Schema Registry integration with YangKit

INSA de Lyon

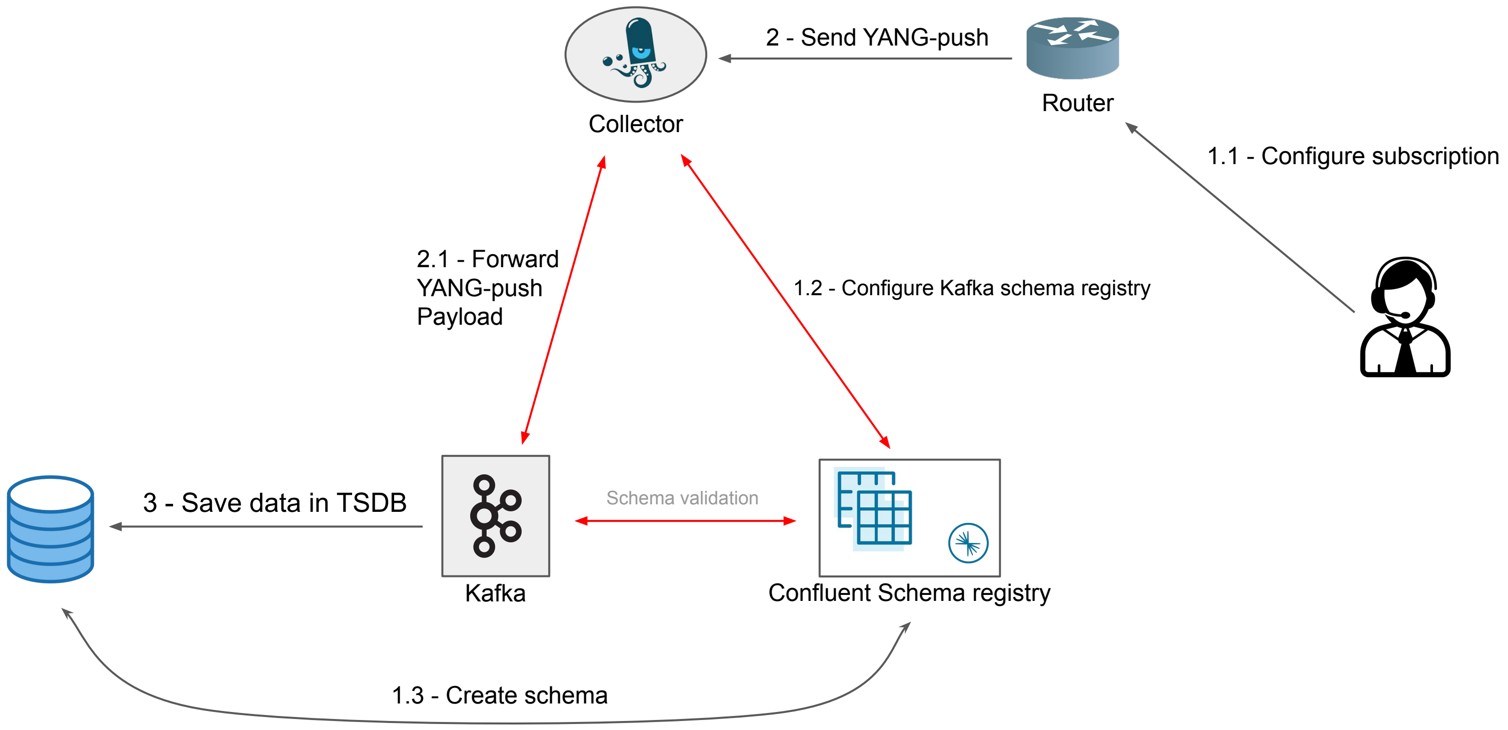
Alex Huang Feng (alex.huang-feng@insa-lyon.fr)
Vivek Boudia (vivekananda.boudia@insa-lyon.fr)
Pierre Francois (pierre.francois@insa-lyon.fr)

Last Updated: 27 october 2023

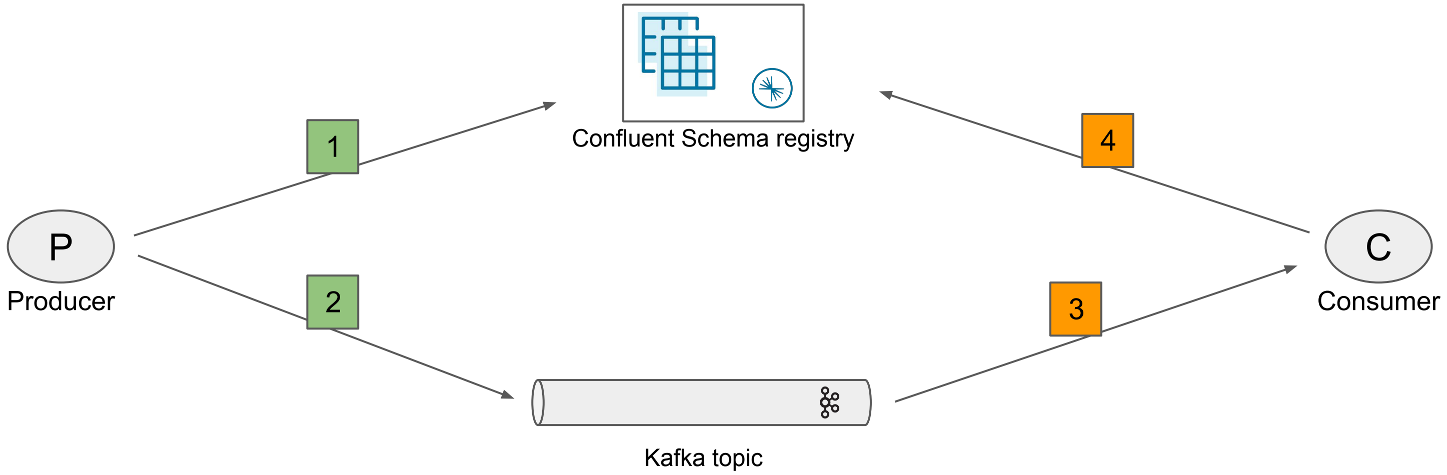
Index

Schema Registry

Full Implementation-level architecture

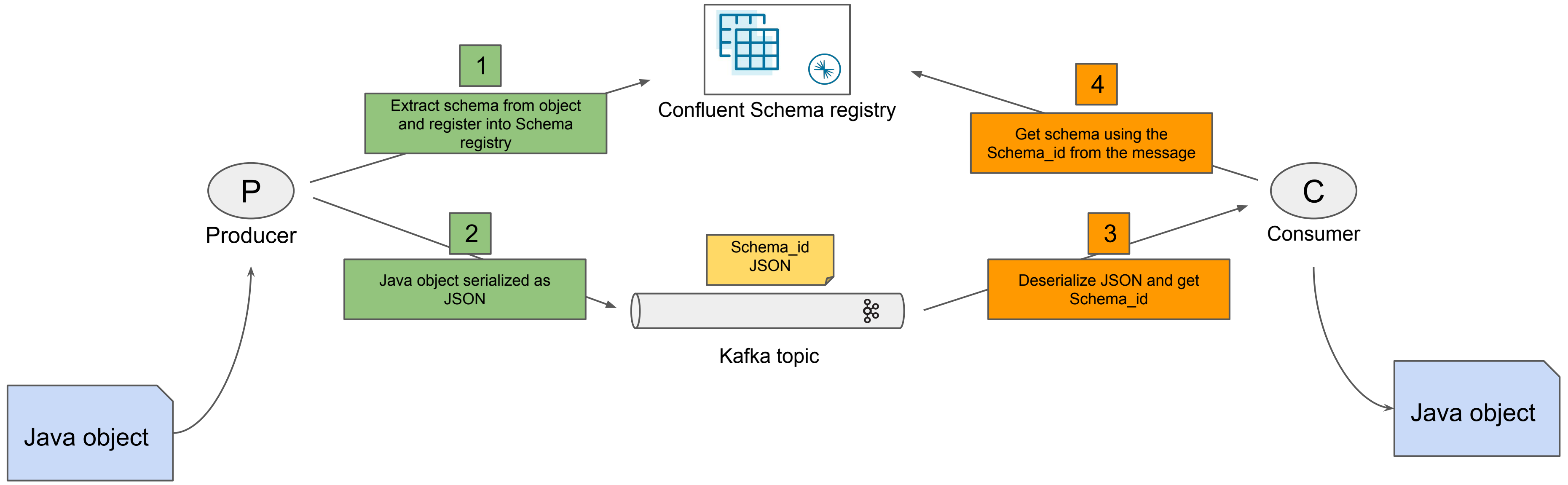

2 - Send YANG-push
Collector
Router
1.1 - Configure subscription
2.1 - Forward YANG-push Payload
1.2 - Configure Kafka schema registry
3 - Save data in TSDB
Schema validation
Kafka
Confluent Schema registry
1.3 - Create schema

How does Schema registry works?

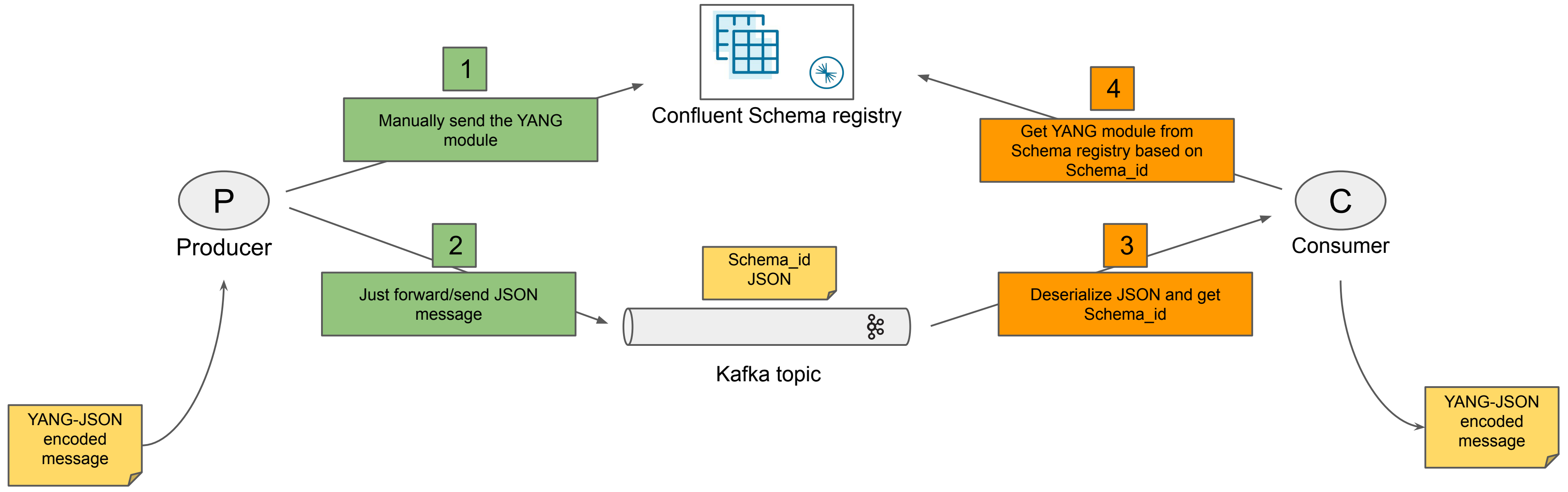


1. register schema
2. Send   <MAGICBYTE, Schema ID, Content>
3. Receive <MAGICBYTE, Schema ID, Content>
4. Get schema

