RESTful API Binary Provisioning Agent (BPA)

This is a Golang application providing a RESTful API to interact with and upload image objects.

The API application source files are in the icn/cmd/bpa-restapi-agent directory.

While the database back-end is extensible, this initial release requires mongodb.

Motivation

The ICN RESTful API intends to provide access to resources inside the system using a uniform interface through the provision of one logical URI. As a constriant, "Servers and clients may also be replaced and developed independently, as long as the interface between them is not altered."

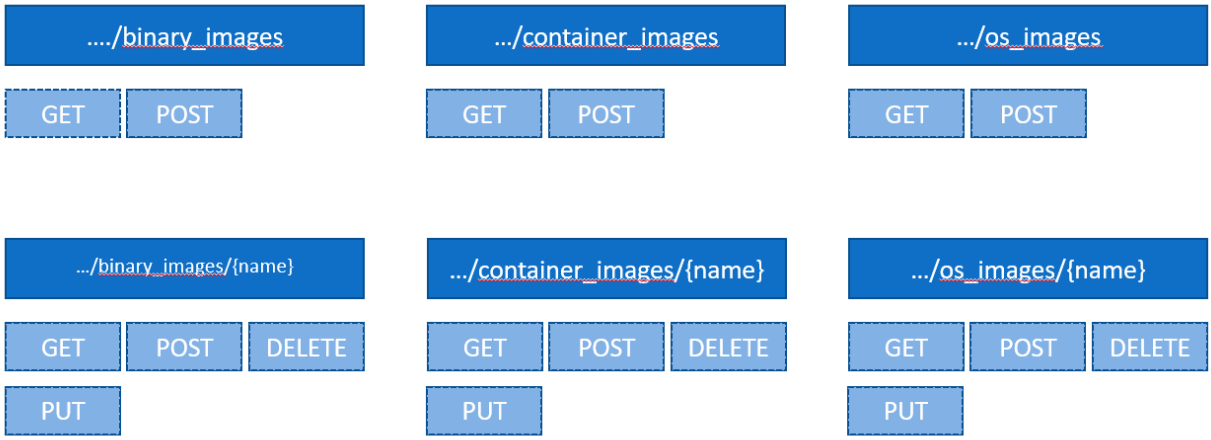
Requirements

This RESTful API service will expose the following resources to a user:

1. Binary Images
2. Container Images
3. OS Images

Design

- Use nouns to describe resources
- In the top URL, identify version, bare-metal cluster name, and resource
- GET - This will be used to list image resources
- POST - This will be used to create image resources
- PUT - This will be used to upload image resources
- DELETE - This will be used to delete image resources



Implementation

Sample **GET** Implementation:

Integrated Cloud Native RESTful API 1.0.0

Addresses deployment of workloads in the edge

Schemes

HTTP

Deployment of Images

GET /v1/baremetalcluster/{clustername}/images/ List all Images.

Endpoint to list all Images.

Parameters

Try it out

Name	Description
clustername * required string (path)	Name of the cluster used to query clustername - Name of the cluster used to query

Responses

Response content type application/json

Code	Description
200	successful operation **Example Value** | Model

```
{
  "ID": "string",
  "image_id": "string",
  "repo": "string",
  "tag": "string",
  "description": "string"
}
```

default	generic error response **Example Value** | Model

```
{
  "code": 0,
  "message": "string"
}
```

GET `/v1/baremetalcluster/{clustername}/images/{name}` Get details of an image.

Endpoint to get details of ICN available images.

