DNSTAPでDNSペイロードを可視化してみる

2019.06.28

DNS Summer Day 2019

株式会社オプテージ　櫻井　俊和

Summary

Introduction

Introduction

[お願い]

- 本発表は参考情報の提供を目的としております。当日限り の資料は公開を控えていただきますようお願いいたします。
- デモ、記載のデータは当社のラボ環境での検証結果です。

Introduction

当日限り

[現状]

- Subscriber
  -> FTTH 1.5M~, Mobile 1M~

- DNS Severs
  ->

- Software
  ->

- Query
  -> peak

- 可視化
  -> DNS Statistics Collector (DSC)

Introduction

[課題]

- デバッグ
  -> 問題発生!!デバッグしないといけない...
  -> どうしてもペイロード情報が欲しいときが...

- DNSペイロードを見る方法
  -> パケットキャプチャ
  -> クエリログをとる (Unboundは?)

Introduction

[DSC]

- https://www.dns-oarc.net/tools/dsc
- Pcap library Base
- Data
  - Qtypes/Rcodes/Opcodes/ Qname
  - Message sizes/EDNS parameters/Known types of DNS
  - Source addressess,subnets /IP transport/TCP,UDP ports

[Pros]

- 欲しいデータはそろっている
- Pcap + C でサーバ負荷にやさしい

[Cons]

- 検索できない
- リアルタイム性にかける
  (cron処理で遅れてやってくる)

-> ぱっと見るには十分だが、調査ツールとして使うはつらいところがある

[今日話すこと]

[Requirement1]

- 欲しいデータがそろっている
- サーバにやさしい
- DNS Softwareに依存しない

DNSTAPがよさげ

[Requirement2]

- リアルタイムに収集
- 検索したい

Messaging +column DBで
`BIGDATA`ぽくできるはず

- > DNSTAPでDNSペイロードを可視化してみる

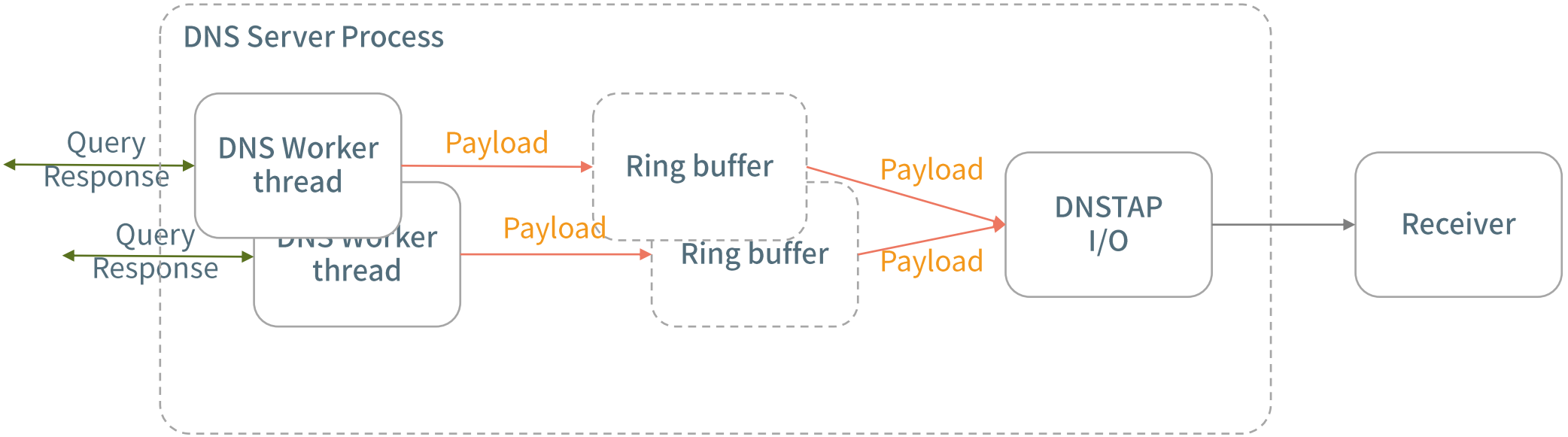
DNSTAP

DNSTAP

[1.概要]

