Multicloud API bindings from documentation

Michał J. Gajda Vitor Vitali Barrozzi Gabriel Araujo
https://www.migamake.com

2021-02-18

Problem

Goals

- Tech stack agility

Goals

- Tech stack agility
  - any language

Goals

- Tech stack agility
  - any language
  - any library stack

Goals

- Tech stack agility
  - any language
  - any library stack
  - fast replacement

Goals

- Tech stack agility
  - any language
  - any library stack
  - fast replacement
- Combine best available microservices

Goals

- Tech stack agility
  - any language
  - any library stack
  - fast replacement
- Combine best available microservices
  - multi-vendor (AWS, GCP)

Goals

- Tech stack agility
  - any language
  - any library stack
  - fast replacement
- Combine best available microservices
  - multi-vendor (AWS, GCP)
  - multi-platform (not just SDK)

Goals

- Tech stack agility
  - any language
  - any library stack
  - fast replacement
- Combine best available microservices
  - multi-vendor (AWS, GCP)
  - multi-platform (not just SDK)
  - no-code and low-code APIs

Solution

- Code generation of API SDK bindings

Solution

- Code generation of API SDK bindings
- Per-API scraper (Python, Scrapy[Evans])

Solution

- Code generation of API SDK bindings
- Per-API scraper (Python, Scrapy[Evans])
- Common format (Haskell)

Solution

- Code generation of API SDK bindings
- Per-API scraper (Python, Scrapy[Evans])
- Common format (Haskell)
- Little generator code (Haskell)

Solution

- Code generation of API SDK bindings
- Per-API scraper (Python, Scrapy[Evans])
- Common format (Haskell)
- Little generator code (Haskell)
- Retargetable to other languages

Issues

Issues (1)

1. REST methods:

- `GET`

Issues (1)

1. REST methods:

- `GET`
- `POST`

Issues (1)

1. REST methods:

- `GET`
- `POST`
- `PUT`

Issues (1)

1. REST methods:

- `GET`
- `POST`
- `PUT`
- `DELETE`

Issues (1)

1. REST methods:

- `GET`
- `POST`
- `PUT`
- `DELETE`
- `PATCH`

Issues (2)

1. REST methods
2. *Parameter passing conventions*

HTTP request path variable:

```
POST /vpc/{vpcid}/status
```

Issues (2)

1. REST methods
2. *Parameter passing conventions*

HTTP request path variable:

`POST /vpc/{vpcid}/status`

URL-encoded query parameter:

`?parameter=name`

Issues (2)

1. REST methods
2. *Parameter passing conventions*

HTTP request path variable:

```
POST /vpc/{vpcid}/status
```

URL-encoded query parameter:

```
?parameter=name
```

HTTP request header:

```
X-AMZ-Transport-encoding: chunked
```

Issues (2.1)

HTTP request path variable

URL-encoded query parameter

HTTP request header:

HTTP cookie:

```
Cookie: AUTH_TOKEN=8498904328099
```

Issues (2.1)

HTTP request path variable

URL-encoded query parameter

HTTP request header:

HTTP cookie:

`Cookie: AUTH_TOKEN=8498904328099`

HTTP auth header

Issues (2.1)

HTTP request path variable

URL-encoded query parameter

HTTP request header:

HTTP cookie:

```
Cookie: AUTH_TOKEN=8498904328099
```

HTTP auth header

HTTP request body

```
{'userid' : 819, name: 'Alice'}
```

Issues (3)

1. REST methods
2. *Parameter passing conventions*
3. Type conversion
- JSON

Issues (3)

1. REST methods
2. *Parameter passing conventions*
3. Type conversion

- JSON
- special strings: `VPC id`, `userid`, `timestamp`

Issues (3)

1. REST methods
2. *Parameter passing conventions*
3. Type conversion

- JSON
- special strings: `VPC id`, `userid`, `timestamp`
- examples only

Issues (3)

1. REST methods
2. *Parameter passing conventions*
3. Type conversion

- JSON
- special strings: `VPC id`, `userid`, `timestamp`
- examples only
- use JSON Autotype

