

# Schema Registry integration with YangKit

INSA de Lyon

Alex Huang Feng ([alex.huang-feng@insa-lyon.fr](mailto:alex.huang-feng@insa-lyon.fr))

Vivek Boudia ([vivekananda.boudia@insa-lyon.fr](mailto:vivekananda.boudia@insa-lyon.fr))

Pierre Francois ([pierre.francois@insa-lyon.fr](mailto:pierre.francois@insa-lyon.fr))

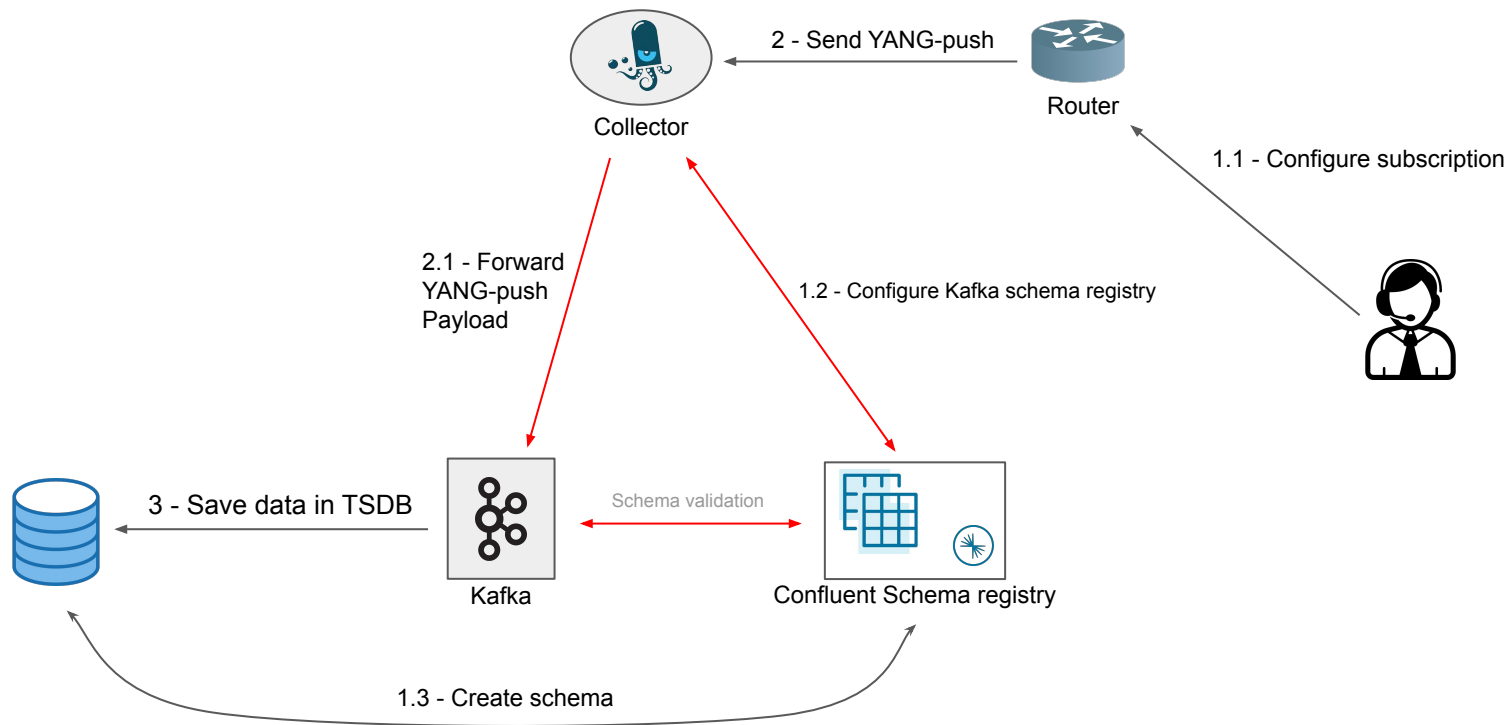
*Last Updated: November 5th 2023*

# Index

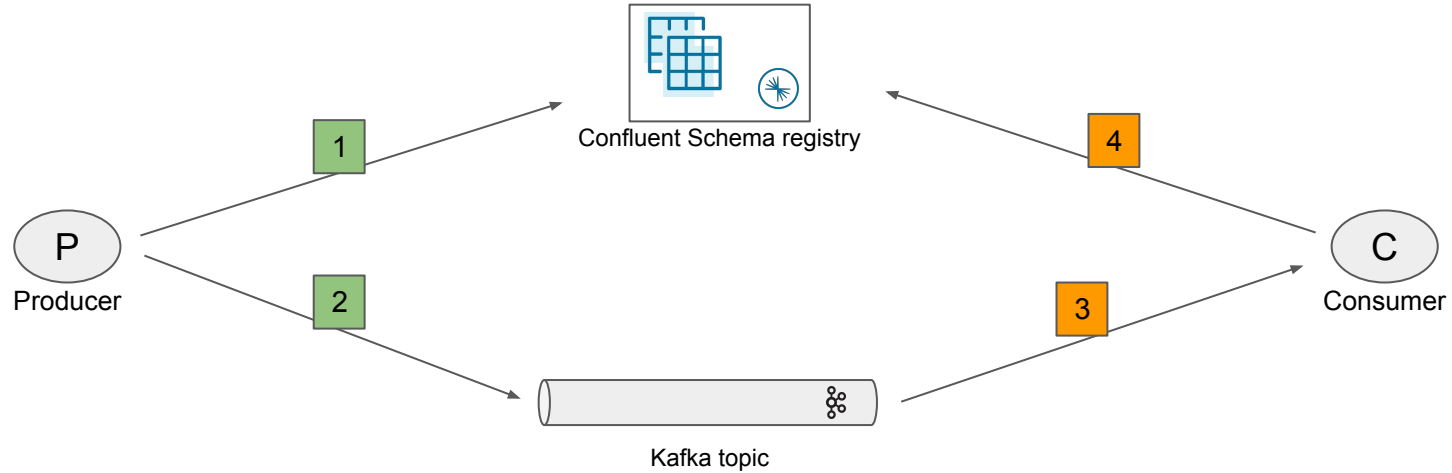
- Schema Registry
  - How does Schema registry works?
  - How YANG push modules are structured
  - Design choices for YANG integration
- YangKit
  - Interface with YangKit (JSON and CBOR)
  - Content encapsulated in “data” node
  - YANG data validation
  - YANG push Notification validation
- Missing pieces: libyangserde (C library adding the MAGIC BYTE and the schema Id into the Kafka message)

