
Building a Toy With ClickHouse

Let's build a demo app on ClickHouse

What for?

— Testing.

— An internal company contest.

— I always like to play with ClickHouse and have fun :)

How?

We will need:

— an idea.

— a nice **dataset**.

— a way to analyze and visialize it.

A Dataset

We want an "internet-scale" dataset. How to get it? 🤔

— get it from the Internet 🤫

DNS

Forward:

```
$ host google.com
google.com has address 142.250.179.174
google.com has IPv6 address 2a00:1450:400e:802::200e
google.com mail is handled by 10 smtp.google.com.
```

Reverse:

```
$ host 142.250.179.174
174.179.250.142.in-addr.arpa domain name pointer
ams15s41-in-f14.1e100.net.
```

DNS

Forward:

```
$ dig @1.1.1.1 google.com -t A
;; ANSWER SECTION:
google.com.             159     IN      A       142.250.179.174
```

Reverse:

```
$ dig @1.1.1.1 174.179.250.142.in-addr.arpa -t PTR
;; ANSWER SECTION:
174.179.250.142.in-addr.arpa. 81998 IN  PTR
     ams15s41-in-f14.1e100.net.net.
```

Why do PTR records exist?

```
$ mtr google.com

 1. 2a02-a210-1111-a500-aef8-ccff-fedc-1234.cable.dynamic.v6.ziggo.nl
 2. (waiting for reply)
 3. 2a02:a200:0:12::1
 4. asd-tr0021-cr101-lo10.core.as9143.net
 5. nl-ams14a-ri1-ae51-0.core.as9143.net
 6. 2001:4860:1:1::2ac
 7. 2a00:1450:8012::1
 8. 2001:4860:0:1::e36
 9. 2001:4860:0:f8b::c
10. 2001:4860::c:4002:c2d0
11. 2001:4860::1:0:cd13
12. 2001:4860:0:f8c::1
13. 2001:4860:0:1::5a85
14. ams17s12-in-x0e.1e100.net
```

Scan All IPv4

... to perform reverse DNS request on all of them.

There are only 2^{32} IP addresses.

Slightly less than 4 billion belongs to Internet.

How to do the scan?

Scan All IPv4

... to perform reverse DNS request on all of them.

There are tools for that:

— MassDNS: https://github.com/blechschmidt/massdns

— GNU adns library: https://www.gnu.org/software/adns/

Advantages: they are fast :)

Disadvantages: you will be blocked :(

Scan All IPv4

Where to do it?

From home?
— your ISP will be surprised by unusual traffic.

From a datacenter?
— my friend tried it from Digital Ocean and was blocked.

In the cloud?
— if you do it in AWS, it is ok if you communicate in advance.

What option did I choose?

DNS Over HTTPS (DoH)

DNS can work:

— over UDP (unreliable and not secure);
— over TCP (reliable but not secure);
— over TLS (reliable and secure, but heavyweight);
— over HTTPS (reliable and secure, but heavyweight);
— DNSCrypt over TCP or UDP (very different);

```
curl -H 'accept: application/dns-json' \
'https://cloudflare-dns.com/dns-query?name=174.179.250.142.in-addr.arpa&type=PTR'
```

```
curl -H 'accept: application/dns-json' \
'https://1.1.1.1/dns-query?name=174.179.250.142.in-addr.arpa&type=PTR'
```

DNS Over HTTPS (DoH)

```
$ curl -H 'accept: application/dns-json' \
'https://1.1.1.1/dns-query?name=174.179.250.142.in-addr.arpa&type=PTR' \
| jq

{
  "Status": 0,
  "TC": false, "RD": true, "RA": true, "AD": false, "CD": false,
  "Question": [
    {
      "name": "174.179.250.142.in-addr.arpa",
      "type": 12
    }
  ],
  "Answer": [
    {
      "name": "174.179.250.142.in-addr.arpa",
      "type": 12,
      "TTL": 77114,
      "data": "ams15s41-in-f14.1e100.net."
    }
  ]
}
```

DNS Over HTTPS (DoH)

Bonus: it can use HTTP 1.0, 1.1, 2, 3.

```
curl --http2 -H 'accept: application/dns-json' \
'https://1.1.1.1/dns-query?name=174.179.250.142.in-addr.arpa&type=PTR' \
--next --http2 -H 'accept: application/dns-json' \
'https://1.1.1.1/dns-query?name=3.121.82.140.in-addr.arpa&type=PTR'
```

CloudFlare, Google, OpenDNS have this service.

Using it for a mass scan could violate ToS,
but at least CloudFlare does not restrict it.

They are processing trillions of requests every day anyway :)

Table For Results

```
CREATE TABLE dns
(
    time DateTime DEFAULT now(),
    json String
)
ENGINE = MergeTree
ORDER BY ();
```

We'll write the responses as is, and parse later.

