NSDL e-Gov

Beckn Level 1 Integration

Adaptor Architecture

Version-1.0 - July 2021

What is Beckn Adaptor?

Beckn Adaptor is a jar file which helps to adapt and comply with beckn specification for a given service. It’s a platform where all the core services like search, on_search, select, on_select, init, on_init… etc originates. Below are operation performed by adaptor:

✓It ensures that beckn specification is followed for each request-response that takes place between BAP and BPP.

✓It converts request(from BAP) and response(from BPP) in the beckn specific json format.

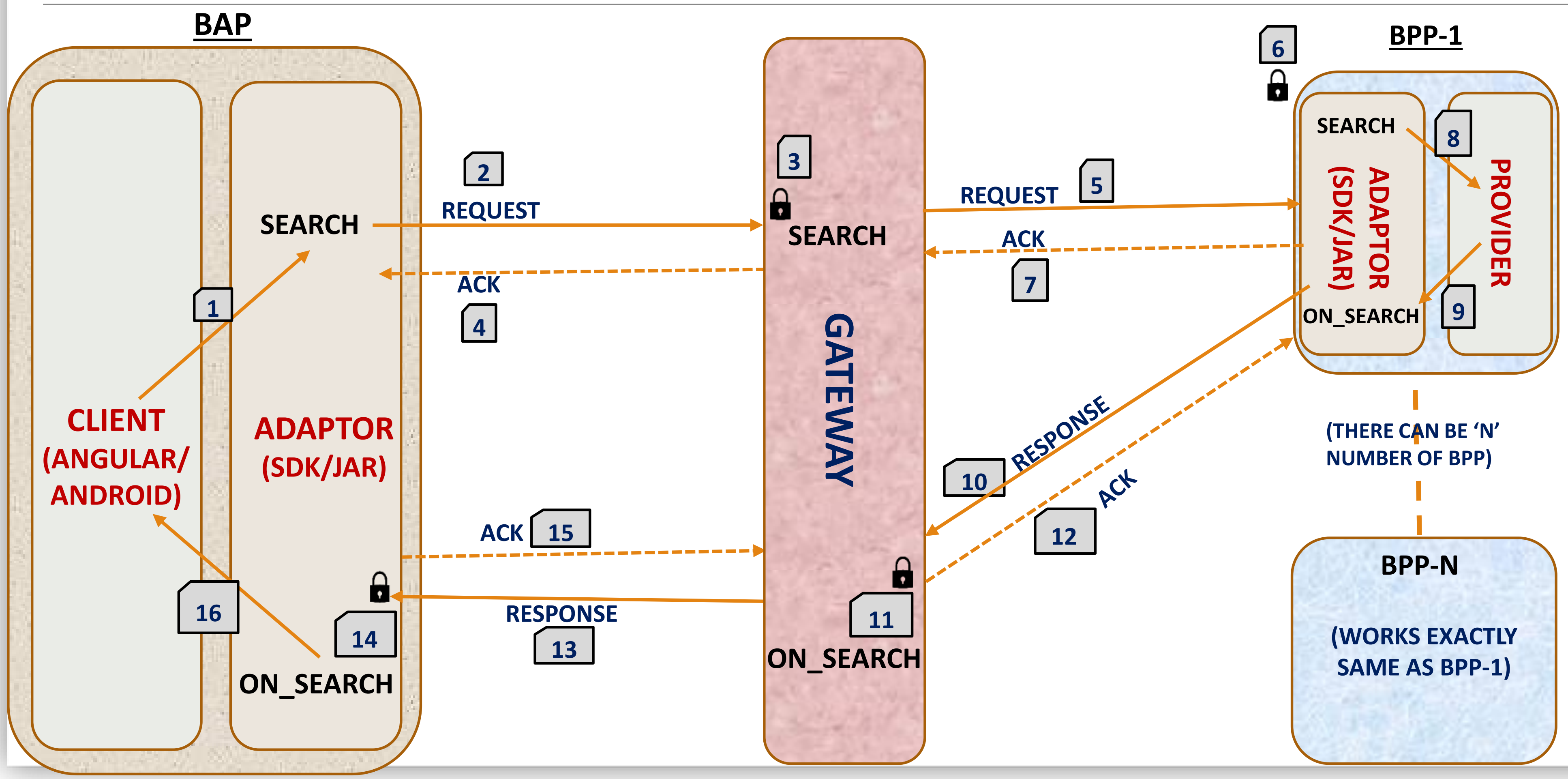
✓It adds required http headers for each request made.

