ELIXIR Beacon Network

Draft Model and API considerations

Jordi Rambla

Sabela de la Torre

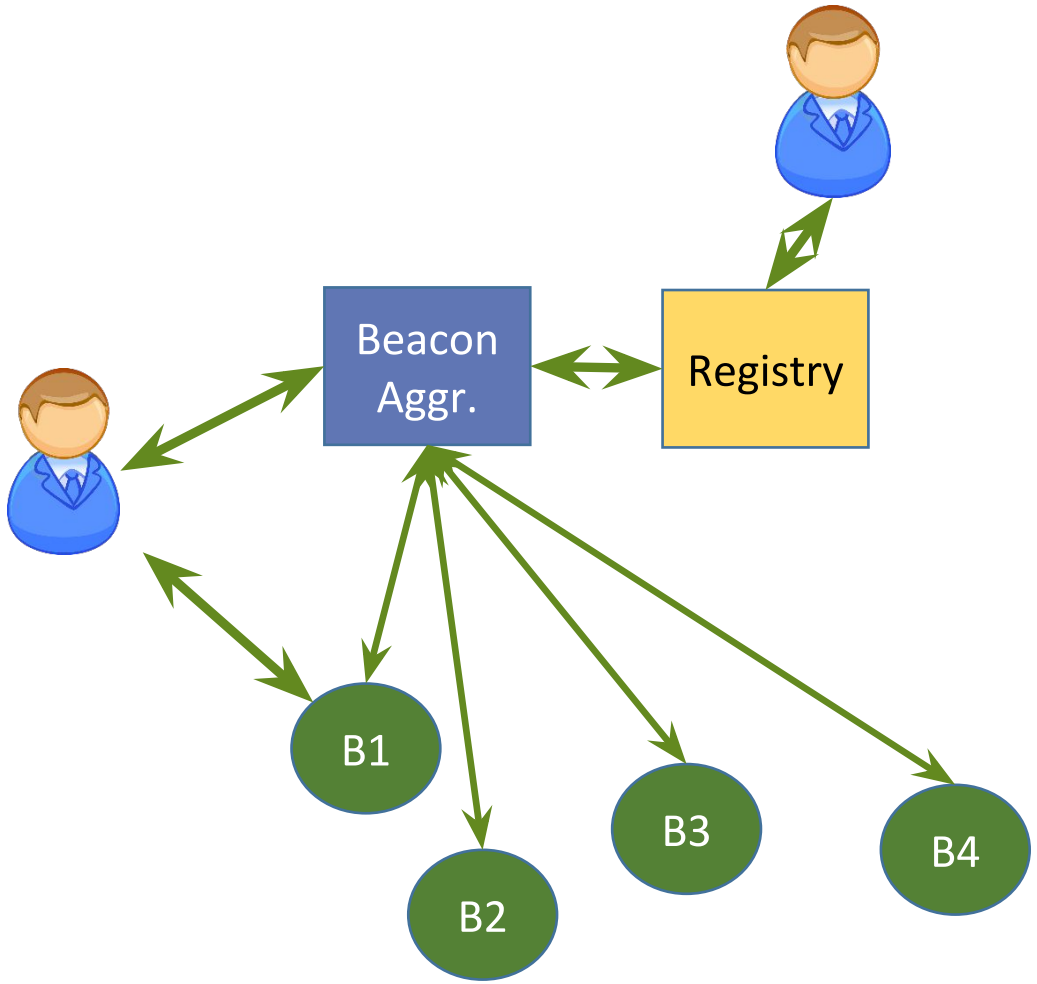
www.elixir-europe.org

Requirements

- Flexible enough for
  - Open, peer to peer networks
  - Closed networks, hierarchy
- Secure
  - *not really covered in this presentation, but very relevant*

