Docker Registry HTTP API V2

Estimated reading time: 116 minutes

Introduction

The *Docker Registry HTTP API* is the protocol to facilitate distribution of images to the docker engine. It interacts with instances of the docker registry, which is a service to manage information about docker images and enable their distribution. The specification covers the operation of version 2 of this API, known as *Docker Registry HTTP API V2*.

While the V1 registry protocol is usable, there are several problems with the architecture that have led to this new version. The main driver of this specification is a set of changes to the docker the image format, covered in docker/docker#8093. The new, self-contained image manifest simplifies image definition and improves security. This specification will build on that work, leveraging new properties of the manifest format to improve performance, reduce bandwidth usage and decrease the likelihood of backend corruption.

For relevant details and history leading up to this specification, please see the following issues:

- docker/docker#8093
- docker/docker#9015
- docker/docker-registry#612

Scope

This specification covers the URL layout and protocols of the interaction between docker registry and docker core. This will affect the docker core registry API and the rewrite of docker-registry. Docker registry implementations may implement other API endpoints, but they are not

covered by this specification.

This includes the following features:

- Namespace-oriented URI Layout
- PUSH/PULL registry server for V2 image manifest format
- Resumable layer PUSH support
- V2 Client library implementation

While authentication and authorization support will influence this specification, details of the protocol will be left to a future specification. Relevant header definitions and error codes are present to provide an indication of what a client may encounter.

Future

There are features that have been discussed during the process of cutting this specification. The following is an incomplete list:

- Immutable image references
- Multiple architecture support
- Migration from v2compatibility representation

These may represent features that are either out of the scope of this specification, the purview of another specification or have been deferred to a future version.

Use Cases

For the most part, the use cases of the former registry API apply to the new version. Differentiating use cases are covered below.

Image Verification

A docker engine instance would like to run verified image named “library/ubuntu”, with the tag “latest”. The engine contacts the registry,

requesting the manifest for “library/ubuntu:latest”. An untrusted registry returns a manifest. Before proceeding to download the individual layers, the engine verifies the manifest’s signature, ensuring that the content was produced from a trusted source and no tampering has occurred. After each layer is downloaded, the engine verifies the digest of the layer, ensuring that the content matches that specified by the manifest.

Resumable Push

Company X’s build servers lose connectivity to docker registry before completing an image layer transfer. After connectivity returns, the build server attempts to re-upload the image. The registry notifies the build server that the upload has already been partially attempted. The build server responds by only sending the remaining data to complete the image file.

Resumable Pull

Company X is having more connectivity problems but this time in their deployment datacenter. When downloading an image, the connection is interrupted before completion. The client keeps the partial data and uses http `Range` requests to avoid downloading repeated data.

Layer Upload De-duplication

Company Y’s build system creates two identical docker layers from build processes A and B. Build process A completes uploading the layer before B. When process B attempts to upload the layer, the registry indicates that its not necessary because the layer is already known.

If process A and B upload the same layer at the same time, both operations will proceed and the first to complete will be stored in the registry (Note: we may modify this to prevent dogpile with some locking mechanism).

Changes

The V2 specification has been written to work as a living document, specifying only what is certain and leaving what is not specified open or to future changes. Only non-conflicting additions should be made to the API and accepted changes should avoid preventing future changes from happening.

This section should be updated when changes are made to the specification, indicating what is different. Optionally, we may start marking parts of the specification to correspond with the versions enumerated here.

Each set of changes is given a letter corresponding to a set of modifications that were applied to the baseline specification. These are merely for reference and shouldn't be used outside the specification other than to identify a set of modifications.

k

- Document use of Accept and Content-Type headers in manifests endpoint.

j

- Add ability to mount blobs across repositories.

i

- Clarified expected behavior response to manifest HEAD request.

h

- All mention of tarsum removed.

g

- Clarify behavior of pagination behavior with unspecified

parameters.

f

- Specify the delete API for layers and manifests.

e

- Added support for listing registry contents.
- Added pagination to tags API.
- Added common approach to support pagination.

d

- Allow repository name components to be one character.
- Clarified that single component names are allowed.

c

- Added section covering digest format.
- Added more clarification that manifest cannot be deleted by tag.

b

- Added capability of doing streaming upload to PATCH blob upload.
- Updated PUT blob upload to no longer take final chunk, now requires entire data or no data.
- Removed `416 Requested Range Not Satisfiable` response status from PUT blob upload.

a

- Added support for immutable manifest references in manifest endpoints.
- Deleting a manifest by tag has been deprecated.

- Specified `Docker-Content-Digest` header for appropriate entities.
- Added error code for unsupported operations.

Overview

This section covers client flows and details of the API endpoints. The URI layout of the new API is structured to support a rich authentication and authorization model by leveraging namespaces. All endpoints will be prefixed by the API version and the repository name:

For example, an API endpoint that will work with the `library/ubuntu` repository, the URI prefix will be:

This scheme provides rich access control over various operations and methods using the URI prefix and http methods that can be controlled in variety of ways.

Classically, repository names have always been two path components where each path component is less than 30 characters. The V2 registry API does not enforce this. The rules for a repository name are as follows:

1. A repository name is broken up into *path components*. A component of a repository name must be at least one lowercase, alpha-numeric characters, optionally separated by periods, dashes or underscores. More strictly, it must match the regular expression `[a-z0-9]+(?:[._-][a-z0-9]+)*`.
2. If a repository name has two or more path components, they must be separated by a forward slash ("/").
3. The total length of a repository name, including slashes, must be less the 256 characters.

These name requirements *only* apply to the registry API and should accept a superset of what is supported by other docker ecosystem components.

All endpoints should support aggressive http caching, compression and range headers, where appropriate. The new API attempts to leverage HTTP semantics where possible but may break from standards to implement targeted features.

For detail on individual endpoints, please see the *Detail* section.

Errors

Actionable failure conditions, covered in detail in their relevant sections, are reported as part of 4xx responses, in a json response body. One or more errors will be returned in the following format:

```
{
    "errors:" [{
            "code": <error identifier>,
            "message": <message describing condition>,
            "detail": <unstructured>
        },
        ...
    ]
}
```

The `code` field will be a unique identifier, all caps with underscores by convention. The `message` field will be a human readable string. The optional `detail` field may contain arbitrary json data providing information the client can use to resolve the issue.

While the client can take action on certain error codes, the registry may add new error codes over time. All client implementations should treat unknown error codes as `UNKNOWN`, allowing future error codes to be added without breaking API compatibility. For the purposes of the specification error codes will only be added and never removed.

For a complete account of all error codes, please see the *Errors* section.

API Version Check

A minimal endpoint, mounted at `/v2/` will provide version support information based on its response statuses. The request format is as follows:

If a `200 OK` response is returned, the registry implements the V2(.1) registry API and the client may proceed safely with other V2 operations. Optionally, the response may contain information about the supported paths in the response body. The client should be prepared to ignore this data.

If a `401 Unauthorized` response is returned, the client should take action based on the contents of the "WWW-Authenticate" header and try the endpoint again. Depending on access control setup, the client may still have to authenticate against different resources, even if this check succeeds.

If `404 Not Found` response status, or other unexpected status, is returned, the client should proceed with the assumption that the registry does not implement V2 of the API.

When a `200 OK` or `401 Unauthorized` response is returned, the "Docker-Distribution-API-Version" header should be set to "registry/2.0". Clients may require this header value to determine if the endpoint serves this API. When this header is omitted, clients may fallback to an older API version.

Content Digests

This API design is driven heavily by content addressability. The core of this design is the concept of a content addressable identifier. It uniquely identifies content by taking a collision-resistant hash of the bytes. Such an identifier can be independently calculated and verified by selection of a common *algorithm*. If such an identifier can be communicated in a secure

manner, one can retrieve the content from an insecure source, calculate it independently and be certain that the correct content was obtained. Put simply, the identifier is a property of the content.

To disambiguate from other concepts, we call this identifier a *digest*. A *digest* is a serialized hash result, consisting of a *algorithm* and *hex* portion. The *algorithm* identifies the methodology used to calculate the digest. The *hex* portion is the hex-encoded result of the hash.

We define a *digest* string to match the following grammar: `digest := algorithm ":" hex algorithm := /[A-Fa-f0-9_+.-]+/ hex := /[A-Fa-f0-9]+/`

Some examples of *digests* include the following:

digest
sha256:6c3c624b58dbbcd3c0dd82b4c53f04194d1247c6eebdaab7c610cf7d66709b

While the *algorithm* does allow one to implement a wide variety of algorithms, compliant implementations should use sha256. Heavy processing of input before calculating a hash is discouraged to avoid degrading the uniqueness of the *digest* but some canonicalization may be performed to ensure consistent identifiers.

Let's use a simple example in pseudo-code to demonstrate a digest calculation: `let C = 'a small string' let B = sha256(C) let D = 'sha256:' + EncodeHex(B) let ID(C) = D`

Above, we have bytestring `C` passed into a function, `SHA256`, that returns a bytestring `B`, which is the hash of `C`. `D` gets the algorithm concatenated with the hex encoding of `B`. We then define the identifier of `C` to `ID(C)` as equal to `D`. A digest can be verified by independently calculating `D` and comparing it with identifier `ID(C)`.

Digest Header

To provide verification of http content, any response may include a `Docker-Content-Digest` header. This will include the digest of the target entity returned in the response. For blobs, this is the entire blob content. For manifests, this is the manifest body without the signature content, also known as the JWS payload. Note that the commonly used canonicalization for digest calculation may be dependent on the mediatype of the content, such as with manifests.

The client may choose to ignore the header or may verify it to ensure content integrity and transport security. This is most important when fetching by a digest. To ensure security, the content should be verified against the digest used to fetch the content. At times, the returned digest may differ from that used to initiate a request. Such digests are considered to be from different *domains*, meaning they have different values for *algorithm*. In such a case, the client may choose to verify the digests in both domains or ignore the server's digest. To maintain security, the client *must* always verify the content against the *digest* used to fetch the content.

> **IMPORTANT:** If a *digest* is used to fetch content, the client should use the same digest used to fetch the content to verify it. The header `Docker-Content-Digest` should not be trusted over the "local" digest.

Pulling An Image

An "image" is a combination of a JSON manifest and individual layer files. The process of pulling an image centers around retrieving these two components.