- 公式 *http://dnstap.info/*
- Query/Responseのペイロードをoutput
- Protocol Buffers Base
- Light weight (drop > block)

DNSTAP の仕組み



[2.Support Ver]

Supported by DNS Software

SW	Ver	自分しらべ
BIND	9.11	http://dnstap.info/
Unbound	1.5.0	http://dnstap.info/
Knot DNS	1.5.0	http://dnstap.info/
Knot Res	1.2.5	http://dnstap.info/
NSD	4.1.23 (latest 4.1.27)	https://www.nlnetlabs.nl/news/2018/Dec/04/nsd-4.1.26-released/
CoreDNS	0.1.0	https://coredns.io/2017/08/03/logging-with-dnstap/
PowerDNS	-	not support ?

[3.Configured] (1/2)
- BIND -

Required library packages protobuf, protobuf-c, fstrm

- Configured:

```
$ ./configure --enable-dnstap
```

- named.conf:

```
options {
   dnstap { all; };
   // dnstap { auth; resolver query; resolver response; };

   /* where to capture to: file or unix (socket) */
   // dnstap-output file "/tmp/named.tap";
   dnstap-output unix "/var/run/dnstap.sock";

   dnstap-identity "tiggr";
   dnstap-version "bind-9.11.2";
};
```

[3.Configured] (2/2)

- Unbound -

Building Unbound with dnstap support requires that the fstrm and protobuf-c libraries be installed first.

- Configured:

```
$ ./configure --enable-dnstap
```

- unbound.conf:

```
dnstap:
    dnstap-enable: yes
    dnstap-socket-path: "/var/run/unbound/dnstap.sock"
    dnstap-send-identity: yes
    dnstap-send-version: yes
    dnstap-log-client-query-messages: yes
    dnstap-log-client-response-messages: yes
    dnstap-log-forwarder-query-messages: yes
    dnstap-log-forwarder-response-messages: yes
    dnstap-log-resolver-query-messages: yes
    dnstap-log-resolver-response-messages: yes
```

[4.Utility tools] (1/4)
ReceiverからDataを読みだすUtility tools

- golang-dnstap @`公式`

```
$ dnstap [-yqj] -u /var/run/unbound/dnstap.sock -w file.tap
```

- dnstap-read @`BIND utility`

```
$ dnstap-read [-m] [-p] [-y] file.tap
```

- dnstap-ldns @`ldns utility`

```
$ dnstap-ldns [-qyx] -r file.tap
```

- > golang-dnstapをつかってみる

[4.Utility tools] (2/4)

golang-dnstap のoutput formatは3種類

1. simple txt

```
$ dnstap [-q] -u /var/run/unbound/dnstap.sock
15:53:33.912506 CQ 127.0.0.1 UDP 28b "google.com." IN A
15:53:34.112489 CQ 127.0.0.1 UDP 28b "google.com." IN AAAA
15:53:33.912524 CR 127.0.0.1 UDP 44b "google.com." IN A
15:53:34.112502 CR 127.0.0.1 UDP 56b "google.com." IN AAAA
```

- > 情報すくなすぎ

[4.Utility tools] (3/4)

golang-dnstap のoutput formatは3種類

2. yml verbose

```
$ dnstap -y -u /var/run/unbound/dnstap.sock
type: MESSAGE
identity: "unbound1"
version: "unbound 1.5.0"
message:
  type: CLIENT_QUERY
  query_time: !!timestamp 2019-06-28 06:53:51.784424
  socket_family: INET
  socket_protocol: UDP
  query_address: 127.0.0.1
  query_port: 46401
  query_message: |
    ;; opcode: QUERY, status: NOERROR, id: 0
    ;; flags: rd; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 0
    ;; QUESTION SECTION:
    ;google.com.       IN      A
```

