

Если вы видите что-то необычное, просто сообщите мне.

# docker push без docker push(пример)

запуск:

```
./uploadImage "~/path/to/saved/image" "http://localhost:8081/link/to/docker/registry" myRepoName 1.0
```

Код приложения:

```
package main

import (
    "bytes"
    "fmt"
    "io/ioutil"
    "net/http"
    "os"
    "path/filepath"
    "strings"
)

func main() {
    imageDir := os.Args[1]
    url := os.Args[2]
    repoName := os.Args[3]
    tag := os.Args[4]
    manifestFile := filepath.Join(imageDir, "manifestCopy")
    configFile := findFileWithExtension(imageDir, ".json")

    prepareLayersForUpload(imageDir, manifestFile)
    setConfigProps(manifestFile, configFile)
    manifestContent, err := ioutil.ReadFile(manifestFile)
    if err != nil {
```

```

    panic(err)
}

uuid := initiateUpload(url, repoName)

layersNames := findLayerFiles(imageDir)
layersSizes := findLayerSizes(imageDir, layersNames)
for i, layerName := range layersNames {
    pathToLayer := findFileWithExactName(imageDir, layerName)
    patchLayer(url, repoName, uuid, pathToLayer, layersSizes[i])
    putLayer(url, repoName, uuid, layerName, pathToLayer, layersSizes[i])
}

configName := getFileNameWithoutExtension(configFile)
patchLayer(url, repoName, uuid, configFile, fileSize(configFile))
putLayer(url, repoName, uuid, configName, configFile, fileSize(configFile))

putManifest(url, repoName, tag, manifestContent)

fmt.Println("Upload completed successfully")
}

func findFileWithExtension(dir, extension string) string {
    files, err := ioutil.ReadDir(dir)
    if err != nil {
        panic(err)
    }
    for _, file := range files {
        if filepath.Ext(file.Name()) == extension {
            return filepath.Join(dir, file.Name())
        }
    }
    panic("File with extension not found")
}

func findFileWithExactName(dir, fileName string) string {
    err := filepath.Walk(dir, func(path string, info os.FileInfo, err error) error {
        if info.Name() == fileName {
            foundPath = path
            return filepath.SkipDir
        }
    })
}

```

```

    return nil
  })
  if err != nil {
    panic(err)
  }
  return foundPath
}

```

```

func findLayerFiles(imageDir string) []string {
  var layerNames []string
  err := filepath.Walk(imageDir, func(path string, info os.FileInfo, err error) error {
    if strings.HasSuffix(info.Name(), "layer.tar") {
      layerNames = append(layerNames, info.Name())
    }
  })
  return nil
  })
  if err != nil {
    panic(err)
  }
  return layerNames
}

```

```

func findLayerSizes(imageDir string, layerNames []string) []int64 {
  var layerSizes []int64
  for _, layerName := range layerNames {
    pathToLayer := findFileWithExactName(imageDir, layerName)
    fileInfo, err := os.Stat(pathToLayer)
    if err != nil {
      panic(err)
    }
    layerSizes = append(layerSizes, fileInfo.Size())
  }
  return layerSizes
}

```

```

func prepareLayersForUpload(imageDir, manifestFile string) {
  infoFile := filepath.Join(imageDir, "info")
  layersNames := findLayerFiles(imageDir)
  layersSizes := findLayerSizes(imageDir, layersNames)

  var buffer bytes.Buffer

```

```

    buffer.WriteString("{}")

    for i, layerName := range layersNames {
        layerSize := layersSizes[i]
        layerDigest := getSha256Sum(findFileWithExactName(imageDir, layerName))
        buffer.WriteString(fmt.Sprintf("{
            \"mediaType\": \"application/vnd.docker.image.rootfs.diff.tar.gzip\",
            \"size\": %d,
            \"digest\": \"sha256:%s\"
        },", layerSize, layerDigest))
    }

    buffer.Truncate(buffer.Len() - 1)
    buffer.WriteString("\n\t}\n}")

    err := ioutil.WriteFile(manifestFile, buffer.Bytes(), 0644)
    if err != nil {
        panic(err)
    }
}

func setConfigProps(manifestFile, configFile string) {
    configSize := fileSize(configFile)
    configName := getFileNameWithoutExtension(configFile)
    manifestContent, err := ioutil.ReadFile(manifestFile)
    if err != nil {
        panic(err)
    }

    manifestContent = bytes.ReplaceAll(manifestContent, []byte("config_size"), []byte(fmt.Sprintf("%d", configSize)))
    manifestContent = bytes.ReplaceAll(manifestContent, []byte("config_hash"), []byte(fmt.Sprintf("%s", configName)))

    err = ioutil.WriteFile(manifestFile, manifestContent, 0644)
    if err != nil {
        panic(err)
    }
}

func initiateUpload(url, repoName string) string {