The first step in pulling an image is to retrieve the manifest. For reference, the relevant manifest fields for the registry are the following:

field	description

name	The name of the image.
tag	The tag for this version of the image.
fsLayers	A list of layer descriptors (including digest)
signature	A JWS used to verify the manifest content

For more information about the manifest format, please see docker/docker#8093.

When the manifest is in hand, the client must verify the signature to ensure the names and layers are valid. Once confirmed, the client will then use the digests to download the individual layers. Layers are stored in as blobs in the V2 registry API, keyed by their digest.

Pulling an Image Manifest

The image manifest can be fetched with the following url:

```
GET /v2/<name>/manifests/<reference>
```

The `name` and `reference` parameter identify the image and are required. The reference may include a tag or digest.

The client should include an Accept header indicating which manifest content types it supports. For more details on the manifest formats and their content types, see manifest-v2-1.md and manifest-v2-2.md. In a successful response, the Content-Type header will indicate which manifest type is being returned.

A `404 Not Found` response will be returned if the image is unknown to the registry. If the image exists and the response is successful, the image manifest will be returned, with the following format (see docker/docker#8093 for details):

```
{
```

```
   "name": <name>,
   "tag": <tag>,
   "fsLayers": [
      {
         "blobSum": <digest>
      },
      ...
    ]
   ],
   "history": <v1 images>,
   "signature": <JWS>
}
```

The client should verify the returned manifest signature for authenticity before fetching layers.

Existing Manifests

The image manifest can be checked for existence with the following url:

```
HEAD /v2/<name>/manifests/<reference>
```

The `name` and `reference` parameter identify the image and are required. The reference may include a tag or digest.

A `404 Not Found` response will be returned if the image is unknown to the registry. If the image exists and the response is successful the response will be as follows:

```
200 OK
Content-Length: <length of manifest>
Docker-Content-Digest: <digest>
```

Pulling a Layer

Layers are stored in the blob portion of the registry, keyed by digest.

Pulling a layer is carried out by a standard http request. The URL is as follows:

```
GET /v2/<name>/blobs/<digest>
```

Access to a layer will be gated by the `name` of the repository but is identified uniquely in the registry by `digest`.

This endpoint may issue a 307 (302 for <HTTP 1.1) redirect to another service for downloading the layer and clients should be prepared to handle redirects.

This endpoint should support aggressive HTTP caching for image layers. Support for Etags, modification dates and other cache control headers should be included. To allow for incremental downloads, `Range` requests should be supported, as well.

Pushing An Image

Pushing an image works in the opposite order as a pull. After assembling the image manifest, the client must first push the individual layers. When the layers are fully pushed into the registry, the client should upload the signed manifest.

The details of each step of the process are covered in the following sections.

Pushing a Layer

All layer uploads use two steps to manage the upload process. The first step starts the upload in the registry service, returning a url to carry out the second step. The second step uses the upload url to transfer the actual data. Uploads are started with a POST request which returns a url that can be used to push data and check upload status.

The `Location` header will be used to communicate the upload location after each request. While it won't change in the this specification, clients should use the most recent value returned by the API.

Starting An Upload

To begin the process, a POST request should be issued in the following format:

```
POST /v2/<name>/blobs/uploads/
```

The parameters of this request are the image namespace under which the layer will be linked. Responses to this request are covered below.

Existing Layers

The existence of a layer can be checked via a `HEAD` request to the blob store API. The request should be formatted as follows:

```
HEAD /v2/<name>/blobs/<digest>
```

If the layer with the digest specified in `digest` is available, a 200 OK response will be received, with no actual body content (this is according to http specification). The response will look as follows:

```
200 OK
Content-Length: <length of blob>
Docker-Content-Digest: <digest>
```

When this response is received, the client can assume that the layer is already available in the registry under the given name and should take no further action to upload the layer. Note that the binary digests may differ for the existing registry layer, but the digests will be guaranteed to match.

Uploading the Layer

If the POST request is successful, a `202 Accepted` response will be returned with the upload URL in the `Location` header:

```
202 Accepted
Location: /v2/<name>/blobs/uploads/<uuid>
Range: bytes=0-<offset>
Content-Length: 0
Docker-Upload-UUID: <uuid>
```

The rest of the upload process can be carried out with the returned url, called the "Upload URL" from the `Location` header. All responses to the upload url, whether sending data or getting status, will be in this format. Though the URI format (`/v2/<name>/blobs/uploads/<uuid>`) for the `Location` header is specified, clients should treat it as an opaque url and should never try to assemble it. While the `uuid` parameter may be an actual UUID, this proposal imposes no constraints on the format and clients should never impose any.

If clients need to correlate local upload state with remote upload state, the contents of the `Docker-Upload-UUID` header should be used. Such an id can be used to key the last used location header when implementing resumable uploads.

Upload Progress

The progress and chunk coordination of the upload process will be coordinated through the `Range` header. While this is a non-standard use of the `Range` header, there are examples of similar approaches in APIs with heavy use. For an upload that just started, for an example with a 1000 byte layer file, the `Range` header would be as follows:

To get the status of an upload, issue a GET request to the upload URL:

```
GET /v2/<name>/blobs/uploads/<uuid>
Host: <registry host>
```

The response will be similar to the above, except will return 204 status:

```
204 No Content
Location: /v2/<name>/blobs/uploads/<uuid>
Range: bytes=0-<offset>
Docker-Upload-UUID: <uuid>
```

Note that the HTTP `Range` header byte ranges are inclusive and that will be honored, even in non-standard use cases.

Monolithic Upload

A monolithic upload is simply a chunked upload with a single chunk and may be favored by clients that would like to avoided the complexity of chunking. To carry out a “monolithic” upload, one can simply put the entire content blob to the provided URL:

```
PUT /v2/<name>/blobs/uploads/<uuid>?digest=<digest>
Content-Length: <size of layer>
Content-Type: application/octet-stream

<Layer Binary Data>
```

The “digest” parameter must be included with the PUT request. Please see the *Completed Upload* section for details on the parameters and expected responses.

Chunked Upload

To carry out an upload of a chunk, the client can specify a range header and only include that part of the layer file:

```
PATCH /v2/<name>/blobs/uploads/<uuid>
Content-Length: <size of chunk>
Content-Range: <start of range>-<end of range>
Content-Type: application/octet-stream

<Layer Chunk Binary Data>
```

There is no enforcement on layer chunk splits other than that the server must receive them in order. The server may enforce a minimum chunk size. If the server cannot accept the chunk, a `416 Requested Range Not Satisfiable` response will be returned and will include a `Range` header indicating the current status:

```
416 Requested Range Not Satisfiable
Location: /v2/<name>/blobs/uploads/<uuid>
Range: 0-<last valid range>
Content-Length: 0
Docker-Upload-UUID: <uuid>
```

If this response is received, the client should resume from the "last valid range" and upload the subsequent chunk. A 416 will be returned under the following conditions:

- Invalid Content-Range header format
- Out of order chunk: the range of the next chunk must start immediately after the "last valid range" from the previous response.

When a chunk is accepted as part of the upload, a `202 Accepted` response will be returned, including a `Range` header with the current upload status:

```
202 Accepted
Location: /v2/<name>/blobs/uploads/<uuid>
Range: bytes=0-<offset>
Content-Length: 0
Docker-Upload-UUID: <uuid>
```

Completed Upload

For an upload to be considered complete, the client must submit a `PUT` request on the upload endpoint with a digest parameter. If it is not provided, the upload will not be considered complete. The format for the final chunk will be as follows:

```
PUT /v2/<name>/blob/uploads/<uuid>?digest=<digest>
Content-Length: <size of chunk>
Content-Range: <start of range>-<end of range>
Content-Type: application/octet-stream

<Last Layer Chunk Binary Data>
```

Optionally, if all chunks have already been uploaded, a `PUT` request with a `digest` parameter and zero-length body may be sent to complete and validated the upload. Multiple "digest" parameters may be provided with different digests. The server may verify none or all of them but *must* notify the client if the content is rejected.

When the last chunk is received and the layer has been validated, the client will receive a `201 Created` response:

```
201 Created
Location: /v2/<name>/blobs/<digest>
Content-Length: 0
Docker-Content-Digest: <digest>
```

The `Location` header will contain the registry URL to access the accepted layer file. The `Docker-Content-Digest` header returns the canonical digest of the uploaded blob which may differ from the provided digest. Most clients may ignore the value but if it is used, the client should verify the value against the uploaded blob data.

Digest Parameter

The “digest” parameter is designed as an opaque parameter to support verification of a successful transfer. For example, an HTTP URI parameter might be as follows:

```
sha256:6c3c624b58dbbcd3c0dd82b4c53f04194d1247c6eebdaab7c610cf7d66709b3
```

Given this parameter, the registry will verify that the provided content does match this digest.

Canceling an Upload

An upload can be cancelled by issuing a DELETE request to the upload endpoint. The format will be as follows:

```
DELETE /v2/<name>/blobs/uploads/<uuid>
```

After this request is issued, the upload uuid will no longer be valid and the registry server will dump all intermediate data. While uploads will time out if not completed, clients should issue this request if they encounter a fatal error but still have the ability to issue an http request.

Cross Repository Blob Mount

A blob may be mounted from another repository that the client has read access to, removing the need to upload a blob already known to the registry. To issue a blob mount instead of an upload, a POST request should be issued in the following format:

```
POST /v2/<name>/blobs/uploads/?mount=<digest>&from=<repository name>
Content-Length: 0
```

If the blob is successfully mounted, the client will receive a `201 Created` response:

```
201 Created
Location: /v2/<name>/blobs/<digest>
Content-Length: 0
Docker-Content-Digest: <digest>
```

The `Location` header will contain the registry URL to access the accepted layer file. The `Docker-Content-Digest` header returns the canonical digest of the uploaded blob which may differ from the provided digest. Most clients may ignore the value but if it is used, the client should verify the value against the uploaded blob data.

If a mount fails due to invalid repository or digest arguments, the registry will fall back to the standard upload behavior and return a `202 Accepted` with the upload URL in the `Location` header:

```
202 Accepted
Location: /v2/<name>/blobs/uploads/<uuid>
Range: bytes=0-<offset>
Content-Length: 0
Docker-Upload-UUID: <uuid>
```

This behavior is consistent with older versions of the registry, which do not recognize the repository mount query parameters.

Note: a client may issue a HEAD request to check existence of a blob in a source repository to distinguish between the registry not supporting blob mounts and the blob not existing in the expected repository.