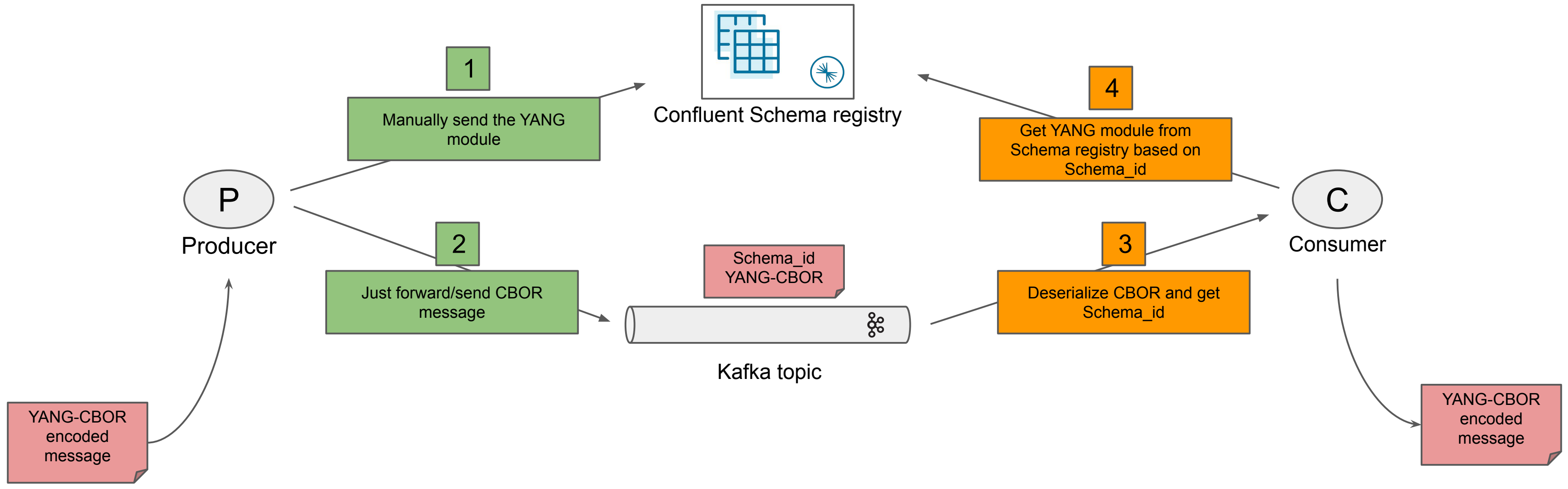
Main usecase for Schema registry: Plain Old Java Objects


1
Extract schema from object and register into Schema registry
Confluent Schema registry
4
Get schema using the Schema_id from the message
P
Producer
C
Consumer
2
Java object serialized as JSON
Schema_id
JSON
Kafka topic
3
Deserialize JSON and get Schema_id
Java object
Java object

Main usecase for YANG: Using YANG-JSON (without POJO)


1
Manually send the YANG module
Confluent Schema registry
4
Get YANG module from Schema registry based on Schema_id
P
Producer
C
Consumer
2
Just forward/send JSON message
Schema_id JSON
Kafka topic
3
Deserialize JSON and get Schema_id
YANG-JSON encoded message
YANG-JSON encoded message

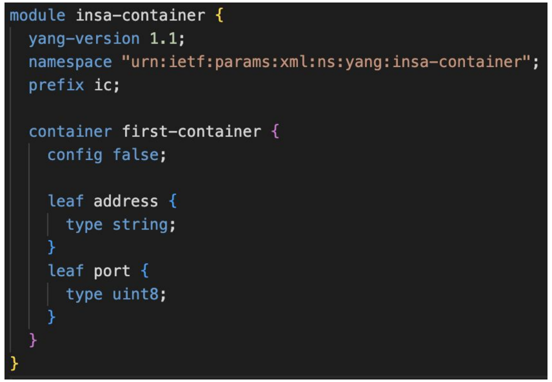
Main usecase for YANG: Using YANG-CBOR (without POJO)


1
Manually send the YANG module
Confluent Schema registry
4
Get YANG module from Schema registry based on Schema_id
P
Producer
2
Just forward/send CBOR message
Schema_id
YANG-CBOR
Kafka topic
3
Deserialize CBOR and get Schema_id
C
Consumer
YANG-CBOR encoded message
YANG-CBOR encoded message

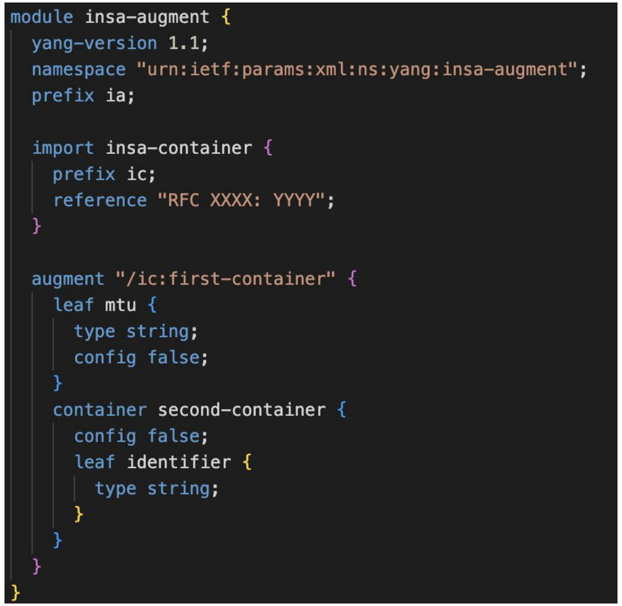
Same as YANG-JSON encoding but in YANG-CBOR

Nature of YANG modules / schemas

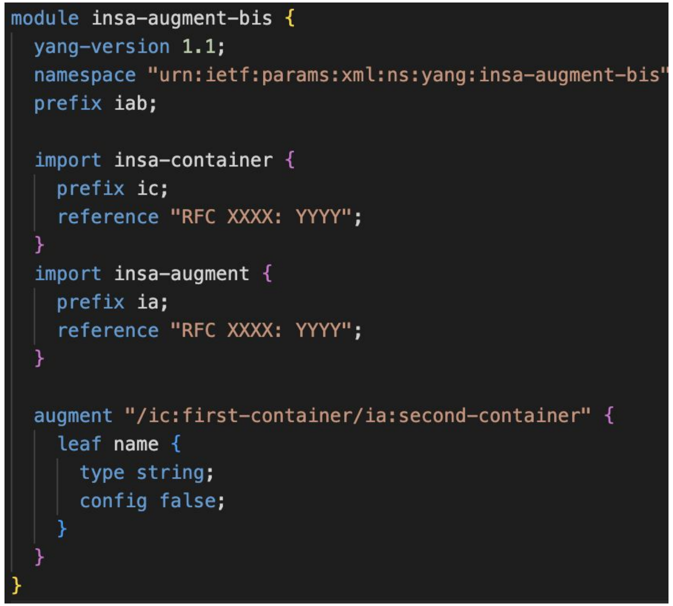
Example:



```
module insa-container {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:insa-container";
  prefix ic;

  container first-container {
    config false;

    leaf address {
      type string;
    }
    leaf port {
      type uint8;
    }
  }
}
```

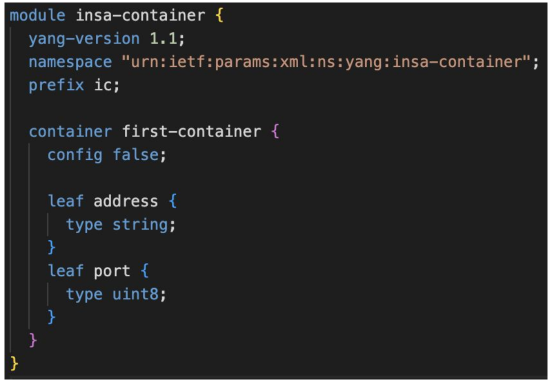


```
module insa-augment {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:insa-augment";
  prefix ia;

  import insa-container {
    prefix ic;
    reference "RFC XXXX: YYYY";
  }

  augment "/ic:first-container" {
    leaf mtu {
      type string;
      config false;
    }
    container second-container {
      config false;
      leaf identifier {
        type string;
      }
    }
  }
}
```

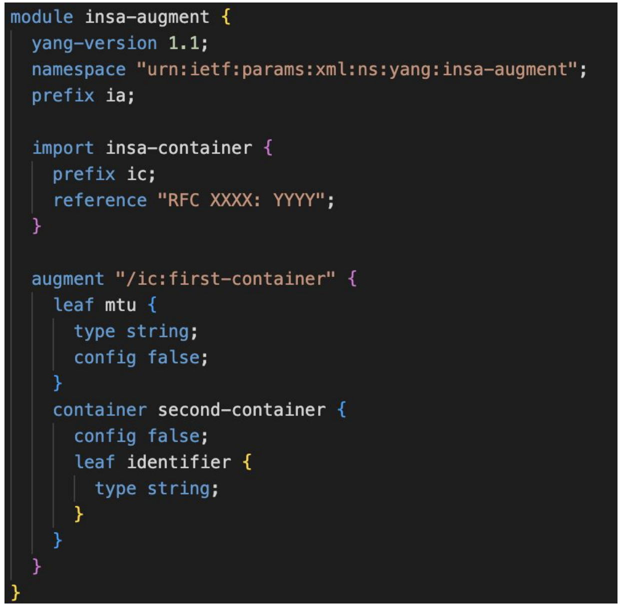


```
module insa-augment-bis {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:insa-augment-bis"
  prefix iab;

  import insa-container {
    prefix ic;
    reference "RFC XXXX: YYYY";
  }
  import insa-augment {
    prefix ia;
    reference "RFC XXXX: YYYY";
  }

  augment "/ic:first-container/ia:second-container" {
    leaf name {
      type string;
      config false;
    }
  }
}
```

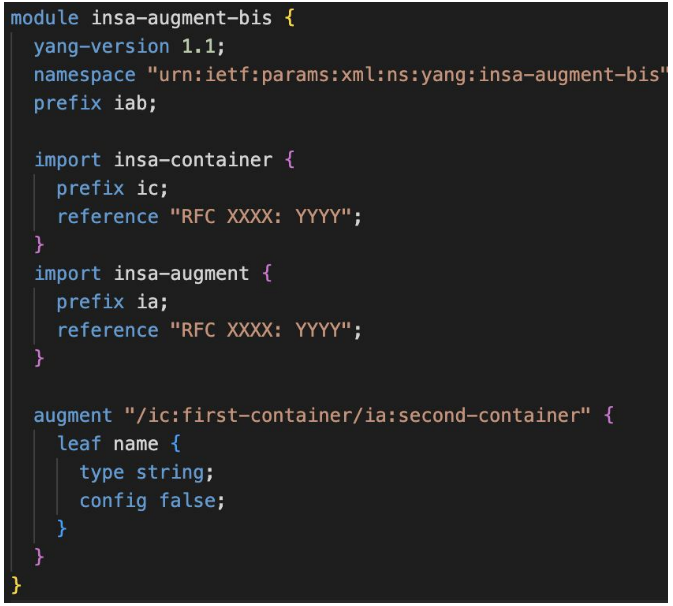
Nature of YANG modules / schemas → Dependencies



```
module insa-container {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:insa-container";
  prefix ic;

  container first-container {
    config false;

    leaf address {
      type string;
    }
    leaf port {
      type uint8;
    }
  }
}
```



```
module insa-augment {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:insa-augment";
  prefix ia;

  import insa-container {
    prefix ic;
    reference "RFC XXXX: YYYY";
  }

  augment "/ic:first-container" {
    leaf mtu {
      type string;
      config false;
    }
    container second-container {
      config false;
      leaf identifier {
        type string;
      }
    }
  }
}
```



```
module insa-augment-bis {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:insa-augment-bis"
  prefix iab;

  import insa-container {
    prefix ic;
    reference "RFC XXXX: YYYY";
  }
  import insa-augment {
    prefix ia;
    reference "RFC XXXX: YYYY";
  }

  augment "/ic:first-container/ia:second-container" {
    leaf name {
      type string;
      config false;
    }
  }
}
```