# Schema Registry

# Full Implementation-level architecture

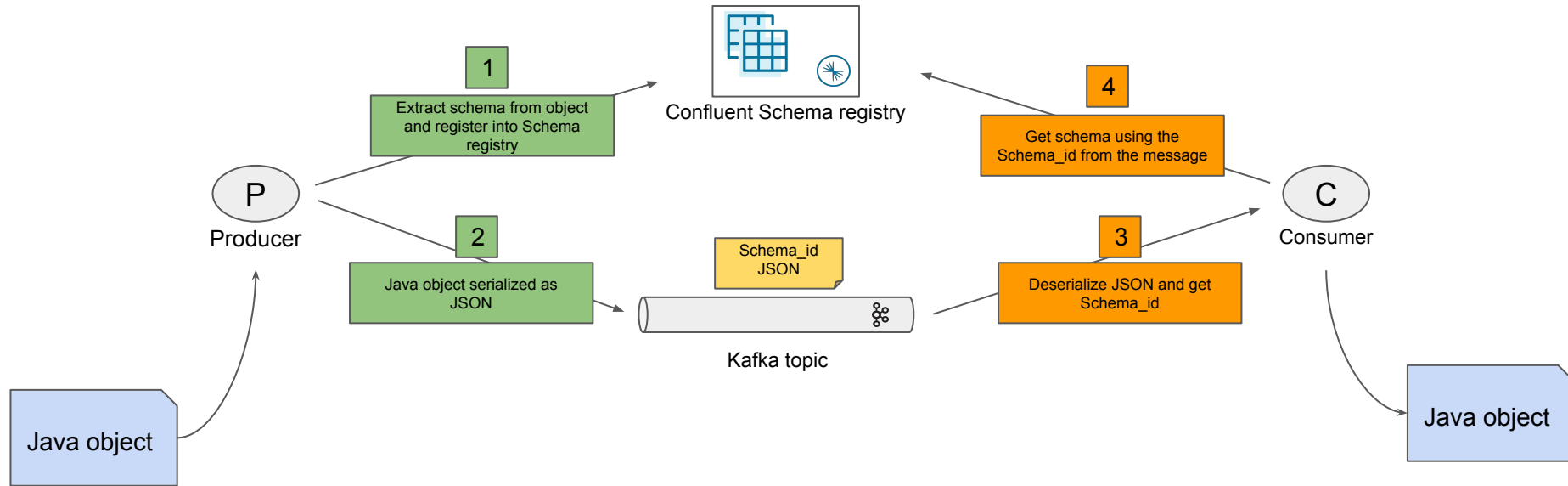


# How does Schema registry works?

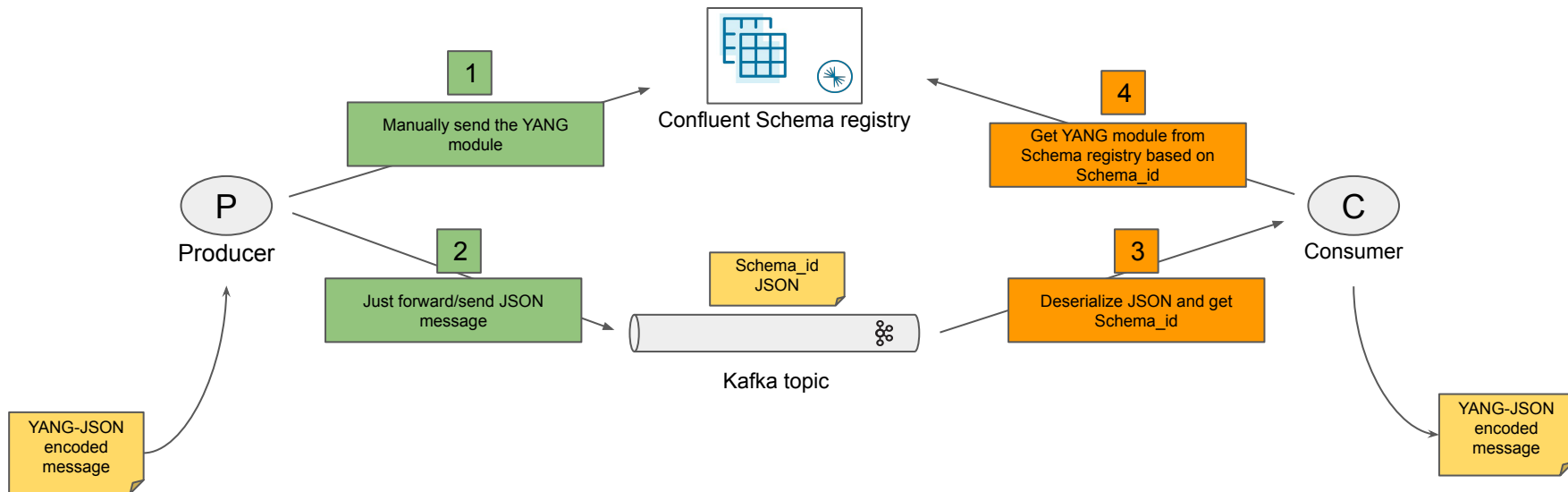


1. register schema
2. Send <MAGICBYTE, Schema ID, Content>
3. Receive <MAGICBYTE, Schema ID, Content>
4. Get schema

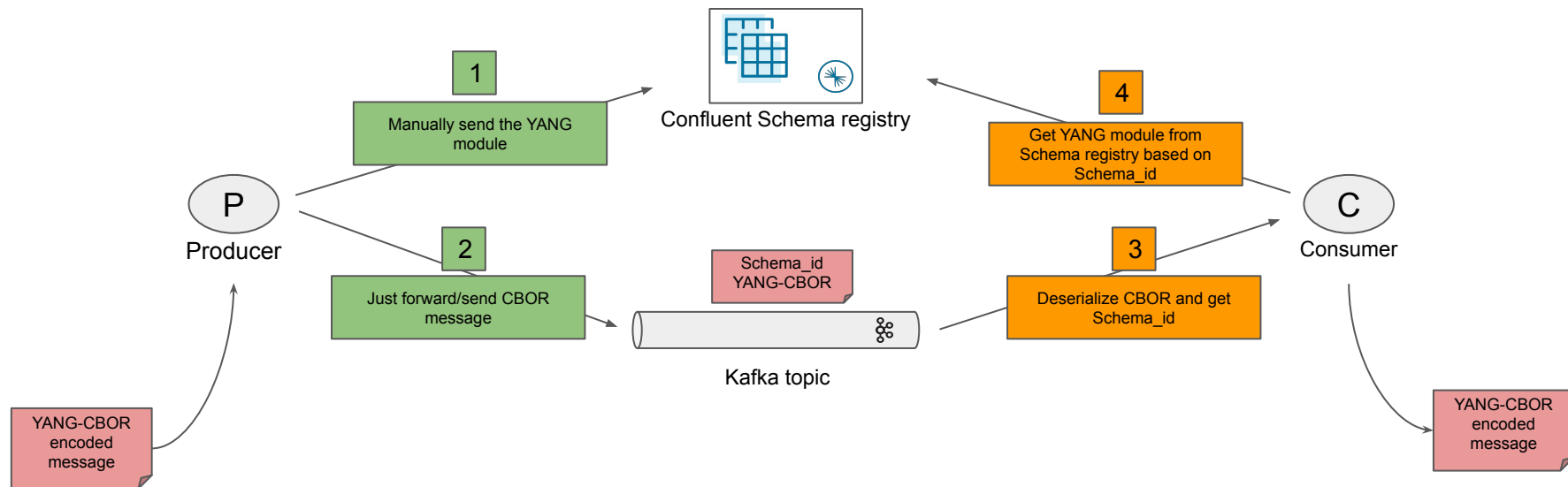
# Main usecase for Schema registry: Plain Old Java Objects