Dashboard

API	ALL	URL	Parameters	cURL	Calls
backblaze	85%	100%	100%	85%	27
cloudflare	85%	100%	85%	97%	488
transferwise	91%	98%	96%	93%	81
gitlab	99%	99%	100%	99%	885
linode	13%	100%	13%	97%	196
azure	17%	99%	100%	17%	5461
site24x7	95%	99%	100%	96%	388
digitalocean	100%	100%	100%	100%	224
dropbox	81%	100%	100%	81%	234
mailgun	100%	100%	100%	100%	81

Implementation

Implementation

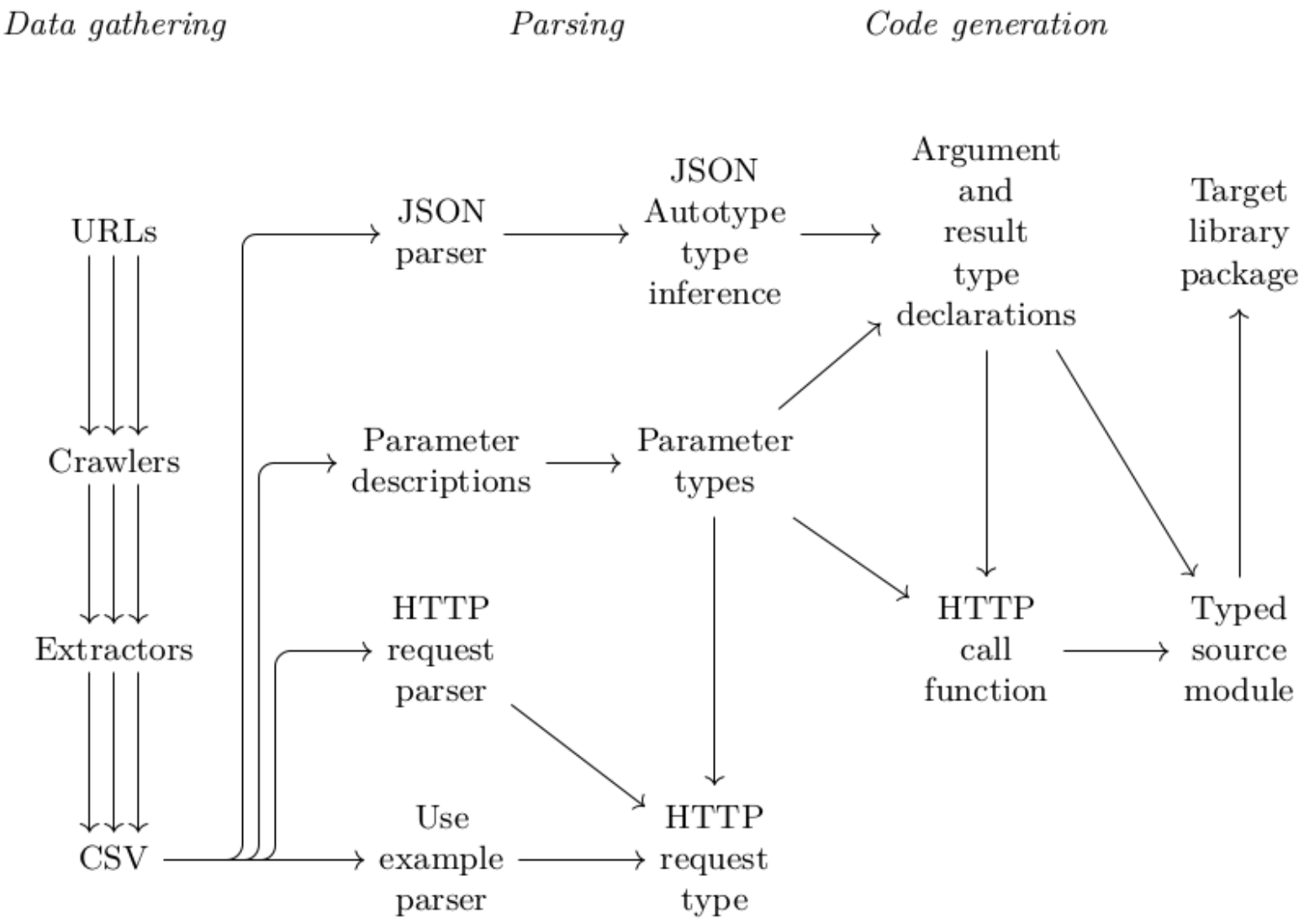


Figure 1: Data flow diagram

Implementation

1. Scraping

Implementation

1. Scraping
2. Parsing and analysis

Implementation

1. Scraping
2. Parsing and analysis
3. Code generation

Implementation (1)

1. Scraping[Evans]
   - Chrome driver

Implementation (1)

1. Scraping[Evans]

- Chrome driver
- XPath[Clark 1999]

Implementation (1)

1. Scraping[Evans]

- Chrome driver
- XPath[Clark 1999]
- CSS[CSS Working Group 2011]

Implementation (1)

1. Scraping[Evans]

- Chrome driver
- XPath[Clark 1999]
- CSS[CSS Working Group 2011]
- Regex, JQ, JSON Path

Implementation (2)

1. Scraping[Evans]
2. Parsing and analysis

- API call path

Implementation (2)

1. Scraping[Evans]
2. Parsing and analysis

- API call path
- JSON type inference

Implementation (2)

1. Scraping[Evans]
2. Parsing and analysis

- API call path
- JSON type inference
- cURL scripts

Implementation (3)

1. Scraping[Evans]
2. Parsing and analysis
3. Code generation:

- Haskell monad stack

Implementation (3)

1. Scraping[Evans]
2. Parsing and analysis
3. Code generation:

- Haskell monad stack
- agile data format

Implementation (3)

1. Scraping[Evans]
2. Parsing and analysis
3. Code generation:

- Haskell monad stack
- agile data format
- retargetable to other languages [Michal J. Gajda 2020, Gajda]

Code

```haskell
-- | From: 'http://documentation.mailgun.com/en/latest/api-
type MailgunDomainsDELETE =
     (:>) "domains"
