Beyond the Download Count: Understanding the Usage of Your Public Python Packages

Avi Press

April 15, 2022

Outline

Outline

Quick notes

- These slides are generated from an org-mode document which is available here.

Outline

Stats from python registries are miles ahead of other languages, but could be much better

PyPI

- Per download:
  - Timestamp
  - Package version
  - Installer name
  - Python version

Stats from python registries are miles ahead of other languages, but could be much better

Conda

- Download by download metrics: `data_source`, time , package version, platform, Python version

GitHub Packages & Google Artifact Registry

- Total downloads for repo
- Total downloads by version

AWS CodeArtifact

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

What else might we want to understand?

Metrics

- Unique downloads (10 downloads from 10 people vs 10 downloads from the same person)
- Downloads by:
  - Host platform
  - Country
  - Architecture
  - Companies

Outline

So what else can the registry see?

- Headers
- Time series information

Headers in Python package downloads

Sample headers a registry will see for a pip download

```
X-Forwarded-For: <ip Address>
User-Agent: pip/21.2.4 {
  "ci":null,
  "cpu":"x86_64",
  "distro":{"name":"macOS","version":"11.3.1"},
  "implementation":{"name":"CPython","version":"3.9.7"},
  "installer":{"name":"pip","version":"21.2.4"},
  "openssl_version":"OpenSSL 1.1.1l  24 Aug 2021",
  "python":"3.9.7",
  "setuptools_version":"60.9.3",
  "system":{"name":"Darwin","release":"20.4.0"}
  }
```

Headers in Python package downloads

Sample headers a registry will see for a pip download

```
X-Forwarded-For: <ip Address>
User-Agent: pip/21.2.4 {
  "ci":null,
  "cpu":"x86_64",
  "distro":{"name":"macOS","version":"11.3.1"},
  "implementation":{"name":"CPython","version":"3.9.7"},
  "installer":{"name":"pip","version":"21.2.4"},
  "openssl_version":"OpenSSL 1.1.1l  24 Aug 2021",
  "python":"3.9.7",
  "setuptools_version":"60.9.3",
  "system":{"name":"Darwin","release":"20.4.0"}
  }
```

This info can tell us

- A notion of uniqueness
- IP request metadata
- Insight into how your users install and use your package

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 and client

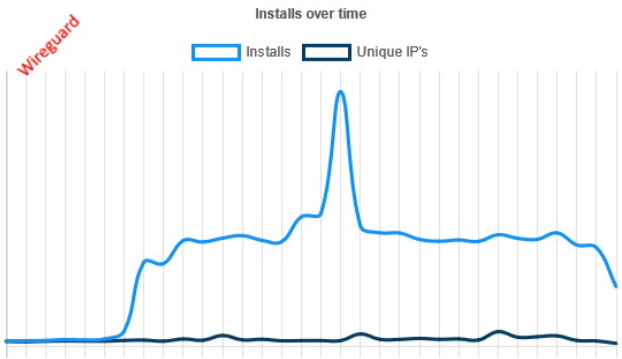
- What is the breakdown of host OS? Architecture?
- Was the client application a registry mirror?

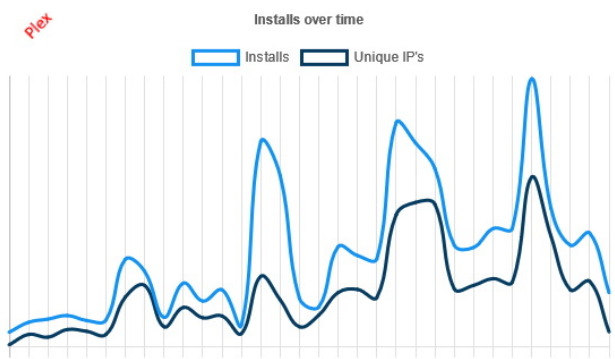
Uniques can be extremely useful

Two users are responsible for 73,000 downloads between them, with the next 10 being responsible for 55,000 between them. Almost half of our downloads 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	Resp	Path
12:00	abc	200	`/simple/django/`
12:00	abc	200	`/simple/django/Django-4.0.3-py3-none-any.whl`
12:05	abc	304	`/simple/django/`
12:10	abc	304	`/simple/django/`
12:15	abc	304	`/simple/django/`
12:20	abc	304	`/simple/django/`

This info can tell us

- Downloads versus download attempts
- Gives clues to activity / behavior

Time series data tells us about usage

Consider this access patterns

Time	Origin	Resp	Path
12:00	abc	200	`/simple/django/`
12:00	abc	200	`/simple/django/Django-4.0.3-py3-none-any.whl`
12:05	abc	304	`/simple/django/`
12:10	abc	304	`/simple/django/`
12:15	abc	304	`/simple/django/`
12:20	abc	304	`/simple/django/`

Time series data tells us about usage

Consider this access patterns

Time	Origin	Resp	Path
12:00	abc	200	`/simple/django/`
12:00	abc	200	`/simple/django/Django-4.0.3-py3-none-any.whl`
12:05	abc	304	`/simple/django/`
12:10	abc	304	`/simple/django/`
12:15	abc	304	`/simple/django/`
12:20	abc	304	`/simple/django/`