✓It authenticate each incoming request and provide acknowledgement depending on the validation status.

✓It has built-in caching functionality which can be switched depending on requirement.

✓It can also store the response in database(phase-2) or redirect response to callback.

Adaptor architecture work flow: search/on_search

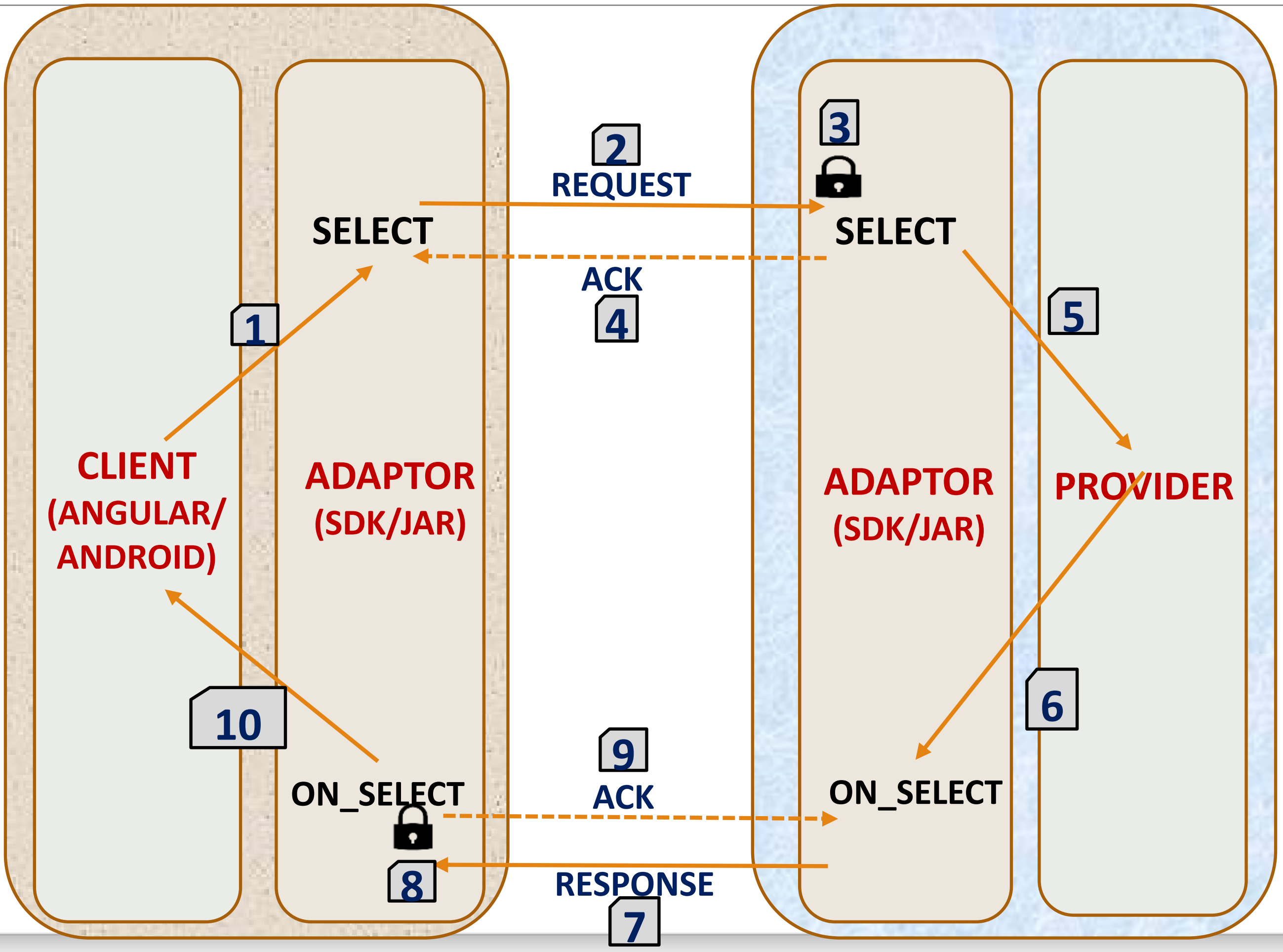


NSDL e-Gov

Adaptor architecture work flow: search/on_search

1. Client makes search call to adaptor.
2. Adaptor converts the message into json in accordance with beckn specification, adds required auth headers and then makes post call to search api of the gateway.
3. Gateway receives the request and authenticate it by validating the headers.
4. Depending on the result of authentication, required acknowledgement is send to the caller(BAP Adaptor).
5. If authentication is ok, then the gateway passes the request to adaptor on bpp side.
6. The adaptor receives the request, validates the headers.
7. Adaptor return appropriate acknowledgment response depending the result of header validation .
8. If authentication is ok, then the adaptor passes the request to actual api in provider side.
9. Provider receives the request, process it and then sends the response to adaptor.
10. Adaptor receives the final response, adds auth headers to it and then calls the on_search of gateway.
11. Gateway receives the request and authenticate it by calling the lookup to registry. The required data for authentication is collected from the headers.
12. Depending on the result of authentication, required acknowledgement is send to the caller(BPP Adaptor).
13. If authentication is ok, then the gateway passes the response to bap adaptor.
14. Bap adaptor receives the request and validates the headers.
15. Adaptor return appropriate acknowledgment response to gateway depending the result of header validation .
16. If authentication is ok, then the adaptor passes the final on_search response to bpp.

Adaptor architecture work flow: select/on_select



Adaptor architecture work flow: select/on_select

1. Client makes post call to adaptor.
2. Adaptor converts the message into json in accordance with beckn specification, adds required auth headers and then makes post call to the adaptor in the BPP side.
3. Adaptor in the BPP side receives the request and authenticate it by calling the lookup to registry. The required data for authentication is collected from the headers.
4. Depending on the result of authentication, required acknowledgement is send to the caller(BAP Adaptor).
5. If authentication is ok, then the adaptor passes the request to actual api in provider side by post call.
6. Provider receives the request, process it and then sends the response to adaptor.
7. Adaptor receives the final response, adds auth headers to it and then calls the required api in BAP side.