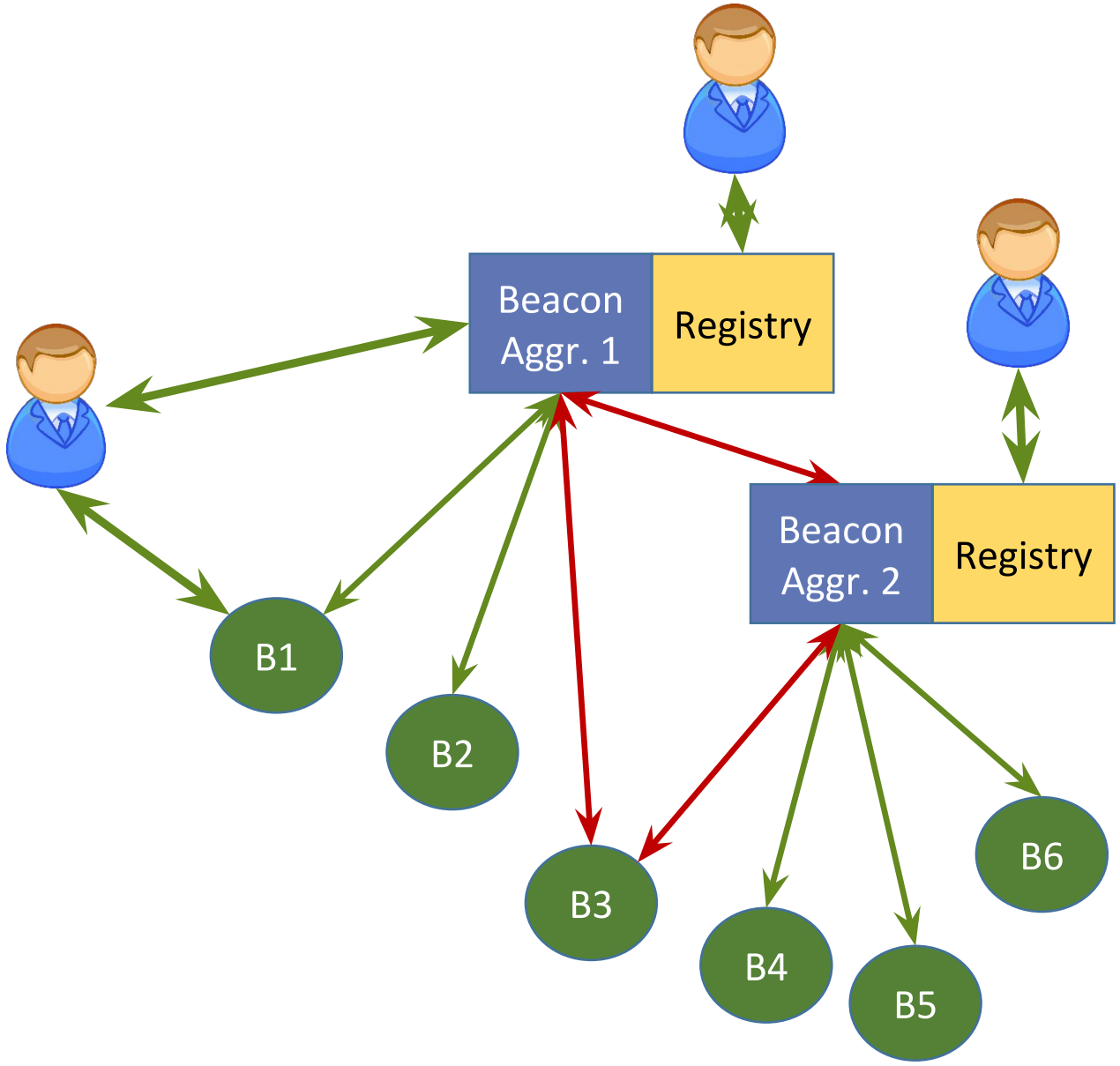
Current model

- ~Beacon Network v0.1
- BN has both Registry and BA roles
- Any user can query any Beacon
  - via the BN or directly to the Beacon
- Registry is manually curated
  - Differences in Beacon API version implemented are dealt at BA



Current model “issues” (adding a 2nd Beacon Network)

- No control of a Beacon that is a BA itself
  - Potential duplication of results
  - A BN head couldn’t recognize if it is querying another network
- Closed networks aren’t possible
- Manual curation of the Registry
- Almost impossible to detect re-identification attacks via distribution of queries (*e.g. querying B3 throught BA1, BA2 and B3 itself*)



Spotted Roles / Service types

- Beacon (example URL: *http://mycohort.org/beacon/v1/*)
  - Manages datasets, accept queries, provide responses
- Beacon Aggregator (e.g. *http://mycohort.org/beacon-network/v1/info*)
  - Accept queries, redirects queries to Beacon services, provide responses aggregating Beacon services responses
  - Does not manage datasets
  - Requires a list of Beacons to query
- Registry (e.g. *http://mycohort.org/registry/v1/*)
  - Manages lists of services

Simple Registry API *draft*

- GET /list -- lists services/nodes by type
- GET /nodes/{id} -- list a service/node details
- POST /nodes
  - Including Beacon info
    - /info
    - /datasets
- PUT /nodes/{id}
- DELETE /nodes/{id}
- GET /servicetypes – enumerator of accepted service types by the registry

Adding a Registry adds new “issues”

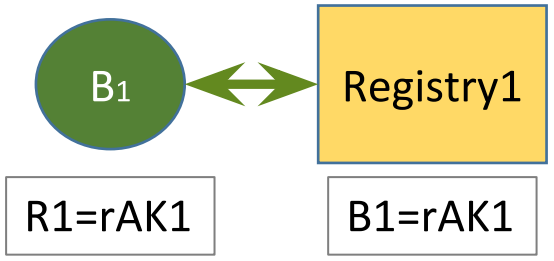
- Adding, updating or removing a Beacon from a Registry could not be performed w/o Authentication
- Doing these CRUD operations automatically means no “real-world user” login should be required
- We are considering using an “Appkey approach” for this functionality