Errors

If an 502, 503 or 504 error is received, the client should assume that the download can proceed due to a temporary condition, honoring the appropriate retry mechanism. Other 5xx errors should be treated as terminal.

If there is a problem with the upload, a 4xx error will be returned indicating the problem. After receiving a 4xx response (except 416, as called out above), the upload will be considered failed and the client should take appropriate action.

Note that the upload url will not be available forever. If the upload uuid is unknown to the registry, a `404 Not Found` response will be returned and the client must restart the upload process.

Deleting a Layer

A layer may be deleted from the registry via its `name` and `digest`. A delete may be issued with the following request format:

```
DELETE /v2/<name>/blobs/<digest>
```

If the blob exists and has been successfully deleted, the following response will be issued:

```
202 Accepted
Content-Length: None
```

If the blob had already been deleted or did not exist, a `404 Not Found` response will be issued instead.

If a layer is deleted which is referenced by a manifest in the registry, then the complete images will not be resolvable.

Pushing an Image Manifest

Once all of the layers for an image are uploaded, the client can upload the image manifest. An image can be pushed using the following request format:

```
PUT /v2/<name>/manifests/<reference>
Content-Type: <manifest media type>

{
   "name": <name>,
   "tag": <tag>,
   "fsLayers": [
      {
         "blobSum": <digest>
      },
      ...
    ]
   ],
   "history": <v1 images>,
   "signature": <JWS>,
   ...
}
```

The `name` and `reference` fields of the response body must match those specified in the URL. The `reference` field may be a “tag” or a “digest”. The content type should match the type of the manifest being uploaded, as specified in manifest-v2-1.md and manifest-v2-2.md.

If there is a problem with pushing the manifest, a relevant 4xx response will be returned with a JSON error message. Please see the *PUT Manifest* section for details on possible error codes that may be returned.

If one or more layers are unknown to the registry, `BLOB_UNKNOWN` errors are returned. The `detail` field of the error response will have a `digest` field identifying the missing blob. An error is returned for each unknown blob. The response format is as follows:

```
{
    "errors:" [{
            "code": "BLOB_UNKNOWN",
            "message": "blob unknown to registry",
            "detail": {
                "digest": <digest>
```

```
            }
        },
        ...
    ]
}
```

Listing Repositories

Images are stored in collections, known as a *repository*, which is keyed by a `name`, as seen throughout the API specification. A registry instance may contain several repositories. The list of available repositories is made available through the *catalog*.

The catalog for a given registry can be retrieved with the following request:

The response will be in the following format:

```
200 OK
Content-Type: application/json

{
  "repositories": [
    <name>,
    ...
  ]
}
```

Note that the contents of the response are specific to the registry implementation. Some registries may opt to provide a full catalog output, limit it based on the user's access level or omit upstream results, if providing mirroring functionality. Subsequently, the presence of a repository in the catalog listing only means that the registry *may* provide access to the repository at the time of the request. Conversely, a missing entry does *not* mean that the registry does not have the repository. More succinctly, the presence of a repository only guarantees that it is there but not that it is *not* there.

For registries with a large number of repositories, this response may be quite large. If such a response is expected, one should use pagination. A registry may also limit the amount of responses returned even if pagination was not explicitly requested. In this case the `Link` header will be returned along with the results, and subsequent results can be obtained by following the link as if pagination had been initially requested.

For details of the `Link` header, please see the *Pagination* section.

Pagination

Paginated catalog results can be retrieved by adding an `n` parameter to the request URL, declaring that the response should be limited to `n` results. Starting a paginated flow begins as follows:

```
GET /v2/_catalog?n=<integer>
```

The above specifies that a catalog response should be returned, from the start of the result set, ordered lexically, limiting the number of results to `n`. The response to such a request would look as follows:

```
200 OK
Content-Type: application/json
Link: <<url>?n=<n from the request>&last=<last repository in response>

{
  "repositories": [
    <name>,
    ...
  ]
}
```

The above includes the *first* `n` entries from the result set. To get the *next* `n` entries, one can create a URL where the argument `last` has the value from `repositories[len(repositories)-1]`. If there are indeed more results, the

URL for the next block is encoded in an RFC5988 `Link` header, as a "next" relation. The presence of the `Link` header communicates to the client that the entire result set has not been returned and another request must be issued. If the header is not present, the client can assume that all results have been received.

> **NOTE:** In the request template above, note that the brackets are required. For example, if the url is `http://example.com/v2/_catalog?n=20&last=b`, the value of the header would be `<http://example.com/v2/_catalog?n=20&last=b>; rel="next"`. Please see RFC5988 for details.

Compliant client implementations should always use the `Link` header value when proceeding through results linearly. The client may construct URLs to skip forward in the catalog.

To get the next result set, a client would issue the request as follows, using the URL encoded in the described `Link` header:

```
GET /v2/_catalog?n=<n from the request>&last=<last repository value fr
```

The above process should then be repeated until the `Link` header is no longer set.

The catalog result set is represented abstractly as a lexically sorted list, where the position in that list can be specified by the query term `last`. The entries in the response start *after* the term specified by `last`, up to `n` entries.

The behavior of `last` is quite simple when demonstrated with an example. Let us say the registry has the following repositories:

If the value of `n` is `2`, *a* and *b* will be returned on the first response. The `Link` header returned on the response will have `n` set to `2` and last set to *b*:

```
Link: <<url>?n=2&last=b>; rel="next"
```

The client can then issue the request with the above value from the `Link` header, receiving the values *c* and *d*. Note that `n` may change on the second to last response or be fully omitted, depending on the server implementation.

Listing Image Tags

It may be necessary to list all of the tags under a given repository. The tags for an image repository can be retrieved with the following request:

```
GET /v2/<name>/tags/list
```

The response will be in the following format:

```
200 OK
Content-Type: application/json

{
    "name": <name>,
    "tags": [
        <tag>,
        ...
    ]
}
```

For repositories with a large number of tags, this response may be quite large. If such a response is expected, one should use the pagination.

Pagination

Paginated tag results can be retrieved by adding the appropriate parameters to the request URL described above. The behavior of tag pagination is identical to that specified for catalog pagination. We cover a

simple flow to highlight any differences.

Starting a paginated flow may begin as follows:

```
GET /v2/<name>/tags/list?n=<integer>
```

The above specifies that a tags response should be returned, from the start of the result set, ordered lexically, limiting the number of results to `n`. The response to such a request would look as follows:

```
200 OK
Content-Type: application/json
Link: <<url>?n=<n from the request>&last=<last tag value from previous

{
  "name": <name>,
  "tags": [
    <tag>,
    ...
  ]
}
```

To get the next result set, a client would issue the request as follows, using the value encoded in the RFC5988 `Link` header:

```
GET /v2/<name>/tags/list?n=<n from the request>&last=<last tag value f
```

The above process should then be repeated until the `Link` header is no longer set in the response. The behavior of the `last` parameter, the provided response result, lexical ordering and encoding of the `Link` header are identical to that of catalog pagination.

Deleting an Image

An image may be deleted from the registry via its `name` and `reference`. A

delete may be issued with the following request format:

```
DELETE /v2/<name>/manifests/<reference>
```

For deletes, `reference` *must* be a digest or the delete will fail. If the image exists and has been successfully deleted, the following response will be issued:

```
202 Accepted
Content-Length: None
```

If the image had already been deleted or did not exist, a `404 Not Found` response will be issued instead.

> **Note** When deleting a manifest from a registry version 2.3 or later, the following header must be used when HEAD or GET-ing the manifest to obtain the correct digest to delete:

```
Accept: application/vnd.docker.distribution.manifest.v2+json
```

> for more details, see: compatibility.md

Detail

> **Note**: This section is still under construction. For the purposes of implementation, if any details below differ from the described request flows above, the section below should be corrected. When they match, this note should be removed.

The behavior of the endpoints are covered in detail in this section, organized by route and entity. All aspects of the request and responses are covered, including headers, parameters and body formats. Examples of requests and their corresponding responses, with success and failure, are

enumerated.

> **Note**: The sections on endpoint detail are arranged with an example request, a description of the request, followed by information about that request.

A list of methods and URIs are covered in the table below:

Method	Path	Entity	Description
GET	`/v2/`	Base	Check that the endpoint implements Docker Registry API V2.
GET	`/v2/<name>/tags/list`	Tags	Fetch the tags under the repository identified by `name`.
GET	`/v2/<name>/manifests/<reference>`	Manifest	Fetch the manifest identified by `name` and `reference` where `reference` can be a tag or digest. A `HEAD` request can also be issued to this endpoint to obtain resource information without receiving all data.
PUT	`/v2/<name>/manifests/<reference>`	Manifest	Put the manifest identified by `name` and `reference` where `reference` can be a tag or digest.
DELETE	`/v2/<name>/manifests/<reference>`	Manifest	Delete the manifest identified by `name` and `reference`. Note that a manifest can *only* be deleted by `digest`.

GET	`/v2/<name>/blobs/<digest>`	Blob	Retrieve the blob from the registry identified by `digest`. A `HEAD` request can also be issued to this endpoint to obtain resource information without receiving all data.
DELETE	`/v2/<name>/blobs/<digest>`	Blob	Delete the blob identified by `name` and `digest`
POST	`/v2/<name>/blobs/uploads/`	Initiate Blob Upload	Initiate a resumable blob upload. If successful, an upload location will be provided to complete the upload. Optionally, if the `digest` parameter is present, the request body will be used to complete the upload in a single request.
GET	`/v2/<name>/blobs/uploads/<uuid>`	Blob Upload	Retrieve status of upload identified by `uuid`. The primary purpose of this endpoint is to resolve the current status of a resumable upload.
PATCH	`/v2/<name>/blobs/uploads/<uuid>`	Blob Upload	Upload a chunk of data for the specified upload.
PUT	`/v2/<name>/blobs/uploads/<uuid>`	Blob Upload	Complete the upload specified by `uuid`, optionally appending the body as the final chunk.

DELETE	`/v2/<name>/blobs/uploads/<uuid>`	Blob Upload	Cancel outstanding upload processes, releasing associated resources. If this is not called, the unfinished uploads will eventually timeout.
GET	`/v2/_catalog`	Catalog	Retrieve a sorted, json list of repositories available in the registry.

The detail for each endpoint is covered in the following sections.