Parameters

Try it out

Name	Description
clustername * required `string` *(path)*	Name of the cluster used to query clustername - Name of the cluster used to query
name * required `string` *(path)*	Name used to query name - Name used to query

Responses

Response content type: application/json

Code	Description
200	successful operation **Example Value** | Model

```
{
  "ID": "string",
  "image_id": "string",
  "repo": "string",
  "tag": "string",
  "description": "string"
}
```

default	generic error response **Example Value** | Model

```
{
  "code": 0,
  "message": "string"
}
```

The sample API

Proposed Usage

Proposed sample command to GET all binary images

```
# Get all binary images
curl -i http://localhost:9015/v1/baremetalcluster/{clustername}/binary_images/

# Get one binary image
curl -i http://localhost:9015/v1/baremetalcluster/{clustername}/binary_images/{name}
```

Proposed sample command to GET all container images

```
# Get all container images
curl -i http://localhost:9015/v1/baremetalcluster/{clustername}/container_images/

# Get one container image
curl -i http://localhost:9015/v1/baremetalcluster/{clustername}/container_images/{name}
```

Proposed sample command to GET all OS images

```
# Get all container images
curl -i http://localhost:9015/v1/baremetalcluster/{clustername}/os_images/

# Get one container image
curl -i http://localhost:9015/v1/baremetalcluster/{clustername}/os_images/{name}
```

How To Use The BPA RESTful API Service

Install and start mongodb. For instructions: https://docs.mongodb.com/manual/installation/

git clone "https://gerrit.akraino.org/r/icn"
cd icn/cmd/bpa-restapi-agent

Run the application
go run main.go

Sample output without a config file:

2019/08/22 14:08:41 Error loading config file. Using defaults
2019/08/22 14:08:41 Starting Integrated Cloud Native API

Cloud Storage with MinIO

Start MinIO server daemon with docker command before run REST API agent, default settings in config/config.go.
AccessKeyID: ICN-ACCESSKEYID
SecretAccessKey: ICN-SECRETACCESSKEY
MinIO Port: 9000

```
$ docker run -p 9000:9000 --name minio1 \
-v /mnt/data:/data \
-v /mnt/config:/root/.minio \
minio/minio server /data
```

MinIO Client will automatic initialize in main.go, and create 3 buckets: binary, container, operatingsystem.
The Upload image will "PUT" to corresponding buckets by HTTP PATCH request url.
You can also check by open browser: http://127.0.0.1:9000/

RESTful API usage examples

Sample POST request format

curl -i -F "metadata=<jsonfile;type=application/json" -F file=@/home/<username>/<dir>/jsonfile -X POST http://NODE_IP:9015//baremetalcluster/{owner}/{clustername}/<image_type>

#image type can be binary_image, container_image, or os_image

Example requests and responses:

Create image - POST

#Using a json file called sample.json
#image_length in sample.json can be determined with the command
ls -al <image_file>

Request

curl -i -F "metadata=<sample.json;type=application/json" -F file=@/home/enyi/workspace/icn/cmd/bpa-restapi-agent/sample.json -X POST http://localhost:9015/v1/baremetalcluster/alpha/beta/container_images

Response

HTTP/1.1 100 Continue

HTTP/1.1 201 Created
Content-Type: application/json
Date: Thu, 22 Aug 2019 22:56:16 GMT
Content-Length: 239

{"owner":"alpha","cluster_name":"beta","type":"container","image_name":"asdf246","image_offset":0,"image_length":29718177,"upload_complete":false,"description":{"image_records":[{"image_record_name":"iuysdi1234","repo":"icn","tag":"1"}]}}

#this creates a database entry for the image, and an empty file in the file system

List image - GET

curl -i -X GET http://localhost:9015/v1/baremetalcluster/{owner}/{clustername}/<image_type>/{imgname}

example:
#continuing with our container image from above

Request

curl -i -X GET http://localhost:9015/v1/baremetalcluster/alpha/beta/container_images/asdf246

Response

HTTP/1.1 200 OK
Content-Type: application/json
Date: Thu, 22 Aug 2019 22:57:10 GMT
Content-Length: 239

{"owner":"alpha","cluster_name":"beta","type":"container","image_name":"asdf246","image_offset":0,"image_length":29718177,"upload_complete":false,"description":{"image_records":[{"image_record_name":"iuysdi1234","repo":"icn","tag":"1"}]}}

Upload container image - PATCH

Request

curl --request PATCH --data-binary "@/home/enyi/workspace/icn/cmd/bpa-restapi-agent/sample_image" http://localhost:9015/v1/baremetalcluster/alpha/beta/container_images/asdf246 --header "Upload-Offset: 0" --header "Expect:" -i