A Script For DNS Scan

```
clickhouse-local --max_block_size 1 --progress --query "
  WITH number DIV 65536 AS a, number DIV 256 MOD 256 AS b, number MOD 256 AS c
  SELECT c::String || '.' || b::String || '.' || a::String
  FROM numbers(16777216)
  WHERE (a > 0 AND a < 127 AND a != 10)
     OR (a >= 128 AND a < 192 AND (a != 172 OR b < 16 OR b > 31)
         AND (a != 169 AND b != 254))
     OR (a >= 192 AND a < 224 AND (a != 192 OR b != 168))
" | xargs -P1000 -I{} bash -c "
  seq 0 255 | sed -r -e '
s@(.*)@--next --http2 -H \"accept: application/dns-json\" \
\"https://1.1.1.1/dns-query?name=\\1.{}.in-addr.arpa\\&type=PTR\"@' \
| tr '\n' ' ' | sed 's/--next/curl -sS/' | bash \
| clickhouse-client --send_timeout 60000 --receive_timeout 60000 --query '
  INSERT INTO dns (json) FORMAT JSONAsString'"
```


38

A Script For DNS Scan

I tried to be gentle and run it from a single machine...

How long did it take?

A Script For DNS Scan

I tried to be gentle and run it from a single machine...

How long did it take?

— around ten days.

— it could be just a few hours with MassDNS.

GREYNOISE

Search for IP Addresses, CVEs, Tags..

TRENDS TODAY TAGS ANALYSIS ALERTS

GREYNOISE TRENDS

DNS OVER HTTPS SCANNER

TAG INTENT
Unknown

TAG CATEGORY
Activity

CVES:
No associated CVEs

IP addresses with this tag have been observed attempting to scan for responses to DNS over HTTPS (DoH) requests.

3 DAYS 10 DAYS •30 DAYS October 01, 2023 - October 31, 2023

15

UNIQUE IPS OBSERVED BY GREYNOISE

I didn't have to do it

There are open datasets of historical DNS scans:

Project Sonar: https://opendata.rapid7.com/sonar.rdns_v2/

Dataset

```
:) SELECT formatReadableQuantity(count()) FROM dns

┌─formatReadableQuantity(count())─┐
│ 3.69 billion                    │
└─────────────────────────────────┘


:) SELECT formatReadableSize(total_bytes)
   FROM system.tables WHERE name = 'dns'

┌─formatReadableSize(total_bytes)─┐
│ 13.69 GiB                       │ -- we are storing JSON as a string
└─────────────────────────────────┘
```

Dataset

```
:) SELECT
    name,
    formatReadableSize(data_compressed_bytes) AS compr,
    formatReadableSize(data_uncompressed_bytes) AS raw,
    round(data_uncompressed_bytes / data_compressed_bytes, 2) AS ratio
FROM system.columns
WHERE table = 'dns' AND name = 'json'

┌─name─┬─compr──────┬─raw─────────┬─ratio─┐
│ json │ 13.66 GiB │ 896.18 GiB │  65.6 │
└──────┴────────────┴─────────────┴───────┘
```

Parse The Dataset

```
CREATE TABLE dns_parsed ENGINE = MergeTree ORDER BY ip
AS
WITH JSONExtract(json,
'Tuple(Status UInt8, TC Bool, RD Bool, RA Bool, AD Bool, CD Bool,
    Question Array(Tuple(name String)),
    Answer Array(Tuple(name String, type UInt8, TTL UInt32, data String)),
    Authority Array(Tuple(name String, type UInt8, TTL UInt32, data String)),
    Comment Array(String))') AS t
SELECT time, t.Status, t.TC, t.RD, t.RA, t.AD, t.CD,
toIPv4(byteSwap(IPv4StringToNum(extract(
  tupleElement(t.Question[1], 'name'), '^(\d+\.\d+\.\d+\.\d+)\.')))) AS ip,
replaceRegexpOne(tupleElement(t.Answer[1], 'data'), '\.$', '') AS domain
FROM dns WHERE t.Answer != ''
SETTINGS allow_experimental_analyzer = 1;
```

Parse The Dataset

```
CREATE TABLE dns_parsed
(
    `time` DateTime,
    `t.Status` UInt8,
    `t.TC` Bool,
    `t.RD` Bool,
    `t.RA` Bool,
    `t.AD` Bool,
    `t.CD` Bool,
    `ip` IPv4,
    `domain` String,
)
ENGINE = MergeTree
ORDER BY ip
```