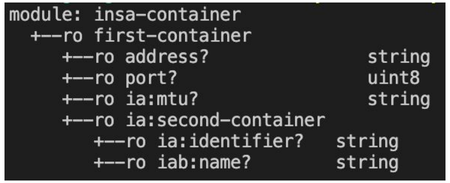
insa-container.yang ← insa-augment.yang ← insa-augment-bis.yang

Nature of YANG modules / schemas: registration in Schema registry

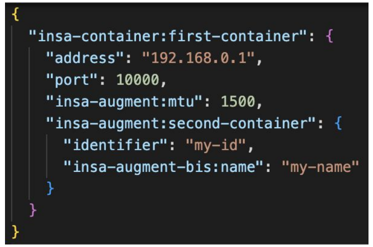
insa-container.yang ← insa-augment.yang ← insa-augment-bis.yang

1 2 3

1. Register `insa-container.yang` (subject **`insa-container`**)
2. Register `insa-augment.yang` (subject **`insa-augment`**)
3. Register `insa-augment-bis.yang` (subject **`insa-augment-bis`**)
4. Use **last subject** (*insa-augment-bis*) to validate message

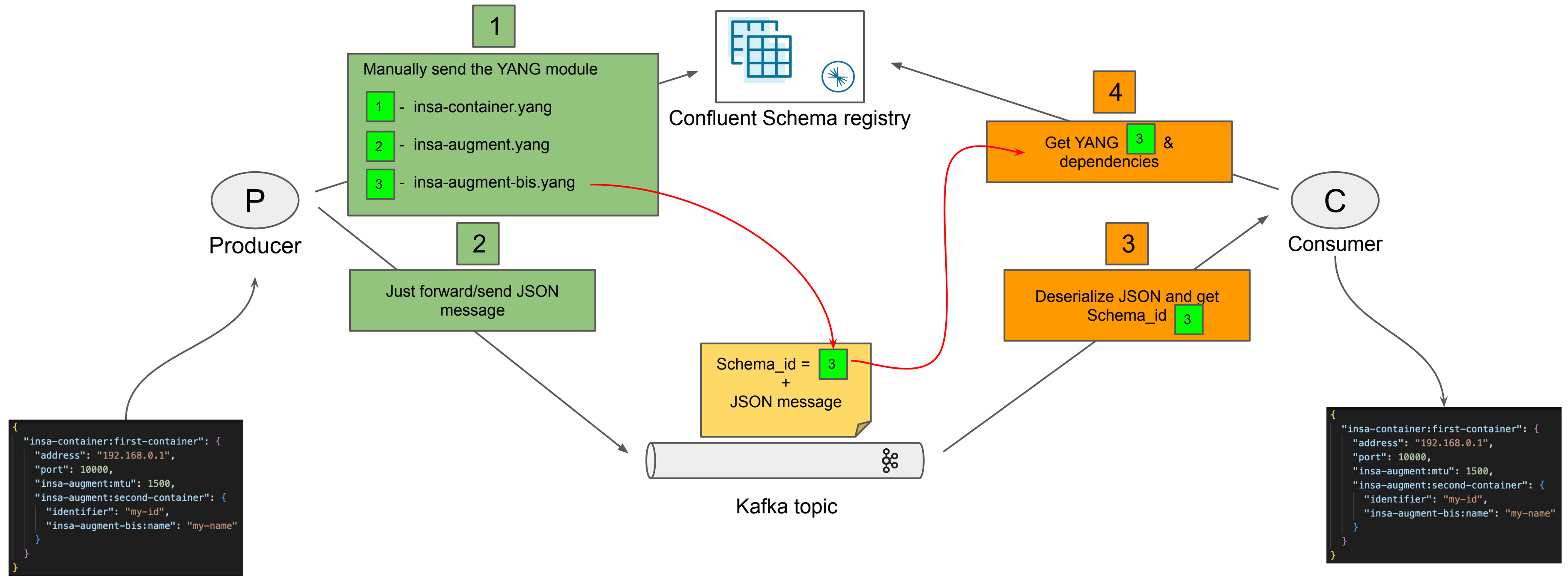


```
module: insa-container
  +--ro first-container
     +--ro address?                 string
     +--ro port?                    uint8
     +--ro ia:mtu?                  string
     +--ro ia:second-container
        +--ro ia:identifier?   string
        +--ro iab:name?        string
```



```
{
  "insa-container:first-container": {
    "address": "192.168.0.1",
    "port": 10000,
    "insa-augment:mtu": 1500,
    "insa-augment:second-container": {
      "identifier": "my-id",
      "insa-augment-bis:name": "my-name"
    }
  }
}
```

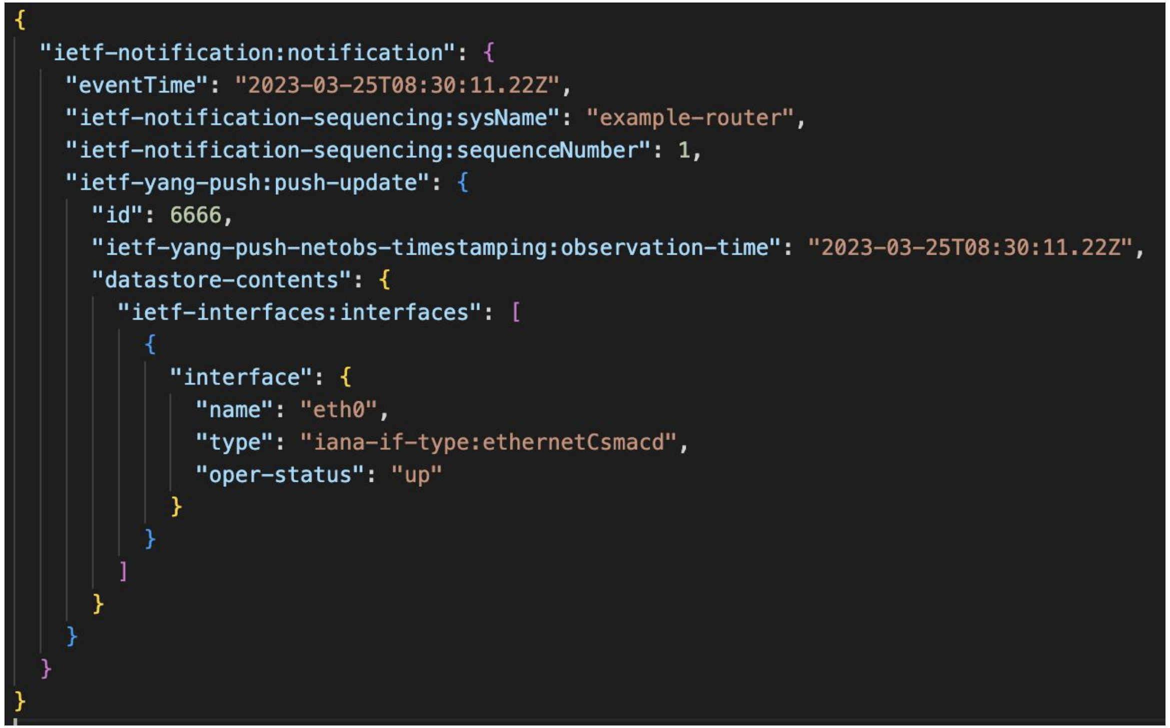
Main usecase for YANG: Using YANG-JSON (without POJO)


1
Manually send the YANG module
1 - insa-container.yang
2 - insa-augment.yang
3 - insa-augment-bis.yang
Confluent Schema registry
4
Get YANG 3 & dependencies
P
Producer
2
Just forward/send JSON message
C
Consumer
3
Deserialize JSON and get Schema_id 3
Schema_id = 3 + JSON message
Kafka topic

```
{
  "insa-container:first-container": {
    "address": "192.168.0.1",
    "port": 10000,
    "insa-augment:mtu": 1500,
    "insa-augment:second-container": {
      "identifier": "my-id",
      "insa-augment-bis:name": "my-name"
    }
  }
}
```

```
{
  "insa-container:first-container": {
    "address": "192.168.0.1",
    "port": 10000,
    "insa-augment:mtu": 1500,
    "insa-augment:second-container": {
      "identifier": "my-id",
      "insa-augment-bis:name": "my-name"
    }
  }
}
```

YANG push in Schema registry: example 1

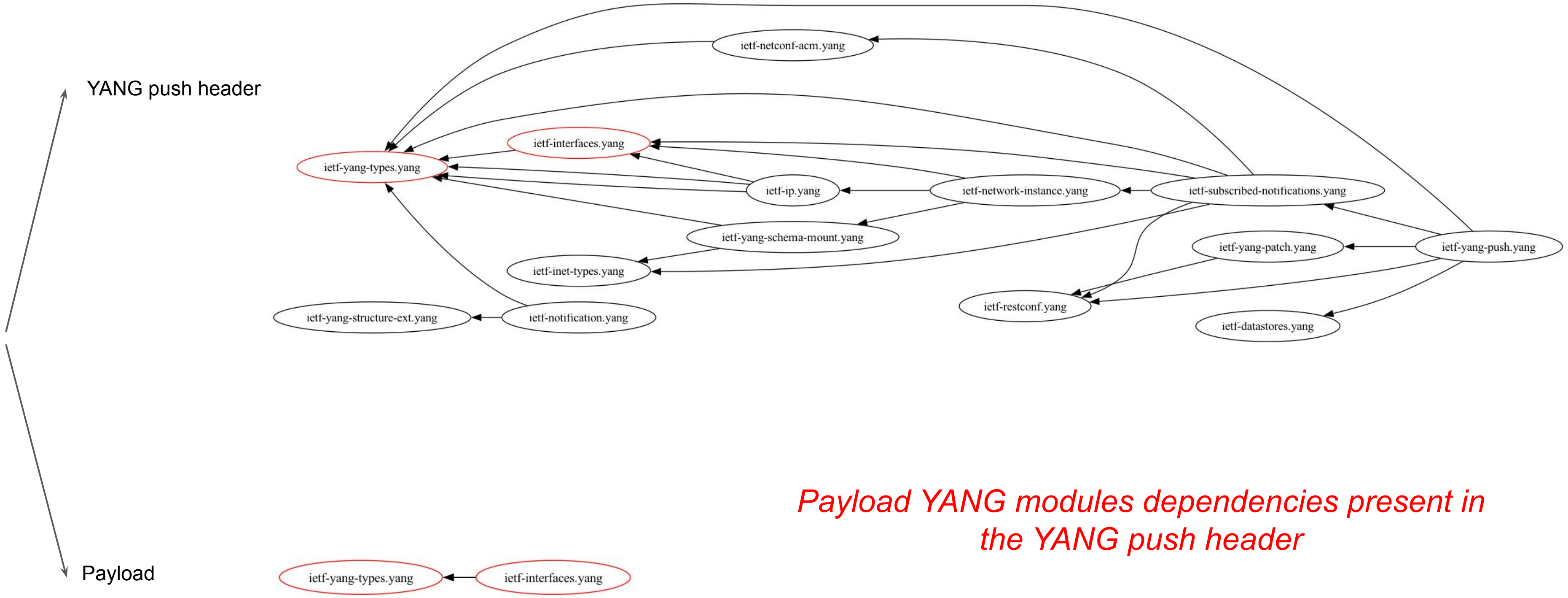


```
{
  "ietf-notification:notification": {
    "eventTime": "2023-03-25T08:30:11.22Z",
    "ietf-notification-sequencing:sysName": "example-router",
    "ietf-notification-sequencing:sequenceNumber": 1,
    "ietf-yang-push:push-update": {
      "id": 6666,
      "ietf-yang-push-netobs-timestamping:observation-time": "2023-03-25T08:30:11.22Z",
      "datastore-contents": {
        "ietf-interfaces:interfaces": [
          {
            "interface": {
              "name": "eth0",
              "type": "iana-if-type:ethernetCsmacd",
              "oper-status": "up"
            }
          }
        ]
      }
    }
  }
}
```