Response

HTTP/1.1 204 No Content
Upload-Offset: 29718177
Date: Thu, 22 Aug 2019 23:19:44 GMT

Check uploaded image - GET

Request

curl -i -X GET http://localhost:9015/v1/baremetalcluster/alpha/beta/container_images/asdf246

Response

HTTP/1.1 200 OK
Content-Type: application/json
Date: Fri, 23 Aug 2019 17:12:07 GMT
Content-Length: 245

{"owner":"alpha","cluster_name":"beta","type":"container","image_name":"asdf246","image_offset":29718177,"image_length":29718177,"upload_complete":true,"description":{"image_records":[{"image_record_name":"iuysdi1234","repo":"icn","tag":"1"}]}}

#after the upload, the image_offset is now the same as image_length and upload_complete changed to true
#if upload was incomplete

Resumable upload -PATCH

#this is the current resumable upload mechanism

Request

curl --request PATCH --data-binary "@/home/enyi/workspace/icn/cmd/bpa-restapi-agent/sample_image" http://localhost:9015/v1/baremetalcluster/alpha/beta/container_images/asdf246 --header "Upload-Offset: 0" --header "Expect:" -i --limit-rate 200K

#the above request limits transfer for testing purposes
#'ctl c' out after a few seconds, to stop file transfer
#check image_offset with a GET

Check upload - GET

Request

curl -i -X GET http://localhost:9015/v1/baremetalcluster/alpha/beta/container_images/asdf246

Response

HTTP/1.1 200 OK
Content-Type: application/json
Date: Sat, 24 Aug 2019 00:30:00 GMT
Content-Length: 245

{"owner":"alpha","cluster_name":"beta","type":"container","image_name":"asdf246","image_offset":4079616,"image_length":29718177,"upload_complete":false,"description":{"image_records":[{"image_record_name":"iuysdi1234","repo":"icn","tag":"2"}]}}

#from our response you can see that image_offset is still less than image_length and #upload_complete is still false
#next we use the dd command (no limiting this time)

Request

dd if=/home/enyi/workspace/icn/cmd/bpa-restapi-agent/sample_image skip=4079616 bs=1 | curl --request PATCH --data-binary @- http://localhost:9015/v1/baremetalcluster/alpha/beta/container_images/asdf246 --header "Upload-Offset: 4079616" --header "Expect:" -i

#the request skips already uploaded 4079616 bytes of data

Response

25638561+0 records in
25638561+0 records out
25638561 bytes (26 MB, 24 MiB) copied, 207.954 s, 123 kB/s
HTTP/1.1 204 No Content
Upload-Offset: 29718177
Date: Sat, 24 Aug 2019 00:43:18 GMT

Update image description - PUT

let's change the tag in description from 1 to latest
once the change is made in sample.json (or your json file)

Request

curl -i -F "metadata=<sample.json;type=application/json" -F file=@/home/enyi/workspace/icn/cmd/bpa-restapi-agent/sample.json -X PUT http://localhost:9015/v1/baremetalcluster/alpha/beta/container_images/asdf246

Response

HTTP/1.1 100 Continue

HTTP/1.1 201 Created
Content-Type: application/json
Date: Fri, 23 Aug 2019 17:21:01 GMT
Content-Length: 239

{"owner":"alpha","cluster_name":"beta","type":"container","image_name":"asdf246","image_offset":0,"image_length":29718177,"upload_complete":false,"description":{"image_records":[{"image_record_name":"iuysdi1234","repo":"icn","tag":"2"}]}}

Delete an image - DELETE

Request

curl -i -X DELETE http://localhost:9015/v1/baremetalcluster/alpha/beta/container_images/asdf246

How To Write ICN Unit Tests in Go using Testify

Go version used is go1.12.9

To install Go, follow the instructions on the official site, (https://golang.org/doc/install).

Testify documentation can be found on Github, https://github.com/stretchr/testify

The first question is, "what to test?" In this case, I'll test a method called GetDirPath in the file image.go.