Dataset

```
:) SELECT formatReadableSize(total_bytes) FROM system.tables
   WHERE name = 'dns_parsed'

┌─formatReadableSize(total_bytes)─┐
│ 4.88 GiB                        │
└─────────────────────────────────┘


:) SELECT name,
   formatReadableSize(data_compressed_bytes) AS compr,
   formatReadableSize(data_uncompressed_bytes) AS raw,
   round(data_uncompressed_bytes / data_compressed_bytes, 2) AS ratio
FROM system.columns WHERE table = 'dns_parsed'
ORDER BY data_compressed_bytes DESC

┌─name─────┬─compr──────┬─raw─────────┬─ratio─┐
│ ip       │ 3.24 GiB   │ 4.78 GiB    │  1.47 │
│ domain   │ 1.54 GiB   │ 42.09 GiB   │ 27.41 │
│ time     │ 31.68 MiB  │ 4.78 GiB    │ 154.43 │
│ t.RD     │ 859.97 KiB │ 1.19 GiB    │ 1456.31 │
│ t.RA     │ 859.97 KiB │ 1.19 GiB    │ 1456.31 │
│ t.Status │ 549.66 KiB │ 11.28 MiB   │ 21.02 │
│ t.AD     │ 151.60 KiB │ 6.03 MiB    │  40.7 │
│ t.TC     │ 96.74 KiB  │ 1.50 MiB    │  15.9 │
│ t.CD     │ 96.74 KiB  │ 1.50 MiB    │  15.9 │
└──────────┴────────────┴─────────────┴───────┘
```

Dataset

```
:) SELECT ip, domain FROM dns_parsed ORDER BY ip LIMIT 10

┌─ip─────────┬─domain──────────────────────┐
│ 1.0.0.1    │ one.one.one.one             │
│ 1.0.4.4    │ ns1.gtelecom.com.au         │
│ 1.0.4.5    │ auth1.gtelecom.com.au       │
│ 1.0.5.5    │ ns2.gtelecom.com.au         │
│ 1.0.6.6    │ ns3.gtelecom.com.au         │
│ 1.0.16.70  │ smtp.kakeibo.tepco.co.jp    │
│ 1.0.16.71  │ smtp01.kakeibo.tepco.co.jp  │
│ 1.0.16.72  │ smtp02.kakeibo.tepco.co.jp  │
│ 1.0.16.73  │ smtp03.kakeibo.tepco.co.jp  │
│ 1.0.16.74  │ smtp04.kakeibo.tepco.co.jp  │
└────────────┴─────────────────────────────┘
```

What are the most popular TLDs? Orgs?

Dataset

```
:) SELECT topLevelDomain(domain) AS d, count() AS c, bar(c, 0, 1e9, 50)
FROM dns_parsed GROUP BY d ORDER BY c DESC LIMIT 10

┌─d──┬─────────c─┬─bar(count(), 0, 1000000000., 50)─┐
│ net │ 406985266 │ ████████████████████             │
│ com │ 281925343 │ ██████████████                   │
│ jp  │  77920764 │ ████                             │
│ br  │  54343724 │ ███                              │
│ de  │  53843959 │ ███                              │
│ it  │  34083123 │ ██                               │
│ fr  │  24368263 │ ▉                                │
│ mx  │  21451597 │ ▉                                │
│ cn  │  18691219 │ ▉                                │
│ ru  │  17262632 │ ▉                                │
└─────┴───────────┴──────────────────────────────────┘
```

Dataset

```
:) SELECT cutToFirstSignificantSubdomain(domain) AS d, count() AS c,
    bar(c, 0, 1e8, 50)
FROM dns_parsed GROUP BY d ORDER BY c DESC LIMIT 10

┌─d──────────────────┬────────c─┬─bar(count(), 0, 100000000., 50)─┐
│ amazonaws.com      │ 56667330 │ ████████████████████████████████▊ │
│ comcast.net        │ 40638365 │ ███████████████████████▋          │
│ bbtec.net          │ 39534782 │ ██████████████████████▉           │
│ spectrum.com       │ 31682763 │ ██████████████████▍               │
│ sbcglobal.net      │ 31406669 │ ██████████████████▎               │
│ myvzw.com          │ 30122993 │ █████████████████▌                │
│ t-ipconnect.de     │ 25722620 │ ██████████████▊                   │
│ rr.com             │ 23476357 │ █████████████▌                    │
│ telecomitalia.it   │ 19438417 │ ███████████▎                      │
│ verizon.net        │ 16205490 │ █████████▍                        │
└────────────────────┴──────────┴──────────────────────────────────┘
```

Dataset

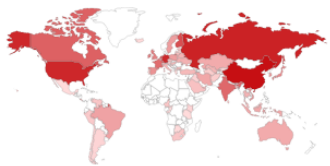
```
:) SELECT domain AS d, count() AS c
FROM dns_parsed WHERE domain LIKE '%clickhouse%'
GROUP BY d ORDER BY d

┌─d──────────────────────────────────────────────────┬─c─┐
│ appodeal-clickhouse-staging                        │ 1 │
│ bettrade.prod.clickhouse-1                         │ 1 │
│ bettrade.stage.clickhouse-1                        │ 1 │
│ bhs-clickhouse-0.justuno.com                       │ 1 │
│ bhs-clickhouse-1.justuno.com                       │ 1 │
│ bhs-clickhouse-2.justuno.com                       │ 1 │
│ bhs-clickhouse-3.justuno.com                       │ 1 │
│ bhs-clickhouse-4.justuno.com                       │ 1 │
│ bi-clickhouse-playground.dev.scraperapi.com        │ 1 │
│ bitmedia.clickhouse-distribution-nyc1              │ 1 │
│ bitmedia.clickhouse-log-nyc1                       │ 1 │
│ bitmedia.clickhouse-r1s1-nyc1                      │ 1 │
│ bitmedia.clickhouse-r1s2-nyc1                      │ 1 │
│ bitmedia.clickhouse-r1s2-nyc1                      │ 1 │
```

SHODAN | Explore | Downloads | Pricing | clickhouse

TOTAL RESULTS

23,505

TOP COUNTRIES



China	5,782
Russian Federation	3,908
United States	3,504
Germany	3,415
Finland	1,319

More...

