

Beyond the Download Count: Understanding the Usage of Your Public Containers

Avi Press

March 22, 2022

Outline

- 1 Before we start. . .
- 2 Container registries today & the data provided
- 3 What is a pull and how does it work
- 4 Pulls are data-rich
- 5 But how can we get this data?

Outline

- 1 Before we start. . .
- 2 Container registries today & the data provided
- 3 What is a pull and how does it work
- 4 Pulls are data-rich
- 5 But how can we get this data?

- These slides are generated from an org-mode document which is available here: <https://github.com/aviaviavi/talks/blob/master/beyond-the-download-count/presentation.org>
- Feel free to jump in with questions as we go!

Outline

- 1 Before we start. . .
- 2 Container registries today & the data provided
- 3 What is a pull and how does it work
- 4 Pulls are data-rich
- 5 But how can we get this data?

Stats from container registries are limited

Docker Hub

- Total pulls for repo

Stats from container registries are limited

Docker Hub

- Total pulls for repo

Others

GitHub

- Total pulls for repo
- Total pulls by tag

Quay.io

- Total pulls for repo
- Total pulls by tag

AWS ECR

- Total pulls for repo
- Time series of pulls by repo

What else might we want to understand?

Metrics

- Unique pulls (10 pulls from 10 people vs 10 pulls from the same person)
- Pulls by:
 - Host platform
 - Container runtime
 - Country
 - Architecture
 - Companies
- Invocations versus downloads of the container?

Granularity

- Should be able to understand metrics as a time-series
- For any given metric, what was the count for:
 - Yesterday?
 - Past week?
 - Past month?
 - Every Tuesday this year?
 - 2021?

Outline

- 1 Before we start. . .
- 2 Container registries today & the data provided
- 3 What is a pull and how does it work
- 4 Pulls are data-rich
- 5 But how can we get this data?

What is a pull, really?

Key Terms

Manifest

An image manifest provides a configuration and set of layers for a single container image for a specific architecture and operating system.

Blob

Data that comprises the layers of the image

Manifest

An image manifest provides a configuration and set of layers for a single container image for a specific architecture and operating system.

```
{
  "schemaVersion": 2,
  "mediaType": "application/vnd.oci.image.manifest.v1+json",
  "config": {
    "mediaType": "application/vnd.oci.image.config.v1+json",
    "size": 7023,
    "digest": "sha256:b5b2b2c507a0944348e0303114d8d93aaaa081732b86451d9bce1f432a537bc7"
  },
  "layers": [
    {
      "mediaType": "application/vnd.oci.image.layer.v1.tar+gzip",
      "size": 32654,
      "digest": "sha256:9834876dcfb05cb167a5c24953eba58c4ac89b1adf57f28f2f9d09af107ee8f0"
    },
    {
      "mediaType": "application/vnd.oci.image.layer.v1.tar+gzip",
      "size": 16724,
      "digest": "sha256:3c3a4604a545cdc127456d94e421cd355bca5b528f4a9c1905b15da2eb4a4c6b"
    },
    {
      "mediaType": "application/vnd.oci.image.layer.v1.tar+gzip",
      "size": 73109,
      "digest": "sha256:ec4b8955958665577945c89419d1af06b5f7636b4ac3da7f12184802ad867736"
    }
  ],
  "annotations": {
    "com.example.key1": "value1",
    "com.example.key2": "value2"
  }
}
```

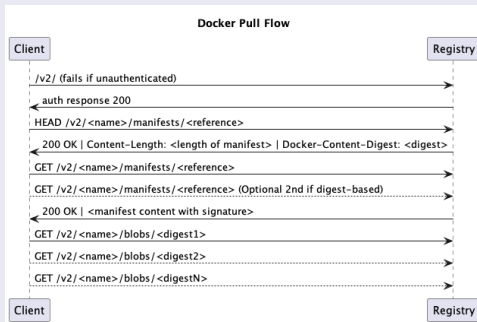
What is a pull, really?

Definition (from Docker)

- A pull request is defined as up to two GET requests on registry manifest URLs (`/v2/*/manifests/*`).
- A normal image pull makes a single manifest request.
- A pull request for a multi-arch image makes two manifest requests.
- HEAD requests are not counted.
 - Docker, <https://docs.docker.com/docker-hub/download-rate-limit/>

Quick example

Call diagram



Notes

- *Name*: eg, organization-name/image-name
- *Reference*: A tag (latest), or a digest (sha256aaabbbcccddd...).