GetDirPath uses the Go core packages *user* and *path* to return a file path, and a directory path.

```
package app

import (
"os/user"
"path"

pkgerrors "github.com/pkg/errors"
)

func (v *ImageClient) GetDirPath(imageName string) (string, string, error) {
        u, err := user.Current()
        if err != nil {
                 return "", "", pkgerrors.Wrap(err, "Current user")
        }
        home := u.HomeDir
        dirPath := path.Join(home, "images", v.storeName)
        filePath := path.Join(dirPath, imageName)

        return filePath, dirPath, err
}
```

Here is the ImageClient type that the method GetDirPath is defined on:

```
type ImageClient struct {
        storeName string
        tagMeta   string
}
```

To test Go packages, Go has a core package called testing which is used with the command *go test*. I'll create the test file image_test.go in the same package as my image.go file. A test for the GetDirPath could look like this:

```
import (
  "testing"
)

func TestGetDirPath(t *testing.T) {

  imageClient := &ImageClient{"test_image", "test_meta"}
  actual, dir, err := imageClient.GetDirPath("test")
  var expected = "/home/enyi/images/test_image/test"
  if actual != expected {
    t.Errorf("Path was incorrect, got: %s, want %s.", actual, expected)
  }
}
```

The TestGetDirPath (test functions begin with 'Test') function uses the testing package to print a message before failing. You can explore the testing package here https://golang.org/pkg/testing/.

Change into your package directory and run:

go test

It should pass.

Next, we'll mock out the user package with Testify and rewrite our test.

If you're using go modules for dependency management, you don't have to install Testify. Otherwise, run:

go get https://github.com/stretchr/testify

To mock a third party application we use an interface. This allows us to create a method, GetUser, that uses the package that we'd like to mock.

So, ultimately, we'll have two definitions of our interface: actual definition and mock definition.

The interface:

```
// Interface for imported packages
type ImportsService interface {
        GetCurrentUser() (*user.User, error)
}
```

Let's make ImageCLient use ImportsService to get user. So, ImageClient implements and uses ImportsService.

```
type ImageClient struct {
        user ImportsService
        storeName string
        tagMeta   string
}
```

Let's rewrite GetDirPath to use GetCurrentUser.

```
func (v *ImageClient) GetDirPath(imageName string) (string, string, error) {
        u, err := v.GetCurrentUser()
        home := u.HomeDir
        dirPath := path.Join(home, "images", v.storeName)
        filePath := path.Join(dirPath, imageName)

        return filePath, dirPath, err
}
```

Let's define GetCurrentUser on ImageClient.

```
// define GetCurrentUser
func (v *ImageClient) GetCurrentUser() (*user.User, error) {
        u, err := user.Current()
        if err != nil {
                return nil, pkgerrors.Wrap(err, "Current user")
        }

        return u, nil
}
```

To test GetDirPath using testify mock, let's re-write image_test.go

```
package app

import (
  "fmt"
  "os/user"
  "testing"

  "github.com/stretchr/testify/mock"
)

type mockImports struct {
  mock.Mock
}

func (m *mockImports) GetCurrentUser() (*user.User, error) {
  fmt.Println("Mocked Get User")
  args := m.Called()

  return args.Get(0).(*user.User), args.Error(1)
}

func TestService(t *testing.T) {
  fakeUser := user.User{}

  myImports := new(mockImports)

  myImports.On("GetCurrentUser").Return(&fakeUser, nil)

  imageClient := ImageClient{myImports, "test_image", "test_meta"}
  _, dir, err := imageClient.GetDirPath("test")
  if err != nil {
    t.Errorf("Path was incorrect, got: %q, want: %q.", dir, "some_path")
  }
}
```

With GetCurrentUser defined on mockImports, inside my TestService, I choose the mockImports object for testing:

```
imageClient := ImageClient{myImports, "test_image", "test_meta"}
```

Run:

go test -v

References:

1. https://restfulapi.net/rest-architectural-constraints/