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



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



*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

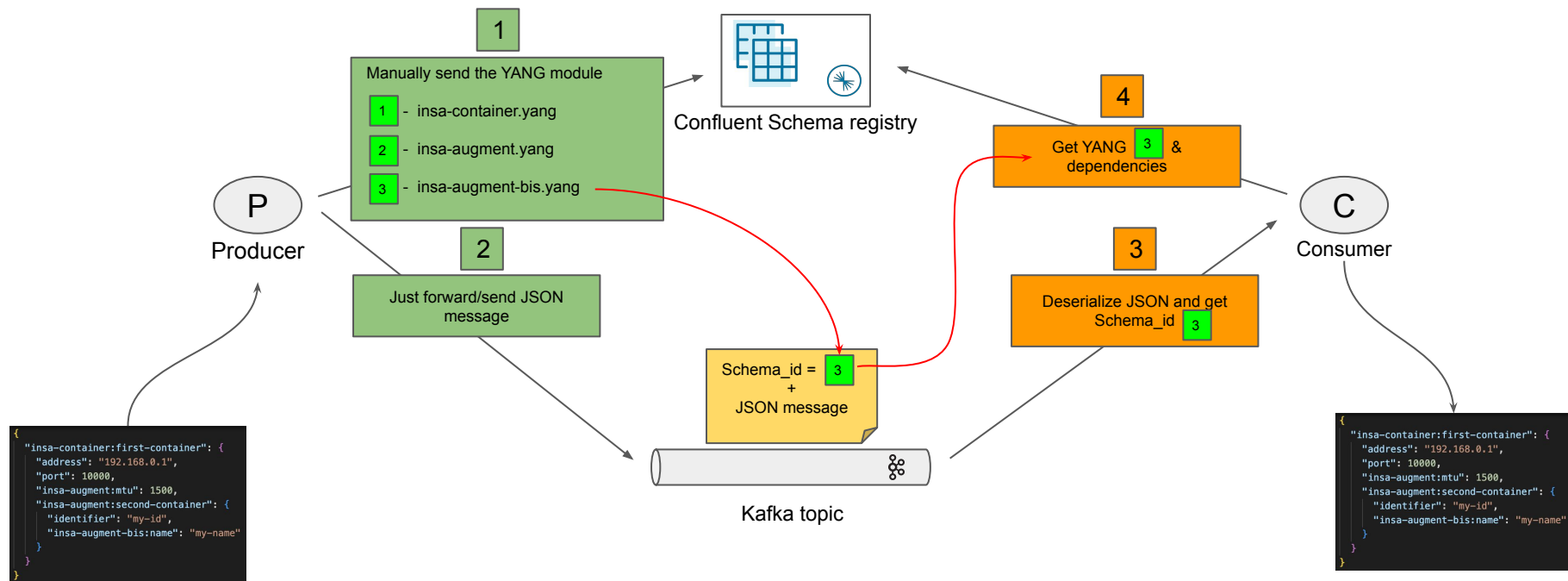
```
module: insa-container
  +-ro first-container
    +-ro address?      string
    +-ro port?         uint8
    +-ro ia:mtu?       string
    +-ro ia:second-container
      +-ro ia:identifier? string
      +-ro iab:name?    string
```



```
{
  "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"
    }
  }
}
```

\*Schema registry Subject: Identifier of the schema used for data validation.

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



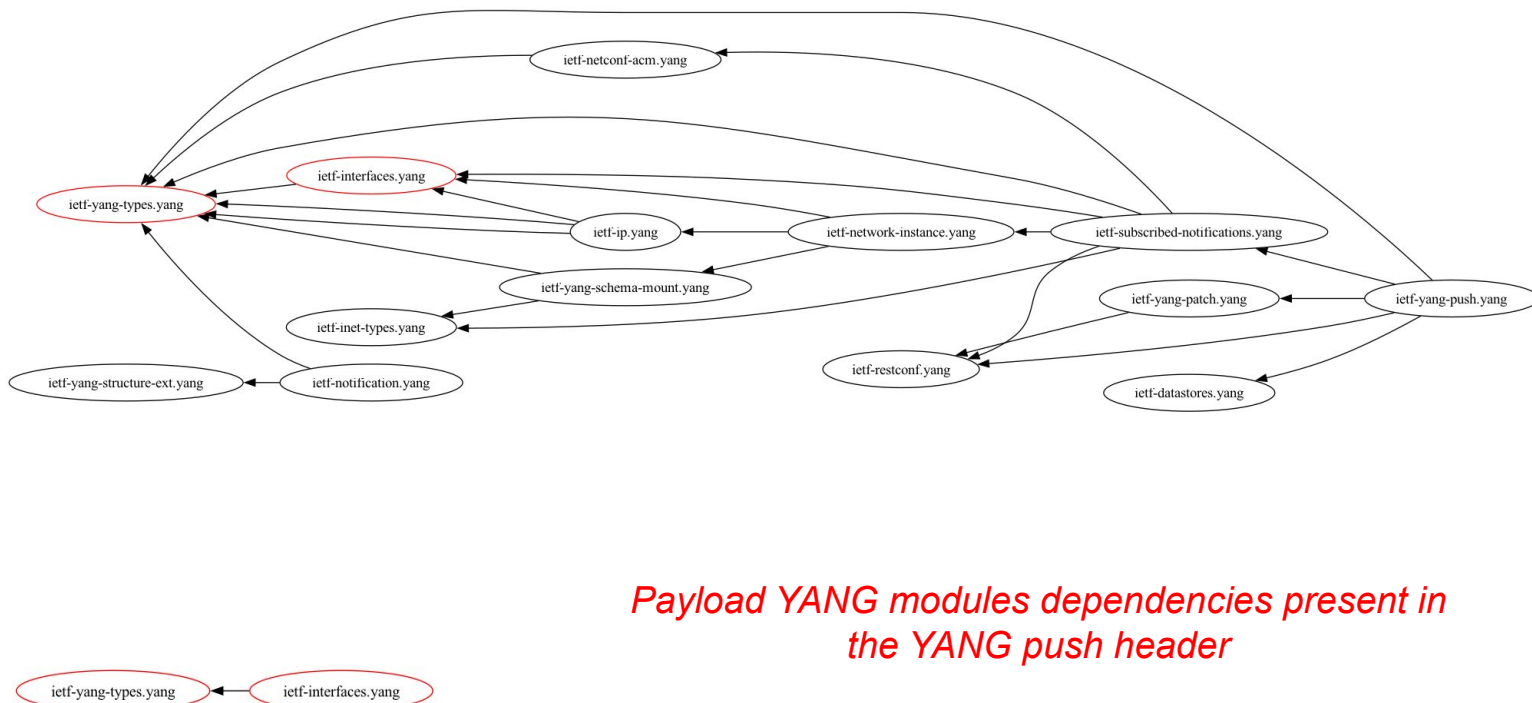
# YANG push in Schema registry: **example 1**

```
{
  "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



## YANG push in Schema registry: **example 2**

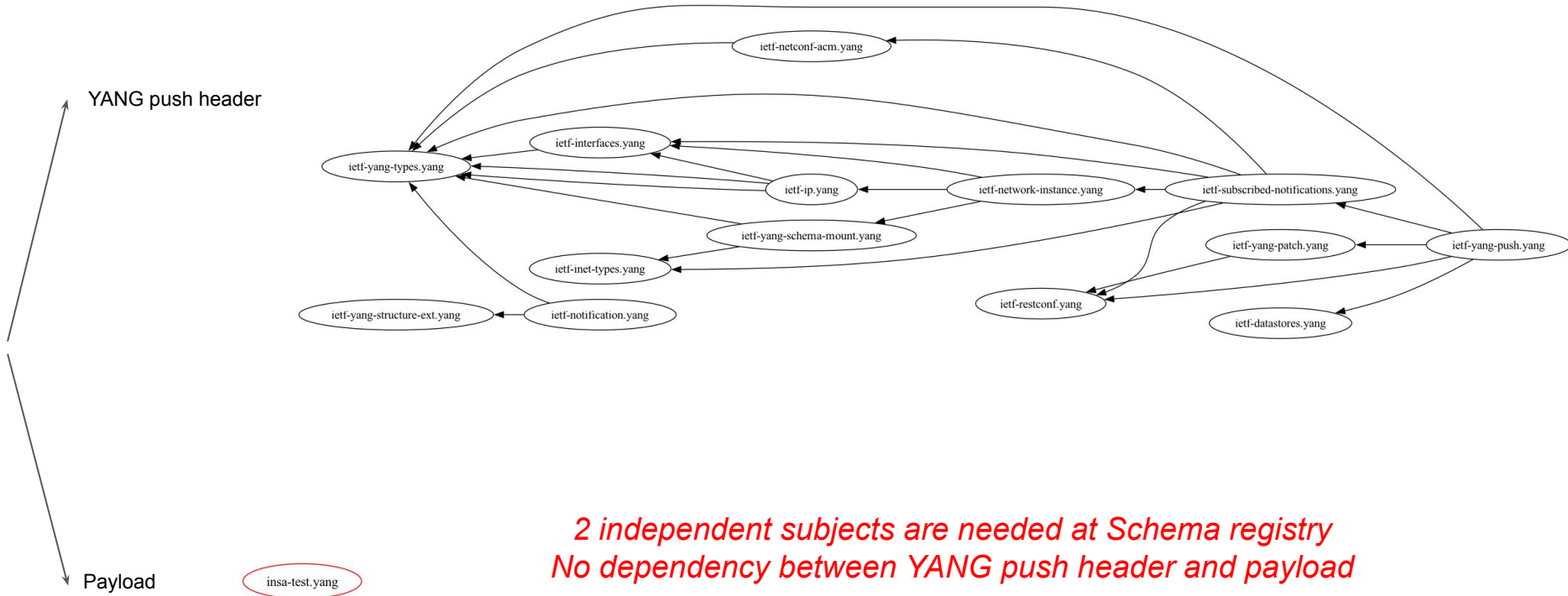
```
{
  "ietf-notification:notification": {
    "eventTime": "2023-03-25T08:30:11.22Z",
    "ietf-notification-sequencing:sysName": "example-router",
    "ietf-notification-sequencing:sequenceNumber": 1,
    "ietf-yang-push:push-update": {
      "id": 6666,
      "ietf-yang-push-netobs-timestamping:observation-time": "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