- > multilineは処理しずらい

[4.Utility tools] (4/4)

golang-dnstap のoutput formatは3種類

3. json verbose (2019.04.05 merge!!)

```
$ dnstap -j -u /var/run/unbound/dnstap.sock
{"type":"MESSAGE","identity":" unbound1 ","version":"unbound 1.5.0","message":{"type":"RESOLVER_QUERY","query_time":"2019-06-28T08:00:52.164606Z","socket_family":"INET","socket_protocol":"UDP","response_address":"156.154.100.5","response_port":53,"query_zone":"jp.","query_message":";; opcode: QUERY, status: NOERROR, id: 5411¥n;; flags: cd; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1¥n¥n;; QUESTION SECTION:¥n;g.dns.jp.¥tIN¥t AAAA¥n¥n;; ADDITIONAL SECTION:¥n¥n;; OPT PSEUDOSECTION:¥n; EDNS: version 0; flags: do; udp: 4096¥n"}}
```

- > 機械処理するならjsonがよさそう
- > nest構造は使いずらい

DNSTAP

[5. positiveとnegative]

[Pros]

- メジャーなSWはsupport済み
  (しかも使いたいVersionで!!)
- Ring Buffers(drop > block)でサーバにやさしい
- socketに投げるので、storage圧迫しない
- golang-dnstapを使えばSW非依存でdata取り出し可能

[Cons]

- 再インストール(compile)必要
- いくつかlibraryが追加で必要
- lostを許容する
- Receiverが死んだらどうなる？
  (process restartが必要?)

可視化

可視化

[1.アーキテクチャ](1/3)

- Clickhouse
  - column-oriented database
  - 公式 *https://clickhouse.yandex/*
  - Pros
    - select/bulk insert/compression is high performance
  - Cons
    - update/delete/join not support
  - Case
    - “cloudflare" 1M dns qps *1)
    - “.cl (cTLD)” Peak of 1.2 Tb/s. *2)

*1) *https://blog.cloudflare.com/how-cloudflare-analyzes-1m-dns-queries-per-second/*
*2) *https://www.usenix.org/sites/default/files/conference/protected-files/srecon18americas_slides_espinoza_bustos.pdf*