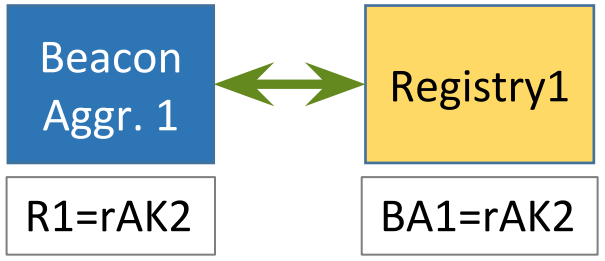
Registry API *draft*

- GET /list -- lists services/nodes by type
- GET /nodes/{id} -- list a service/node details
- POST /nodes
  - AuthN/Z -- manually generated Appkey (shared secret from this point on)
  - Including
    - Beacon info: /info /datasets
    - Appkey at HTTP Request Header (https required)
    - *Currently missed:*
      - *Is authentication supported/required?*
      - *Which access levels (P, R, C) are supported?*
- PUT /nodes/{id}
  - AuthN/Z -- Appkey (shared secret)
  - Info like in POST
- DELETE /nodes/{id}
  - AuthN/Z -- Appkey (shared secret)
- GET /servicetypes – enumerator of accepted service types by the registry

Registration process and Appkey generation

- Beacon1 (B1) wants to register at Registry1 (R1)
  - RegisterAppKey (rAK1) is generated manually or programmatically
  - Both B1 and R1 store rAK1
- BeaconAggregator1 (BA1) wants to register at Registry1 (R1)
  - RegisterAppKey (rAK2) is generated manually or programmatically
  - Both BA1 and R1 store rAK2
- The RegisterAppKey is required at any subsequent communication





Potential syntaxes

```
Beacon1 {
   Registries {
      Registry1=asf3qs03
   }
}
```

```
BeaconAggregator1 {
   Registries {
      Registry1=f3as03qs
   }
}
```

Registry Automatic Curation

- PULL
  - A heartbeat gathers info about the Beacon info (version, etc.)
- PUSH
  - Each Beacon can update itself the registries where it is enlisted
- *Both models could co-exist*

Extensions to the Beacon API

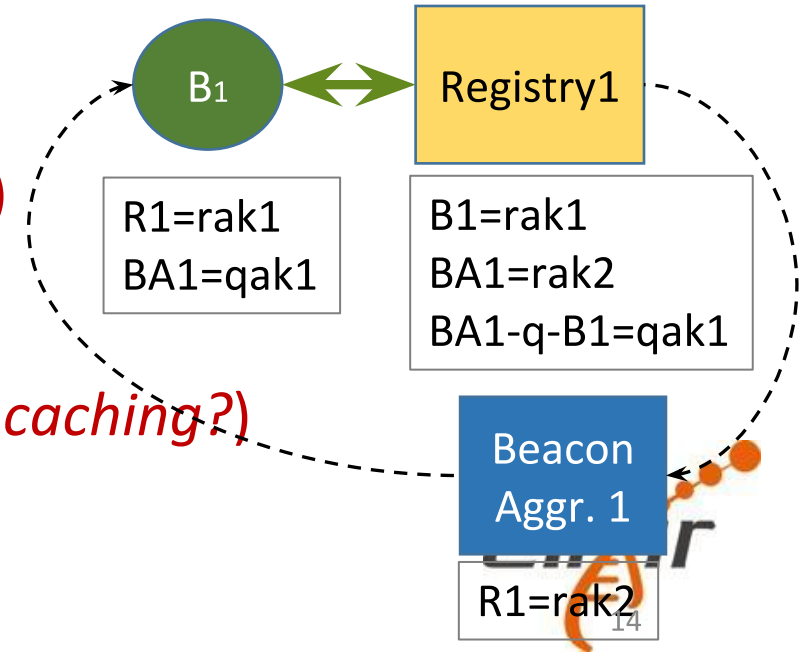
- Additional requirements on top of regular Beacon duties
  - To know who should be updated when changes happen
  - To know which Appkey to use for each service interaction
    - For Registries
    - For Beacon Aggregators (*see later*)
- GET /networks – lists all the networks / registries the Beacon is enlisted in
- GET /networks/{id} – details on a given network/registry

If…

- we already have a “closed” conversation mechanism between Beacon and Registry, why not leverage it for “closing” conversations at *query time*?