# Complexity coming from the IETF Specification

## RFC8641: YANG push

notifications:

```
+---n push-update
|   +---ro id?                sn:subscription-id
|   +---ro datastore-contents? <anydata>
|   +---ro incomplete-update? empty
+---n push-change-update {on-change}?
    +---ro id?                sn:subscription-id
    +---ro datastore-changes
    |   +---ro yang-patch
    |   |   +---ro patch-id    string
    |   |   +---ro comment?    string
    |   |   +---ro edit* [edit-id]
    |   |       +---ro edit-id    string
    |   |       +---ro operation enumeration
    |   |       +---ro target    target-resource-offset
    |   |       +---ro point?    target-resource-offset
    |   |       +---ro where?    enumeration
    |   |       +---ro value?    <anydata>
    |   +---ro incomplete-update? empty
```

## RFC7950: YANG 1.1

### 7.10. The "anydata" Statement

The "anydata" statement defines an interior node in the schema tree. It takes one argument, which is an identifier, followed by a block of substatements that holds detailed anydata information.

The "anydata" statement is used to represent an unknown set of nodes that can be modeled with YANG, except anyxml, but for which the data model is not known at module design time. It is possible, though not required, for the data model for anydata content to become known through protocol signaling or other means that are outside the scope of this document.

An example of where anydata can be useful is a list of received notifications where the specific notifications are not known at design time.

An anydata node cannot be augmented (see [Section 7.17](#)).

An anydata node exists in zero or one instance in the data tree.

An implementation may or may not know the data model used to model a specific instance of an anydata node.


Since the use of anydata limits the manipulation of the content, the "anydata" statement SHOULD NOT be used to define configuration data.

# 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)

*\*Schema Context: Instance in memory registering all YANG modules for data validation (of a message)*

## Design 1: Craft a new YANG module based on the modules coming from the routers



```
ietf-inet-types@2021-02-22.yang
ietf-yang-types@2023-01-23.yang
ietf-restconf@2017-01-26.yang
ietf-datastores@2018-02-14.yang
insa-test@2023-09-05.yang
ietf-yang-structure-ext@2020-06-17.yang
ietf-interfaces@2018-02-20.yang
ietf-yang-patch@2017-02-22.yang
ietf-netconf-acm@2018-02-14.yang
ietf-yang-schema-mount@2019-01-14.yang
ietf-ip@2018-02-22.yang
ietf-network-instance@2019-01-21.yang
ietf-notification@2023-07-23.yang
ietf-subscribed-notifications@2019-09-09.yang
ietf-yang-push@2019-09-09.yang
```

`crafted-yang-push@2023-11-07.yang`

\*Includes all leaves and types defined in the set of YANG modules

### Observations:

- the YANG modules from the routers are not used anymore once the crafted YANG is created
- Filtering (datatree/Xpath) can be managed directly from the YANG module

## Design 1: Craft a new YANG module based on the modules coming from the routers

crafted-yang-push@2023-11-07.yang ← Validate message →

```
{
  "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
        }
      }
    }
  }
}
```

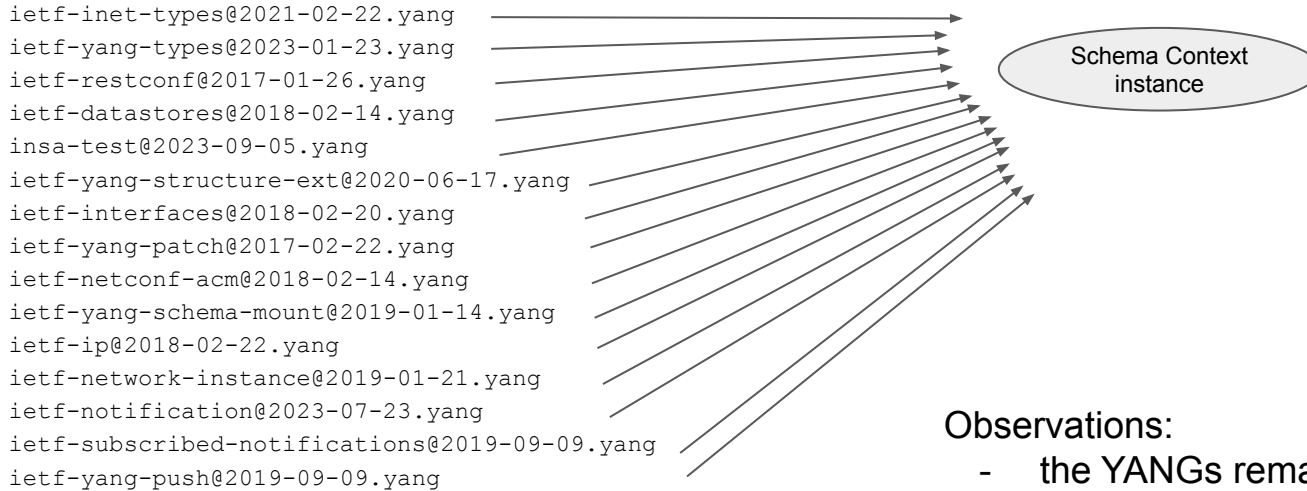
Validation happens directly against a single YANG module