TOP PORTS

8123	8,869
9000	7,720
9009	3,570
8443	1,528
9004	513

More...

View Report | View on Map

Access Granted: Want to get more out of your existing Shodan account? Check out everything you have access to.

116.202.46.146

static.146.46.202.116.clients.your-server.de
Hetzner Online GmbH
Germany, Nürnberg

database

```
HTTP/1.0 400 Bad Request

Port 9000 is for clickhouse-client program
You must use port 8123 for HTTP.

ClickHouse:
  Required Login: True
```

18.119.43.215

ec2-18-119-43-215.us-east-2.compute.amazonaws.com
Amazon Technologies Inc.
United States, Hilliard

cloud | database

```
HTTP/1.0 400 Bad Request

Port 9000 is for clickhouse-client program
You must use port 8123 for HTTP.

ClickHouse:
  Required Login: True
```

91.126.149.240

h-91-126-149-240.wholesale.adamo.es
Adamo Telecom Iberia S.A.U.
Spain, Barcelona

database

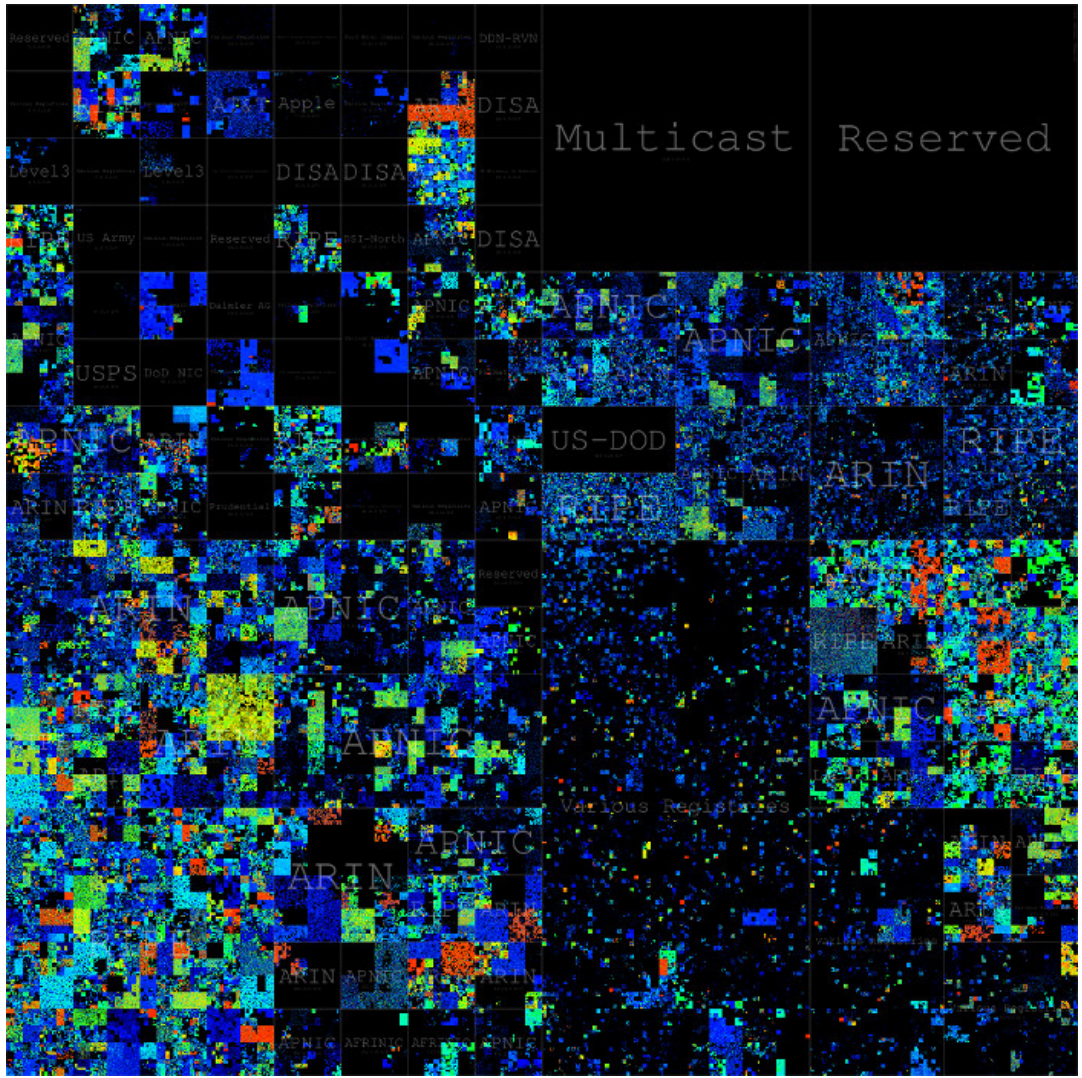
```
HTTP/1.1 200 OK
Date: Tue, 31 Oct 2023 08:38:24 GMT
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Keep-Alive: timeout=60
X-ClickHouse-Summary: {"read_rows":"0","read_bytes":"0","written_rows":"0","written_bytes":"0
```

Visualization

I want nice pictures out of this dataset!

