device-discovery GENERIC-HTTP

Interface Design Description

Abstract
This document describes the **GENERIC-HTTP** service interface of **device-discovery** service, which offers communication via http protocol with JSON payload encoding.

Contents

1 Overview

This document describes the **GENERIC-HTTP** service interface of **device-discovery**, which enables both application and core/support systems to lookup, register and revoke devices on which the Local Cloud's systems are running. Device representation is not necessary for the base functionalities of a Local Cloud but in certain use cases (e.g. enabling onboarding) is needed. It's implemented using protocol, encoding as stated in the following table:

Profile type	Type	Version
Transfer protocol	HTTP	1.1
Data encryption	N/A	-
Encoding	JSON	RFC 8259 [1]
Compression	N/A	-

Table 1: Communication and semantics details

This document provides the Interface Design Description IDD to the *device-discovery – Service Description* document. For further details about how this service is meant to be used, please consult that document.

The rest of this document describes how to realize the **device-discovery** service **GENERIC-HTTP** interface in details.

2 Interface Description

2.1 operation register

The service operation request requires an authorization bearer header and a `DeviceRegistrationRequest` JSON encoded body.

```
POST /serviceregistry/device-registry/register HTTP/1.1
Authorization: Bearer <authorization-info>

{
   "name": "string",
   "metadata": {
      "additionalProp1": "string",
      "additionalProp2": {},
      "additionalProp3": []
   },
   "addresses": [
      "string"
   ]
}
```

Listing 1: A register request.

The service operation responses with the status code `200 Ok` if called successfully and the device entity is already existing or `201 Create` if the entity was newly created. The response also contains a `DeviceRegistrationResponse` JSON encoded body.

```
{
  "name": "string",
  "metadata": {
    "additionalProp1": "string",
    "additionalProp2": {},
    "additionalProp3": []
  },
  "addresses": [
    {
      "type": "string",
      "address": "string"
    }
  ],
  "createdAt": "string",
  "updatedAt": "string"
}
```

Listing 2: A register success response.

The error codes are, `400 Bad Request` if request is malformed, `403 Forbidden` if requester authentication was unsuccessful, `401 Unauthorized` if the authenticated requester has no permission and `500 Internal Server Error` if an unexpected error happens. The error response also contains an `ErrorResponse` JSON encoded body.

```
{
  "errorMessage": "string",
  "errorCode": 0,
  "exceptionType": "string",
  "origin": "string"
}
```

Listing 3: A register error response.

2.2 operation lookup

The service operation request requires an authorization bearer header and may optionally include a `DeviceLookupRequest` JSON encoded body.

```
POST /serviceregistry/device-registry/lookup HTTP/1.1
Authorization: Bearer <authorization-info>

{
  "deviceNames": [
    "string"
  ],
  "addresses": [
    "string"
  ],
  "addressType": "string",
  "metadataRequirementList": [
    {
      "empty": true,
      "additionalProp1": {},
      "additionalProp2": {},
      "additionalProp3": {}
    }
  ]
}
```

Listing 4: A lookup request.

The service operation responses with the status code `200 Ok` if called successfully and with a `DeviceLookupResponse` JSON encoded body.

```
{
  "entries": [
    {
      "name": "string",
      "metadata": {
        "additionalProp1": "string",
        "additionalProp2": {},
        "additionalProp3": []
      },
      "addresses": [
        {
          "type": "string",
          "address": "string"
        }
      ],
      "createdAt": "string",
      "updatedAt": "string"
    }
  ],
  "count": 1
}
```

Listing 5: A lookup success response.

The error codes are, `400 Bad Request` if request is malformed, `403 Forbidden` if requester authentication was unsuccessful, `401 Unauthorized` if the authenticated requester has no permission and `500 Internal Server Error` if an unexpected error happens. The error response also contains an `ErrorResponse` JSON encoded body.

```
{
  "errorMessage": "string",
  "errorCode": 0,
  "exceptionType": "string",
  "origin": "string"
}
```

Listing 6: A lookup error response.

2.3 operation revoke

The service operation request requires an authorization bearer header and a device `name` as path parameter.

```
DELETE /serviceregistry/device-registry/revoke/<name> HTTP/1.1
Authorization: Bearer <authorization-info>
```

Listing 7: A revoke request.

The service operation responses with the status code `200 Ok` if called successfully and an existing device entity was removed and `204 No Content` if no matching entity was found. The success response not contains any response body.

The error codes are, `400 Bad Request` if request is malformed, `403 Forbidden` if requester authentication was unsuccessful, `401 Unauthorized` if the authenticated requester has no permission, `423 Locked` if entity is not removable and `500 Internal Server Error` if an unexpected error happens. The error response also contains an `ErrorResponse` JSON encoded body.

```
{
  "errorMessage": "string",
  "errorCode": 0,
  "exceptionType": "string",
  "origin": "string"
}
```

Listing 8: A revoke error response.

3 Data Models

Here, all data objects that can be part of the service interface are listed and must be respected by the implementing systems. Note that each subsection, which describes one type of object, begins with the *struct* keyword, which is meant to denote a JSON Object that must contain certain fields, or names, with values conforming to explicitly named types. As a complement to the primary types defined in this section, there is also a list of secondary types in Section 3.9.1, which are used to represent things like hashes, identifiers and texts.

3.1 struct DeviceRegistrationRequest

Field	Type	Mandatory	Description
name	Name	yes	Unique identifier of the device.
metadata	Metadata	no	Additional information about the device.
addresses	List<Address>	yes	Different kind of addresses of the device.

3.2 struct Metadata

An Object which maps String keys to primitive, Object or List values.