# 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)

\*Schema Context: Instance in memory registering all YANG modules for data validation (of a message)

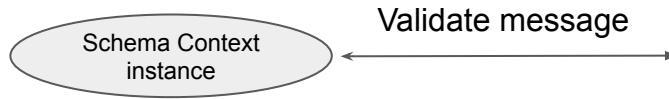
## Design 2: Register all YANG modules to an Schema context



### Observations:

- the YANGs remain intact from the routers
- YANG push message need to be validated against a Schema context, not a schema as defined in Schema registry

## Design 2: Register all YANG modules to an Schema context



```
{
  "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
        }
      }
    }
  }
}
```

Schema context needs to be instantiated on a message version basis

# 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)

*\*Schema Context: Instance in memory registering all YANG modules for data validation (of a message)*



# Complexity coming from the IETF Specification

## RFC8641: YANG push

notifications:

```
+---n push-update
|   +---ro id?                sn:subscription-id
|   +---ro datastore-contents? <anydata>
|   +---ro incomplete-update? empty
+---n push-change-update {on-change}?
    +---ro id?                sn:subscription-id
    +---ro datastore-changes
    |   +---ro yang-patch
    |   |   +---ro patch-id    string
    |   |   +---ro comment?    string
    |   |   +---ro edit* [edit-id]
    |   |       +---ro edit-id    string
    |   |       +---ro operation enumeration
    |   |       +---ro target    target-resource-offset
    |   |       +---ro point?    target-resource-offset
    |   |       +---ro where?    enumeration
    |   |       +---ro value?    <anydata>
    |   +---ro incomplete-update? empty
```

## RFC7950: YANG 1.1

### 7.10. The "anydata" Statement

The "anydata" statement defines an interior node in the schema tree. It takes one argument, which is an identifier, followed by a block of substatements that holds detailed anydata information.

The "anydata" statement is used to represent an unknown set of nodes that can be modeled with YANG, except anyxml, but for which the data model is not known at module design time. It is possible, though not required, for the data model for anydata content to become known through protocol signaling or other means that are outside the scope of this document.

An example of where anydata can be useful is a list of received notifications where the specific notifications are not known at design time.

An anydata node cannot be augmented (see [Section 7.17](#)).

An anydata node exists in zero or one instance in the data tree.

An implementation may or may not know the data model used to model a specific instance of an anydata node.

Since the use of anydata limits the manipulation of the content, the "anydata" statement SHOULD NOT be used to define configuration data.

# 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)

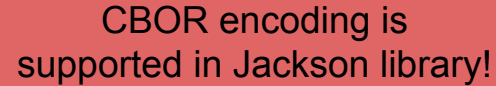
Output:

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

# Interface with YangKit (CBOR)

Input:

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



CBOR encoding is supported in Jackson library!

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

Expected: **Throw an error**

Reality: 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**

Expected: **Throw an error**

Reality: 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


Expected: Throw an error

Reality: 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: **Not defined**
- Content in the data: **Present**