Registration process and Appkey generation

- Beacon1 (B1) wants to register at Registry1 (R1)
  - Manual or automatic RegisterAppKey (rAK1) is generated
  - Both B1 and R1 store rAK1
- BeaconAggregator1 (BA1) wants to register at Registry1 (R1)
  - Manual or automatic RegisterAppKey (rAK2) is generated
  - Both BA1 and R1 store rAK2
- B1 wants to accept queries *only* from BA1
  - B1 asks BA1 for its registries (*master, preferred, all???*)
  - B1 posts/gets a QueryAppKey (q=qAK1) from R1
  - Both B1 and R1 store qAK1, linked to BA1
  - BA1 gets qAK1 from R1 when querying is needed (*add caching?*)



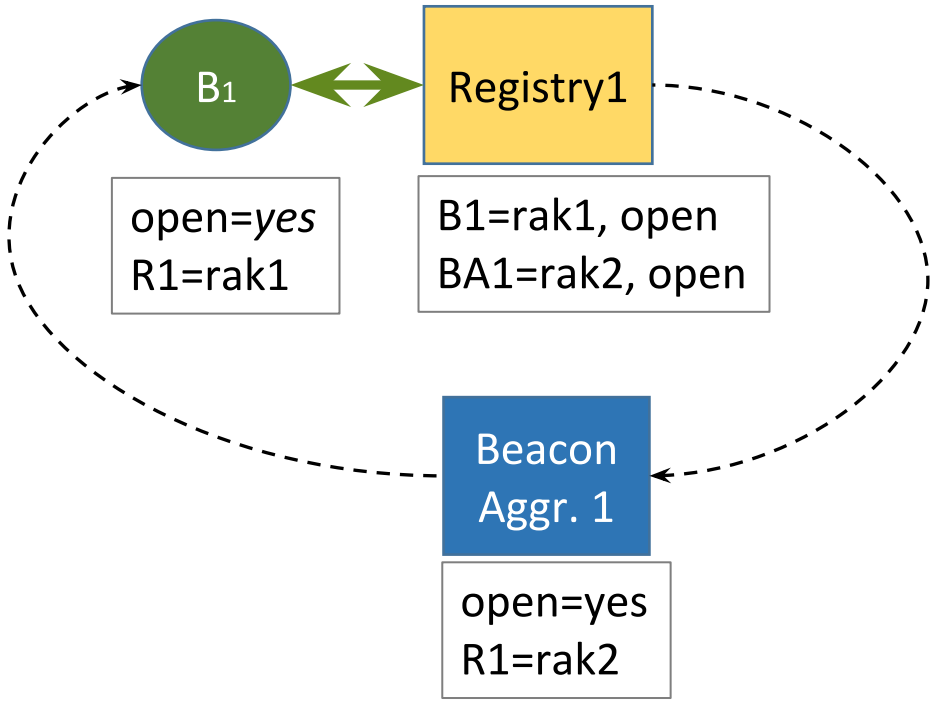
Potential syntaxes, *including openness*

```
Beacon1 {
    open = no,
    Registries {
        Registry1=asf3qs03
        },
    Requesters {
        BeaconAggregator1=3d4fa,
        BeaconAggregator2=f4ggf7
        }
    }
```

```
BeaconAggregator1 {
    open = yes,
    Registries {
        Registry1=asf3qs03
        },
    Querying {
        Beacon1=3d4fa
        }
    }
```

What if we want open queries?

- Beacon1 (B1) registers at Registry1 (R1)
  - States openness with an specific syntax/statement, e.g.:
    - B1 = open *or*
    - B1 = closed *(adding explicit keys)*
      - BA1=qak1
      - BA2=qak2
  - Better avoid making it implicit at Beacon, as errors could let to “null” responses and thus opening a Beacon to any requester
    - *~rule of explicit authorization*

