a name instead of just an IP
    - assign certificates to an Openflow channel
  - MACsec model
  - PoE model
- ...

What configures gNMI?

What needs to be configured?

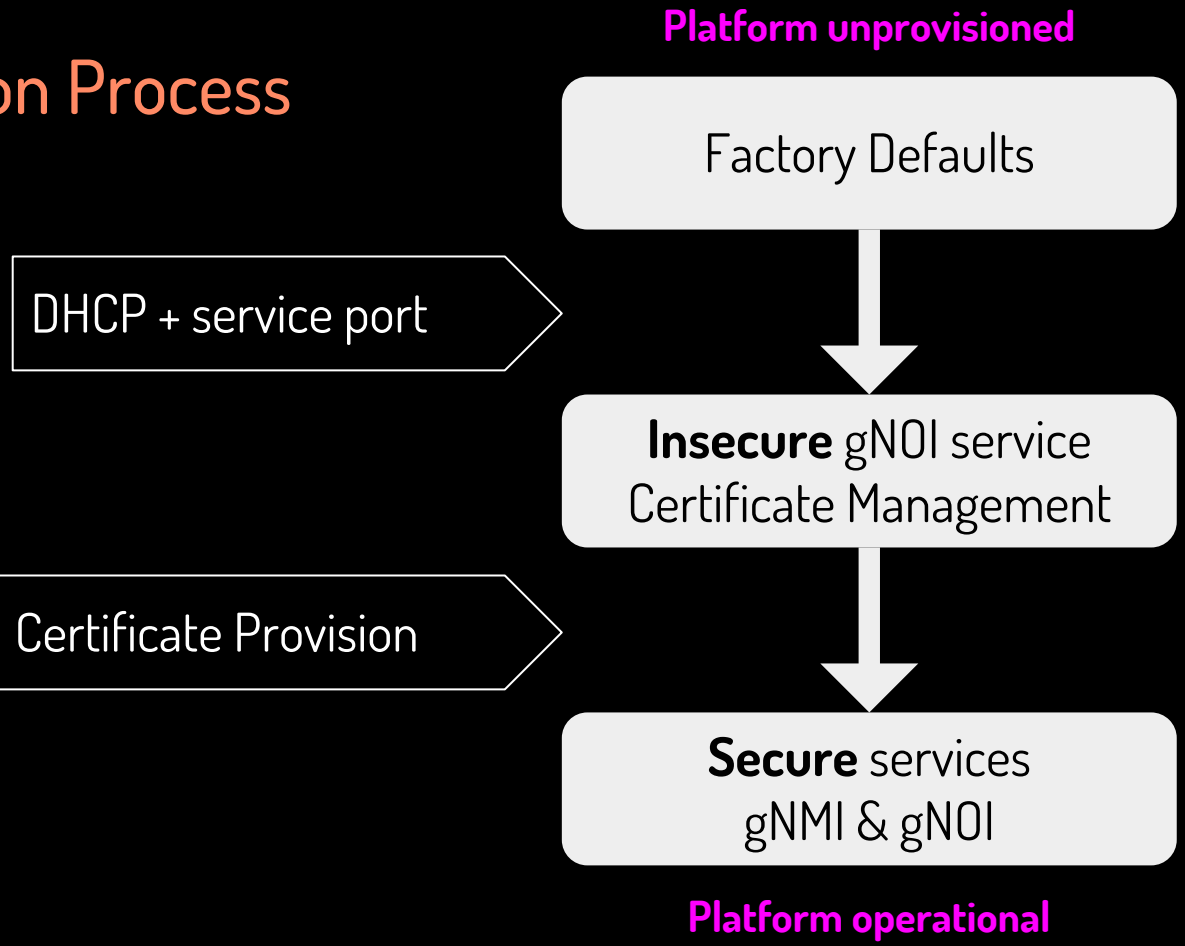
1. Admin interface IP Address
   - ➢ DHCP ✓
2. Enable service & TCP Port
   - ➢ DHCP Option ✓
3. Certificates
   - ➢ gNOI !

gNOI - gRPC Network Operations Interface

```
service CertificateManagement {
  rpc Rotate(stream RotateCertificateRequest) returns (stream RotateCertificateResponse);
  rpc Install(stream InstallCertificateRequest) returns (stream InstallCertificateResponse);
  rpc GetCertificates(GetCertificatesRequest) returns (GetCertificatesResponse);
  rpc RevokeCertificates(RevokeCertificatesRequest) returns (RevokeCertificatesResponse);
  rpc CanGenerateCSR(CanGenerateCSRRequest) returns (CanGenerateCSRResponse);
}
service File {
  <...>
}
service System {
  <...>
  rpc SetPackage(SetPackageRequest) returns (SetPackageResponse) {}
  rpc Reboot(RebootRequest) returns (RebootResponse) {}
}
```

Platform Provision Process

Provision process assumes a secure environment.



What's Next?

1. Using gNMI to configure an Access Point;

2. gNMI reference implementation;
   - github.com/google/gnxi

3. Docker instance with running example;
   - github.com/faucetsdn/Dockerfile.gnmi

Thank you!