Expected: **Throw an error**

Result: 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";
  }
}
```

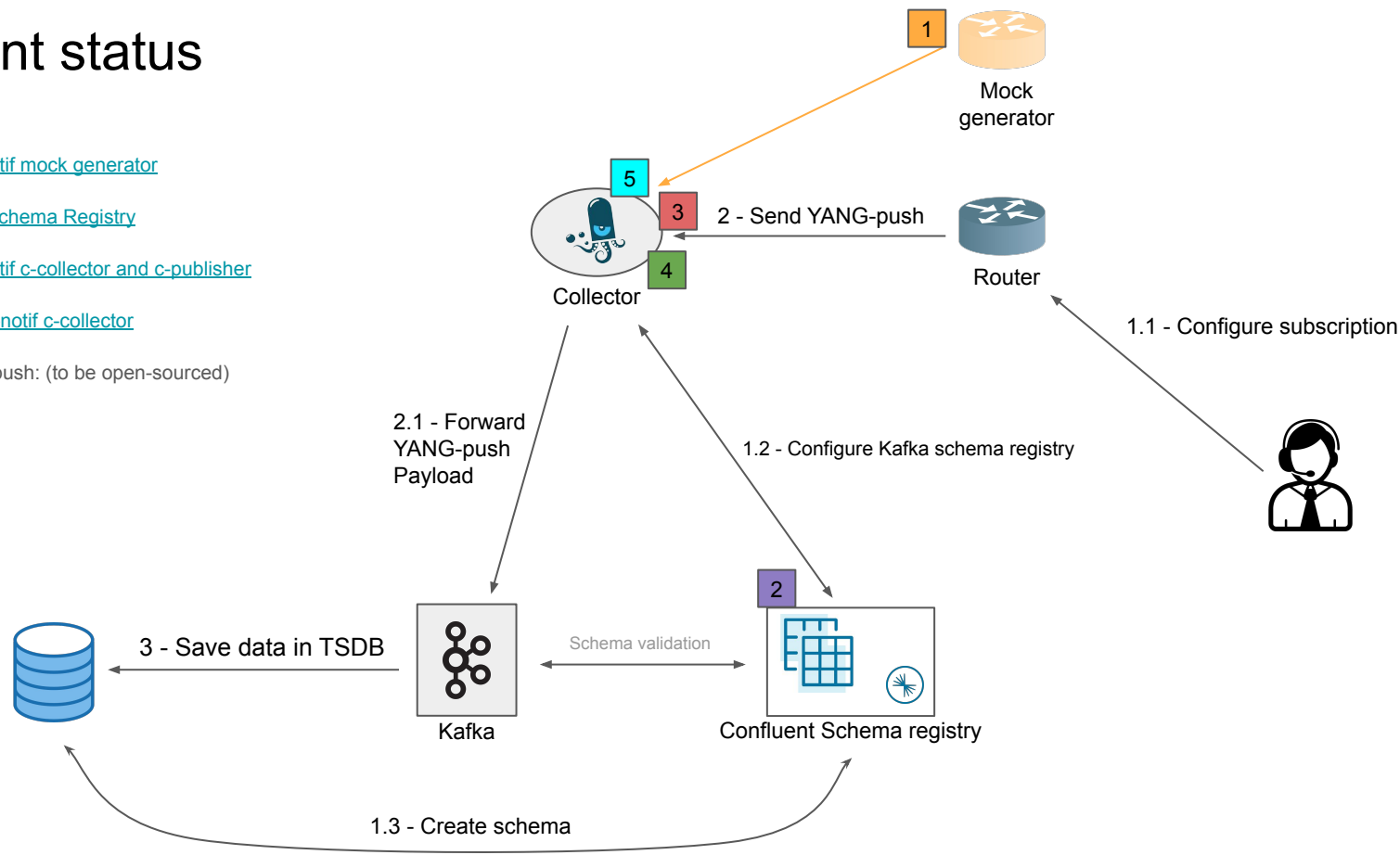
# Missing pieces

libyangserdes



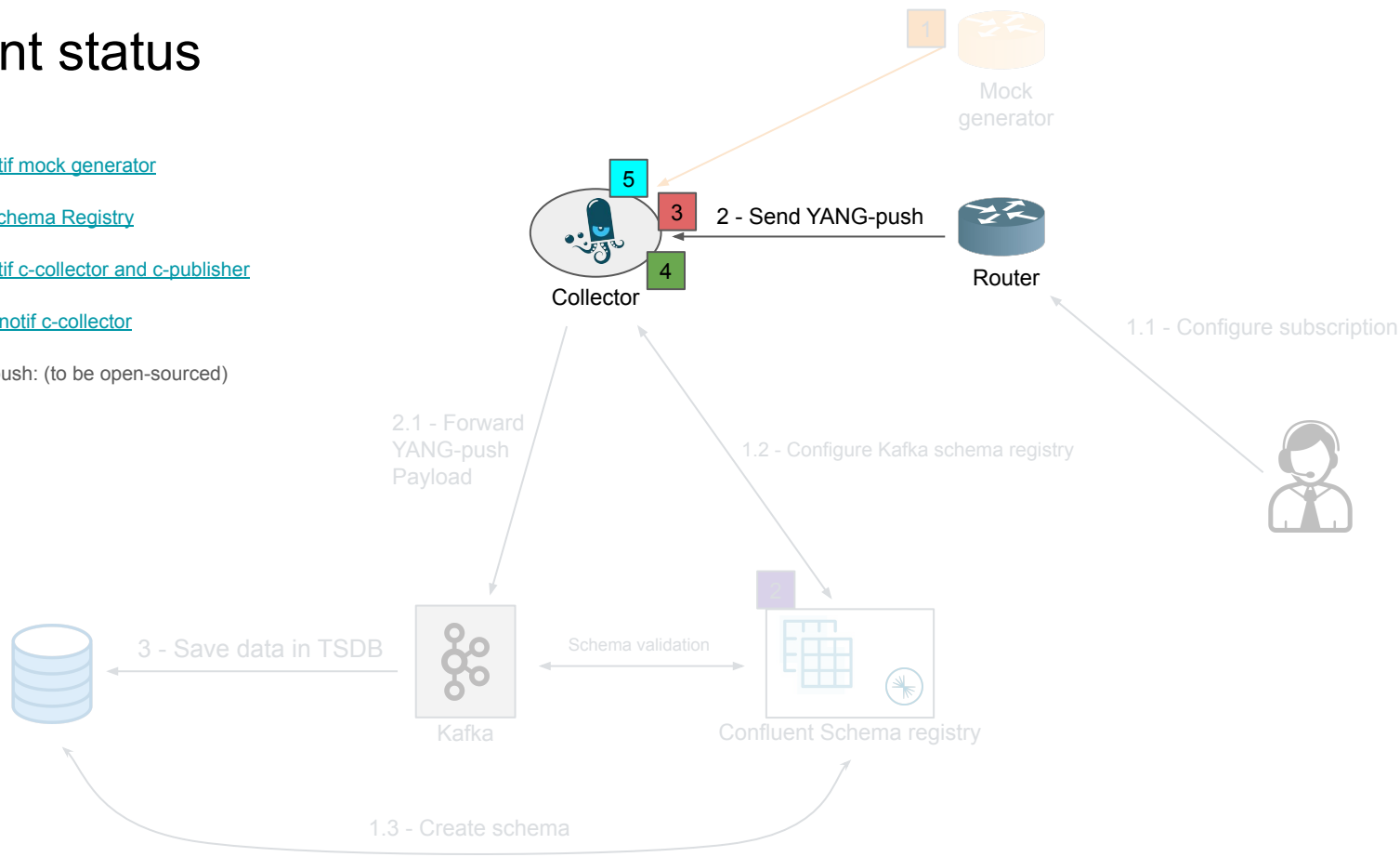
# Current status

- 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)

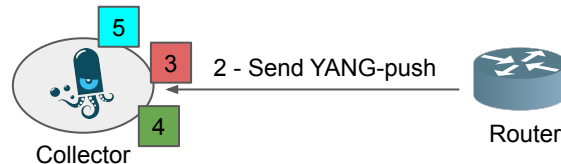


# Current status

- 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)



## Current status: router - collector



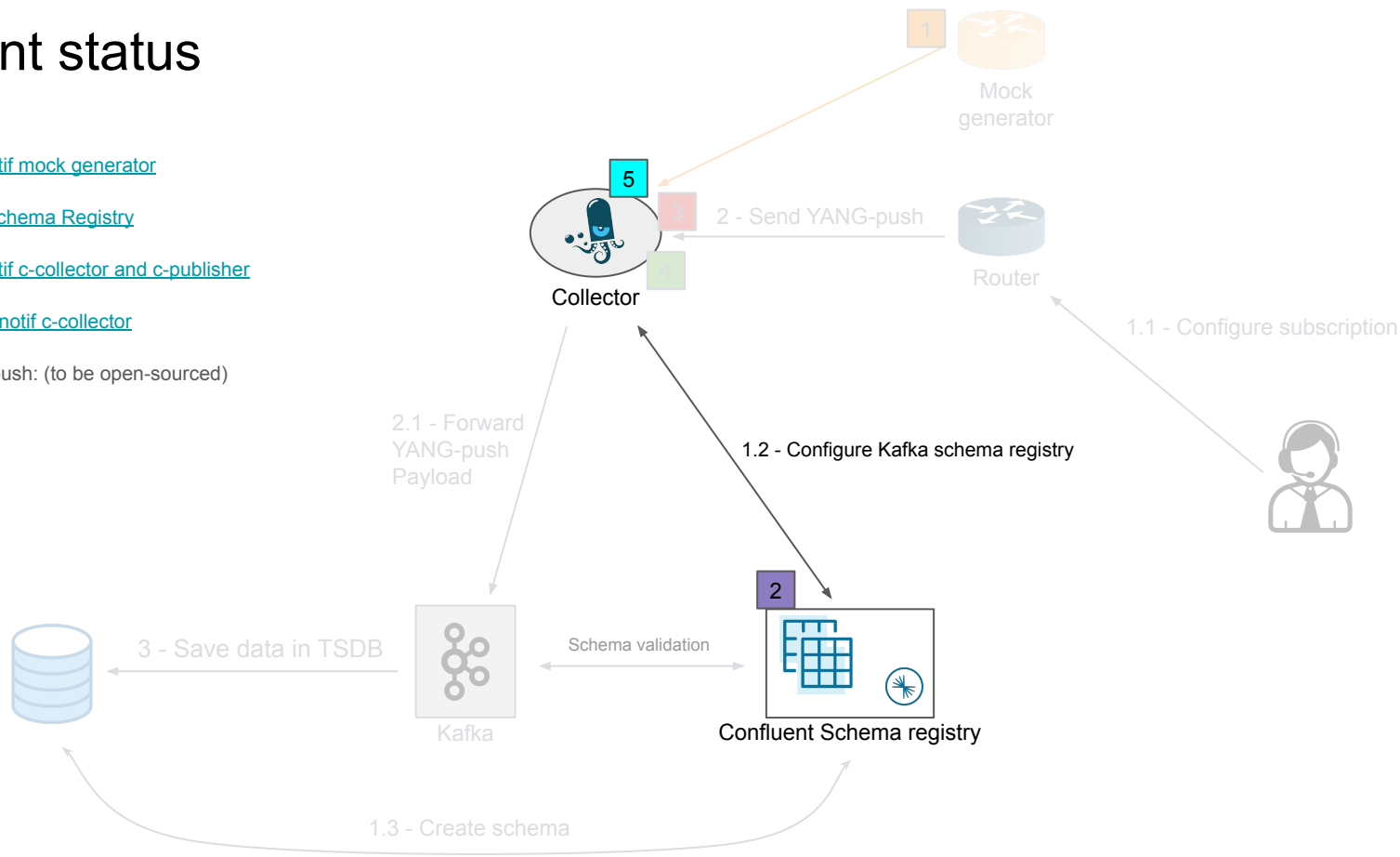
- Router
  - able to craft UDP-notif messages using: [UDP-notif c-collector and c-publisher](#)
- Collector
  - able to collect UDP-notif messages using: [UDP-notif c-collector and c-publisher](#)