Challenges with measuring pulls

- A "pull" spans multiple API calls
 - Event processors must be stateful, and must include a notion of identity at the request level.
- A "normal" image pull vs "abnormal"
- Manifests vs blobs
- Different clients have different behavior

Theory vs practice

In theory

Clients will call HEAD for the manifest and only call GET when updates are needed.

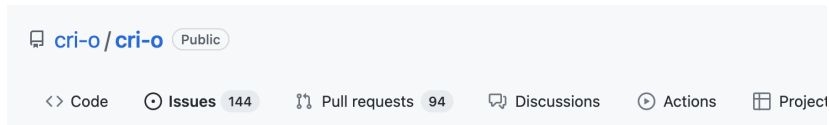
In practice

Many clients will just call GET every time. Examples:

- GoHttpClient
- Older versions of Watchtower
- Older versions of Diujn
- Older versions of Docker for Mac

It's actually even worse

Some clients fetch manifests twice (eg, cri-o)



cri-o / cri-o Public

<> Code **Issues 144** Pull requests 94 Discussions Actions Projects

Image manifests are requested twice #5039

Closed

fabioluz opened this issue on Jun 28, 2021 · 28 comments



wgahnagl commented on Nov 22, 2021

Contributor  ...

manifests being requested twice is intentional due to an optimization, and switching over to libimage doesn't appear to be able to resolve the issue either without getting rid of the optimization.



wgahnagl closed this on Nov 22, 2021

Outline

- 1 Before we start. . .
- 2 Container registries today & the data provided
- 3 What is a pull and how does it work
- 4 Pulls are data-rich**
- 5 But how can we get this data?

So what else can the registry see?

- Headers
- Time series information

Headers in Docker pulls

Headers per request

```
X-Request-ID: <request id>
X-Forwarded-For: <ip>
authorization: Bearer <token>
accept: application/vnd.docker.distribution.manifest.v2+json
accept: application/vnd.docker.distribution.manifest.list.v2+json
accept: application/vnd.docker.distribution.manifest.v1+json
user-agent: docker/20.10.6 go/go1.13.15 git-commit/a3dc69e6b9
           os/windows arch/amd64 UpstreamClient(Go-http-client/1.1)
```

Headers in Docker pulls

Headers per request

```
X-Request-ID: <request id>
X-Forwarded-For: <ip>
authorization: Bearer <token>
accept: application/vnd.docker.distribution.manifest.v2+json
accept: application/vnd.docker.distribution.manifest.list.v2+json
accept: application/vnd.docker.distribution.manifest.v1+json
user-agent: docker/20.10.6 go/go1.13.15 git-commit/a3dc69e6b9
           os/windows arch/amd64 UpstreamClient(Go-http-client/1.1)
```

This info can tell us

- A notion of uniqueness(!!)
- IP request metadata
- Container runtime
- Platform

Headers are rich in information

A notion of uniqueness

You may have had 1000 downloads today but from only 5 distinct sources

IP request metadata

- Where are your users distributed geographically?
- Are your downloads coming from companies or individuals? Which companies?
- Laptops or CI?
- Which clouds?

Platform

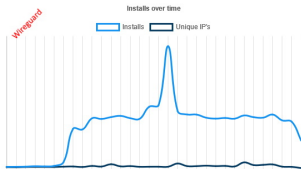
- Container runtime: Docker, containerd, cri-o, Watchertower?
 - If lots old versions are involved, your pull counts are likely inflated!
- What is the breakdown of host OS? Architecture?

Uniques can be extremely useful

Two users are responsible for 73,000 pulls between them, with the next 10 being responsible for 55,000 between them. Almost half of our pulls through Scarf can be attributed to 20 users with misconfigured or overly aggressive deployment/update services

- *LinuxServer.io Blog*

link - <https://www.linuxserver.io/blog/unravelling-some-stats>



So what else can the registry see?