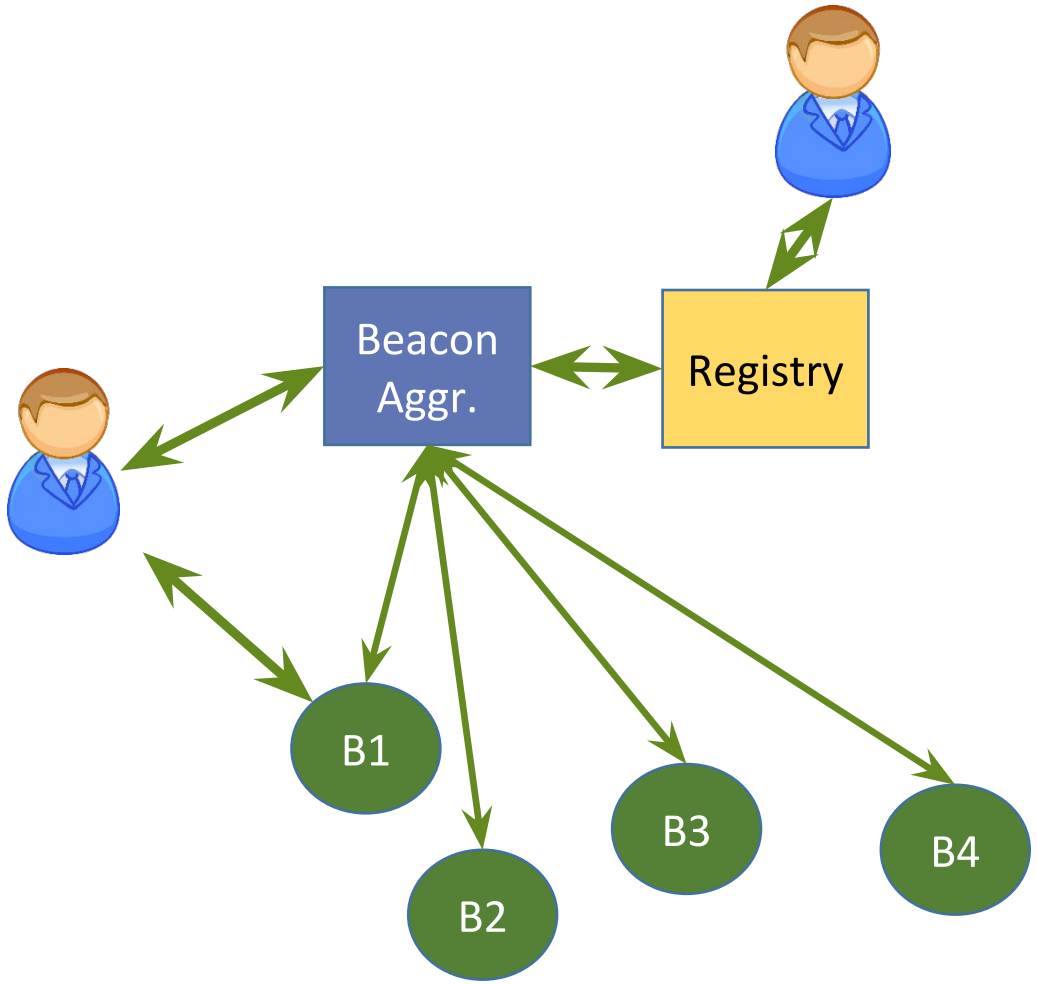


Scenarios

- GA4GH like network
- Closed network
- Peer-to-peer network
- Hierarchical network