Errors

The error codes encountered via the API are enumerated in the following table:

Code	Message	Description
`BLOB_UNKNOWN`	blob unknown to registry	This error may be returned when a blob is unknown to the registry in a specified repository. This can be returned with a standard get or if a manifest references an unknown layer during upload.
`BLOB_UPLOAD_INVALID`	blob upload invalid	The blob upload encountered an error and can no longer proceed.
`BLOB_UPLOAD_UNKNOWN`	blob upload unknown to registry	If a blob upload has been cancelled or was never started, this error code may be returned.
`DIGEST_INVALID`	provided digest did not match uploaded content	When a blob is uploaded, the registry will check that the content matches the digest provided by the client. The error may include a detail structure with the key “digest”, including the invalid digest string. This error may also be returned when a manifest includes an invalid layer digest.

MANIFEST_BLOB_UNKNOWN	blob unknown to registry	This error may be returned when a manifest blob is unknown to the registry.
MANIFEST_INVALID	manifest invalid	During upload, manifests undergo several checks ensuring validity. If those checks fail, this error may be returned, unless a more specific error is included. The detail will contain information the failed validation.
MANIFEST_UNKNOWN	manifest unknown	This error is returned when the manifest, identified by name and tag is unknown to the repository.
MANIFEST_UNVERIFIED	manifest failed signature verification	During manifest upload, if the manifest fails signature verification, this error will be returned.
NAME_INVALID	invalid repository name	Invalid repository name encountered either during manifest validation or any API operation.
NAME_UNKNOWN	repository name not known to registry	This is returned if the name used during an operation is unknown to the registry.
SIZE_INVALID	provided length did not match content length	When a layer is uploaded, the provided size will be checked against the uploaded content. If they do not match, this error will be returned.
TAG_INVALID	manifest tag did not match URI	During a manifest upload, if the tag in the manifest does not match the uri tag, this error will be returned.
UNAUTHORIZED	authentication required	The access controller was unable to authenticate the client. Often this will be accompanied by a Www-Authenticate HTTP response header indicating how to authenticate.
DENIED	requested access to the resource is denied	The access controller denied access for the operation on a resource.
	The operation	The operation was unsupported due

`UNSUPPORTED`	is unsupported.	to a missing implementation or invalid set of parameters.

Base

Base V2 API route. Typically, this can be used for lightweight version checks and to validate registry authentication.

GET Base

Check that the endpoint implements Docker Registry API V2.

```
GET /v2/
Host: <registry host>
Authorization: <scheme> <token>
```

The following parameters should be specified on the request:

Name	Kind	Description
`Host`	header	Standard HTTP Host Header. Should be set to the registry host.
`Authorization`	header	An RFC7235 compliant authorization header.

On Success: OK

The API implements V2 protocol and is accessible.

On Failure: Not Found

The registry does not implement the V2 API.

On Failure: Authentication Required

```
401 Unauthorized
WWW-Authenticate: <scheme> realm="<realm>", ..."
Content-Length: <length>
Content-Type: application/json; charset=utf-8
```

```
{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The client is not authenticated.

The following headers will be returned on the response:

Name	Description
`WWW-Authenticate`	An RFC7235 compliant authentication challenge header.
`Content-Length`	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`UNAUTHORIZED`	authentication required	The access controller was unable to authenticate the client. Often this will be accompanied by a Www-Authenticate HTTP response header indicating how to authenticate.

Tags

Retrieve information about tags.

GET Tags

Fetch the tags under the repository identified by `name`.

Tags

```
GET /v2/<name>/tags/list
Host: <registry host>
Authorization: <scheme> <token>
```

Return all tags for the repository

The following parameters should be specified on the request:

Name	Kind	Description
`Host`	header	Standard HTTP Host Header. Should be set to the registry host.
`Authorization`	header	An RFC7235 compliant authorization header.
`name`	path	Name of the target repository.

On Success: OK

```
200 OK
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
    "name": <name>,
    "tags": [
        <tag>,
        ...
    ]
}
```

A list of tags for the named repository.

The following headers will be returned with the response:

Name	Description
`Content-Length`	Length of the JSON response body.

On Failure: Authentication Required

```
401 Unauthorized
WWW-Authenticate: <scheme> realm="<realm>", ..."
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The client is not authenticated.

The following headers will be returned on the response:

Name	Description
`WWW-Authenticate`	An RFC7235 compliant authentication challenge header.
`Content-Length`	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`UNAUTHORIZED`	authentication required	The access controller was unable to authenticate the client. Often this will be accompanied by a Www-Authenticate HTTP response header indicating how to authenticate.

On Failure: No Such Repository Error

```
404 Not Found
Content-Length: <length>
Content-Type: application/json; charset=utf-8
```

```
{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The repository is not known to the registry.

The following headers will be returned on the response:

Name	Description
`Content-Length`	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`NAME_UNKNOWN`	repository name not known to registry	This is returned if the name used during an operation is unknown to the registry.

On Failure: Access Denied

```
403 Forbidden
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
```

```
        ...
    ]
}
```

The client does not have required access to the repository.

The following headers will be returned on the response:

Name	Description
`Content-Length`	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`DENIED`	requested access to the resource is denied	The access controller denied access for the operation on a resource.

Tags Paginated

```
GET /v2/<name>/tags/list?n=<integer>&last=<integer>
```

Return a portion of the tags for the specified repository.

The following parameters should be specified on the request:

Name	Kind	Description
`name`	path	Name of the target repository.
`n`	query	Limit the number of entries in each response. It not present, all entries will be returned.
`last`	query	Result set will include values lexically after last.

On Success: OK

```
200 OK
```

```
Content-Length: <length>
Link: <<url>?n=<last n value>&last=<last entry from response>>; rel="n
Content-Type: application/json; charset=utf-8

{
    "name": <name>,
    "tags": [
        <tag>,
        ...
    ],
}
```

A list of tags for the named repository.

The following headers will be returned with the response:

Name	Description
`Content-Length`	Length of the JSON response body.
`Link`	RFC5988 compliant rel='next' with URL to next result set, if available

On Failure: Authentication Required

```
401 Unauthorized
WWW-Authenticate: <scheme> realm="<realm>", ..."
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The client is not authenticated.

The following headers will be returned on the response:

Name	Description
`WWW-Authenticate`	An RFC7235 compliant authentication challenge header.
`Content-Length`	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`UNAUTHORIZED`	authentication required	The access controller was unable to authenticate the client. Often this will be accompanied by a Www-Authenticate HTTP response header indicating how to authenticate.

On Failure: No Such Repository Error

```
404 Not Found
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The repository is not known to the registry.

The following headers will be returned on the response:

Name	Description
`Content-Length`	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`NAME_UNKNOWN`	repository name not known to registry	This is returned if the name used during an operation is unknown to the registry.

On Failure: Access Denied

```
403 Forbidden
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The client does not have required access to the repository.

The following headers will be returned on the response:

Name	Description
`Content-Length`	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
DENIED	requested access to the resource is denied	The access controller denied access for the operation on a resource.

Manifest

Create, update, delete and retrieve manifests.

GET Manifest

Fetch the manifest identified by `name` and `reference` where `reference` can be a tag or digest. A `HEAD` request can also be issued to this endpoint to obtain resource information without receiving all data.

```
GET /v2/<name>/manifests/<reference>
Host: <registry host>
Authorization: <scheme> <token>
```

The following parameters should be specified on the request:

Name	Kind	Description
Host	header	Standard HTTP Host Header. Should be set to the registry host.
Authorization	header	An RFC7235 compliant authorization header.
name	path	Name of the target repository.
reference	path	Tag or digest of the target manifest.

On Success: OK

```
200 OK
Docker-Content-Digest: <digest>
Content-Type: <media type of manifest>

{
   "name": <name>,
   "tag": <tag>,
```

```
   "fsLayers": [
      {
         "blobSum": "<digest>"
      },
      ...
    ]
   ],
   "history": <v1 images>,
   "signature": <JWS>
}
```

The manifest identified by `name` and `reference`. The contents can be used to identify and resolve resources required to run the specified image.

The following headers will be returned with the response:

Name	Description
`Docker-Content-Digest`	Digest of the targeted content for the request.

On Failure: Bad Request

```
400 Bad Request
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The name or reference was invalid.

The error codes that may be included in the response body are enumerated

below:

Code	Message	Description
`NAME_INVALID`	invalid repository name	Invalid repository name encountered either during manifest validation or any API operation.
`TAG_INVALID`	manifest tag did not match URI	During a manifest upload, if the tag in the manifest does not match the uri tag, this error will be returned.

On Failure: Authentication Required

```
401 Unauthorized
WWW-Authenticate: <scheme> realm="<realm>", ..."
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The client is not authenticated.

The following headers will be returned on the response:

Name	Description
`WWW-Authenticate`	An RFC7235 compliant authentication challenge header.
`Content-Length`	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
UNAUTHORIZED	authentication required	The access controller was unable to authenticate the client. Often this will be accompanied by a Www-Authenticate HTTP response header indicating how to authenticate.

On Failure: No Such Repository Error

```
404 Not Found
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The repository is not known to the registry.

The following headers will be returned on the response:

Name	Description
Content-Length	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
NAME_UNKNOWN	repository name not known to registry	This is returned if the name used during an operation is unknown to the registry.

On Failure: Access Denied

```
403 Forbidden
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The client does not have required access to the repository.

The following headers will be returned on the response:

Name	Description
`Content-Length`	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`DENIED`	requested access to the resource is denied	The access controller denied access for the operation on a resource.

PUT Manifest

Put the manifest identified by `name` and `reference` where `reference` can be a tag or digest.

```
PUT /v2/<name>/manifests/<reference>
```

```
Host: <registry host>
Authorization: <scheme> <token>
Content-Type: <media type of manifest>

{
   "name": <name>,
   "tag": <tag>,
   "fsLayers": [
      {
         "blobSum": "<digest>"
      },
      ...
    ]
   ],
   "history": <v1 images>,
   "signature": <JWS>
}
```

The following parameters should be specified on the request:

Name	Kind	Description
`Host`	header	Standard HTTP Host Header. Should be set to the registry host.
`Authorization`	header	An RFC7235 compliant authorization header.
`name`	path	Name of the target repository.
`reference`	path	Tag or digest of the target manifest.

On Success: Created

```
201 Created
Location: <url>
Content-Length: 0
Docker-Content-Digest: <digest>
```

The manifest has been accepted by the registry and is stored under the specified `name` and `tag`.

The following headers will be returned with the response:

Name	Description
`Location`	The canonical location url of the uploaded manifest.
`Content-Length`	The `Content-Length` header must be zero and the body must be empty.
`Docker-Content-Digest`	Digest of the targeted content for the request.