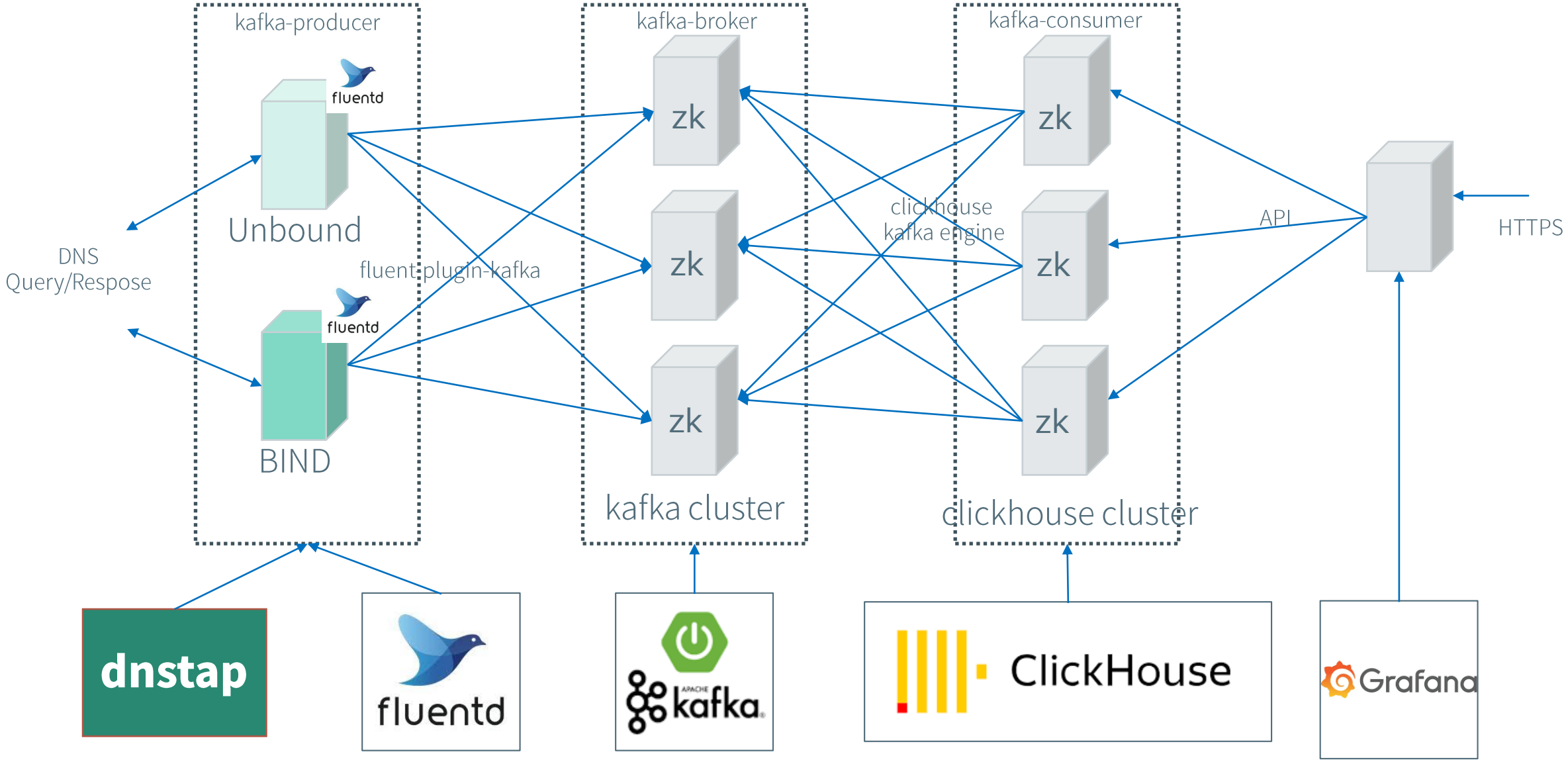
可視化

[1.アーキテクチャ](2/3)

- Apache Kafka
  - pub/sub messaging system
  - 公式 *https://kafka.apache.org/*
  - Pros *1)
    - Low Latency/high performance/Scalability
  - Cons *1)
    - No Complete Set of Monitoring Tools/Dependency on Zookeeper
  - Case
    - “cloudflare" 1M dns qps *2)
    - “LINE” 250 billlion/day. *3)

*1) *https://data-flair.training/blogs/advantages-and-disadvantages-of-kafka/*
*2) *https://blog.cloudflare.com/how-cloudflare-analyzes-1m-dns-queries-per-second/*
*3) *https://www.slideshare.net/linecorp/multitenancy-kafka-cluster-for-line-services-with-250-billion-daily-messages*

[1.アーキテクチャ](3/3) 構想編



[2.performance設計]
Performance設計をこのくらいに設定

Performance Requirements / qps

Peak	Now	Design
All node		
One node		

- > Query/Responseともに取得すると2倍のレコード転送が必要

[3.アーキテクチャ(Re:)](1/7)

Fluentdではだめだった(つらかった..)

- exec input plugin

```
<source>
@type exec
command sudo -u unbound /usr/local/go/bin/dnstap -u /var/unbound/dnstap.sock
format none
tag dnstap.hostname
</source>
```

- tail input plugin

```
<source>
@type tail
format none
tag dnstap.hostname
path /tmp/hogehoge.txt
pos_file /tmp/test/dnstap.tail.pos
</source>
```

- > multiproccesに対応してない

[3.アーキテクチャ(Re:)](2/7)

Fluentdではだめだった(つらかった..)