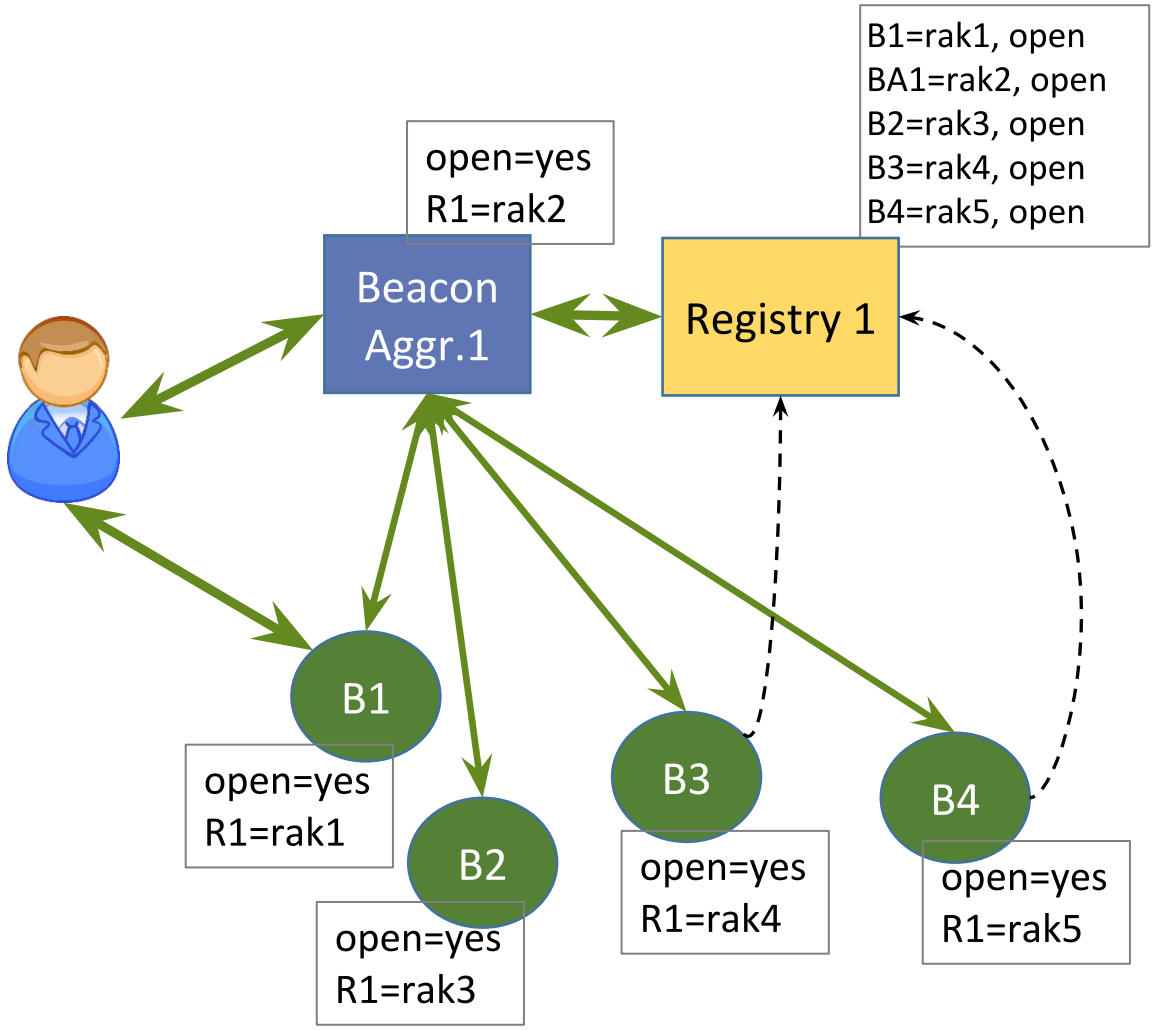
Scenario 1: original

- GA4GH Beacon Network v0.1 (DNAStack)
- BN has Registry and BA roles
- Any user can query any Beacon
  - via the BN or directly to the Beacon
- Registry is manually curated



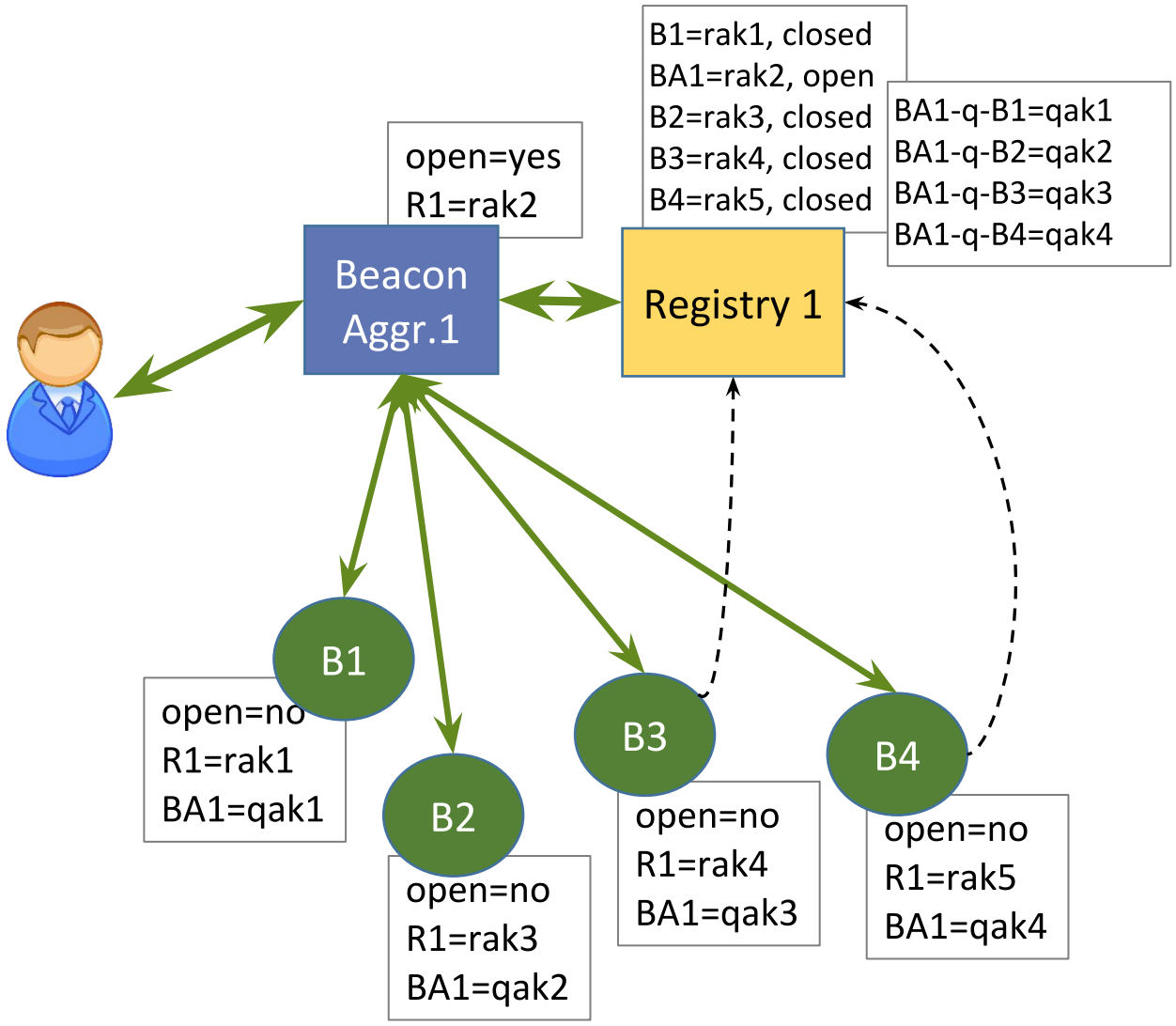
Scenario 1: Elixir BN version

- (following ELIXIR Beacon Network spec)
- BN has Registry and a BA roles
- Any user can query any Beacon
- Registry could be automatically curated