On Failure: Invalid Manifest

```
400 Bad Request
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The received manifest was invalid in some way, as described by the error codes. The client should resolve the issue and retry the request.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`NAME_INVALID`	invalid repository name	Invalid repository name encountered either during manifest validation or any API operation.
`TAG_INVALID`	manifest tag did not match URI	During a manifest upload, if the tag in the manifest does not match the uri tag, this error will be returned.
		During upload, manifests undergo several checks ensuring validity. If those checks

MANIFEST_INVALID	manifest invalid	fail, this error may be returned, unless a more specific error is included. The detail will contain information the failed validation.
MANIFEST_UNVERIFIED	manifest failed signature verification	During manifest upload, if the manifest fails signature verification, this error will be returned.
BLOB_UNKNOWN	blob unknown to registry	This error may be returned when a blob is unknown to the registry in a specified repository. This can be returned with a standard get or if a manifest references an unknown layer during upload.

On Failure: Authentication Required

```
401 Unauthorized
WWW-Authenticate: <scheme> realm="<realm>", ..."
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The client is not authenticated.

The following headers will be returned on the response:

Name	Description
WWW-Authenticate	An RFC7235 compliant authentication challenge header.
Content-Length	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
UNAUTHORIZED	authentication required	The access controller was unable to authenticate the client. Often this will be accompanied by a Www-Authenticate HTTP response header indicating how to authenticate.

On Failure: No Such Repository Error

```
404 Not Found
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The repository is not known to the registry.

The following headers will be returned on the response:

Name	Description
Content-Length	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
	repository name	

NAME_UNKNOWN	not known to registry	This is returned if the name used during an operation is unknown to the registry.

On Failure: Access Denied

```
403 Forbidden
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The client does not have required access to the repository.

The following headers will be returned on the response:

Name	Description
Content-Length	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
DENIED	requested access to the resource is denied	The access controller denied access for the operation on a resource.

On Failure: Missing Layer(s)

```
400 Bad Request
```

```
Content-Type: application/json; charset=utf-8

{
    "errors:" [{
            "code": "BLOB_UNKNOWN",
            "message": "blob unknown to registry",
            "detail": {
                "digest": "<digest>"
            }
        },
        ...
    ]
}
```

One or more layers may be missing during a manifest upload. If so, the missing layers will be enumerated in the error response.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
BLOB_UNKNOWN	blob unknown to registry	This error may be returned when a blob is unknown to the registry in a specified repository. This can be returned with a standard get or if a manifest references an unknown layer during upload.

On Failure: Not allowed

Manifest put is not allowed because the registry is configured as a pull-through cache or for some other reason

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
UNSUPPORTED	The operation is unsupported.	The operation was unsupported due to a missing implementation or invalid set of parameters.

DELETE Manifest

Delete the manifest identified by `name` and `reference`. Note that a manifest can *only* be deleted by `digest`.

```
DELETE /v2/<name>/manifests/<reference>
Host: <registry host>
Authorization: <scheme> <token>
```

The following parameters should be specified on the request:

Name	Kind	Description
`Host`	header	Standard HTTP Host Header. Should be set to the registry host.
`Authorization`	header	An RFC7235 compliant authorization header.
`name`	path	Name of the target repository.
`reference`	path	Tag or digest of the target manifest.

On Success: Accepted

On Failure: Invalid Name or Reference

```
400 Bad Request
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The specified `name` or `reference` were invalid and the delete was unable to

proceed.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`NAME_INVALID`	invalid repository name	Invalid repository name encountered either during manifest validation or any API operation.
`TAG_INVALID`	manifest tag did not match URI	During a manifest upload, if the tag in the manifest does not match the uri tag, this error will be returned.

On Failure: Authentication Required

```
401 Unauthorized
WWW-Authenticate: <scheme> realm="<realm>", ..."
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The client is not authenticated.

The following headers will be returned on the response:

Name	Description
`WWW-Authenticate`	An RFC7235 compliant authentication challenge header.
`Content-Length`	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
UNAUTHORIZED	authentication required	The access controller was unable to authenticate the client. Often this will be accompanied by a Www-Authenticate HTTP response header indicating how to authenticate.

On Failure: No Such Repository Error

```
404 Not Found
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The repository is not known to the registry.

The following headers will be returned on the response:

Name	Description
Content-Length	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
	repository name	

`NAME_UNKNOWN`	not known to registry	This is returned if the name used during an operation is unknown to the registry.

On Failure: Access Denied

```
403 Forbidden
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The client does not have required access to the repository.

The following headers will be returned on the response:

Name	Description
`Content-Length`	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`DENIED`	requested access to the resource is denied	The access controller denied access for the operation on a resource.

On Failure: Unknown Manifest

```
404 Not Found
```

```
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The specified `name` or `reference` are unknown to the registry and the delete was unable to proceed. Clients can assume the manifest was already deleted if this response is returned.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`NAME_UNKNOWN`	repository name not known to registry	This is returned if the name used during an operation is unknown to the registry.
`MANIFEST_UNKNOWN`	manifest unknown	This error is returned when the manifest, identified by name and tag is unknown to the repository.

On Failure: Not allowed

Manifest delete is not allowed because the registry is configured as a pull-through cache or `delete` has been disabled.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description

UNSUPPORTED	The operation is unsupported.	The operation was unsupported due to a missing implementation or invalid set of parameters.

Blob

Operations on blobs identified by `name` and `digest`. Used to fetch or delete layers by digest.

GET Blob

Retrieve the blob from the registry identified by `digest`. A `HEAD` request can also be issued to this endpoint to obtain resource information without receiving all data.

Fetch Blob

```
GET /v2/<name>/blobs/<digest>
Host: <registry host>
Authorization: <scheme> <token>
```

The following parameters should be specified on the request:

Name	Kind	Description
Host	header	Standard HTTP Host Header. Should be set to the registry host.
Authorization	header	An RFC7235 compliant authorization header.
name	path	Name of the target repository.
digest	path	Digest of desired blob.

On Success: OK

```
200 OK
Content-Length: <length>
Docker-Content-Digest: <digest>
Content-Type: application/octet-stream
```

```
<blob binary data>
```

The blob identified by `digest` is available. The blob content will be present in the body of the request.

The following headers will be returned with the response:

Name	Description
`Content-Length`	The length of the requested blob content.
`Docker-Content-Digest`	Digest of the targeted content for the request.

On Success: Temporary Redirect

```
307 Temporary Redirect
Location: <blob location>
Docker-Content-Digest: <digest>
```

The blob identified by `digest` is available at the provided location.

The following headers will be returned with the response:

Name	Description
`Location`	The location where the layer should be accessible.
`Docker-Content-Digest`	Digest of the targeted content for the request.

On Failure: Bad Request

```
400 Bad Request
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
```

```
            "detail": ...
        },
        ...
    ]
}
```

There was a problem with the request that needs to be addressed by the client, such as an invalid `name` or `tag`.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`NAME_INVALID`	invalid repository name	Invalid repository name encountered either during manifest validation or any API operation.
`DIGEST_INVALID`	provided digest did not match uploaded content	When a blob is uploaded, the registry will check that the content matches the digest provided by the client. The error may include a detail structure with the key “digest”, including the invalid digest string. This error may also be returned when a manifest includes an invalid layer digest.

On Failure: Not Found

```
404 Not Found
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The blob, identified by `name` and `digest`, is unknown to the registry.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`NAME_UNKNOWN`	repository name not known to registry	This is returned if the name used during an operation is unknown to the registry.
`BLOB_UNKNOWN`	blob unknown to registry	This error may be returned when a blob is unknown to the registry in a specified repository. This can be returned with a standard get or if a manifest references an unknown layer during upload.

On Failure: Authentication Required

```
401 Unauthorized
WWW-Authenticate: <scheme> realm="<realm>", ..."
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The client is not authenticated.

The following headers will be returned on the response:

Name	Description
`WWW-Authenticate`	An RFC7235 compliant authentication challenge header.

`Content-Length`	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`UNAUTHORIZED`	authentication required	The access controller was unable to authenticate the client. Often this will be accompanied by a Www-Authenticate HTTP response header indicating how to authenticate.

On Failure: No Such Repository Error

```
404 Not Found
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The repository is not known to the registry.

The following headers will be returned on the response:

Name	Description
`Content-Length`	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
NAME_UNKNOWN	repository name not known to registry	This is returned if the name used during an operation is unknown to the registry.

On Failure: Access Denied

```
403 Forbidden
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The client does not have required access to the repository.

The following headers will be returned on the response:

Name	Description
Content-Length	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
DENIED	requested access to the resource is denied	The access controller denied access for the operation on a resource.

Fetch Blob Part

```
GET /v2/<name>/blobs/<digest>
Host: <registry host>
Authorization: <scheme> <token>
Range: bytes=<start>-<end>
```

This endpoint may also support RFC7233 compliant range requests. Support can be detected by issuing a HEAD request. If the header `Accept-Range: bytes` is returned, range requests can be used to fetch partial content.

The following parameters should be specified on the request:

Name	Kind	Description
`Host`	header	Standard HTTP Host Header. Should be set to the registry host.
`Authorization`	header	An RFC7235 compliant authorization header.
`Range`	header	HTTP Range header specifying blob chunk.
`name`	path	Name of the target repository.
`digest`	path	Digest of desired blob.

On Success: Partial Content

```
206 Partial Content
Content-Length: <length>
Content-Range: bytes <start>-<end>/<size>
Content-Type: application/octet-stream

<blob binary data>
```

The blob identified by `digest` is available. The specified chunk of blob content will be present in the body of the request.

The following headers will be returned with the response:

Name	Description

`Content-Length`	The length of the requested blob chunk.
`Content-Range`	Content range of blob chunk.

On Failure: Bad Request

```
400 Bad Request
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

There was a problem with the request that needs to be addressed by the client, such as an invalid `name` or `tag`.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`NAME_INVALID`	invalid repository name	Invalid repository name encountered either during manifest validation or any API operation.
`DIGEST_INVALID`	provided digest did not match uploaded content	When a blob is uploaded, the registry will check that the content matches the digest provided by the client. The error may include a detail structure with the key “digest”, including the invalid digest string. This error may also be returned when a manifest includes an invalid layer digest.