       ((:>) (Capture "domain" DomainType)
          ((:>) (BasicAuth "*" ())
             (Delete '[JSON] MailgunDomainsDELETEResponse))
data MailgunDomainsDELETEArgs = MailgunDomainsDELETEArgs
  { mailgunDomainsDELETEArgsBasicAuth :: BasicAuthType
  , mailgunDomainsDELETEArgsDomain    :: DomainType }
-- | Delete a domain from your account.
mailgunDomainsDELETE :: MailgunDomainsDELETEArgs
                     -> SC.ClientM MailgunDomainsDELETEResp
mailgunDomainsDELETE args
  = SC.client (P.Proxy :: P.Proxy MailgunDomainsDELETE)
      (Just (mailgunDomainsDELETEArgsDomain args))
      (mailgunDomainsDELETEArgsBasicAuth args)
```

Limitations

- specialized components:
  - custom HTTP transport
  - custom auth
- replicating bugs from documentation
- some companies only seem to provide language-specific SDKs, no REST documentation # Benefits
- thousands of API calls available within months;

Limitations

- specialized components:
  - custom HTTP transport
  - custom auth
- replicating bugs from documentation
- some companies only seem to provide language-specific SDKs, no REST documentation # Benefits
- thousands of API calls available within months;
- easily retargetable;

Limitations

- specialized components:
  - custom HTTP transport
  - custom auth
- replicating bugs from documentation
- some companies only seem to provide language-specific SDKs, no REST documentation # Benefits
- thousands of API calls available within months;
- easily retargetable;
- easy scaling to other APIs;

Limitations

- specialized components:
  - custom HTTP transport
  - custom auth
- replicating bugs from documentation
- some companies only seem to provide language-specific SDKs, no REST documentation # Benefits
- thousands of API calls available within months;
- easily retargetable;
- easy scaling to other APIs;
- No longer depends on vendor support;