8. Adaptor in the BAP receives the request and authenticate it by calling the lookup to registry. The required data for authentication is collected from the headers.
9. Depending on the result of authentication, required acknowledgement is send to the caller(BPP Adaptor).
10. If authentication is ok, then the adaptor passes the final response to client.

How to configure the Adaptor?

There is an Adaptor for each core beckn service like search, select, init, etc. The callback api's like on_search, on_select, on_init are bundled together with core api in the same jar. The adaptors are very flexible. For a given api, the same jar file can act as adaptor for both BAP or BPP depending on configuration that needs to be provided in the jar file.

Before starting the jar, configure these 2 files correctly:

1. application.yml
2. adaptor-config-bap.json / adaptor-config-bpp.json

Configuring the Adaptor for BAP

Below are the Configurations that need to be provided depending on the entity type BAP:

1) application.yml

```
a) server:
    port: 8080
b) beckn:
    persistence:
      type: http|db-postgres
    entity:
      type: bap
```

Description :

1) port: it is server port number on which this jar will run
2) beckn.persistence.type: the pipe separated value for persistence strategy. Currently allowed values are http & db-postgres. Value http means response will be pushed to the url mentioned in the `http_entity_endpoint` parameter. If db-postgres used then response will be saved in database. Any other value is not allowed.
3) beckn.entity.type: the allowed values are bap or bpp. Depending on the value provided, jar will all auto configuration internally and start working accordingly . Any other value is not allowed.

Configuring the Adaptor for BAP

2) adaptor-config-bap.json

```
{
        "keyid": “xxx",
        "callbackurladaptor": “xxx",
        "callbackurlapi": “xxx",
        "privatekey": “xxx",
        "algo": "ed25519",
        "timeout": 1000,
        "retrycount": 3,
        "authenticate": true
}
```

Description:

1) keyid: it is the id that is used while registering as BAP to beckn.
2) callbackurladaptor: it is the call back url on which the BPP will send the response.
3) callbackurlapi: it is the url of BAP to which the adaptor will finally send the response of callback. If not provided the callback response will end in adaptor.
4) privatekey: it is the private key of the BAP. This will be used while signing the authorization header.
5) algo: it is the algorithm used while signing the authorization header. Should be ed25519.
6) timeout: it is the timeout in milliseconds used while making post call to BPP.
7) retrycount: number of time the retry should occur incase of timeout.
8) authenticate: check for verification of authorization header. It takes only boolean value.