YANG-push header

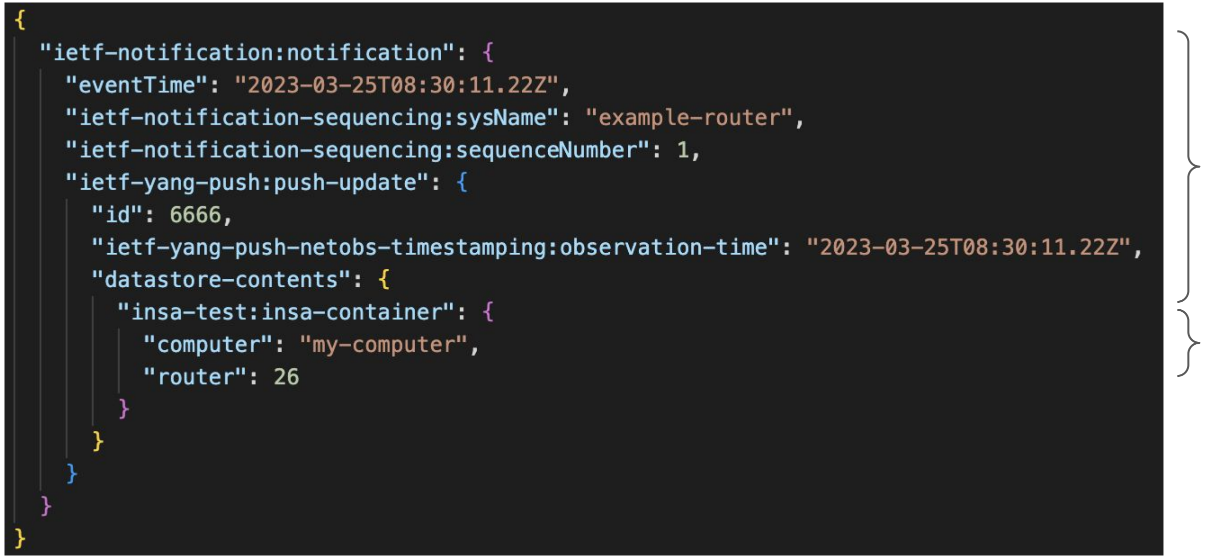
YANG-push payload

YANG push in Schema registry: dependencies



Payload YANG modules dependencies present in the YANG push header

YANG push in Schema registry: example 2



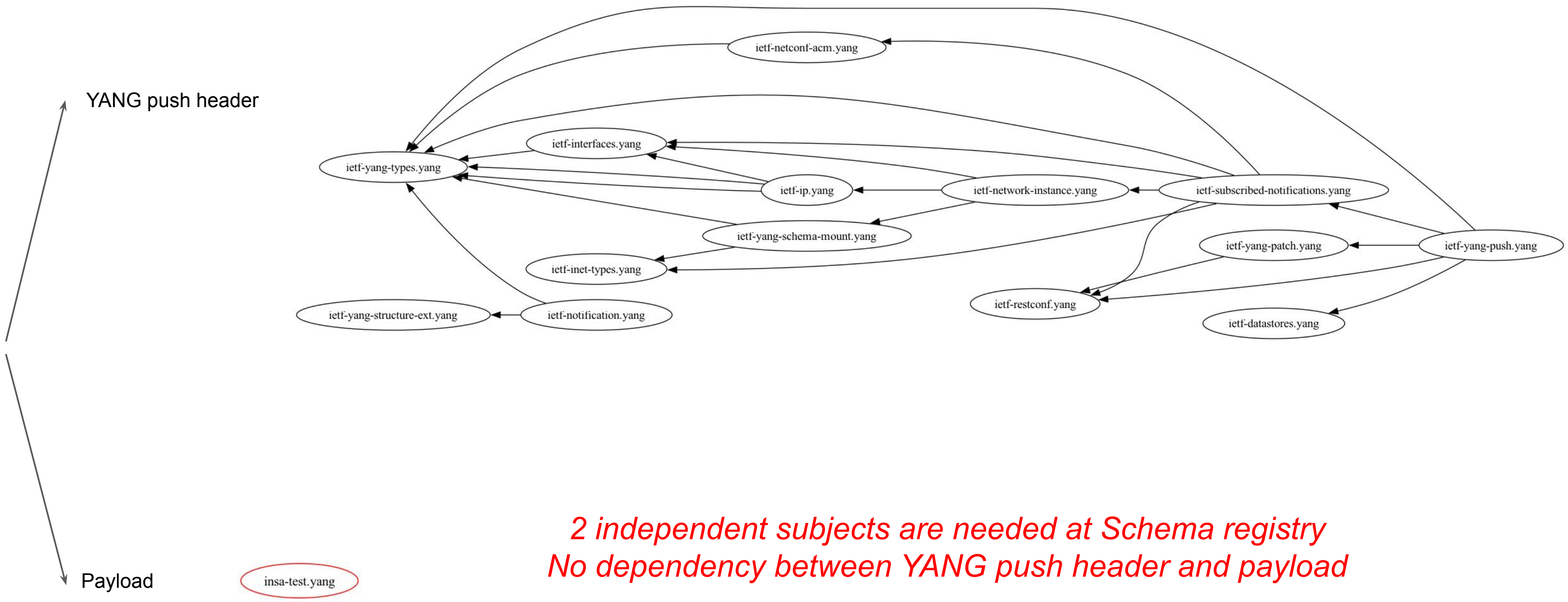
```
{
  "ietf-notification:notification": {
    "eventTime": "2023-03-25T08:30:11.22Z",
    "ietf-notification-sequencing:sysName": "example-router",
    "ietf-notification-sequencing:sequenceNumber": 1,
    "ietf-yang-push:push-update": {
      "id": 6666,
      "ietf-yang-push-netobs-timestamping:observation-time": "2023-03-25T08:30:11.22Z",
      "datastore-contents": {
        "insa-test:insa-container": {
          "computer": "my-computer",
          "router": 26
        }
      }
    }
  }
}
```

YANG-push header

Independent YANG

Note: `insa-test.yang` *is a non-standard YANG module for this example. Same usecase as if OpenConfig YANG modules were used.*

YANG push in Schema registry: dependencies



2 independent subjects are needed at Schema registry
No dependency between YANG push header and payload

Designs for integrating YANG to Schema registry

(1) Create a YANG module integrating all the YANG push header and payload
   - First approach proposed at Hackathon 117
   - **Issue**: Creating a new YANG module changes the namespace of the encoded message.

(2) Register all the YANG modules to a Schema Context
   - Approach taken by YangKit (*currently testing this approach at INSA*)
   - **Issue**: Schema registry needs to support “one subject is associated to multiple models” (not only augmentations, but independent modules)

Impact on Schema registry using Design (2)

Register all the YANG modules to a Schema Context

- **Issue**: Schema registry needs to support “one subject is associated to multiple models” (not only augmentations, but independent modules)
  - Solution 1: Implement support in schema registry
    - API is changed and not “compliant” to Confluent approach (meaning we are implementing a new API)
  - Solution 2: (workaround) Global SchemaContext per Schema Registry
    - API is not changed, but upon request, the global SchemaContext is used
    - YANG versioning and BC/NBC checks cannot be supported

Yangkit

Interfaces

Gaps

Interface with YangKit (JSON)

Input:

- JsonNode (Jackson library)
- SchemaContext (Class having all Serialised yang modules)

Output:

- Notification is valid/invalid
- Data is valid/invalid

Interface with YangKit (CBOR)

CBOR encoding is supported in Jackson library!

Input:

- JsonNode (Jackson library)
- SchemaContext (Class having all Serialised yang modules)

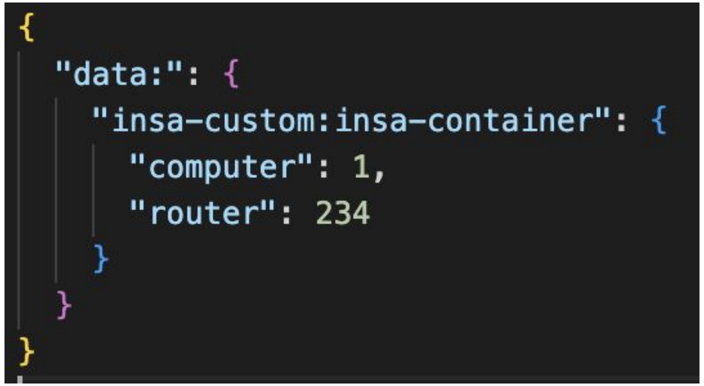
Output:

- Notification is valid/invalid
- Data is valid/invalid

Validating YANG-JSON **data**

YANG data is wrapped in “data” node

- *Is it a Standard/expected behavior?*
- *Behavior coming from NETCONF RFC6241?*
- Easy to change



```
{
    "data:": {
        "insa-custom:insa-container": {
            "computer": 1,
            "router": 234
        }
    }
}
```

Yangkit validation gaps

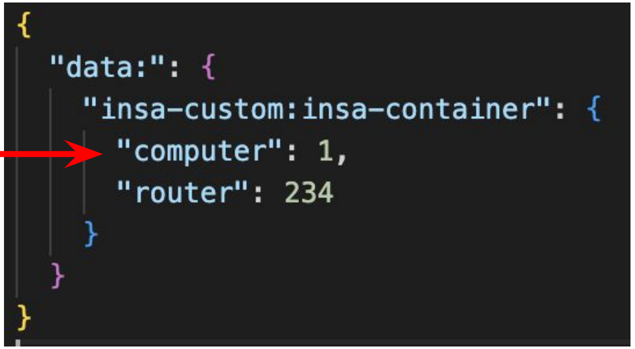
- Type validation
- Mandatory
- Lists
- Unknown elements