What to do?

← I want something like this

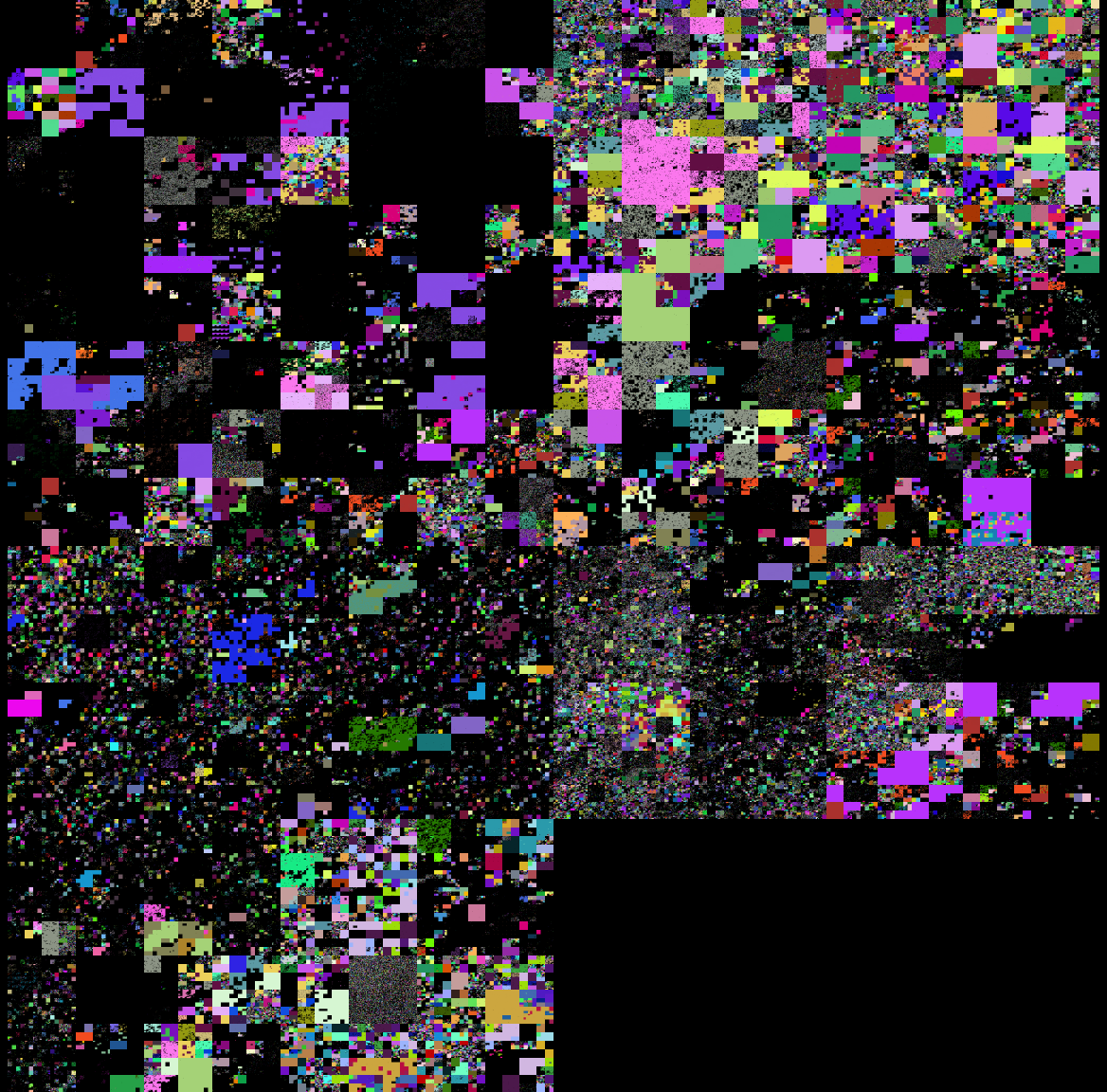


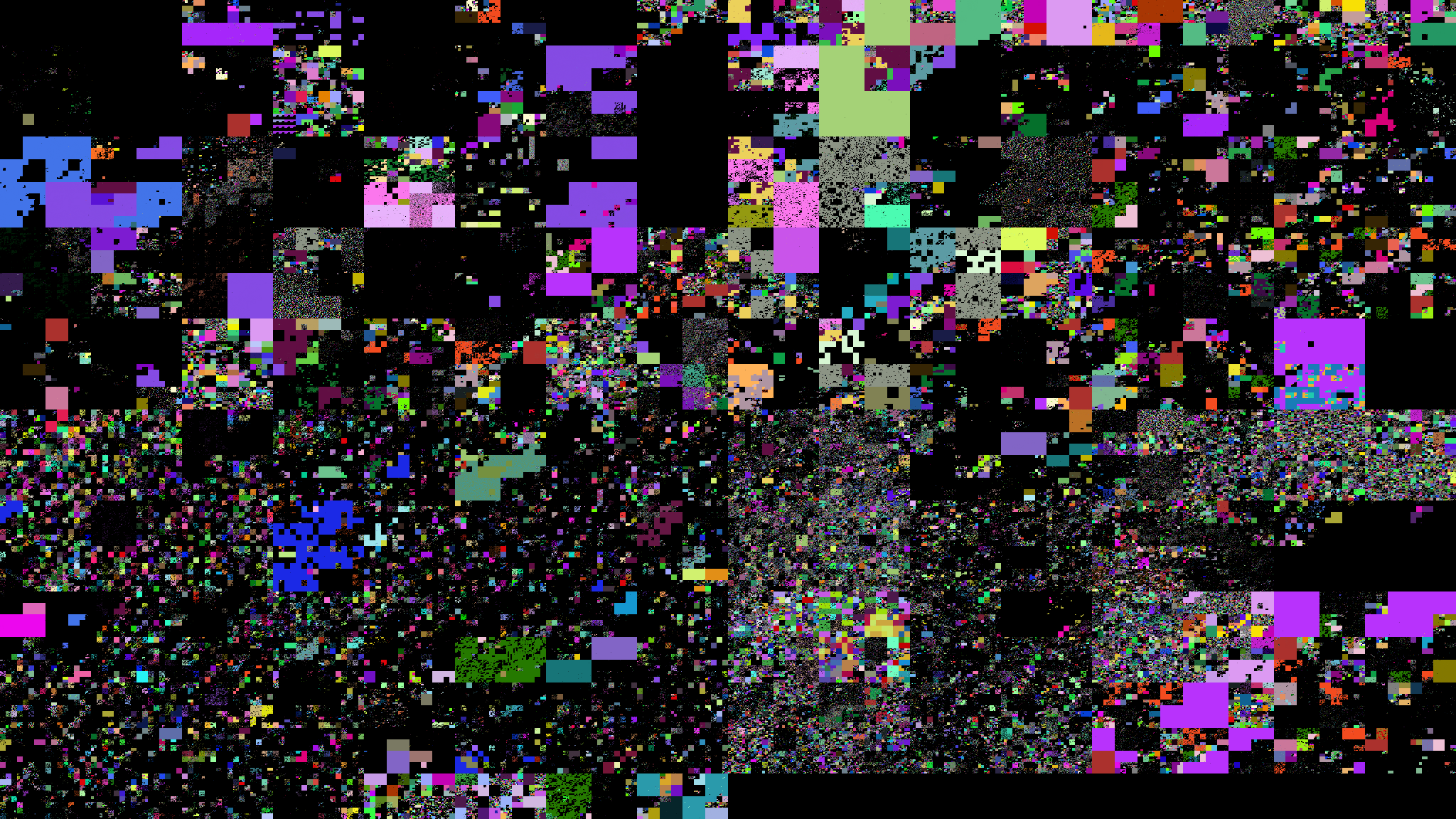
← I want this
but bigger and better

2018, Ben Cartwright-Cox
https://blog.benjojo.co.uk/post/scan-ping-the-internet-hilbert-curve

Drawing a Picture With ClickHouse

```
(echo 'P3 4096 4096 255';
clickhouse client --host ... --query "
  WITH 4096 AS w, 4096 AS h, w * h AS pixels,
    cutToFirstSignificantSubdomain(domain) AS tld,
    sipHash64(tld) AS hash,
    hash MOD 256 AS r,
    hash DIV 256 MOD 256 AS g,
    hash DIV 65536 MOD 256 AS b,
    toUInt32(ip) AS num,
    num DIV (0x100000000 DIV pixels) AS idx,
    mortonDecode(2, idx) AS coord
  SELECT avg(r)::UInt8, avg(g)::UInt8, avg(b)::UInt8
  FROM dns_parsed GROUP BY coord
  ORDER BY coord.2 * w + coord.1
    WITH FILL FROM 0 TO 4096*4096
" --progress
) | pnmtopng > image.png
```





Visualization

So what?

It is just 4K resolution.

I want a full picture of 4 gigapixel, 65536×65536!

It works — we can generate 4 gigapixel picture straight away.

But the picture does not open in many viewers.

Solution: Tiles

Let's represent a giant picture in the same way
as Google Maps — as a set of tiles.

And make a web page that will load the tiles on demand.

But now we have to use JavaScript...