```

```

    resp, err := http.Post(fmt.Sprintf("%s/v2/%s/blobs/uploads/", url, repoName), "", nil)
    if err != nil {
        panic(err)
    }
    uuid := resp.Header.Get("Docker-Upload-Uuid")
    if uuid == "" {
        panic("Failed to initiate upload")
    }
    return uuid
}

func patchLayer(url, repoName, uuid, filePath string, fileSize int64) {
    file, err := os.Open(filePath)
    if err != nil {
        panic(err)
    }
    defer file.Close()

    _, err = httpPatch(fmt.Sprintf("%s/v2/%s/blobs/uploads/%s", url, repoName, uuid), file, fileSize, "Content-Type:
    application/octet-stream")
    if err != nil {
        panic(err)
    }
}

func putLayer(url, repoName, uuid, layerName, filePath string, fileSize int64) {
    _, err := httpPut(fmt.Sprintf("%s/v2/%s/blobs/uploads/%s?digest=sha256:%s", url, repoName, uuid, layerName),
    filePath, fileSize, "Content-Type: application/octet-stream")
    if err != nil {
        panic(err)
    }
}

func putManifest(url, repoName, tag string, manifestContent []byte) {
    _, err := httpPut(fmt.Sprintf("%s/v2/%s/manifests/%s", url, repoName, tag), bytes.NewReader(manifestContent),
    int64(len(manifestContent)), "Content-Type: application/vnd.docker.distribution.manifest.v2+json")
    if err != nil {
        panic(err)
    }
}

```

```

func fileSize(filePath string) int64 {
    fileInfo, err := os.Stat(filePath)
    if err != nil {
        panic(err)
    }
    return fileInfo.Size()
}

func getSha256Sum(filePath string) string {
    file, err := os.Open(filePath)
    if err != nil {
        panic(err)
    }
    defer file.Close()

    hash := sha256.New()
    if _, err := io.Copy(hash, file); err != nil {
        panic(err)
    }
    return hex.EncodeToString(hash.Sum(nil))
}

func getFileNameWithoutExtension(filePath string) string {
    return strings.TrimSuffix(filepath.Base(filePath), filepath.Ext(filePath))
}

func httpPatch(url string, body io.Reader, contentLength int64, headers ...string) ([]byte, error) {
    req, err := http.NewRequest(http.MethodPatch, url, body)
    if err != nil {
        return nil, err
    }

    addHeaders(req, contentLength, headers...)

    resp, err := http.DefaultClient.Do(req)
    if err != nil {
        return nil, err
    }
    defer resp.Body.Close()

    responseBody, err := ioutil.ReadAll(resp.Body)

```

```

    if err != nil {
        return nil, err
    }

    if resp.StatusCode >= http.StatusBadRequest {
        return nil, fmt.Errorf("HTTP %d: %s", resp.StatusCode, resp.Status)
    }

    return responseBody, nil
}

func httpPut(url string, filePath string, contentLength int64, headers ...string) ([]byte, error) {
    file, err := os.Open(filePath)
    if err != nil {
        return nil, err
    }
    defer file.Close()

    req, err := http.NewRequest(http.MethodPut, url, file)
    if err != nil {
        return nil, err
    }

    addHeaders(req, contentLength, headers...)

    resp, err := http.DefaultClient.Do(req)
    if err != nil {
        return nil, err
    }
    defer resp.Body.Close()

    responseBody, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        return nil, err
    }

    if resp.StatusCode >= http.StatusBadRequest {
        return nil, fmt.Errorf("HTTP %d: %s", resp.StatusCode, resp.Status)
    }

    return responseBody, nil
}

```

```
}

func addHeaders(req *http.Request, contentLength int64, headers ...string) {
    req.ContentLength = contentLength
    for _, header := range headers {
        parts := strings.SplitN(header, ":", 2)
        if len(parts) != 2 {
            continue
        }
        req.Header.Set(strings.TrimSpace(parts[0]), strings.TrimSpace(parts[1]))
    }
}
```

---

Revision #2

Created 12 December 2023 06:44:58 by gasick

Updated 12 December 2023 06:58:06 by gasick