Relevant info

- Highly regular intervals, rebuilding and/or polling for latest version

Possible explanations

- Production deployment
- Internal tooling deployment

Time series data tells us about usage

Versus this one

Time	Origin	Resp	Path
09:00	abc	200	`/simple/django/`
09:00	abc	200	`/simple/django/Django-4.0.3-py3-none-any.whl`
09:03	abc	304	`/simple/django/`
10:13	abc	304	`/simple/django/`
10:14	abc	304	`/simple/django/`
12:00	abc	304	`/simple/django/`
13:50	abc	200	`/simple/django/Django-4.0.0-py3-none-any.whl`
13:50	abc	304	`/simple/django/`
14:11	abc	304	`/simple/django/`
14:15	abc	200	`/simple/django/Django-4.0.3-py3-none-any.whl`

Time series data tells us about usage

Versus this one

Time	Origin	Resp	Path
09:00	abc	200	`/simple/django/`
09:00	abc	200	`/simple/django/Django-4.0.3-py3-none-any.whl`
09:03	abc	304	`/simple/django/`
10:13	abc	304	`/simple/django/`
10:14	abc	304	`/simple/django/`
12:00	abc	304	`/simple/django/`
13:50	abc	200	`/simple/django/Django-4.0.0-py3-none-any.whl`
13:50	abc	304	`/simple/django/`
14:11	abc	304	`/simple/django/`
14:15	abc	200	`/simple/django/Django-4.0.3-py3-none-any.whl`

Relevant info

- Irregular intervals
- Multiple versions

Possible explanations

- Local development

Outline

Convince your registry to give it you

Let me know how it goes!

Host a registry

Point end-users to a registry you control

```
$ pip install --extra-index-url yourdomain.com/simple your-package
```

Pros

- Open source solutions (devpi)
- 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 full package set to your users in Mumbai?*

Registry Gateway

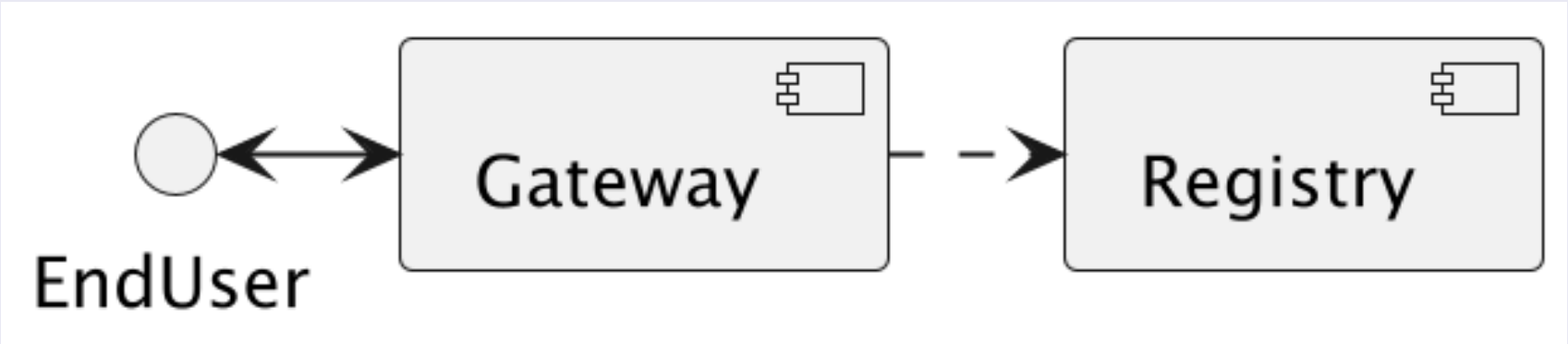
Idea

Put a service in front of the registry that:

- Passes traffic transparently to the registry that hosts the package via a redirect
- Processes traffic to process download data

Point end-users to a registry gateway you control

```
$ pip install --extra-index-url yourdomain.com/simple your-package
```



Registry Gateway

Pros

- 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 Python packages!

Cons

- Added complexity
  - Failure point
  - Performance choke point

Simple!(*)

Just a little nginx config and we're done!

```
server {
  server_name a.domain.com
  listen 443;
  rewrite (.*) https://pypi.org$1 permanent;
}
```

. . . Almost

- Gateway needs to be available and fast globally
- Processing logs and storing data properly are non-trivial and likely high volume
- `--extra-index-url` has quirks

–extra-index-url is not powerful enough

Consider a `requirements.txt` file

```
--extra-index-url https://organization.org/simple
--extra-index-url https://company.com/simple

company-package==1.0.0
organization-package=2.0.1
numpy
```

Question

Which registry will we go to for each package?

–extra-index-url is not powerful enough

Consider a `requirements.txt` file

```
--extra-index-url https://organization.org/simple
--extra-index-url https://company.com/simple

company-package==1.0.0
organization-package=2.0.1
numpy
```

Question

Which registry will we go to for each package?

Answer

- We can't say! (Behavior in this scenario is undefined in Pip)
- If the client ends up going to a different registry, you're missing data

How Scarf built its package registry gateway

How Scarf built its Python 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

Data privacy

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 packages 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
- Rigorously scrubbing PII
- Many competing package installers -> edge-case bugs

Tying it together

- Registry data can be useful!
- Your current registry provider doesn't provide access to all the data you should have, 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	$\text{avi}_{\text{press}}$
GitHub	aviaviavi
LinkedIn	link

Scarf

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