Resm

Resm is a simple resource manager that provides resources on demand.

Background

Users can ask the service to allocate a resource, or deallocate it if it is not needed anymore.

Resources are allocated by the user name and deallocated by the resource name.

User can request the list of all the allocated and deallocated resources, or just the list of allocated resources by him only.

Resources number (capacity) is limited. If all the resources were allocated, the server rejects the user's request until some other (or the same) user deallocates a resource.

The service provides an ability to reset its state so that all the resources become deallocated.

Functional requirements

1. The service should provide the following list of actions:
    - Allocate
    - Deallocate
    - List
    - Reset
2. The service should provide REST API.
   The API should be implemented as described below in the REST API section.
3. The service uses JSON format (http://json.org) to encode responses
4. Web service port and resources number should be configurable.
5. Build, clean, run and run unit tests should have a Makefile interface.

REST API

The REST API is described in the following way:

- State
- Request
- Response

State describes the state of the service.

Request describes the user action in terms of REST API.

Response describes expected HTTP response code and message body (if any).

Example:

- `{allocated:{},deallocated:[r1,r2,r3]}`
- `GET allocate/alice HTTP/1.1`
- `201 Created`
- `r1`

From this scenario we can see that all the resources are available and user Alice is trying to allocate a resource. As expected she gets first available resource which is r1.

Here response contains HTTP code (201 Created) and message body (r1).

Allocate

1.

- `{allocated:{},deallocated:[r1,r2,r3]}`
- `GET allocate/alice HTTP/1.1`
- `201 Created`
- `r1`

2.

- `{allocated:{r1:alice,r2:bob,r3:alice},deallocated:[]}`
- `GET allocate/bob`
- `503 Service Unavailable HTTP/1.1`
- `Out of resources.`

Deallocate

1.

- `{allocated:{r2:bob,r3:alice},deallocated:[r1]}`
- `GET deallocate/r2 HTTP/1.1`
- `204 No Content`

2.

- `{allocated:{r3:alice},deallocated:[r1,r2]}`
- `GET deallocate/r1 HTTP/1.1`
- `404 Not Found`
- `Not allocated.`

3.

- `{allocated:{r3:alice},deallocated:[r1,r2]}`
- `GET deallocate/any HTTP/1.1`
- `404 Not Found`
- `Not allocated.`

List

1.

- `{allocated:[],deallocated:[r1,r2,r3]}`
- `GET list HTTP/1.1`
- `200 OK`
- `{"allocated":[],"deallocated":["r1","r2","r3"]}`

2.

- `{allocated:{r1:alice,r2:bob},deallocated:[r3]}`
- `GET list HTTP/1.1`
- `200 OK`
- `{"allocated":{"r1":"alice","r2":"bob"},"deallocated":["r3"]}`

3.

- `{allocated:{r1:alice,r2:bob,r3:alice},deallocated:[]}`
- `GET list/alice HTTP/1.1`
- `200 OK`
- `["r1","r3"]`

4.

- `{allocated:{r1:alice,r3:alice},deallocated:[r2]}`
- `GET list/bob HTTP/1.1`
- `200 OK`
- `[]`

Reset

1.

- `{allocated:{r1:alice,r2:bob,r3:alice},deallocated:[]}`
- `GET reset HTTP/1.1`
- `204 No Content`

Bad request

1.

- `[Any state]`
- `[Any HTTP method and path other than described above]`
- `400 Bad Request`
- `Bad request.`

Common requirements

1. The project should be implemented in any of the following programming languages: Erlang, Python, C++.
   The other languages are not strictly prohibited though, please let us know if you are about to do that in different language.
2. The service should respond to HTTP requests.
3. It should be possible to build and run the application under the GNU/Linux OS (Ubuntu/Centos).
4. Unit tests should be provided.
5. The project should contain a readme file with instructions for build, configure, test and run the application.
   Preferably in English.
6. Service should have an ability to run in a Docker container:
   https://docs.docker.com/userguide/dockerizing/

Additional requirements (at will, considered as a plus)

1. Ability to run the application as an init.d service.
2. Ability to build a package for the application (rpm/deb).