- kafka oputput plugin

```
<match dnstap.**>
  @type kafka2
  brokers kafka11:9092,kafka12:9092,kafka13:9092
  default_topic test
  <format>
    @type json
  </format>
  <buffer>
    @type file
    path /tmp/td-agent.¥*.buffer
  </buffer>
  required_acks 1
</match>
```

- > 30Krcord/secくらいが限界？

[3.アーキテクチャ(Re:)](3/7)

1. Fluentdに代わるoutput
- fluent-agent-lite
  - pros: perl、low cost(cpu,memory), high performance
  - cons: fluent -> kafkaの実装必要

- Flume
  - pros: high performance
  - cons: jvm実装。front-endにJVMをいれるの抵抗ある？

2. Data整形のつらさ
   Jsonネストを整形するため、 record/sec を外部parseでinput/outputは無理がある

[3.アーキテクチャ(Re:)](4/7)

結局..

golang-dnstap kafka output moduleの実装

- > golang-dnstap をforkした
  - parse --> golang-dnstap内で実装
  - output --> golang-dnstap内で実装

[3.アーキテクチャ(Re:)](5/7)

parse処理して、filtering + flat に変換


```
{
 “type”: "MESSAGE",
 "identity": "unbound1",
 "version": "unbound 1.5.8",
 "message": {
   "type": "CLIENT_RESPONSE",
   "response_time": "2019-06-28T10:13:37.760803Z",
   "socket_family": "INET",
   "socket_protocol": "UDP",
   "query_address": "127.0.0.1",
   "query_port": 35434,
   "response_message": ";; opcode: QUERY, status: NOERROR, id: 24¥n;;
flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL:
0¥n¥n;; QUESTION SECTION:¥n;google.co.jp.¥tIN¥t A¥n¥n;; ANSWER
SECTION:¥ngoogle.co.jp.¥t298¥tIN¥tA¥t216.58.196.227¥n"
  }
}
```

```
{
 "type": "MESSAGE",
 "identity": "unbound1",
 "version": "unbound 1.5.8",
 “msg_type": "CLIENT_RESPONSE",
 "timestamp": "2019-06-28T10:13:37.760803Z",
 "socket_family": "INET",
 "socket_protocol": "UDP",
 "query_address": "127.0.0.1",
 "query_port": 35434,
 “rcode”: “NOERROR”,
 “id”: 24,
 “query_type” : “A”,
 “query_name” : “google.co.jp”,
 “flags_aa”: 0,
 “flags_tc”: 0,
 “flags_rd”: 0,
 “flags_ra”: 0,
 “flags_ad”: 0,
 “flags_cd”: 0,
}
```

可視化

[3.アーキテクチャ(Re:)](6/7)

https://github.com/st3930/golang-dnstap/tree/feature/toml_kafka

* 興味あるかたは是非つかってください!!
* 必要なフィールドはこれだ!! など教えてください

可視化

[3.アーキテクチャ(Re:)](6/7)