Configuring the Adaptor for BPP

Below are the Configurations that need to be provided depending on the entity type BPP:

1) application.yml

```
a) server:
        port: 8080
b) beckn:
        persistence:
            type: http|db-postgres
        entity:
            type: bpp
```

Description :

1) port: it is server port number on which this jar will run
2) beckn.persistence.type: the pipe separated value for persistence strategy. Currently allowed values are http & db-postgres. Value http means response will be pushed to the url mentioned in the `http_entity_endpoint` parameter. If db-postgres used then response will be saved in database. Any other value is not allowed.
3) beckn.entity.type: the allowed values are bap or bpp. Depending on the value provided, jar will all auto configuration internally and start working accordingly . Any other value is not allowed.

NSDL e-Gov

Configuring the Adaptor for BPP

2) adaptor-config-bpp.json

```
{
        "keyid": “xxx",
        "callbackurladaptor": “xxx",
        "callbackurlapi": “xxx",
        "privatekey": “xxx",
        "algo": "ed25519",
        "timeout": 2000,
        "retrycount": 3,
        "authenticate": true
}
```

Description:

1) keyid: it is the id that is used while registering as BPP to beckn.
2) callbackurladaptor: it is the call back url on which the BPP will send the response.
3) callbackurlapi: it is the url of BPP to which the adaptor will forward the request after validation.
4) privatekey: it is the private key of the BPP. This will be used while signing the authorization header.
5) algo: it is the algorithm used while signing the authorization header. Should be ed25519.
6) timeout: it is the timeout in milliseconds used while making post call to BAP as part of callback.
7) retrycount: number of time the retry should occur incase of timeout.
8) authenticate: check for verification of authorization header. It takes only boolean value.

NSDL e-Gov

Notes and Sample files:

1) Allowed api name are search, select, cancel, confirm, init, rating, status, support, track, update

Sample files:

adaptor-config-bpp.json file:

```
{
        "keyid": "nsdl.co.in|nsdl_bpp_1",
        "private_key": "xxxxx_xxxxx_xxxxx_xxxxx",
        "api": [
                {
                        "name": "search",
                        "http_entity_endpoint": "http://localhost:8082/bpp/mock/search",
                        "http_timeout": 1000,
                        "http_retry_count": 3,
                        "header_validity": 600000,
                        "header_authentication": true
                },
                {
                        "name": "select",
                        "http_entity_endpoint": "http://localhost:8082/bpp/mock/select",
                        "http_timeout": 1000,
                        "http_retry_count": 3,
                        "header_validity": 600000,
                        "header_authentication": true
                },
                {
                        "name": "cancel",
                        "http_entity_endpoint": "http://localhost:8082/bpp/mock/cancel",
                        "http_timeout": 1000,
                        "http_retry_count": 3,
                        "header_validity": 600000,
                        "header_authentication": true
                },
{
        "name": "confirm",
        "http_entity_endpoint": "http://localhost:8082/bpp/mock/confirm",
        "http_timeout": 1000,
        "http_retry_count": 3,
        "header_validity": 600000,
        "header_authentication": true
},
{
        "name": "init",
        "http_entity_endpoint": "http://localhost:8082/bpp/mock/init",
        "http_timeout": 1000,
        "http_retry_count": 3,
        "header_validity": 600000,
        "header_authentication": true
},

{
        "name": "rating",
        "http_entity_endpoint": "http://localhost:8082/bpp/mock/rating",
        "http_timeout": 1000,
        "http_retry_count": 3,
        "header_validity": 600000,
        "header_authentication": true
},
{
        "name": "status",
        "http_entity_endpoint": "http://localhost:8082/bpp/mock/status",
        "http_timeout": 1000,
        "http_retry_count": 3,
        "header_validity": 600000,
                "header_authentication": true
            },
            {
                "name": "support",
                "http_entity_endpoint": "http://localhost:8082/bpp/mock/support",
                "http_timeout": 1000,
                "http_retry_count": 3,
                "header_validity": 600000,
                "header_authentication": true
            },
            {
                "name": "track",
                "http_entity_endpoint": "http://localhost:8082/bpp/mock/track",
                "http_timeout": 1000,
                "http_retry_count": 3,
                "header_validity": 600000,
                "header_authentication": true
            },
            {
                "name": "update",
                "http_entity_endpoint": "http://localhost:8082/bpp/mock/update",
                "http_timeout": 1000,
                "http_retry_count": 3,
                "header_validity": 600000,
                "header_authentication": true
            }
        ]
}
```