3.3 struct DeviceRegistrationResponse

Field	Type	Description
name	Name	Unique identifier of the registered device.
metadata	Metadata	Additional information about the registered device.
addresses	List<AddressDescriptor>	Different kind of addresses of the registered device.
createdAt	DateTime	Device was registered at this timestamp.
updatedAt	DateTime	Device was modified at this timestamp.

3.4 struct AddressDescriptor

Field	Type	Description
type	AddressType	Type of the address.
address	Address	Address.

3.5 struct ErrorResponse

Field	Type	Description
errorMessage	String	Description of the error.
errorCode	Number	Numerical code of the error. Same as HTTP response code.
type	ErrorType	Type value of the error.
origin	String	Origin of the error.

3.6 struct DeviceLookupRequest

Field	Type	Mandatory	Description
deviceNames	List<Name>	no	Requester is looking for devices with any of the specified names.
addresses	List<Address>	no	Requester is looking for devices with any of the specified addresses.
addressType	AddressType	no	Requester is looking for devices with the specified type of address.
metadataRequirementsList	List<MetadataRequirements>	no	Requester is looking for devices that are matching any of the specified metadata requirements.

3.7 struct MetadataRequirements

A special Object which maps String keys to Object, primitive or list values, where

- Keys can be paths (or multi-level keys) which access a specific value in a Metadata structure, where parts of the path are delimited with dot character (e.g. in case of ”key.subkey” path we are looking for the key named ”key” in the metadata, which is associated with an embedded object and in this object we are looking for the key named ”subkey”).
- Values are special Objects with two fields: an operation (e.g. less than) and an actual value (e.g. a number). A metadata is matching a requirement if the specified operation returns true using the metadata value referenced by a key path as first and the actual value as second operands.
- Alternatively, values can be ordinary primitives, lists or Objects. In this case the operation is equals by default.

3.8 struct DeviceLookupResponse

Field	Type	Description
entries	List<DeviceLookupResult>	List of device results.
count	Number	Number of returned devices.

3.9 struct DeviceLookupResult

Field	Type	Description
name	Name	Unique identifier of the device.
metadata	Metadata	Additional information about the device.
addresses	List<AddressDescriptor>	Different kind of addresses of the device.
createdAt	DateTime	Device was registered at this timestamp.
updatedAt	DateTime	Device was modified at this timestamp.

3.9.1 Primitives

As all message payloads are encoded using the JSON format [2], the following primitive constructs, part of that standard, become available. Note that the official standard is defined in terms of parsing rules, while this list only concerns syntactic information.

JSON Type	Description
Value	Any out of Object, Array, String, Number, Boolean or Null.
Object <A>	An unordered collection of [String: Value] pairs, where each Value conforms to type A.
Array <A>	An ordered collection of Value elements, where each element conforms to type A.
String	An arbitrary UTF-8 string.
Number	Any IEEE 754 binary64 floating point number [3], except for *+Inf*, *-Inf* and *NaN*.
Boolean	One out of `true` or `false`.
Null	Must be `null`.

With these primitives now available, we proceed to define all the types specified in the **device-discovery** SD document without a direct equivalent among the JSON types. Concretely, we define the **device-discovery** SD primitives either as *aliases* or *structs*. An *alias* is a renaming of an existing type, but with some further details about how it is intended to be used. Structs are described in the beginning of the parent section.

3.9.2 alias Address = String

A string representation of a network address. An address can be a version 4 IP address, a version 6 IP address, DNS name or MAC address.

3.9.3 alias AddressType = String

String value of a network address type. Could be only `HOSTNAME`, `IPV4`, `IPV6` or `MAC`.

3.9.4 alias DateTime = String

Pinpoints a moment in time in the format of ISO8601 standard ”yyyy-mm-ddThh:mm:ssZ”, where ”yyy” denotes year (4 digits), ”mm” denotes month starting from 01, ”dd” denotes day starting from 01, ”T” is the separator between date and time part, ”hh” denotes hour in the 24-hour format (00-23), ”MM” denotes minute (00-59), ”SS” denotes second (00-59). ”Z” indicates that the time is in UTC. An example of a valid date/time string is ”2024-12-05T12:00:00Z”

3.9.5 alias ErrorType = String

String value of the error type. Could be `ARROWHEAD`, `INVALID_PARAMETER`, `AUTH`, `FORBIDDEN`, `DATA_NOT_FOUND`, `TIMEOUT`, `LOCKED`, `INTERNAL_SERVER_ERROR` or `EXTERNAL_SERVER_ERROR`.

3.9.6 alias List <A> = Array<A>

There is no difference.

3.9.7 alias Name = String

A String indentifier that is intended to be both human and machine-readable.

4 References

[1] T. Bray, “The JavaScript Object Notation (JSON) Data Interchange Format,” RFC 8259, Dec. 2017. [Online]. Available: https://rfc-editor.org/rfc/rfc8259.txt

[2] ——, “The JavaScript Object Notation (JSON) Data Interchange Format,” RFC 7159, 2014, RFC Editor. [Online]. Available: https://doi.org/10.17487/RFC7159

[3] M. Cowlishaw, “IEEE Standard for Floating-Point Arithmetic,” *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, July 2019. [Online]. Available: https://doi.org/10.1109/IEEESTD.2019.8766229

5 Revision History

5.1 Amendments

No.	Date	Version	Subject of Amendments	Author
1	YYYY-MM-DD	5.0.0		Xxx Yyy

5.2 Quality Assurance

No.	Date	Version	Approved by
1	YYYY-MM-DD	5.0.0	Xxx Yyy