  - able to collect HTTPS-notif messages using: [HTTPS-notif c-collector](#)
  - able to get YANG modules 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*

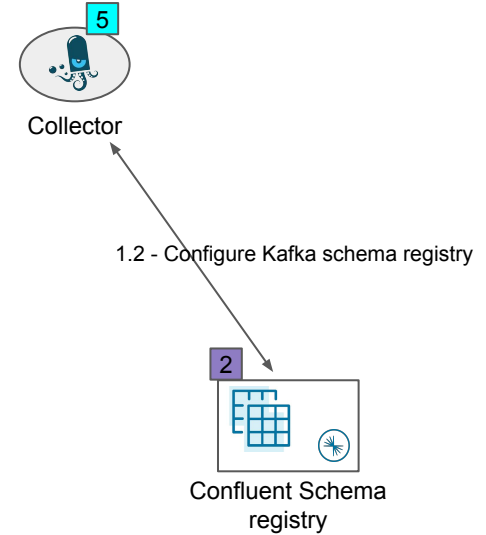
# Current status

- 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)



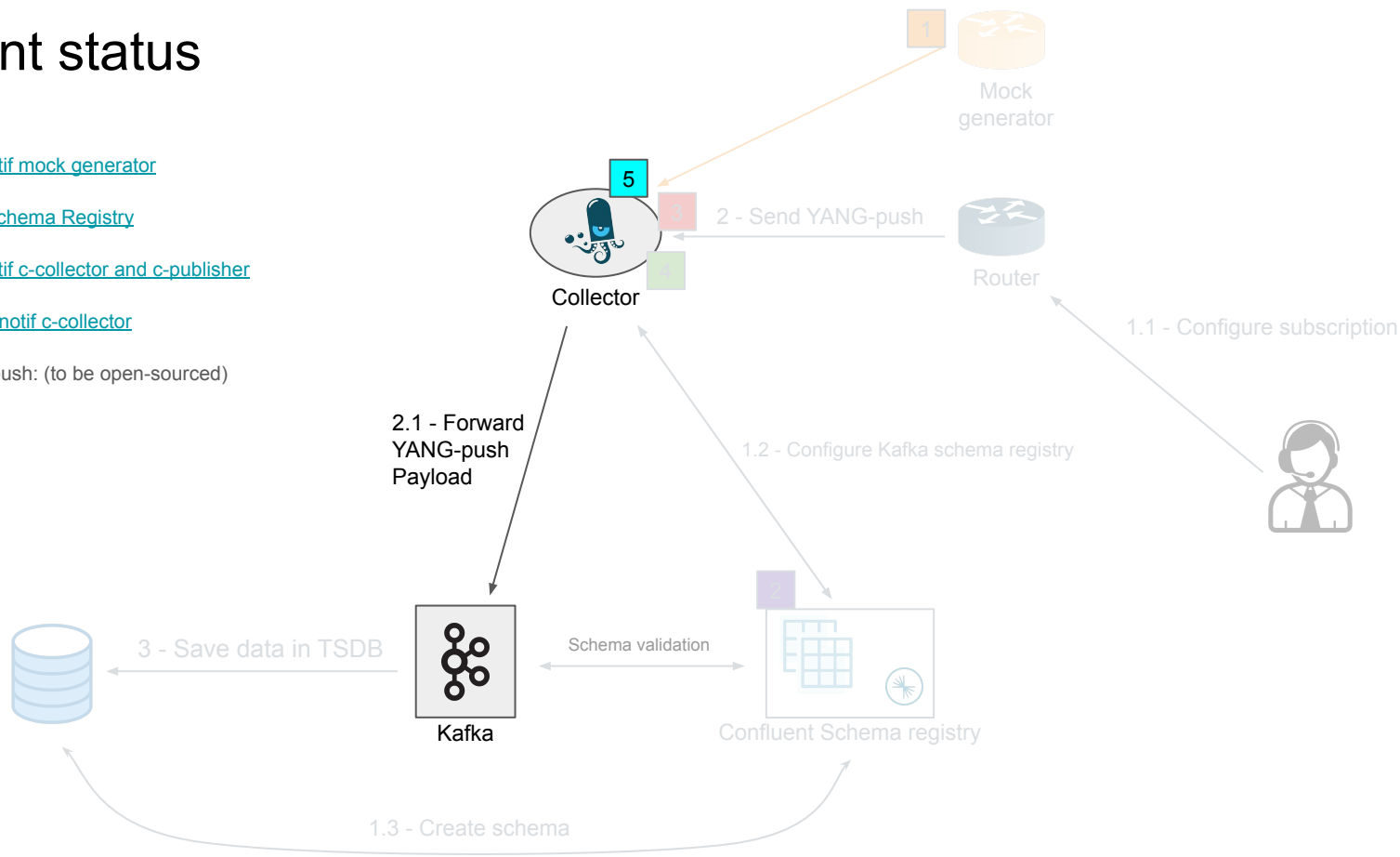
# Current status: collector - schema registry

- 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*



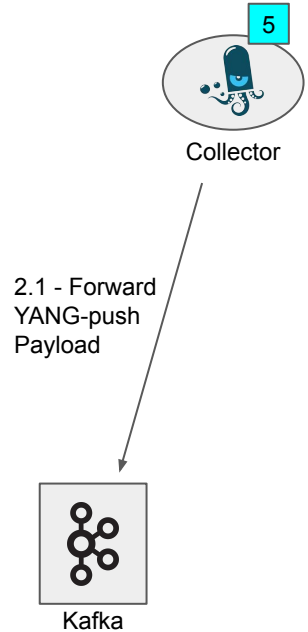
# Current status

- 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)



# 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



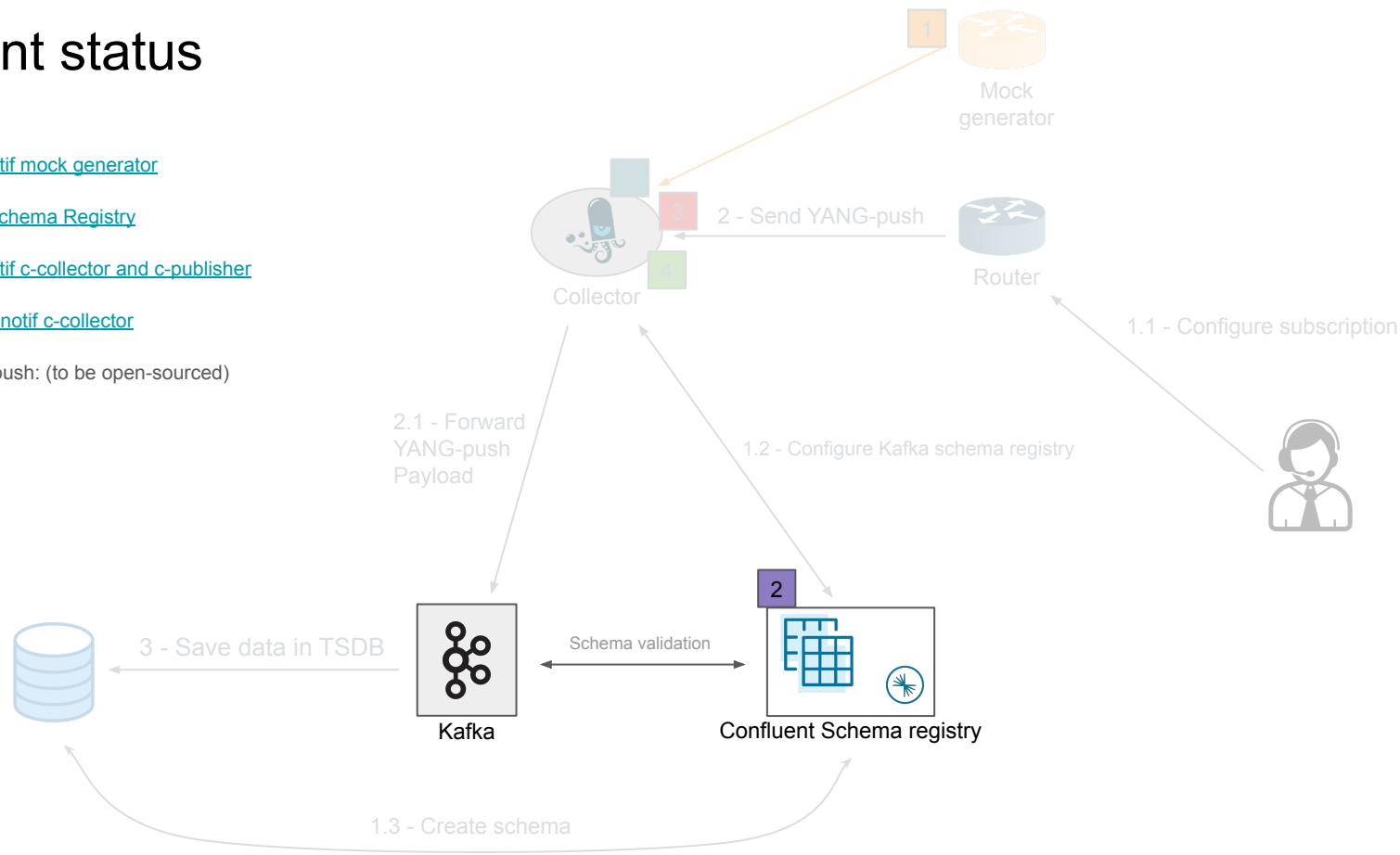
# libyangserdes

- 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