Libraries for viewing large images:
— Leaflet: https://leafletjs.com/
— OpenSeaDragon: https://openseadragon.github.io/

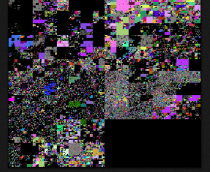
Tiles

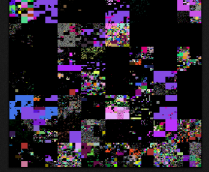
```
0-0-0.png  1-0-0.png  1-0-1.png  1-1-0.png  1-1-1.png  2-0-0.png
2-0-1.png  2-0-2.png  2-0-3.png  2-1-0.png  2-1-1.png  2-1-2.png
2-1-3.png  2-2-0.png  2-2-1.png  2-2-2.png  2-2-3.png  2-3-0.png
2-3-1.png  2-3-2.png  2-3-3.png  3-0-0.png  3-0-1.png  3-0-2.png
3-0-3.png  3-0-4.png  3-0-5.png  3-0-6.png  3-0-7.png  3-1-0.png
3-1-1.png  3-1-2.png  3-1-3.png  3-1-4.png  3-1-5.png  3-1-6.png
3-1-7.png  3-2-0.png  3-2-1.png  3-2-2.png  3-2-3.png  3-2-4.png
3-2-5.png  3-2-6.png  3-2-7.png  3-3-0.png  3-3-1.png  3-3-2.png
3-3-3.png  3-3-4.png  3-3-5.png  3-3-6.png  3-3-7.png  3-4-0.png
3-4-1.png  3-4-2.png  3-4-3.png  3-4-4.png  3-4-5.png  3-4-6.png
3-4-7.png  3-5-0.png  3-5-1.png  3-5-2.png  3-5-3.png  3-5-4.png
...
```

{zoom}-{x}-{y}.png

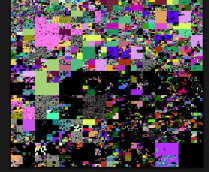
Tiles

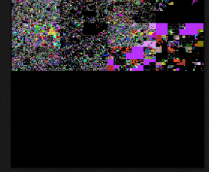
```
for zoom in {0..6}
do
  num_tiles=$((2**zoom))
  for tile_x in $(seq 0 $(($num_tiles - 1)))
  do
    for tile_y in $(seq 0 $(($num_tiles - 1)))
    do
      filename="tiles/${zoom}-${tile_x}-${tile_y}.png"
      echo $filename

      [ -s "$filename" ] ||
      (
        echo 'P3 1024 1024 255';
        clickhouse client --host driel7jwie.eu-west-1.aws.clickhouse-sta
          WITH 1024 AS w, 1024 AS h, w * h AS pixels,
            cutToFirstSignificantSubdomain(domain) AS tld,
            sipHash64(tld) AS hash,
            hash MOD 256 AS r, hash DIV 256 MOD 256 AS g, hash DIV 65536
```