Validating YANG-JSON data: Type validation

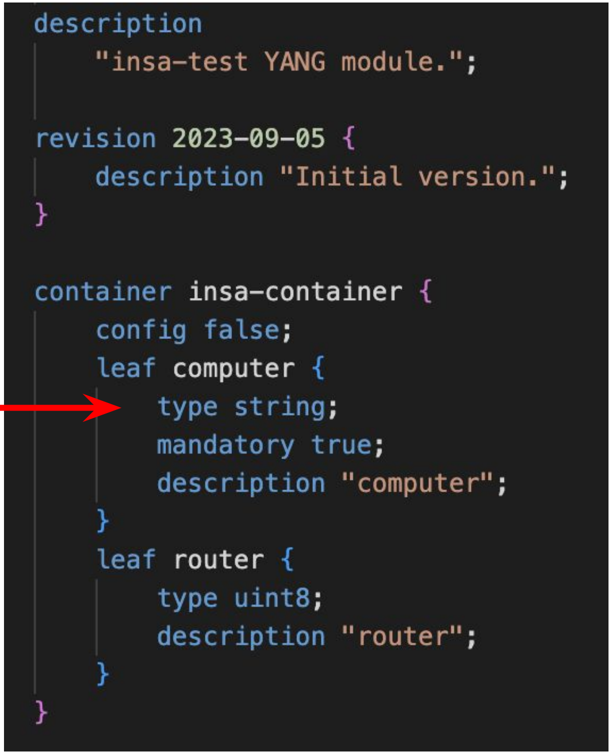
Leaf computer:

- YANG definition: **string**
- Content in the data: **integer**

→ Validator doesn't throw any error



```
{
  "data:": {
    "insa-custom:insa-container": {
      "computer": 1,
      "router": 234
    }
  }
}
```



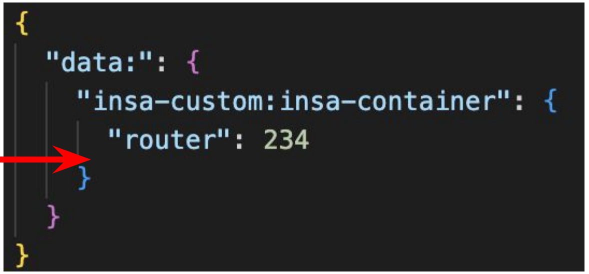
```
description
    "insa-test YANG module.";

revision 2023-09-05 {
    description "Initial version.";
}

container insa-container {
    config false;
    leaf computer {
        type string;
        mandatory true;
        description "computer";
    }
    leaf router {
        type uint8;
        description "router";
    }
}
```

Validating YANG-JSON data: Mandatory leaves

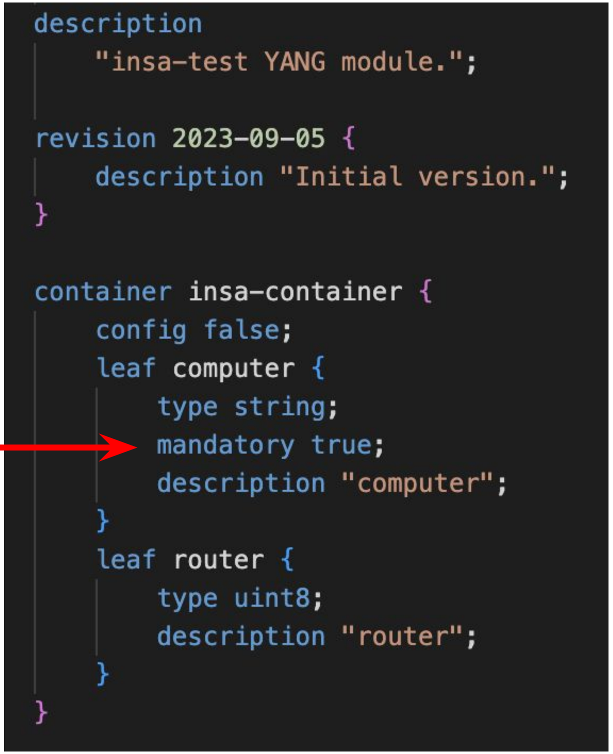
Leaf `computer`:

- YANG definition: **mandatory**
- Content in the data: **not present**

→ Validator doesn't throw any error



```
{
  "data:": {
    "insa-custom:insa-container": {
      "router": 234
    }
  }
}
```



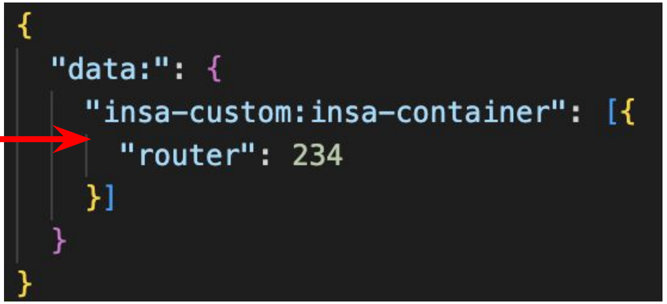
```
description
    "insa-test YANG module.";

revision 2023-09-05 {
    description "Initial version.";
}

container insa-container {
    config false;
    leaf computer {
        type string;
        mandatory true;
        description "computer";
    }
    leaf router {
        type uint8;
        description "router";
    }
}
```

Validating YANG-JSON data: Lists (missing keys)

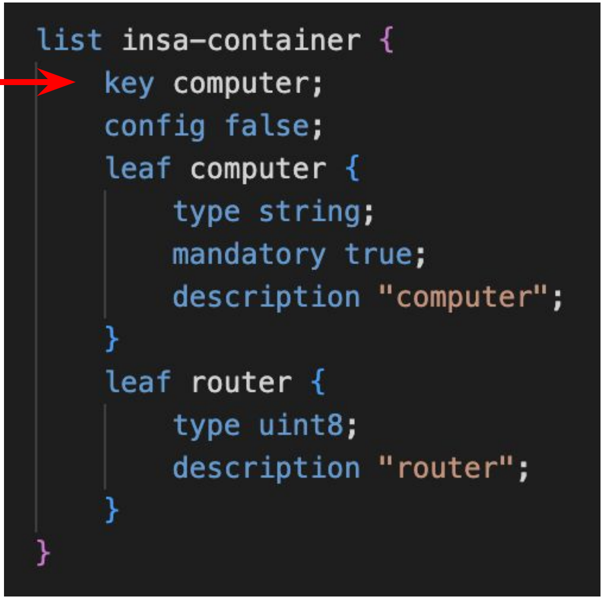
Leaf `computer`:

- YANG definition: **Key of a list**
- Content in the data: **not present**

→ Validator throws a **“missing-element”** error



```
{
  "data:": {
    "insa-custom:insa-container": [{
      "router": 234
    }]
  }
}
```



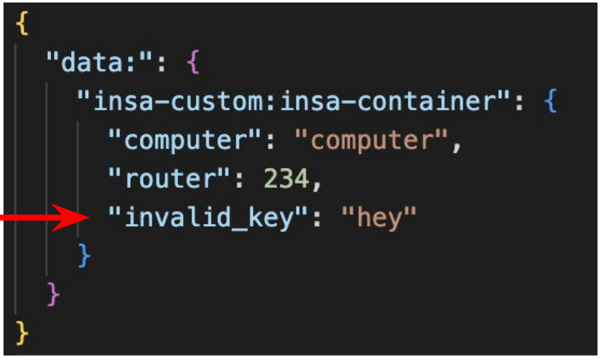
```
list insa-container {
    key computer;
    config false;
    leaf computer {
        type string;
        mandatory true;
        description "computer";
    }
    leaf router {
        type uint8;
        description "router";
    }
}
```

Validating YANG-JSON data: Unknown elements

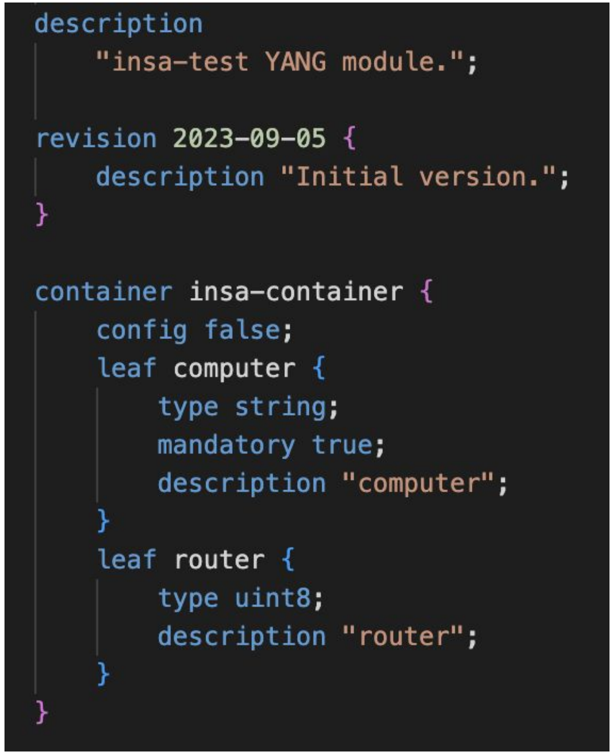
Leaf `invalid_key`:

- YANG definition: **<u>Not defined</u>**
- Content in the data: **<u>Present</u>**

→ Validator throws an "**unknown element**" error



```
{
  "data:": {
    "insa-custom:insa-container": {
      "computer": "computer",
      "router": 234,
      "invalid_key": "hey"
    }
  }
}
```



```
description
    "insa-test YANG module.";

revision 2023-09-05 {
    description "Initial version.";
}

container insa-container {
    config false;
    leaf computer {
        type string;
        mandatory true;
        description "computer";
    }
    leaf router {
        type uint8;
        description "router";
    }
}
```

Validating YANG push Notification

- Accepts a YANG module defining a structure
- Can separate YANG push header from the payload
- Data Validation: similar to YANG data

Corner cases to have in mind

XPath is not managed by Yangkit

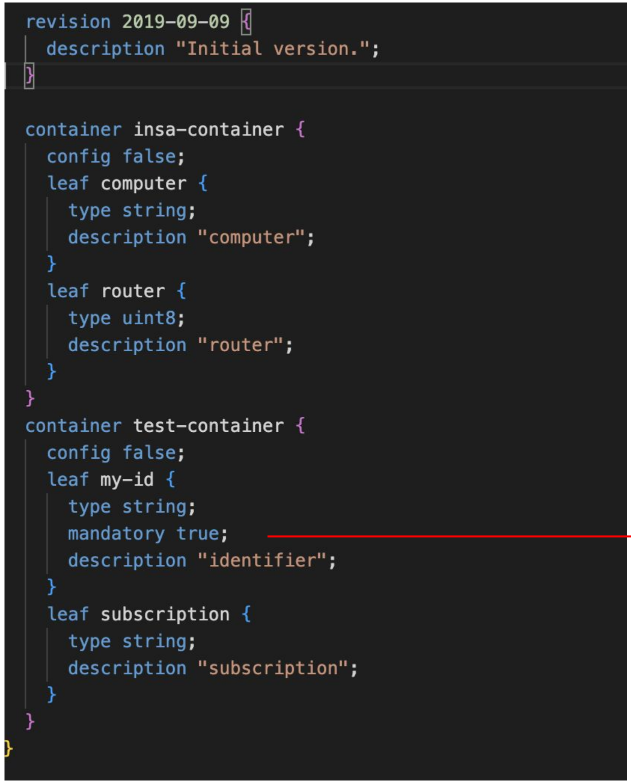
- YANG push subscription need to validate the data defined by the XPath
- The rest of the YANG module, the data should not be present

→ Yangkit approach: validate against a general YANG tree defined by **SchemaContext**