On Failure: Not Found

```
404 Not Found
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
NAME_UNKNOWN	repository name not known to registry	This is returned if the name used during an operation is unknown to the registry.
BLOB_UNKNOWN	blob unknown to registry	This error may be returned when a blob is unknown to the registry in a specified repository. This can be returned with a standard get or if a manifest references an unknown layer during upload.

On Failure: Requested Range Not Satisfiable

```
416 Requested Range Not Satisfiable
```

The range specification cannot be satisfied for the requested content. This can happen when the range is not formatted correctly or if the range is outside of the valid size of the content.

On Failure: Authentication Required

```
401 Unauthorized
WWW-Authenticate: <scheme> realm="<realm>", ..."
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The client is not authenticated.

The following headers will be returned on the response:

Name	Description
`WWW-Authenticate`	An RFC7235 compliant authentication challenge header.
`Content-Length`	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`UNAUTHORIZED`	authentication required	The access controller was unable to authenticate the client. Often this will be accompanied by a Www-Authenticate HTTP response header indicating how to authenticate.

On Failure: No Such Repository Error

```
404 Not Found
Content-Length: <length>
Content-Type: application/json; charset=utf-8
```

```
{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The repository is not known to the registry.

The following headers will be returned on the response:

Name	Description
`Content-Length`	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`NAME_UNKNOWN`	repository name not known to registry	This is returned if the name used during an operation is unknown to the registry.

On Failure: Access Denied

```
403 Forbidden
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
```

```
        },
        ...
    ]
}
```

The client does not have required access to the repository.

The following headers will be returned on the response:

Name	Description
`Content-Length`	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`DENIED`	requested access to the resource is denied	The access controller denied access for the operation on a resource.

DELETE Blob

Delete the blob identified by `name` and `digest`

```
DELETE /v2/<name>/blobs/<digest>
Host: <registry host>
Authorization: <scheme> <token>
```

The following parameters should be specified on the request:

Name	Kind	Description
`Host`	header	Standard HTTP Host Header. Should be set to the registry host.
`Authorization`	header	An RFC7235 compliant authorization header.
`name`	path	Name of the target repository.
`digest`	path	Digest of desired blob.

On Success: Accepted

```
202 Accepted
Content-Length: 0
Docker-Content-Digest: <digest>
```

The following headers will be returned with the response:

Name	Description
`Content-Length`	0
`Docker-Content-Digest`	Digest of the targeted content for the request.

On Failure: Invalid Name or Digest

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`DIGEST_INVALID`	provided digest did not match uploaded content	When a blob is uploaded, the registry will check that the content matches the digest provided by the client. The error may include a detail structure with the key “digest”, including the invalid digest string. This error may also be returned when a manifest includes an invalid layer digest.
`NAME_INVALID`	invalid repository name	Invalid repository name encountered either during manifest validation or any API operation.

On Failure: Not Found

```
404 Not Found
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
```

```
            "detail": ...
        },
        ...
    ]
}
```

The blob, identified by `name` and `digest`, is unknown to the registry.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`NAME_UNKNOWN`	repository name not known to registry	This is returned if the name used during an operation is unknown to the registry.
`BLOB_UNKNOWN`	blob unknown to registry	This error may be returned when a blob is unknown to the registry in a specified repository. This can be returned with a standard get or if a manifest references an unknown layer during upload.

On Failure: Method Not Allowed

```
405 Method Not Allowed
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

Blob delete is not allowed because the registry is configured as a pull-through cache or `delete` has been disabled

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`UNSUPPORTED`	The operation is unsupported.	The operation was unsupported due to a missing implementation or invalid set of parameters.

On Failure: Authentication Required

```
401 Unauthorized
WWW-Authenticate: <scheme> realm="<realm>", ..."
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The client is not authenticated.

The following headers will be returned on the response:

Name	Description
`WWW-Authenticate`	An RFC7235 compliant authentication challenge header.
`Content-Length`	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description

`UNAUTHORIZED`	authentication required	The access controller was unable to authenticate the client. Often this will be accompanied by a Www-Authenticate HTTP response header indicating how to authenticate.

On Failure: No Such Repository Error

```
404 Not Found
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The repository is not known to the registry.

The following headers will be returned on the response:

Name	Description
`Content-Length`	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`NAME_UNKNOWN`	repository name not known to registry	This is returned if the name used during an operation is unknown to the registry.

On Failure: Access Denied

```
403 Forbidden
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The client does not have required access to the repository.

The following headers will be returned on the response:

Name	Description
`Content-Length`	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`DENIED`	requested access to the resource is denied	The access controller denied access for the operation on a resource.

Initiate Blob Upload

Initiate a blob upload. This endpoint can be used to create resumable uploads or monolithic uploads.

POST Initiate Blob Upload

Initiate a resumable blob upload. If successful, an upload location will be provided to complete the upload. Optionally, if the `digest` parameter is

present, the request body will be used to complete the upload in a single request.

Initiate Monolithic Blob Upload

```
POST /v2/<name>/blobs/uploads/?digest=<digest>
Host: <registry host>
Authorization: <scheme> <token>
Content-Length: <length of blob>
Content-Type: application/octect-stream

<binary data>
```

Upload a blob identified by the `digest` parameter in single request. This upload will not be resumable unless a recoverable error is returned.

The following parameters should be specified on the request:

Name	Kind	Description
`Host`	header	Standard HTTP Host Header. Should be set to the registry host.
`Authorization`	header	An RFC7235 compliant authorization header.
`Content-Length`	header	
`name`	path	Name of the target repository.
`digest`	query	Digest of uploaded blob. If present, the upload will be completed, in a single request, with contents of the request body as the resulting blob.

On Success: Created

```
201 Created
Location: <blob location>
Content-Length: 0
Docker-Upload-UUID: <uuid>
```

The blob has been created in the registry and is available at the provided location.

The following headers will be returned with the response:

Name	Description
`Location`	
`Content-Length`	The `Content-Length` header must be zero and the body must be empty.
`Docker-Upload-UUID`	Identifies the docker upload uuid for the current request.

On Failure: Invalid Name or Digest

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`DIGEST_INVALID`	provided digest did not match uploaded content	When a blob is uploaded, the registry will check that the content matches the digest provided by the client. The error may include a detail structure with the key “digest”, including the invalid digest string. This error may also be returned when a manifest includes an invalid layer digest.
`NAME_INVALID`	invalid repository name	Invalid repository name encountered either during manifest validation or any API operation.

On Failure: Not allowed

Blob upload is not allowed because the registry is configured as a pull-through cache or for some other reason

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
	The operation	

`UNSUPPORTED`	is unsupported.	The operation was unsupported due to a missing implementation or invalid set of parameters.

On Failure: Authentication Required

```
401 Unauthorized
WWW-Authenticate: <scheme> realm="<realm>", ..."
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The client is not authenticated.

The following headers will be returned on the response:

Name	Description
`WWW-Authenticate`	An RFC7235 compliant authentication challenge header.
`Content-Length`	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`UNAUTHORIZED`	authentication required	The access controller was unable to authenticate the client. Often this will be accompanied by a Www-Authenticate HTTP response header indicating how to authenticate.

On Failure: No Such Repository Error

```
404 Not Found
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The repository is not known to the registry.

The following headers will be returned on the response:

Name	Description
`Content-Length`	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`NAME_UNKNOWN`	repository name not known to registry	This is returned if the name used during an operation is unknown to the registry.

On Failure: Access Denied

```
403 Forbidden
Content-Length: <length>
Content-Type: application/json; charset=utf-8
```

```
{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The client does not have required access to the repository.

The following headers will be returned on the response:

Name	Description
`Content-Length`	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`DENIED`	requested access to the resource is denied	The access controller denied access for the operation on a resource.

Initiate Resumable Blob Upload

```
POST /v2/<name>/blobs/uploads/
Host: <registry host>
Authorization: <scheme> <token>
Content-Length: 0
```

Initiate a resumable blob upload with an empty request body.

The following parameters should be specified on the request:

Name	Kind	Description

`Host`	header	Standard HTTP Host Header. Should be set to the registry host.
`Authorization`	header	An RFC7235 compliant authorization header.
`Content-Length`	header	The `Content-Length` header must be zero and the body must be empty.
`name`	path	Name of the target repository.

On Success: Accepted

```
202 Accepted
Content-Length: 0
Location: /v2/<name>/blobs/uploads/<uuid>
Range: 0-0
Docker-Upload-UUID: <uuid>
```

The upload has been created. The `Location` header must be used to complete the upload. The response should be identical to a `GET` request on the contents of the returned `Location` header.

The following headers will be returned with the response:

Name	Description
`Content-Length`	The `Content-Length` header must be zero and the body must be empty.
`Location`	The location of the created upload. Clients should use the contents verbatim to complete the upload, adding parameters where required.
`Range`	Range header indicating the progress of the upload. When starting an upload, it will return an empty range, since no content has been received.
`Docker-Upload-UUID`	Identifies the docker upload uuid for the current request.

On Failure: Invalid Name or Digest

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
DIGEST_INVALID	provided digest did not match uploaded content	When a blob is uploaded, the registry will check that the content matches the digest provided by the client. The error may include a detail structure with the key “digest”, including the invalid digest string. This error may also be returned when a manifest includes an invalid layer digest.
NAME_INVALID	invalid repository name	Invalid repository name encountered either during manifest validation or any API operation.

On Failure: Authentication Required

```
401 Unauthorized
WWW-Authenticate: <scheme> realm="<realm>", ..."
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The client is not authenticated.

The following headers will be returned on the response:

Name	Description
WWW-Authenticate	An RFC7235 compliant authentication challenge header.
Content-Length	Length of the JSON response body.

The error codes that may be included in the response body are enumerated

below:

Code	Message	Description
`UNAUTHORIZED`	authentication required	The access controller was unable to authenticate the client. Often this will be accompanied by a Www-Authenticate HTTP response header indicating how to authenticate.

On Failure: No Such Repository Error

```
404 Not Found
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The repository is not known to the registry.

The following headers will be returned on the response:

Name	Description
`Content-Length`	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`NAME_UNKNOWN`	repository name not known to	This is returned if the name used during an

	registry	operation is unknown to the registry.