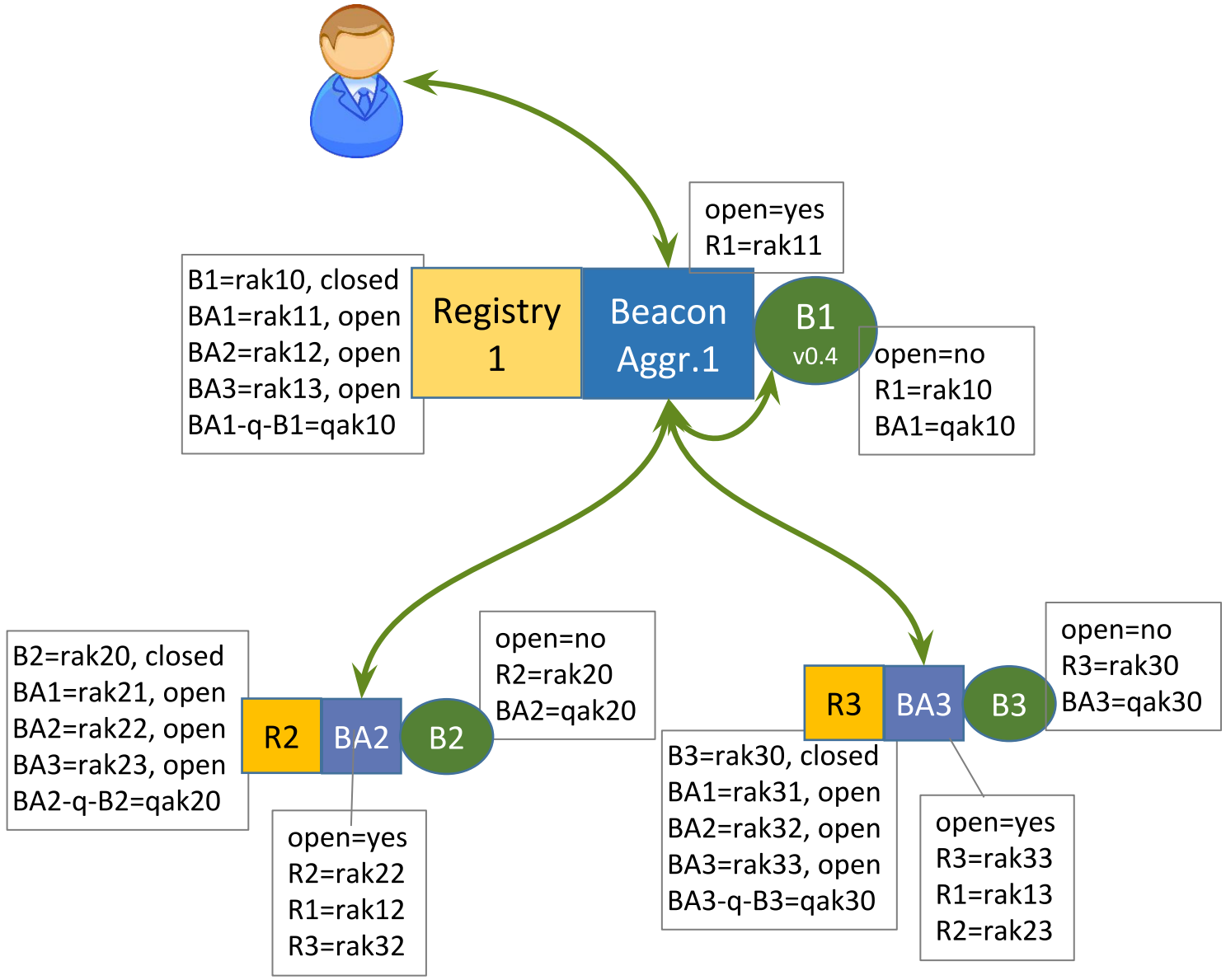
Scenario 2: Closed Network

- BN has Registry and a BA roles
- Beacon only accepts queries from a given Beacon Aggregator
- Beacon /query requires Appkey in the HTTPRequest
  - And at every other endpoint