Corner cases to have in mind

Xpath=/example:insa-container

Xpath=/example:test-container



```
revision 2019-09-09 {
  description "Initial version.";
}

container insa-container {
  config false;
  leaf computer {
    type string;
    description "computer";
  }
  leaf router {
    type uint8;
    description "router";
  }
}
container test-container {
  config false;
  leaf my-id {
    type string;
    mandatory true;
    description "identifier";
  }
  leaf subscription {
    type string;
    description "subscription";
  }
}
}
```

Mandatory field

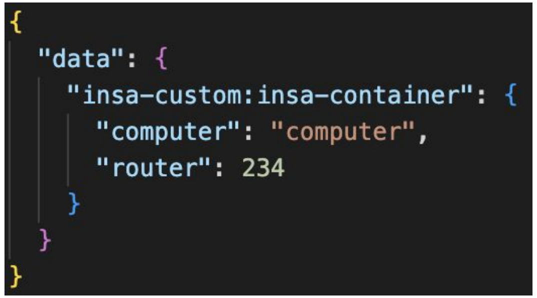
Corner cases to have in mind

Subscription to

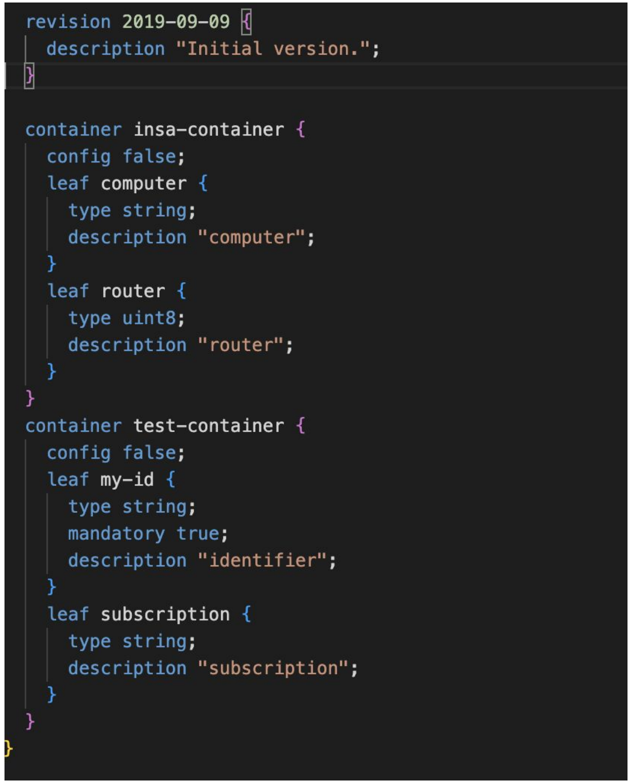
- Xpath=/example:insa-container

Yangkit default behavior (Checked with Frank):

- Data not having /test-container/my-id should fail



```
{
  "data": {
    "insa-custom:insa-container": {
      "computer": "computer",
      "router": 234
    }
  }
}
```



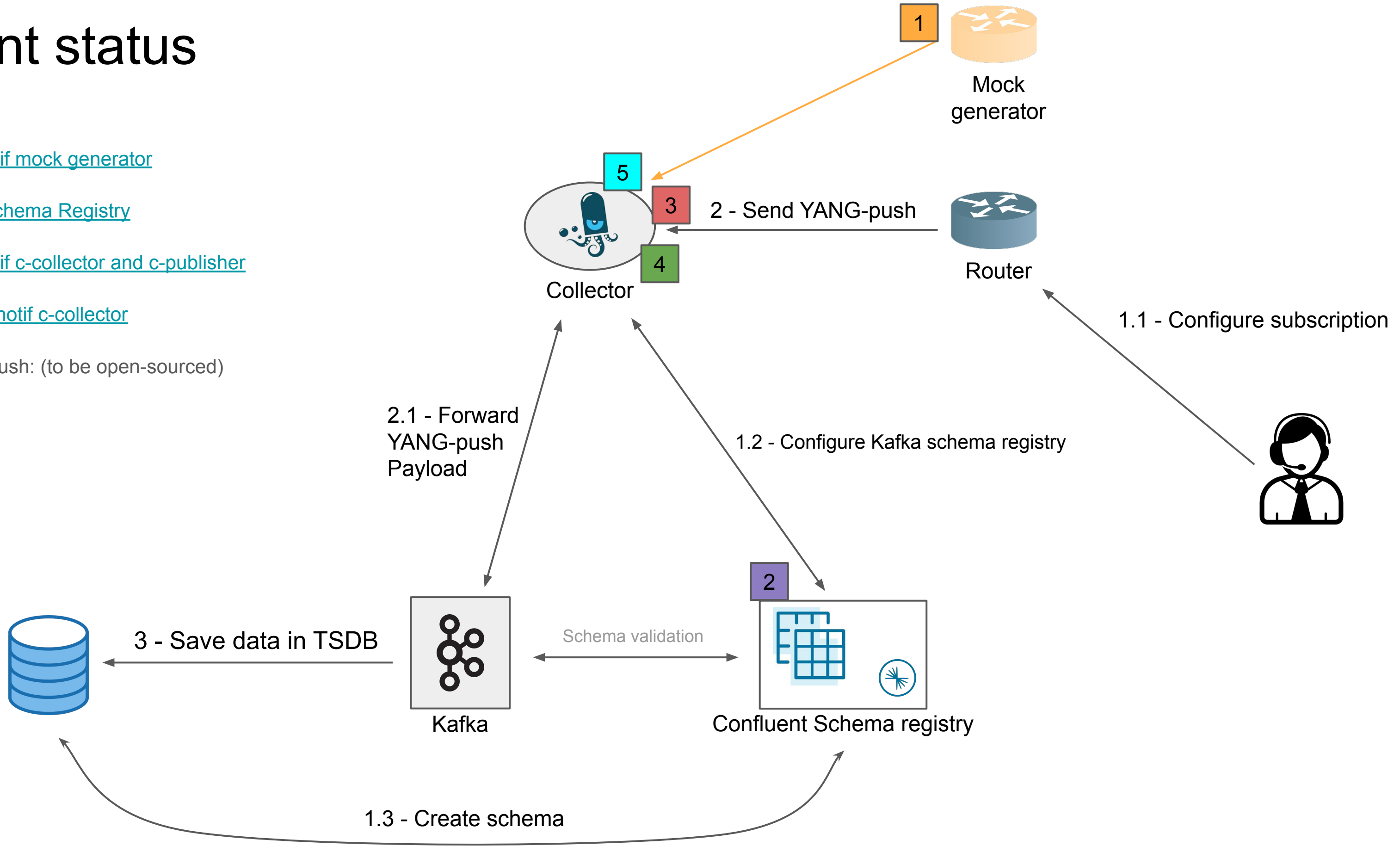
```
revision 2019-09-09 {
  description "Initial version.";
}

container insa-container {
  config false;
  leaf computer {
    type string;
    description "computer";
  }
  leaf router {
    type uint8;
    description "router";
  }
}
container test-container {
  config false;
  leaf my-id {
    type string;
    mandatory true;
    description "identifier";
  }
  leaf subscription {
    type string;
    description "subscription";
  }
}
}
```

Missing pieces

libyangserdes

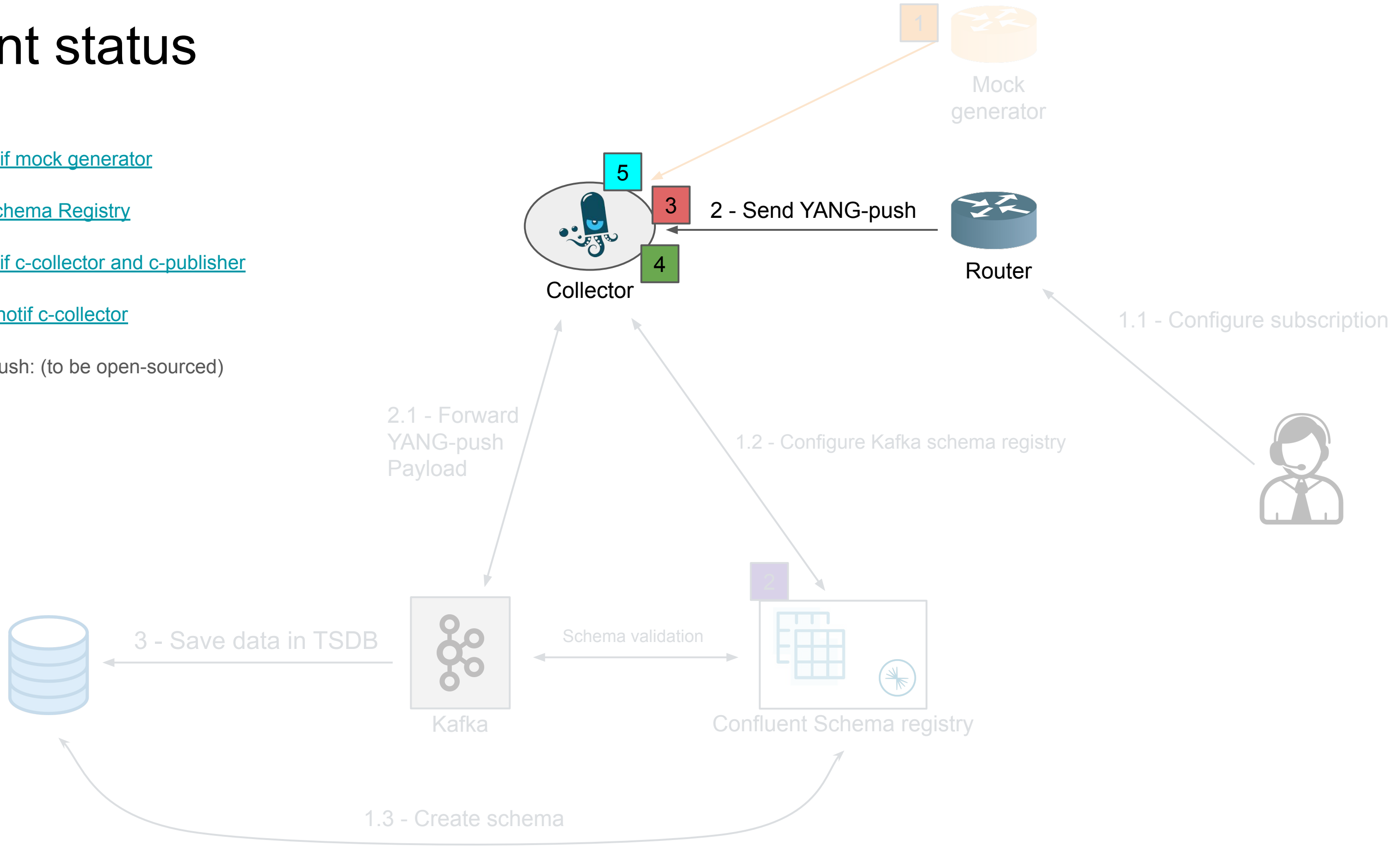
Current status

1	UDP-notif mock generator
2	Kafka Schema Registry
3	UDP-notif c-collector and c-publisher
4	HTTPS-notif c-collector
5	libyangpush: (to be open-sourced)

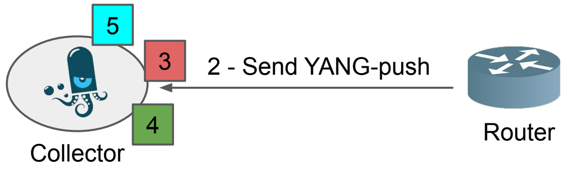

1
Mock generator
5
3
2 - Send YANG-push
Router
4
Collector
1.1 - Configure subscription
2.1 - Forward YANG-push Payload
1.2 - Configure Kafka schema registry
2
3 - Save data in TSDB
Schema validation
Kafka
Confluent Schema registry
1.3 - Create schema