On Failure: Access Denied

```
403 Forbidden
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The client does not have required access to the repository.

The following headers will be returned on the response:

Name	Description
`Content-Length`	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`DENIED`	requested access to the resource is denied	The access controller denied access for the operation on a resource.

Mount Blob

```
POST /v2/<name>/blobs/uploads/?mount=<digest>&from=<repository name>
Host: <registry host>
```

```
Authorization: <scheme> <token>
Content-Length: 0
```

Mount a blob identified by the `mount` parameter from another repository.

The following parameters should be specified on the request:

Name	Kind	Description
`Host`	header	Standard HTTP Host Header. Should be set to the registry host.
`Authorization`	header	An RFC7235 compliant authorization header.
`Content-Length`	header	The `Content-Length` header must be zero and the body must be empty.
`name`	path	Name of the target repository.
`mount`	query	Digest of blob to mount from the source repository.
`from`	query	Name of the source repository.

On Success: Created

```
201 Created
Location: <blob location>
Content-Length: 0
Docker-Upload-UUID: <uuid>
```

The blob has been mounted in the repository and is available at the provided location.

The following headers will be returned with the response:

Name	Description
`Location`	
`Content-Length`	The `Content-Length` header must be zero and the body must be empty.
`Docker-Upload-UUID`	Identifies the docker upload uuid for the current request.

On Failure: Invalid Name or Digest

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
DIGEST_INVALID	provided digest did not match uploaded content	When a blob is uploaded, the registry will check that the content matches the digest provided by the client. The error may include a detail structure with the key “digest”, including the invalid digest string. This error may also be returned when a manifest includes an invalid layer digest.
NAME_INVALID	invalid repository name	Invalid repository name encountered either during manifest validation or any API operation.

On Failure: Not allowed

Blob mount is not allowed because the registry is configured as a pull-through cache or for some other reason

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
UNSUPPORTED	The operation is unsupported.	The operation was unsupported due to a missing implementation or invalid set of parameters.

On Failure: Authentication Required

```
401 Unauthorized
WWW-Authenticate: <scheme> realm="<realm>", ..."
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
```

```
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The client is not authenticated.

The following headers will be returned on the response:

Name	Description
`WWW-Authenticate`	An RFC7235 compliant authentication challenge header.
`Content-Length`	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`UNAUTHORIZED`	authentication required	The access controller was unable to authenticate the client. Often this will be accompanied by a Www-Authenticate HTTP response header indicating how to authenticate.

On Failure: No Such Repository Error

```
404 Not Found
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
```

```
        ...
    ]
}
```

The repository is not known to the registry.

The following headers will be returned on the response:

Name	Description
`Content-Length`	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`NAME_UNKNOWN`	repository name not known to registry	This is returned if the name used during an operation is unknown to the registry.

On Failure: Access Denied

```
403 Forbidden
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The client does not have required access to the repository.

The following headers will be returned on the response:

Name	Description
`Content-Length`	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`DENIED`	requested access to the resource is denied	The access controller denied access for the operation on a resource.

Blob Upload

Interact with blob uploads. Clients should never assemble URLs for this endpoint and should only take it through the `Location` header on related API requests. The `Location` header and its parameters should be preserved by clients, using the latest value returned via upload related API calls.

GET Blob Upload

Retrieve status of upload identified by `uuid`. The primary purpose of this endpoint is to resolve the current status of a resumable upload.

```
GET /v2/<name>/blobs/uploads/<uuid>
Host: <registry host>
Authorization: <scheme> <token>
```

Retrieve the progress of the current upload, as reported by the `Range` header.

The following parameters should be specified on the request:

Name	Kind	Description
`Host`	header	Standard HTTP Host Header. Should be set to the registry host.

`Authorization`	header	An RFC7235 compliant authorization header.
`name`	path	Name of the target repository.
`uuid`	path	A uuid identifying the upload. This field can accept characters that match `[a-zA-Z0-9-_.=]+`.

On Success: Upload Progress

```
204 No Content
Range: 0-<offset>
Content-Length: 0
Docker-Upload-UUID: <uuid>
```

The upload is known and in progress. The last received offset is available in the `Range` header.

The following headers will be returned with the response:

Name	Description
`Range`	Range indicating the current progress of the upload.
`Content-Length`	The `Content-Length` header must be zero and the body must be empty.
`Docker-Upload-UUID`	Identifies the docker upload uuid for the current request.

On Failure: Bad Request

```
400 Bad Request
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
```

```
    ]
}
```

There was an error processing the upload and it must be restarted.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
DIGEST_INVALID	provided digest did not match uploaded content	When a blob is uploaded, the registry will check that the content matches the digest provided by the client. The error may include a detail structure with the key “digest”, including the invalid digest string. This error may also be returned when a manifest includes an invalid layer digest.
NAME_INVALID	invalid repository name	Invalid repository name encountered either during manifest validation or any API operation.
BLOB_UPLOAD_INVALID	blob upload invalid	The blob upload encountered an error and can no longer proceed.

On Failure: Not Found

```
404 Not Found
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The upload is unknown to the registry. The upload must be restarted.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
BLOB_UPLOAD_UNKNOWN	blob upload unknown to registry	If a blob upload has been cancelled or was never started, this error code may be returned.

On Failure: Authentication Required

```
401 Unauthorized
WWW-Authenticate: <scheme> realm="<realm>", ..."
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The client is not authenticated.

The following headers will be returned on the response:

Name	Description
WWW-Authenticate	An RFC7235 compliant authentication challenge header.
Content-Length	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
UNAUTHORIZED	authentication required	The access controller was unable to authenticate the client. Often this will be accompanied by a Www-Authenticate HTTP response header indicating how to authenticate.

On Failure: No Such Repository Error

```
404 Not Found
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The repository is not known to the registry.

The following headers will be returned on the response:

Name	Description
Content-Length	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
NAME_UNKNOWN	repository name not known to registry	This is returned if the name used during an operation is unknown to the registry.

On Failure: Access Denied

```
403 Forbidden
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The client does not have required access to the repository.

The following headers will be returned on the response:

Name	Description
`Content-Length`	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`DENIED`	requested access to the resource is denied	The access controller denied access for the operation on a resource.

PATCH Blob Upload

Upload a chunk of data for the specified upload.

Stream upload

```
PATCH /v2/<name>/blobs/uploads/<uuid>
```

```
Host: <registry host>
Authorization: <scheme> <token>
Content-Type: application/octet-stream

<binary data>
```

Upload a stream of data to upload without completing the upload.

The following parameters should be specified on the request:

Name	Kind	Description
`Host`	header	Standard HTTP Host Header. Should be set to the registry host.
`Authorization`	header	An RFC7235 compliant authorization header.
`name`	path	Name of the target repository.
`uuid`	path	A uuid identifying the upload. This field can accept characters that match `[a-zA-Z0-9-_.=]+`.

On Success: Data Accepted

```
204 No Content
Location: /v2/<name>/blobs/uploads/<uuid>
Range: 0-<offset>
Content-Length: 0
Docker-Upload-UUID: <uuid>
```

The stream of data has been accepted and the current progress is available in the range header. The updated upload location is available in the `Location` header.

The following headers will be returned with the response:

Name	Description
`Location`	The location of the upload. Clients should assume this changes after each request. Clients should use the contents verbatim to complete the upload, adding parameters where required.

`Range`	Range indicating the current progress of the upload.
`Content-Length`	The `Content-Length` header must be zero and the body must be empty.
`Docker-Upload-UUID`	Identifies the docker upload uuid for the current request.

On Failure: Bad Request

```
400 Bad Request
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

There was an error processing the upload and it must be restarted.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`DIGEST_INVALID`	provided digest did not match uploaded content	When a blob is uploaded, the registry will check that the content matches the digest provided by the client. The error may include a detail structure with the key “digest”, including the invalid digest string. This error may also be returned when a manifest includes an invalid layer digest.
`NAME_INVALID`	invalid repository name	Invalid repository name encountered either during manifest validation or any API operation.

BLOB_UPLOAD_INVALID	blob upload invalid	The blob upload encountered an error and can no longer proceed.

On Failure: Not Found

```
404 Not Found
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The upload is unknown to the registry. The upload must be restarted.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
BLOB_UPLOAD_UNKNOWN	blob upload unknown to registry	If a blob upload has been cancelled or was never started, this error code may be returned.

On Failure: Authentication Required

```
401 Unauthorized
WWW-Authenticate: <scheme> realm="<realm>", ..."
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
```

```
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The client is not authenticated.

The following headers will be returned on the response:

Name	Description
`WWW-Authenticate`	An RFC7235 compliant authentication challenge header.
`Content-Length`	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`UNAUTHORIZED`	authentication required	The access controller was unable to authenticate the client. Often this will be accompanied by a Www-Authenticate HTTP response header indicating how to authenticate.

On Failure: No Such Repository Error

```
404 Not Found
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
```

```
        ...
    ]
}
```

The repository is not known to the registry.

The following headers will be returned on the response:

Name	Description
`Content-Length`	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`NAME_UNKNOWN`	repository name not known to registry	This is returned if the name used during an operation is unknown to the registry.

On Failure: Access Denied

```
403 Forbidden
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The client does not have required access to the repository.

The following headers will be returned on the response:

Name	Description
`Content-Length`	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`DENIED`	requested access to the resource is denied	The access controller denied access for the operation on a resource.

Chunked upload

```
PATCH /v2/<name>/blobs/uploads/<uuid>
Host: <registry host>
Authorization: <scheme> <token>
Content-Range: <start of range>-<end of range, inclusive>
Content-Length: <length of chunk>
Content-Type: application/octet-stream

<binary chunk>
```

Upload a chunk of data to specified upload without completing the upload. The data will be uploaded to the specified Content Range.

The following parameters should be specified on the request:

Name	Kind	Description
`Host`	header	Standard HTTP Host Header. Should be set to the registry host.
`Authorization`	header	An RFC7235 compliant authorization header.
`Content-Range`	header	Range of bytes identifying the desired block of content represented by the body. Start must the end offset retrieved via status check plus one. Note that this is a non-standard use of the `Content-Range` header.
`Content-`	header	Length of the chunk being uploaded, corresponding

`Length`		the length of the request body.
`name`	path	Name of the target repository.
`uuid`	path	A uuid identifying the upload. This field can accept characters that match `[a-zA-Z0-9-_.=]+`.

On Success: Chunk Accepted

```
204 No Content
Location: /v2/<name>/blobs/uploads/<uuid>
Range: 0-<offset>
Content-Length: 0
Docker-Upload-UUID: <uuid>
```

The chunk of data has been accepted and the current progress is available in the range header. The updated upload location is available in the `Location` header.

The following headers will be returned with the response:

Name	Description
`Location`	The location of the upload. Clients should assume this changes after each request. Clients should use the contents verbatim to complete the upload, adding parameters where required.
`Range`	Range indicating the current progress of the upload.
`Content-Length`	The `Content-Length` header must be zero and the body must be empty.
`Docker-Upload-UUID`	Identifies the docker upload uuid for the current request.

On Failure: Bad Request

```
400 Bad Request
Content-Type: application/json; charset=utf-8

{
        "errors:" [
```

```
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

There was an error processing the upload and it must be restarted.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`DIGEST_INVALID`	provided digest did not match uploaded content	When a blob is uploaded, the registry will check that the content matches the digest provided by the client. The error may include a detail structure with the key “digest”, including the invalid digest string. This error may also be returned when a manifest includes an invalid layer digest.
`NAME_INVALID`	invalid repository name	Invalid repository name encountered either during manifest validation or any API operation.
`BLOB_UPLOAD_INVALID`	blob upload invalid	The blob upload encountered an error and can no longer proceed.