Limitations

- specialized components:
  - custom HTTP transport
  - custom auth
- replicating bugs from documentation
- some companies only seem to provide language-specific SDKs, no REST documentation # Benefits
- thousands of API calls available within months;
- easily retargetable;
- easy scaling to other APIs;
- No longer depends on vendor support;
- Reduce supported code base

Limitations

- specialized components:
  - custom HTTP transport
  - custom auth
- replicating bugs from documentation
- some companies only seem to provide language-specific SDKs, no REST documentation # Benefits
- thousands of API calls available within months;
- easily retargetable;
- easy scaling to other APIs;
- No longer depends on vendor support;
- Reduce supported code base
- Available as paid service from https://migamake.com/

Running example

```
data Record = Record {
    description :: String
  , amount      :: Int
  , date        :: CalendarDay
  }
```

Parsing interface

Parsing CSV row:

```
parseFields :: [String] -> Record
```

Parsing interface may fail

```
parseFields :: [String] -> Record
```

Parsing interface may fail

```
parseFields :: [String] -> Record

parseTable :: [[String]] -> Maybe [Record]
```

Parsing interface may fail

```
parseFields :: [String] -> Record

parseTable :: [[String]] -> Maybe [Record]
```

This declaration is the first important mistake you may make.

Parsing interface with error handling

```
parseFields :: [String] -> Maybe Record

parseFields :: [String] -> Either Why Record

newtype Why = Why String
```

Parsing interface with error handling

```
parseFields :: [String] -> Maybe Record
```

```
parseFields :: [String] -> Either Why Record
```

```
newtype Why = Why String
```

Maybe is root of ~~of all evil~~ ***opaque errors***.

Parsing interface with error handling

```
parseFields :: [String] -> Maybe Record
```

```
parseFields :: [String] -> Either Why Record
```

```
newtype Why = Why String
```

Maybe is root of ~~of all evil~~ ***opaque errors****.*

And this is error handling mistake that most Haskell libraries make.

Finding errors

```
parseFields :: [String] -> Either String Record

parseFields :: [String] -> Either Failure Record

data Failure  = Failure Where Why

newtype Where = Where { lineNo :: Int }
newtype Why   = Message String
```

Finding errors

```
parseFields :: [String] -> Either String Record

parseFields :: [String] -> Either Failure Record

data Failure  = Failure Where Why

newtype Where = Where { lineNo :: Int }
newtype Why   = Message String
```

Such error handling is totally unsuitable for large data sets.

Parsing interface for large datasets

Assume chance of single-record failure at 0.01

Parsing interface for large datasets

Assume chance of single-record failure at 0.01

Records	File size	Successful processing of entire file
10	1kB	99.9%
1000	100kB	90.5%
10k	1MB	36.8%
1M	100MB	$>$0.01%

Parsing interface for large datasets

```
parseFile :: FilePath -> IO ([Why], [Record])

parseRecord :: [String] -> Either [Why] Record

validateRecord :: Record -> ([Why], ValidRecord)
```

Parsing interface for larger datasets

```
newtype Why = Why [(Where, String)]

parseFields :: [String] -> (Issues, Successes)

type Successes = [Record]

newtype Issues = Issues [Issue]

data Issue  = Issue Where Why

newtype Where = Where { lineNo :: Int }
newtype Why   = Message String
```

Parsing interface for larger datasets

```
newtype Why = Why [(Where, String)]

parseFields :: [String] -> (Issues, Successes)

type Successes = [Record]

newtype Issues = Issues [Issue]

data Issue  = Issue Where Why

newtype Where = Where { lineNo :: Int }
newtype Why   = Message String

instance Monoid Issues where
  Issues   earlierFailures <>
    Issues laterFailures    = Failures (firstFailure <>
                                        secondFailure)
  mempty = Failures []
```

Did we just lose a **monad**!?

```
parseFields :: [String] -> (Failures, Successes)
```

Did we just lose a **monad**!?

```
parseFields :: [String] -> (Failures, Successes)

parseFields :: MonadWriter Failures m => [String] -> m Fail

class (Monoid w, Monad m) => MonadWriter w m | m -> w where
  tell :: w -> m () -- report a failure
```

Did we just lose a **monad**!?

```
parseFields :: [String] -> (Failures, Successes)

parseFields :: MonadWriter Failures m => [String] -> m Fai

class (Monoid w, Monad m) => MonadWriter w m | m -> w where
  tell :: w -> m () -- report a failure

mapData f =  concat
         <$> mapM (f >>= return ([]:)
                        `catch` (\issue -> tell >> return []))
```

Issue report is not just an error message

Did you ever see `head of empty list`?