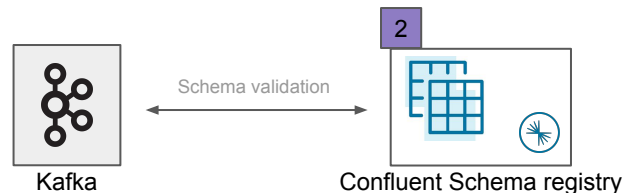


# Current status

- 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)



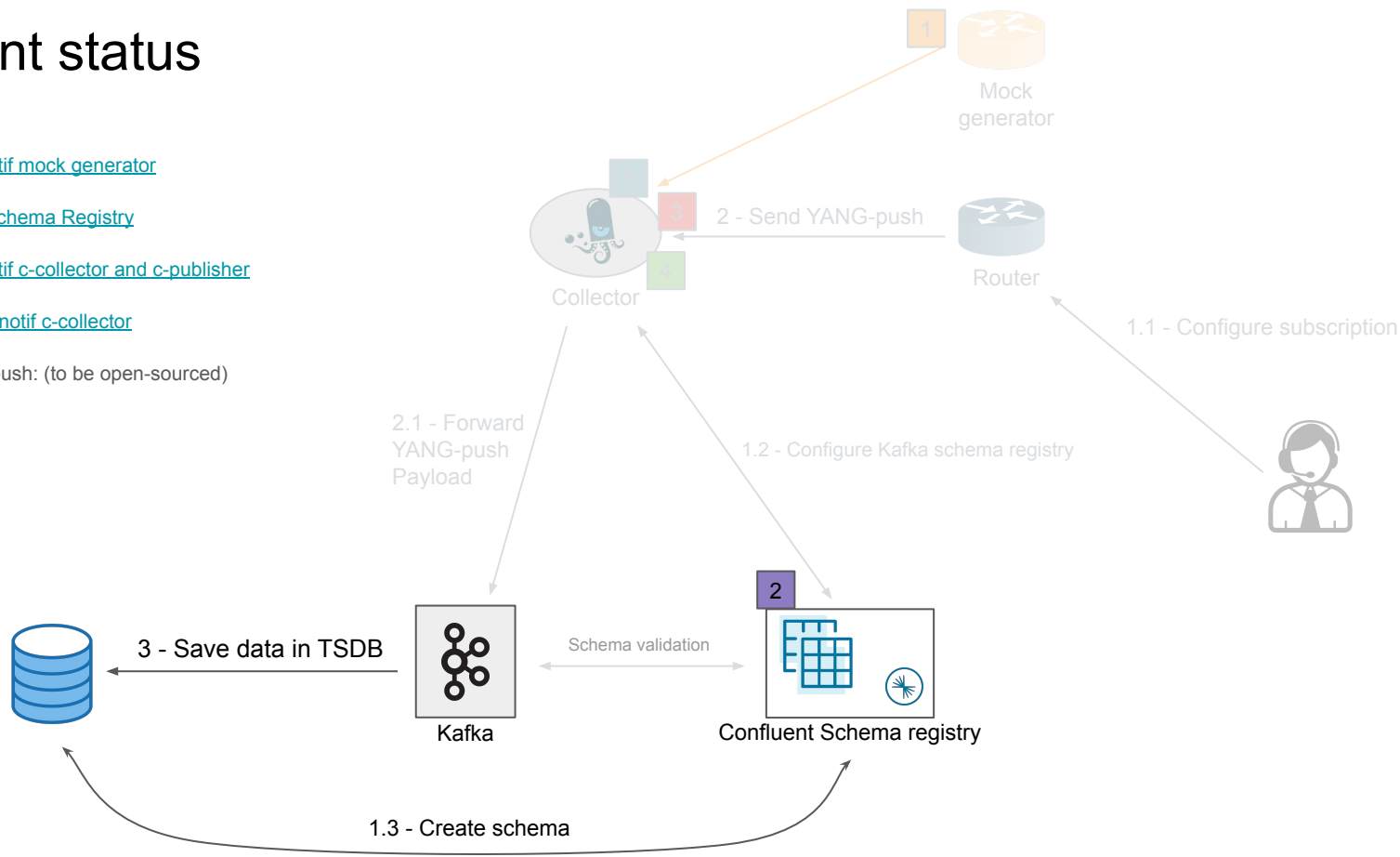
# Current status: Kafka - Schema Registry



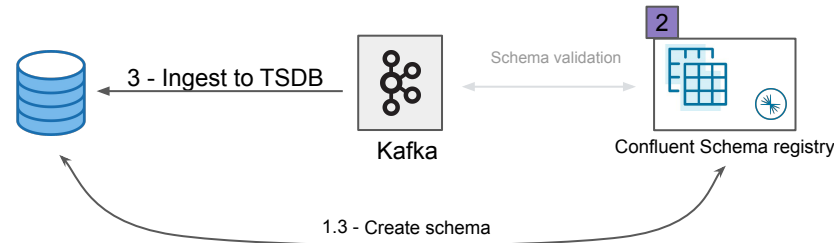
- Kafka
  - is able to get the Schema\_id from the message
  - is able to request the schema from the Schema Registry and validate the content
  - WIP: YANG validation with Yangkit
- Confluent Schema Registry
  - is able to provide the schema / schema context
  - WIP: Current discussions if current API 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)



# 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