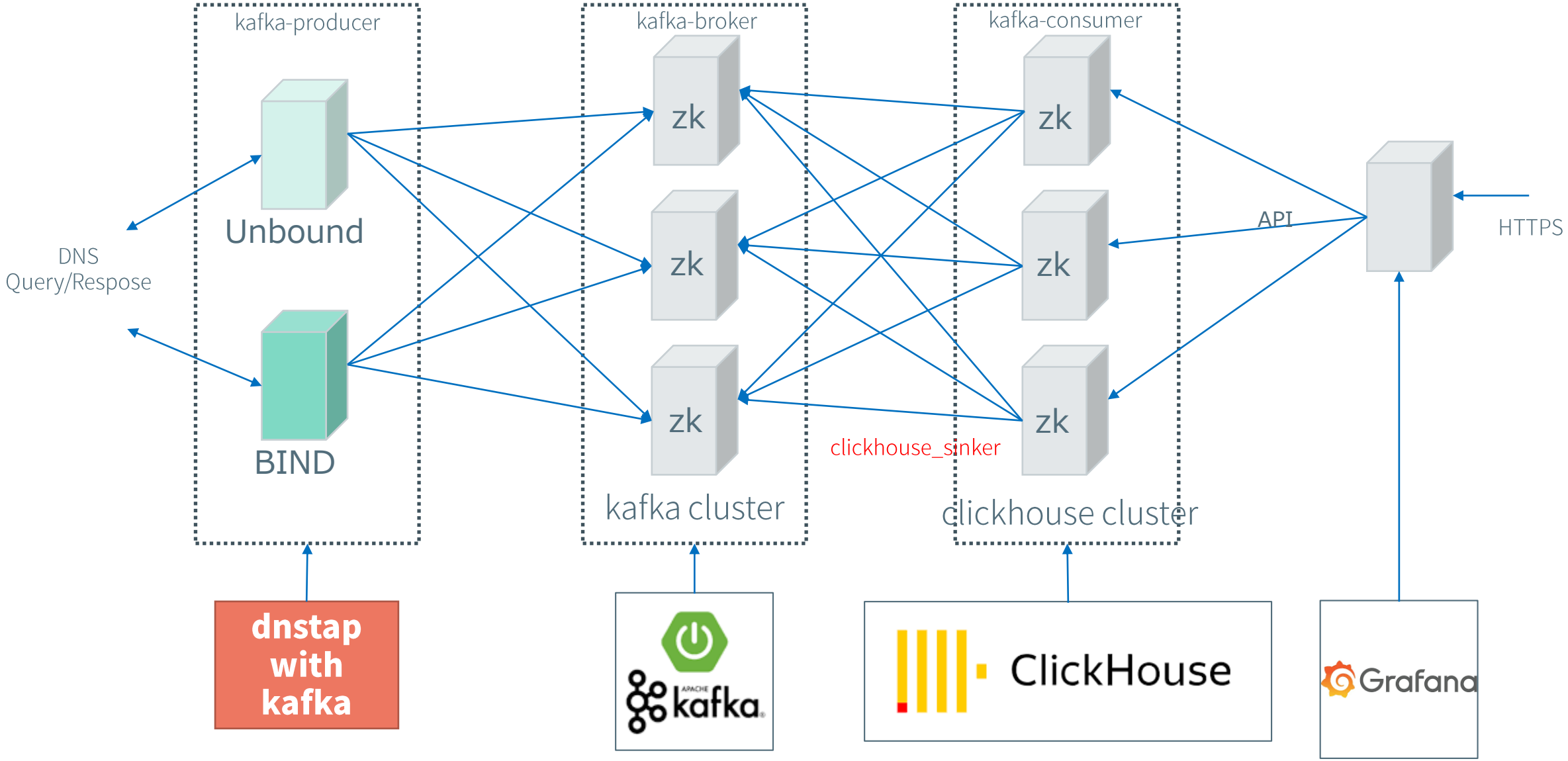
kafka からclickhouseへの取り込み
- > “clickhouse kafka engine”ではだめだった
    - offsetの取り方がうまくいかない？ Segmentの最後に欠損
    - clickhouse_sinker*1) を採用

* 知見あるかたは是非フィードバックを!!

1) https://github.com/housepower/clickhouse_sinker

デモ

デモ　(1/2)



デモ (2/2)

- dnsperfで
  - BIND <-- qps
  - Unbound <-- qps

- Grafanaで見てみる
  - ほぼリアルタイムで

まとめ

- DNSTAPは、ペイロード情報を取得するtoolsとしてかなり有力な候補になる
- 可視化するにはストリーム処理に関するノウハウが必要 (DNSとは直接関係ない!!)

Appendix

golang-dnstap

- golang-dnstap kafka outputの install

```
$ sudo yum install golang --enablerepo=epel

$ export GOPATH=`gopath`
$ sudo -E go get -d github.com/Shopify/sarama
$ sudo -E go get -d github.com/BurntSushi/toml
$ sudo -E go get -d github.com/st3930/golang-dnstap

$ cd $GOPATH/src/github.com/st3930/golang-dnstap
$ sudo git checkout -b feature/toml_kafka remotes/origin/feature/toml_kafka
$ git branch -a
$ sudo -E go install github.com/st3930/golang-dnstap/dnstap
$ cp -p $GOPATH/src/github.com/st3930/golang-dnstap/toml/dnstap.toml /etc/dnstap.toml

$ sudo -u [unbound|named] -c /etc/dnstap.toml -u `unix_socket_path'
```