Issue report is not just an error message

Did you ever see `head of empty list`?

```
newtype WhereInFile = WhereInFile { lineNo :: Int }
```

Issue report is not just an error message

Did you ever see `head of empty list`?

```
newtype WhereInFile = WhereInFile { lineNo :: Int }

data WhereInFiles = WhereInFiles { filename :: FilePath, li
```

Issue report is not just an error message

Did you ever see `head of empty list`?

```
newtype WhereInFile = WhereInFile { lineNo :: Int }

data WhereInFiles = WhereInFiles { filename :: FilePath, li

class Profunctor p where
  lmap :: (b -> a) -> p a c -> p b c
  rmap :: (c -> d) -> p a c -> p a d
```

Adding path to error failure

```
import Data.Aeson

prependFailure :: String -> Parser a -> Parser a

instance FromJSON Coord where
  parseJSON = withObject "Coord" $ \v -> Coord
        <$> prependFailure "failure parsing x" (v .: "x")
        <*> prependFailure "failure parsing y" (v .: "y")
```

For general error reporting, look at contravariant logging solutions.

Parsing - judge by impact

RECORD	ALL	URL	Parameter	cURL	#Calls
backblaze	96%	100%	100%	96%	27
office	3%	86%	100%	3%	1494
ibmcloud	0%	0%	0%	0%	0
cloudflare	86%	100%	87%	97%	488
transferwise	34%	98%	96%	37%	81
gitlab	98%	99%	100%	98%	2436
linode	13%	100%	13%	98%	196
azure	17%	99%	100%	17%	5461
site24x7	92%	99%	100%	93%	385
digitalocean	100%	100%	100%	100%	224
dropbox	94%	100%	100%	94%	234
mailgun	100%	100%	100%	100%	81
vultr	100%	100%	100%	100%	155

Aggregation of errors

Error message	Quantity	Severity
Failed to parse URL: ...	300	Error
Parameter of unknown type: ...	450	Warning
Failed to parse example response: ...	50	Error
Parameter not described: ...	50	Error

Aggregation of errors in Haskell

```
groupErrors :: [Why] -> [(Why, [Why])]
groupErrors = groupBy ((==)     `on` prefix)
            . sortBy  (compare `on` prefix)

prefix = head . breakOn (==":")
```

For more general solution, look at clustering with prefix trees.

Higher kinded data types

```
data Record f = Record {
    description :: f String
  , amount      :: f Int
  , date        :: f CalendarDay
  }

newtype Unparsed a = Unparsed String

type InputRecord = Record Unparsed

newtype Identity a = Identity a

type ValidRecord = Record Identity
```