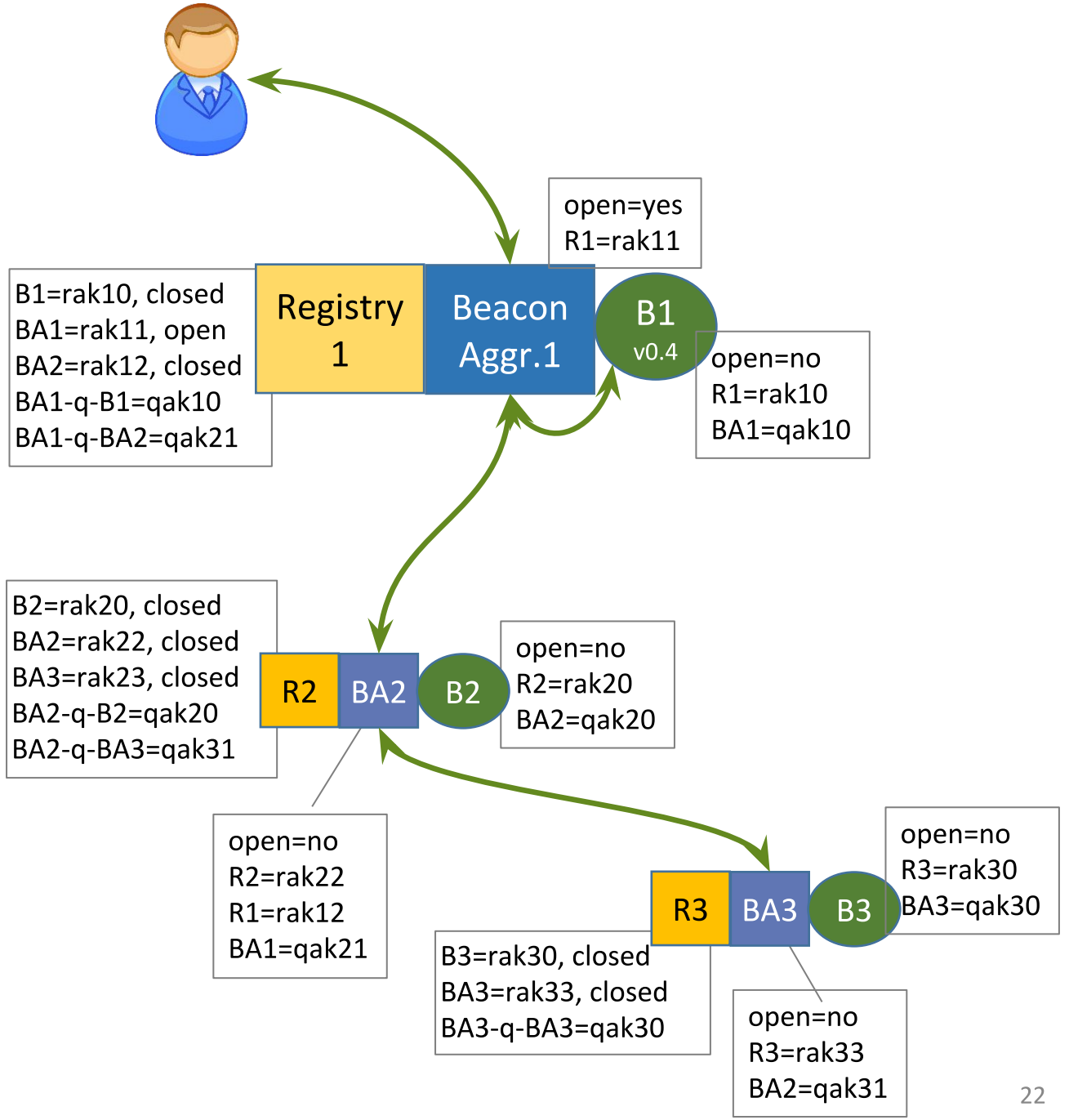
Scenario 3: Peer to peer

- Any user or BA can query any other “Beacon”
- Every site has Registry, BA and B roles
  - As every B should know where other B are and aggregate their responses, they are actually acting as BA
  - *BA wraps the actual Beacon*
- Different possible approaches
  - local Beacon could be open, then should be registered at every registry, which is cumbersome
  - Local Beacon could be *private* to its “wrapping” BA



Scenario 4 (Hierarchical)

- Any BA would only accept queries from *a parent*
- Every site has Registry, BA and B roles
  - As every BA should know where other BA are and aggregate their responses, they are actually acting as BA
  - *BA wraps the actual Beacon*



Beacon Aggregator API considerations

- Equal to extended Beacon API
  - Would also register at a service
  - Needs to know about datasets at each Beacon if it should answer calls to the Beacon API */datasets* endpoint
- Should allow for requests to query just specific/selected beacons from its list?
- Must consolidate the responses from duplicated datasets?
- How to support different AuthX models?
- How to support different access levels?