On Failure: Not Found

```
404 Not Found
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
```

```
        },
        ...
    ]
}
```

The upload is unknown to the registry. The upload must be restarted.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
BLOB_UPLOAD_UNKNOWN	blob upload unknown to registry	If a blob upload has been cancelled or was never started, this error code may be returned.

On Failure: Requested Range Not Satisfiable

```
416 Requested Range Not Satisfiable
```

The `Content-Range` specification cannot be accepted, either because it does not overlap with the current progress or it is invalid.

On Failure: Authentication Required

```
401 Unauthorized
WWW-Authenticate: <scheme> realm="<realm>", ..."
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
```

```
}
```

The client is not authenticated.

The following headers will be returned on the response:

Name	Description
`WWW-Authenticate`	An RFC7235 compliant authentication challenge header.
`Content-Length`	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`UNAUTHORIZED`	authentication required	The access controller was unable to authenticate the client. Often this will be accompanied by a Www-Authenticate HTTP response header indicating how to authenticate.

On Failure: No Such Repository Error

```
404 Not Found
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The repository is not known to the registry.

The following headers will be returned on the response:

Name	Description
`Content-Length`	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`NAME_UNKNOWN`	repository name not known to registry	This is returned if the name used during an operation is unknown to the registry.

On Failure: Access Denied

```
403 Forbidden
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The client does not have required access to the repository.

The following headers will be returned on the response:

Name	Description
`Content-Length`	Length of the JSON response body.

The error codes that may be included in the response body are enumerated

below:

Code	Message	Description
`DENIED`	requested access to the resource is denied	The access controller denied access for the operation on a resource.

PUT Blob Upload

Complete the upload specified by `uuid`, optionally appending the body as the final chunk.

```
PUT /v2/<name>/blobs/uploads/<uuid>?digest=<digest>
Host: <registry host>
Authorization: <scheme> <token>
Content-Length: <length of data>
Content-Type: application/octet-stream

<binary data>
```

Complete the upload, providing all the data in the body, if necessary. A request without a body will just complete the upload with previously uploaded content.

The following parameters should be specified on the request:

Name	Kind	Description
`Host`	header	Standard HTTP Host Header. Should be set to the registry host.
`Authorization`	header	An RFC7235 compliant authorization header.
`Content-Length`	header	Length of the data being uploaded, corresponding to the length of the request body. May be zero if no data is provided.
`name`	path	Name of the target repository.
`uuid`	path	A uuid identifying the upload. This field can accept characters that match `[a-zA-Z0-9-_.=]+`.
`digest`	query	Digest of uploaded blob.

On Success: Upload Complete

```
204 No Content
Location: <blob location>
Content-Range: <start of range>-<end of range, inclusive>
Content-Length: 0
Docker-Content-Digest: <digest>
```

The upload has been completed and accepted by the registry. The canonical location will be available in the `Location` header.

The following headers will be returned with the response:

Name	Description
`Location`	The canonical location of the blob for retrieval
`Content-Range`	Range of bytes identifying the desired block of content represented by the body. Start must match the end of offset retrieved via status check. Note that this is a non-standard use of the `Content-Range` header.
`Content-Length`	The `Content-Length` header must be zero and the body must be empty.
`Docker-Content-Digest`	Digest of the targeted content for the request.

On Failure: Bad Request

```
400 Bad Request
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
```

```
}
```

There was an error processing the upload and it must be restarted.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
DIGEST_INVALID	provided digest did not match uploaded content	When a blob is uploaded, the registry will check that the content matches the digest provided by the client. The error may include a detail structure with the key “digest”, including the invalid digest string. This error may also be returned when a manifest includes an invalid layer digest.
NAME_INVALID	invalid repository name	Invalid repository name encountered either during manifest validation or any API operation.
BLOB_UPLOAD_INVALID	blob upload invalid	The blob upload encountered an error and can no longer proceed.
UNSUPPORTED	The operation is unsupported.	The operation was unsupported due to a missing implementation or invalid set of parameters.

On Failure: Not Found

```
404 Not Found
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
```

```
}
```

The upload is unknown to the registry. The upload must be restarted.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
BLOB_UPLOAD_UNKNOWN	blob upload unknown to registry	If a blob upload has been cancelled or was never started, this error code may be returned.

On Failure: Authentication Required

```
401 Unauthorized
WWW-Authenticate: <scheme> realm="<realm>", ..."
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The client is not authenticated.

The following headers will be returned on the response:

Name	Description
WWW-Authenticate	An RFC7235 compliant authentication challenge header.
Content-Length	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
UNAUTHORIZED	authentication required	The access controller was unable to authenticate the client. Often this will be accompanied by a Www-Authenticate HTTP response header indicating how to authenticate.

On Failure: No Such Repository Error

```
404 Not Found
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The repository is not known to the registry.

The following headers will be returned on the response:

Name	Description
Content-Length	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
	repository name	This is returned if the name used during an

NAME_UNKNOWN	not known to registry	operation is unknown to the registry.

On Failure: Access Denied

```
403 Forbidden
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The client does not have required access to the repository.

The following headers will be returned on the response:

Name	Description
Content-Length	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
DENIED	requested access to the resource is denied	The access controller denied access for the operation on a resource.

DELETE Blob Upload

Cancel outstanding upload processes, releasing associated resources. If this is not called, the unfinished uploads will eventually timeout.

```
DELETE /v2/<name>/blobs/uploads/<uuid>
Host: <registry host>
Authorization: <scheme> <token>
Content-Length: 0
```

Cancel the upload specified by `uuid`.

The following parameters should be specified on the request:

Name	Kind	Description
`Host`	header	Standard HTTP Host Header. Should be set to the registry host.
`Authorization`	header	An RFC7235 compliant authorization header.
`Content-Length`	header	The `Content-Length` header must be zero and the body must be empty.
`name`	path	Name of the target repository.
`uuid`	path	A uuid identifying the upload. This field can accept characters that match `[a-zA-Z0-9-_.=]+`.

On Success: Upload Deleted

```
204 No Content
Content-Length: 0
```

The upload has been successfully deleted.

The following headers will be returned with the response:

Name	Description
`Content-Length`	The `Content-Length` header must be zero and the body must be empty.

On Failure: Bad Request

```
400 Bad Request
```

```
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

An error was encountered processing the delete. The client may ignore this error.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
NAME_INVALID	invalid repository name	Invalid repository name encountered either during manifest validation or any API operation.
BLOB_UPLOAD_INVALID	blob upload invalid	The blob upload encountered an error and can no longer proceed.

On Failure: Not Found

```
404 Not Found
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
```

```
        ...
    ]
}
```

The upload is unknown to the registry. The client may ignore this error and assume the upload has been deleted.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
BLOB_UPLOAD_UNKNOWN	blob upload unknown to registry	If a blob upload has been cancelled or was never started, this error code may be returned.

On Failure: Authentication Required

```
401 Unauthorized
WWW-Authenticate: <scheme> realm="<realm>", ..."
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The client is not authenticated.

The following headers will be returned on the response:

Name	Description

`WWW-Authenticate`	An RFC7235 compliant authentication challenge header.
`Content-Length`	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
`UNAUTHORIZED`	authentication required	The access controller was unable to authenticate the client. Often this will be accompanied by a Www-Authenticate HTTP response header indicating how to authenticate.

On Failure: No Such Repository Error

```
404 Not Found
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The repository is not known to the registry.

The following headers will be returned on the response:

Name	Description
`Content-Length`	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
NAME_UNKNOWN	repository name not known to registry	This is returned if the name used during an operation is unknown to the registry.

On Failure: Access Denied

```
403 Forbidden
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "errors:" [
            {
            "code": <error code>,
            "message": "<error message>",
            "detail": ...
        },
        ...
    ]
}
```

The client does not have required access to the repository.

The following headers will be returned on the response:

Name	Description
Content-Length	Length of the JSON response body.

The error codes that may be included in the response body are enumerated below:

Code	Message	Description
DENIED	requested access to the resource is denied	The access controller denied access for the operation on a resource.

Catalog

List a set of available repositories in the local registry cluster. Does not provide any indication of what may be available upstream. Applications can only determine if a repository is available but not if it is not available.

GET Catalog

Retrieve a sorted, json list of repositories available in the registry.

Catalog Fetch

Request an unabridged list of repositories available. The implementation may impose a maximum limit and return a partial set with pagination links.

On Success: OK

```
200 OK
Content-Length: <length>
Content-Type: application/json; charset=utf-8

{
        "repositories": [
                <name>,
                ...
        ]
}
```

Returns the unabridged list of repositories as a json response.

The following headers will be returned with the response:

Name	Description
`Content-Length`	Length of the JSON response body.

Catalog Fetch Paginated

```
GET /v2/_catalog?n=<integer>&last=<integer>
```

Return the specified portion of repositories.

The following parameters should be specified on the request:

Name	Kind	Description
`n`	query	Limit the number of entries in each response. It not present, all entries will be returned.
`last`	query	Result set will include values lexically after last.

On Success: OK

```
200 OK
Content-Length: <length>
Link: <<url>?n=<last n value>&last=<last entry from response>>; rel="n
Content-Type: application/json; charset=utf-8

{
        "repositories": [
                <name>,
                ...
        ]
        "next": "<url>?last=<name>&n=<last value of n>"
}
```

The following headers will be returned with the response:

Name	Description
`Content-Length`	Length of the JSON response body.
`Link`	RFC5988 compliant rel='next' with URL to next result set, if available