Time series of requests

Time	Origin ID	Request Type	Path
12:00	abc	HEAD	/v2/image-name/manifests/latest
12:00	abc	GET	/v2/image-name/manifests/latest
12:05	abc	HEAD	/v2/image-name/manifests/latest
12:10	abc	HEAD	/v2/image-name/manifests/latest
12:15	abc	HEAD	/v2/image-name/manifests/latest
12:20	abc	HEAD	/v2/image-name/manifests/latest

This info can tell us

- Invocations of the container vs downloads of the container
- Gives clues to activity / behavior

Time series data tells us about usage

Consider this access patterns

Time	Origin ID	Request Type	Path
12:00	abc	HEAD	/v2/image-name/manifests/latest
12:00	abc	GET	/v2/image-name/manifests/latest
12:05	abc	HEAD	/v2/image-name/manifests/latest
12:10	abc	HEAD	/v2/image-name/manifests/latest
12:15	abc	HEAD	/v2/image-name/manifests/latest
12:20	abc	HEAD	/v2/image-name/manifests/latest

Time series data tells us about usage

Consider this access patterns

Time	Origin ID	Request Type	Path
12:00	abc	HEAD	/v2/image-name/manifests/latest
12:00	abc	GET	/v2/image-name/manifests/latest
12:05	abc	HEAD	/v2/image-name/manifests/latest
12:10	abc	HEAD	/v2/image-name/manifests/latest
12:15	abc	HEAD	/v2/image-name/manifests/latest
12:20	abc	HEAD	/v2/image-name/manifests/latest

Relevant info

- Highly regular intervals, polling for latest version

Possible explanations

- Production deployment
- Internal tooling deployment

Time series data tells us about usage

Versus this one

Time	Origin ID	Request Type	Path
12:00	abc	HEAD	/v2/image-name/manifests/latest
12:00	abc	GET	/v2/image-name/manifests/latest
13:09	abc	HEAD	/v2/image-name/manifests/latest
13:09	abc	HEAD	/v2/image-name/manifests/latest
13:10	abc	HEAD	/v2/image-name/manifests/latest
09:01	abc	HEAD	/v2/image-name/manifests/1.0.1
09:01	abc	GET	/v2/image-name/manifests/1.0.1
09:03	abc	HEAD	/v2/image-name/manifests/latest
09:10	abc	HEAD	/v2/image-name/manifests/latest

Time series data tells us about usage

Versus this one

Time	Origin ID	Request Type	Path
12:00	abc	HEAD	/v2/image-name/manifests/latest
12:00	abc	GET	/v2/image-name/manifests/latest
13:09	abc	HEAD	/v2/image-name/manifests/latest
13:09	abc	HEAD	/v2/image-name/manifests/latest
13:10	abc	HEAD	/v2/image-name/manifests/latest
09:01	abc	HEAD	/v2/image-name/manifests/1.0.1
09:01	abc	GET	/v2/image-name/manifests/1.0.1
09:03	abc	HEAD	/v2/image-name/manifests/latest
09:10	abc	HEAD	/v2/image-name/manifests/latest

Relevant info

- Irregular intervals
- Multiple versions

Possible explanations

- Local development

Outline

- 1 Before we start. . .
- 2 Container registries today & the data provided
- 3 What is a pull and how does it work
- 4 Pulls are data-rich
- 5 But how can we get this data?


Convince your registry to give it you

Let me know how it goes!

[Code](#) [Issues 199](#) [Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#)

Detailed download statistics per image #1047





[Open](#) michaelcoburn opened this issue on May 17, 2017 · 27 comments

 **michaelcoburn** commented on May 17, 2017

Hi, is there a way to see detailed download statistics of a given image? For example <https://hub.docker.com/r/percona/pmm-server/> reports +100k pulls, but that is difficult for an image maintainer to really understand adoption as the number is so rounded.

- Is this a feature currently supported for logged-in administrators of images?
- or perhaps something in the higher paid-tiers?

Thanks,

  74  11  6

Host a registry

```
$ docker pull yourdomain.com/your-image
```

Pros

- Open source solutions (eg distribution)
- Distribute from your own domain
- Full access (publishing, data handling, insights, etc)

Cons

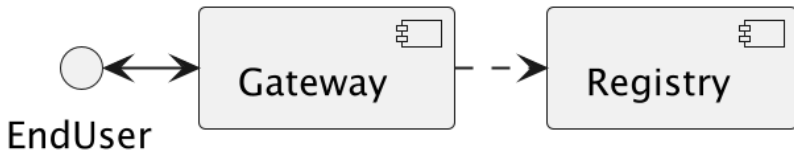
- Bandwidth is expensive
- Availability and performance are on you
 - *How long will it take your us-west-2 machine to stream a 1GB container to a user in Mumbai?*

Idea

Put a service in front of the registry that:

- Passes traffic transparently to the registry that hosts the container via a redirect
- Processes traffic to process pull data

```
$ docker pull yourdomain.com/your-image
```



Pros

- Can access all request data
- Lightweight service - redirection can be very dumb
- Robust to API changes from the the client/registry
- Simply(*) redirecting rather than proxying means minimal overhead (bandwidth and speed)
- Decoupling from registry
- Distribute from your own domain
- Can work for things besides containers!