0-0-0.png

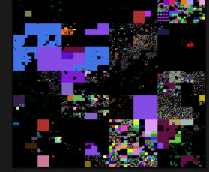

1-0-0.png

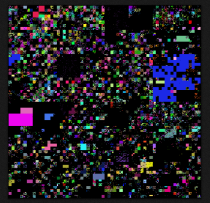

1-0-1.png

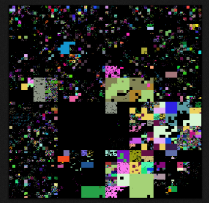

1-1-0.png

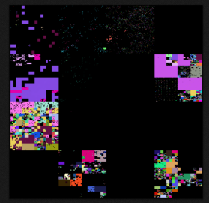

1-1-1.png

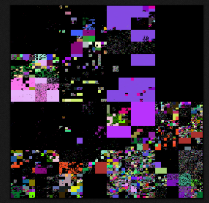

2-0-0.png

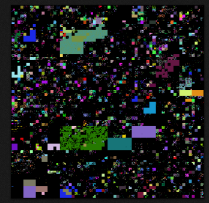

2-0-1.png

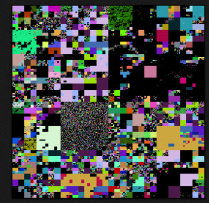

2-0-2.png

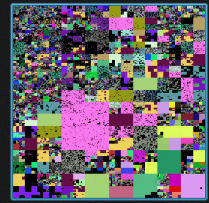

2-0-3.png

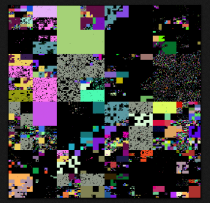

2-1-0.png

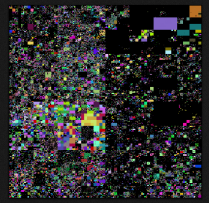

2-1-1.png


2-1-2.png

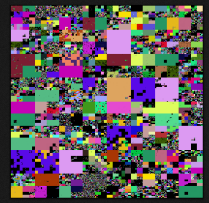

2-1-3.png

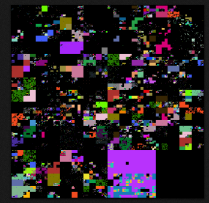

2-2-0.png

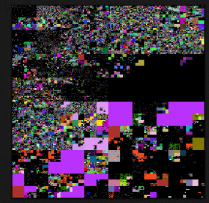

2-2-1.png


2-2-2.png

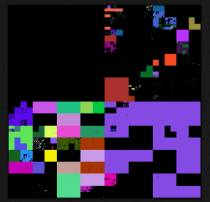

2-2-3.png

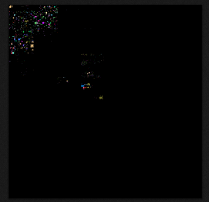

2-3-0.png

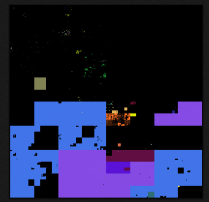

2-3-1.png

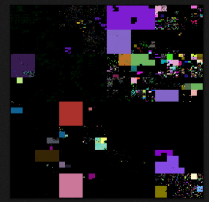

2-3-2.png

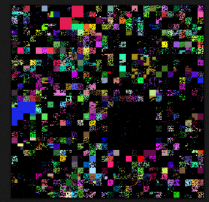

2-3-3.png

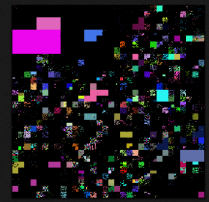

3-0-0.png

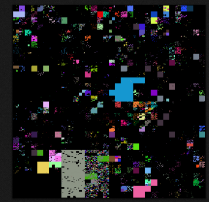

3-0-1.png

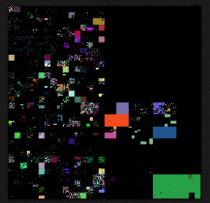

3-0-2.png

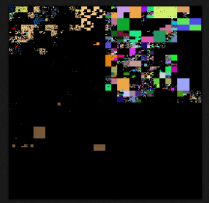

3-0-3.png

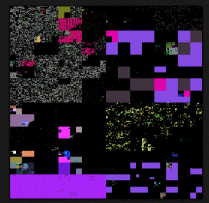

3-0-4.png

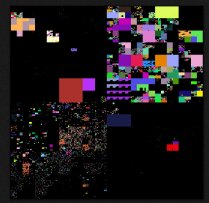

3-0-5.png

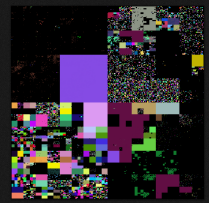

3-0-6.png

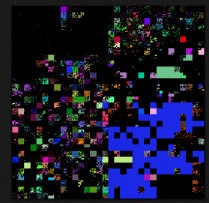

3-0-7.png

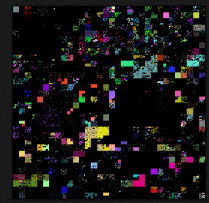

3-1-0.png

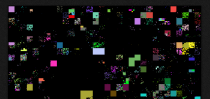

3-1-1.png

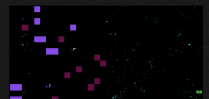

3-1-2.png


3-1-3.png


3-1-4.png


3-1-5.png





ReverseDNS.space

https://reversedns.space/ ⟵ click here

ReverseDNS.space

Every click generates an SQL query to ClickHouse:

```
const response = fetch(
    `https://driel7jwie.eu-west-1.aws.clickhouse-staging.com/?user=website`,
    {
        method: 'POST',
        body: `SELECT domain FROM dns_parsed WHERE ip = '${ip_str}' FORMAT JSON`
    }).then(response => response.json().then(o => {

        let host = o.rows ? o.data[0].domain : 'no response';

        if (ip_str == current_requested_ip) {
            popup.setContent(`<b>${ip_str}</b><br/>${host}`);
        }
    }));
```

ReverseDNS.space

Don't forget to set up users and quotas:

```
CREATE USER website IDENTIFIED WITH ...
SETTINGS
    add_http_cors_header = 1 READONLY,
    limit = 1 READONLY,
    offset = 0 READONLY,
    max_result_rows = 1 READONLY

GRANT SELECT ON default.dns_parsed TO website;

CREATE QUOTA website
    KEYED BY ip_address
    FOR RANDOMIZED INTERVAL 1 MINUTE MAX query_selects = 100,
    FOR RANDOMIZED INTERVAL 1 HOUR MAX query_selects = 3000,
    FOR RANDOMIZED INTERVAL 1 DAY MAX query_selects = 30000
    TO website;
```

Takeaways

Don't be afraid — you can use ClickHouse for everything.

You will find something new in the process.

Maybe you fill find a creative way to break something.

Source code:
https://github.com/ClickHouse/reversedns.space/

Source: Eh Bee Family, 2014,
https://vine.co/v/OwTOez0W6wg


Q&A