Current status

- 1 UDP-notif mock generator
- 2 Kafka Schema Registry
- 3 UDP-notif c-collector and c-publisher
- 4 HTTPS-notif c-collector
- 5 libyangpush: (to be open-sourced)


1
Mock generator
5
3
4
Collector
2 - Send YANG-push
Router
1.1 - Configure subscription
2.1 - Forward YANG-push Payload
1.2 - Configure Kafka schema registry
2
3 - Save data in TSDB
Kafka
Schema validation
Confluent Schema registry
1.3 - Create schema

Current status: router - collector



- Router
  - able to craft UDP-notif messages using: UDP-notif c-collector and c-publisher

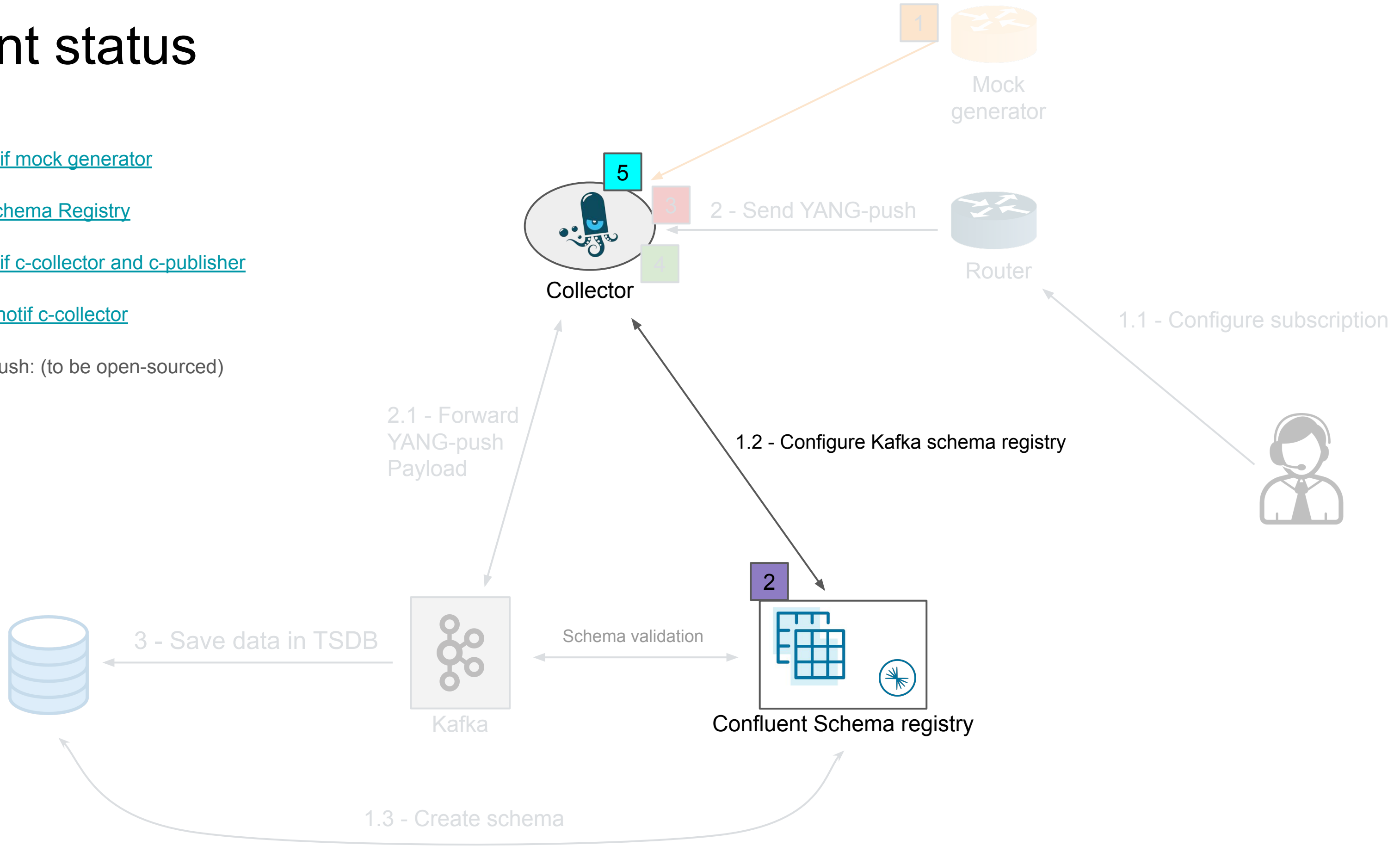
- Collector
  - able to collect UDP-notif messages using: UDP-notif c-collector and c-publisher
  - able to collect HTTPS-notif messages using: HTTPS-notif c-collector
  - able to get YANG modules dependencies using: libyangpush (to be open-sourced)

YANG-JSON and YANG-CBOR encoding is generated at Router independently from the transport protocol.

*Current Transports already support **YANG-JSON** and **YANG-CBOR** MediaType*

Current status

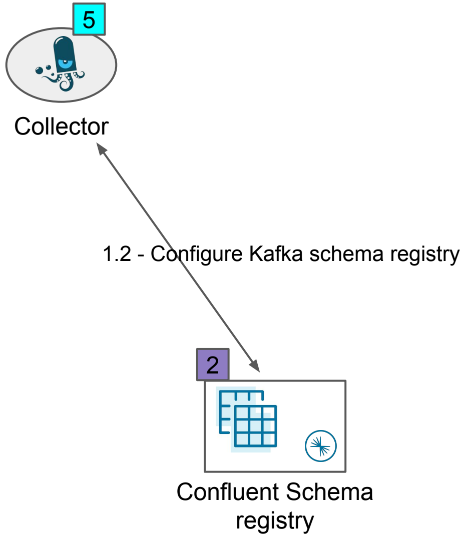
- 🟧 1 UDP-notif mock generator
- 🟪 2 Kafka Schema Registry
- 🟥 3 UDP-notif c-collector and c-publisher
- 🟩 4 HTTPS-notif c-collector
- 🟦 5 libyangpush: (to be open-sourced)


1
Mock generator
5
Collector
3
2 - Send YANG-push
Router
4
1.1 - Configure subscription
2.1 - Forward YANG-push Payload
1.2 - Configure Kafka schema registry
2
3 - Save data in TSDB
Kafka
Schema validation
Confluent Schema registry
1.3 - Create schema

Current status: collector - schema registry

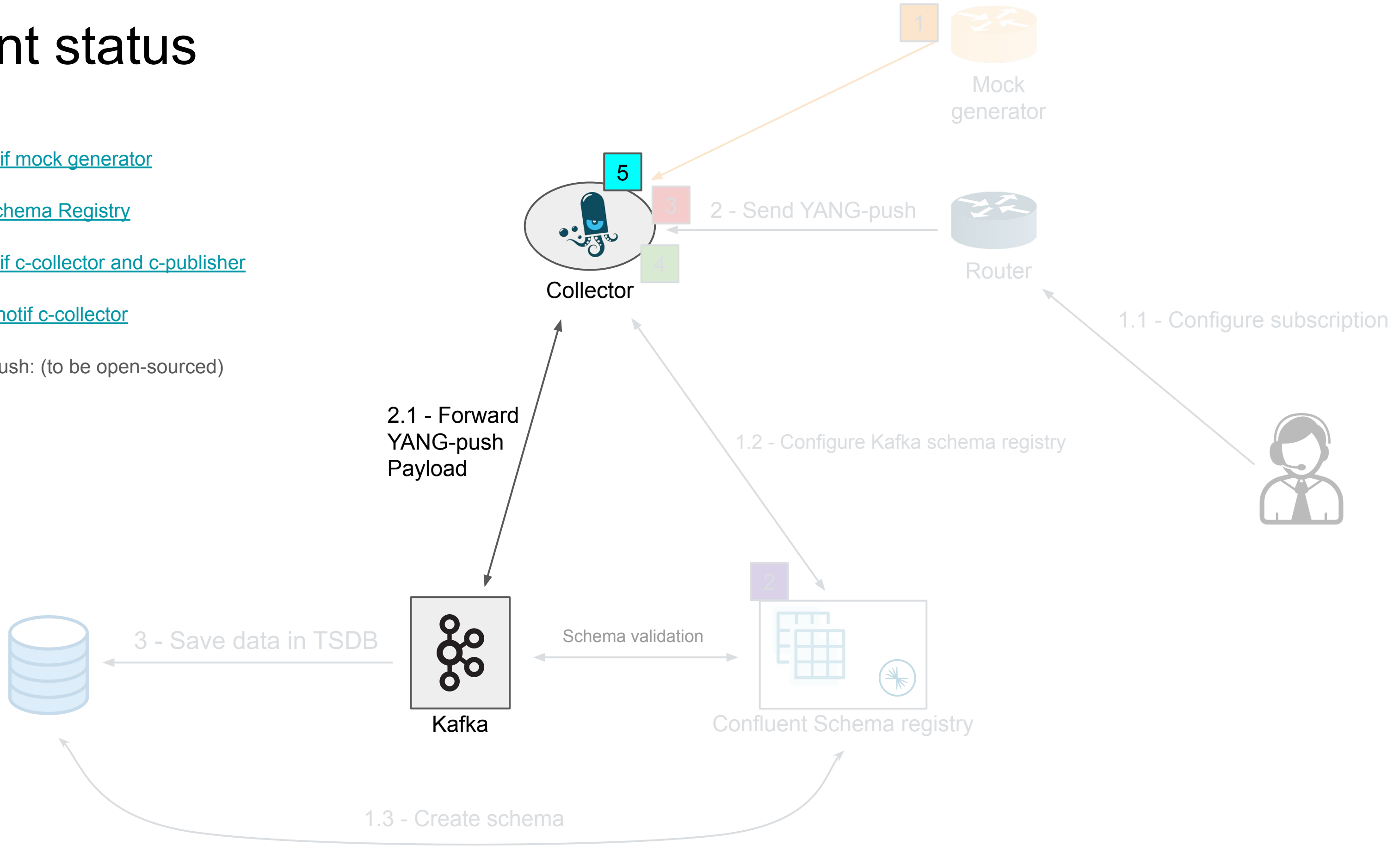
- Collector
  - when the YANG push message is received, we are able to get all YANG modules from the router using libyangpush
  - Generate registration to Schema Registry using libyangpush

- Schema registry
  - Support **YANG** as a schema type
  - *WIP: one subject defines one YANG module or one Schema Context*


5
Collector
1.2 - Configure Kafka schema registry
2
Confluent Schema registry

Current status

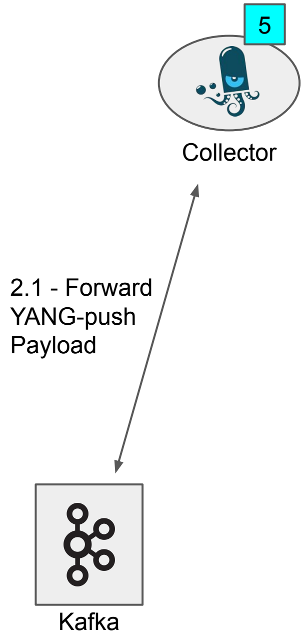
1	UDP-notif mock generator
2	Kafka Schema Registry
3	UDP-notif c-collector and c-publisher
4	HTTPS-notif c-collector
5	libyangpush: (to be open-sourced)