Cons

- Added complexity
 - Failure point
 - Performance choke point

Simple!(*)

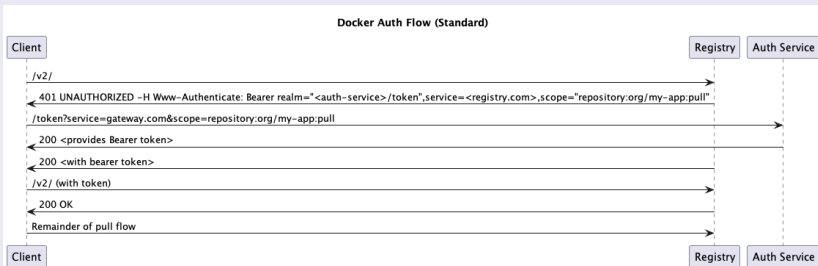
```
server {  
    server_name a.domain.com  
    listen 443;  
    rewrite (.* ) https://registry-1.docker.io$1 permanent;  
}
```

...Almost

- Gateway still needs to be available and fast globally
- Can't actually always redirect :'(

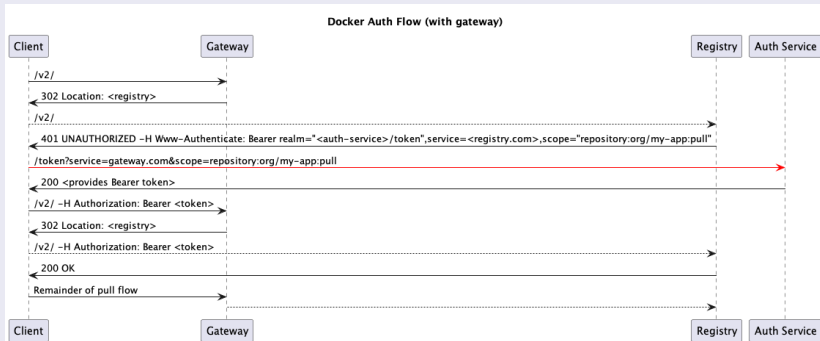
In certain cases, container registry gateways must proxy

Standard auth & pull



In certain cases, container registry gateways must proxy

Auth & pull with Gateway



Some clients mess up the `/token` request when redirected, confusing service address & redirect target.

How Scarf built its container registry gateway

Scarf Gateway



SCARF | Gateway



How Scarf built its container registry gateway

(to be open-sourced soon)

Phase 1

A general recommended approach to anyone wanting to get started building their own

- Nginx
 - Send access logs to storage (we were using AWS Cloudwatch)
 - Lua for any custom business logic you might want, eg reading configs from Redis
- Process logs asynchronously to generate analytics & insights

Phase 2

- Server as hand-written Haskell code
- Configuration in-memory
- Send access logs to time series storage, eg Kafka
- distribution as a pull-through-cache when we are forced to proxy

This can be done while still completely preserving end-user privacy.

- Depending on how you store and process this data, you may or may not run into compliance considerations like GDPR
- Recommendations:
 - Don't touch PII you don't need
 - Delete it once you are done processing it
 - Leverage 3rd parties to handle it on your behalf
 - Consult legal counsel

Other benefits of the gateway approach

- Distribute from your own domain, not someone else's
- Ability to switch registries on-the-fly without breaking anything downstream.
 - Dual publishing can keep your containers online when primary registry goes down

Notable challenges

- Easy to build, harder to scale
 - Multi-region availability, redundancy, etc is where the real complexity lives
- Proxying as little as possible
- Many competing container runtimes / clients -> edge-case bugs

Tying it together

- Registry data can be useful!
- Your current registry provider doesn't provide access to pull data, but there are still ways to get to it.
- Registry gateways can be a reasonable option

Thank you!

Avi Press

Website	https://avi.press
Twitter	@avi__press
GitHub	aviaviavi
LinkedIn	link

Scarf

Website	https://scarf.sh
Twitter	scarf _{oss}
GitHub	scarf-sh
LinkedIn	link