1
Mock generator
5
Collector
3
4
2 - Send YANG-push
Router
1.1 - Configure subscription
2.1 - Forward YANG-push Payload
1.2 - Configure Kafka schema registry
2
3 - Save data in TSDB
Kafka
Schema validation
Confluent Schema registry
1.3 - Create schema

Current status: collector - Kafka topic

- Collector
    - Once the collector got the Schema_id serialize the message to Kafka using **libyangserdes** (Craft Kafka message with MAGICBYTE and Schema_id)

- Kafka
    - is able to receive JSON or CBOR messages

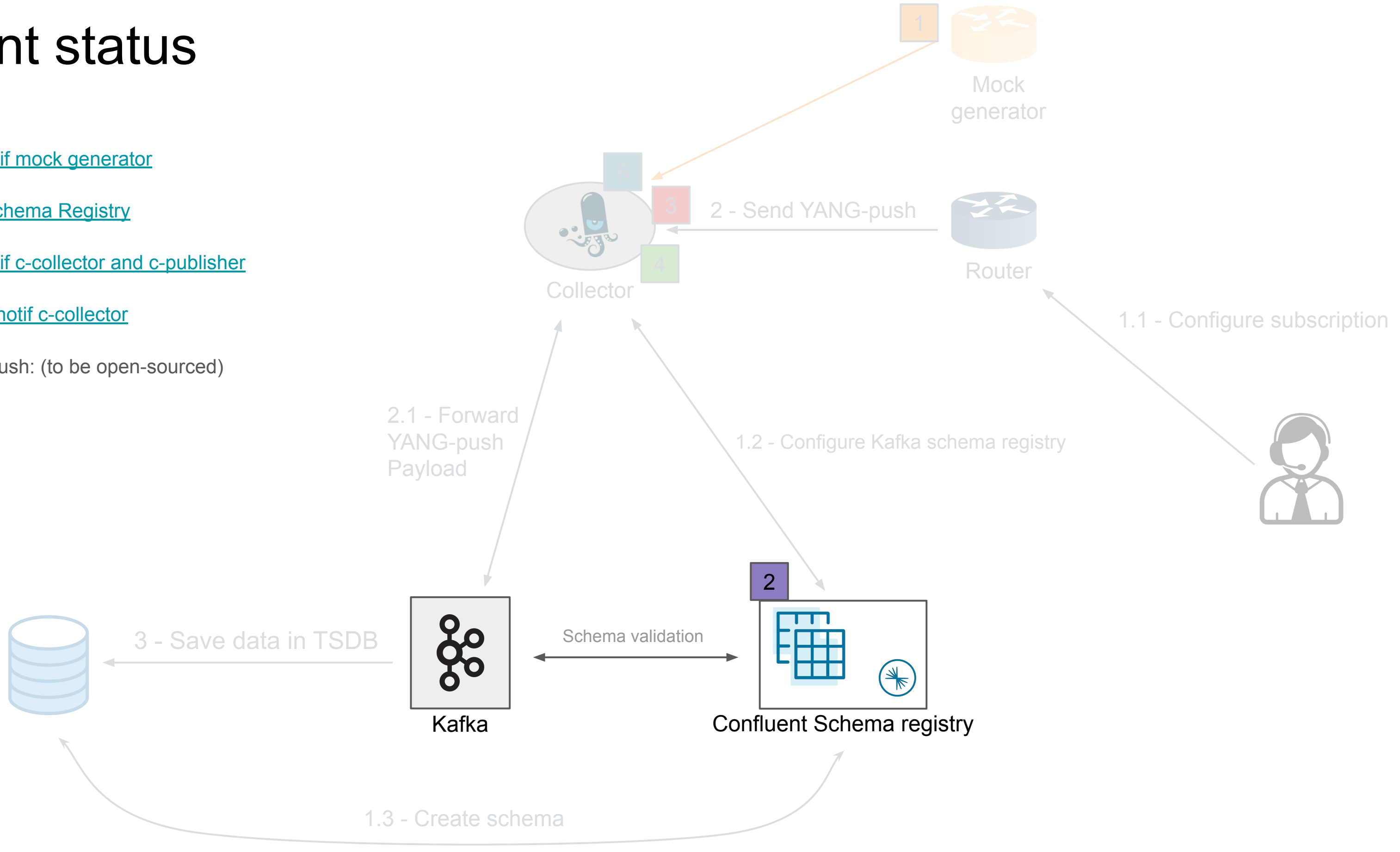


libyangserdes

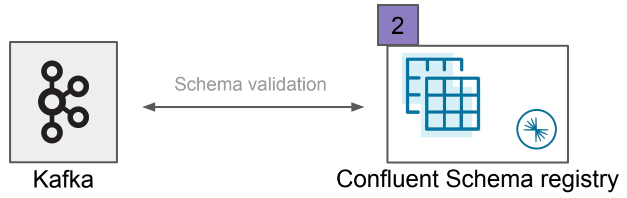
- Client uses `libyangpush` to register all the YANG modules to Schema Registry
- Client uses `libyangserdes` to craft the kafka serialised message including the ***MAGICBYTE*** and the ***Schema_id***
- libyangserdes must not modify the content of the message

Current status

1	UDP-notif mock generator
2	Kafka Schema Registry
3	UDP-notif c-collector and c-publisher
4	HTTPS-notif c-collector
5	libyangpush: (to be open-sourced)


1
Mock generator
2 - Send YANG-push
3
4
Router
Collector
1.1 - Configure subscription
2.1 - Forward YANG-push Payload
1.2 - Configure Kafka schema registry
2
3 - Save data in TSDB
Schema validation
Kafka
Confluent Schema registry
1.3 - Create schema

Current status: Kafka - Schema Registry

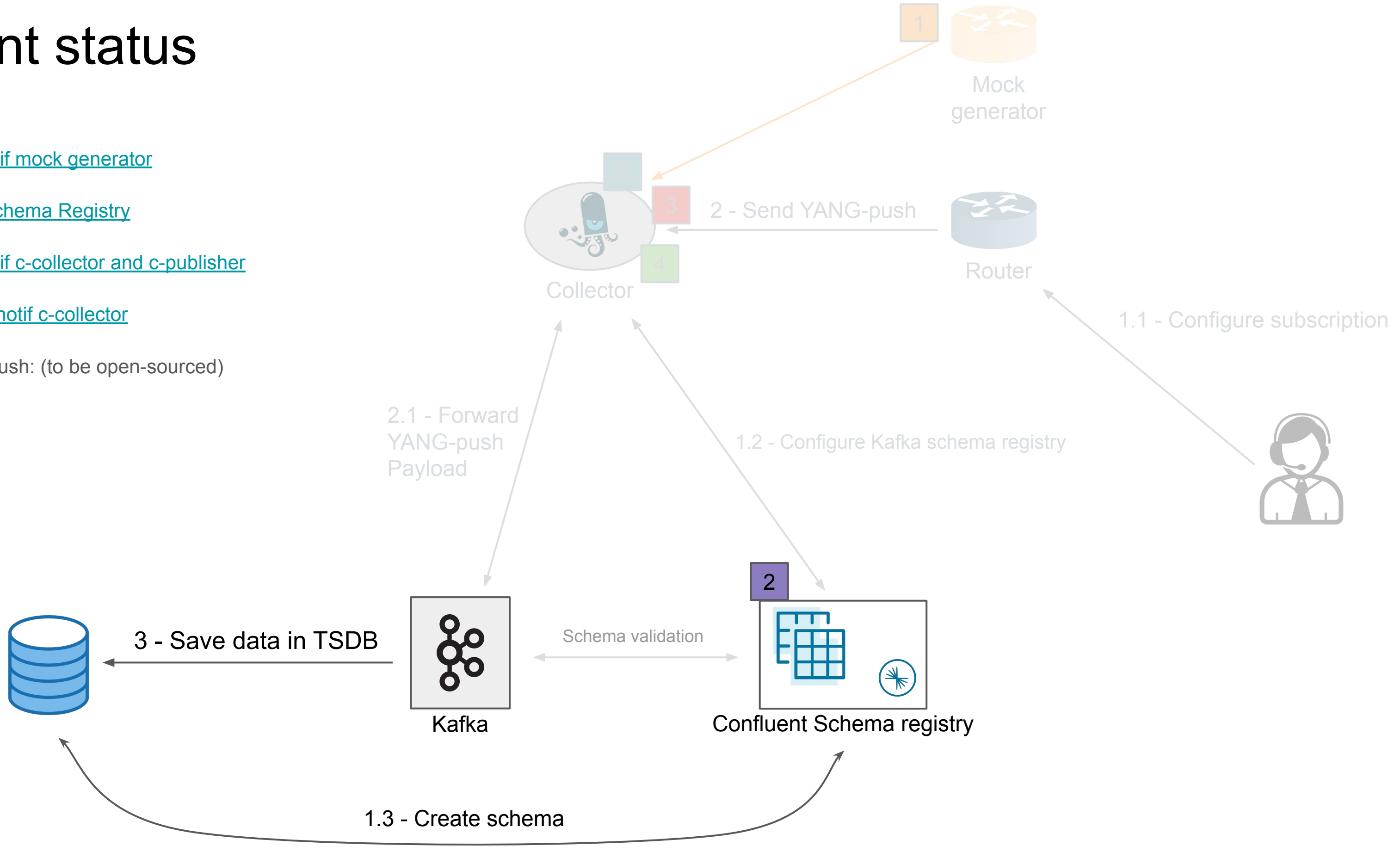


- Kafka
  - is able to get the Schema_id from the message
  - is able to request the schema from the Schema Registry and validate the content
  - WIP: YANG validation with Yangkit

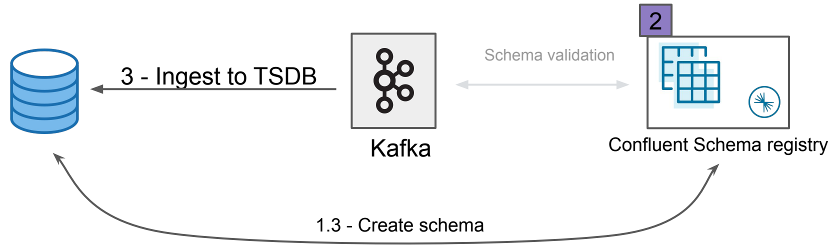
- Confluent Schema Registry
  - is able to provide the schema / schema context
  - WIP: Current discussions if current API need to be modified to accommodate YANG subjects

Current status

- 🟧 1 [UDP-notif mock generator]
- 🟪 2 [Kafka Schema Registry]
- 🟥 3 [UDP-notif c-collector and c-publisher]
- 🟩 4 [HTTPS-notif c-collector]
- 🟦 5 libyangpush: (to be open-sourced)


1
Mock generator
3
4
Collector
2 - Send YANG-push
Router
1.1 - Configure subscription
2.1 - Forward YANG-push Payload
1.2 - Configure Kafka schema registry
3 - Save data in TSDB
Kafka
Schema validation
2
Confluent Schema registry
1.3 - Create schema

Current status: TSDB ingestion



- TSDB
  - Druid uses Kafka connect to create/ingest data to the database
  - Missing: YANG connector (part of Druid)
- Kafka
  - is able to provide the schema / schema context
  - WIP: Current discussions if current API need to be modified to accommodate YANG subjects
- Confluent Schema registry
  - is able to provide the schema / schema context
  - WIP: Current discussions if current API need to be modified to accommodate YANG subjects