Intermediate data formats

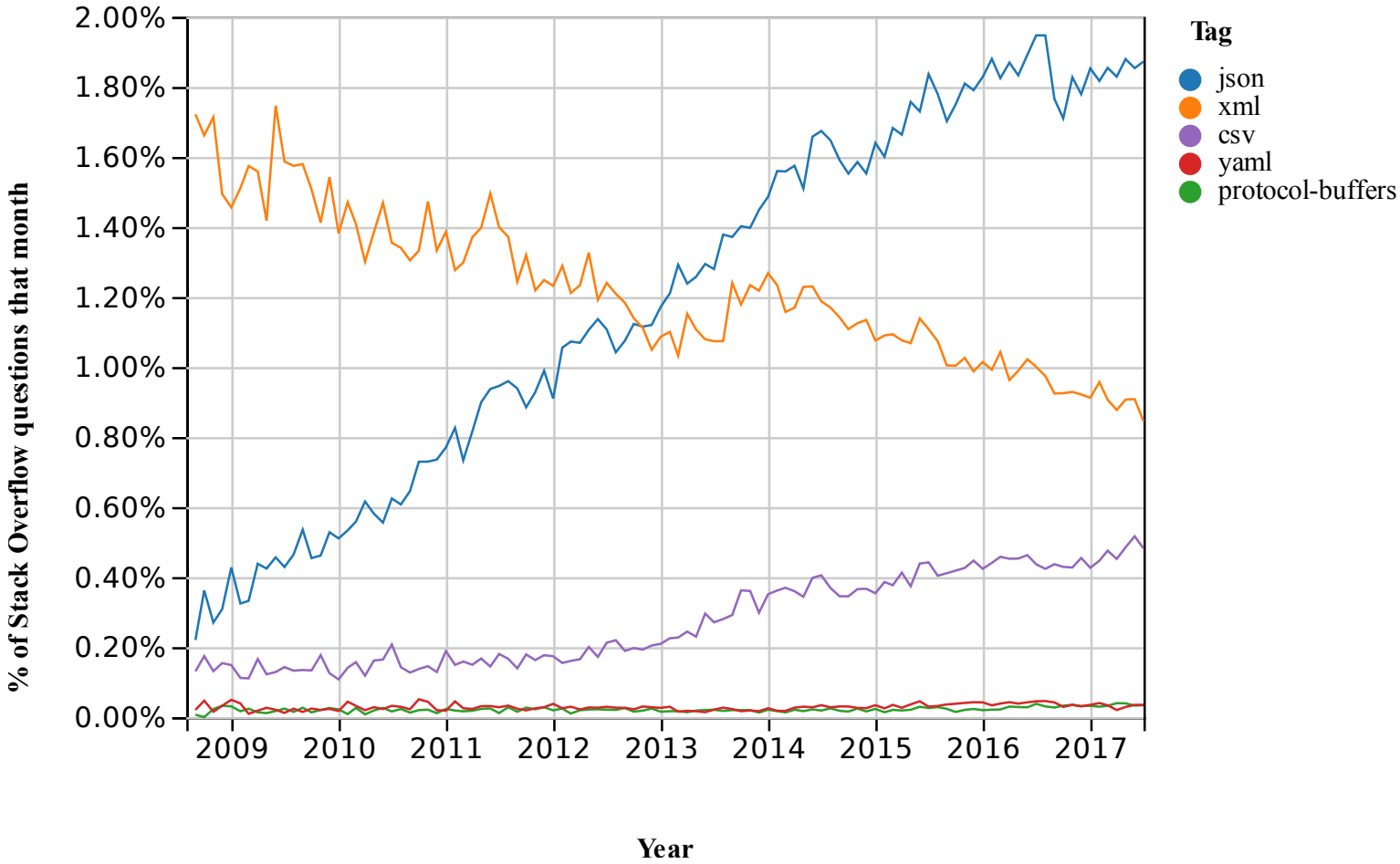


Figure 2: CSV growth

https://twobithistory.org/2017/09/21/the-rise-and-rise-of-json.html

Standard data formats

- Always browse data in standard formats
- Tables capture relations easier
- `Data.Csv.ToNamedRecord` are `Show` for tabular format
- Use pivot tables in your favourite office app

```
class ToNamedRecord a where
  toNamedRecord :: a -> NamedRecord
```

Standard data formats

- Always browse data in standard formats
- Tables capture relations easier
- `Data.Csv.ToNamedRecord` are `Show` for tabular format
- Use pivot tables in your favourite office app

```
class ToNamedRecord a where
  toNamedRecord :: a -> NamedRecord

class DefaultOrdered a where
  toNamedRecord :: a -> NamedRecord

type NamedRecord = HashMap ByteString ByteString
```

Standard field exports

```
instance (Csv.DefaultOrdered a
         ,Csv.ToNamedRecord a)
      =>  Csv.DefaultOrdered (Either String a) where
  headerOrder (_ :: Either String a) =
    Csv.header [Data.Vector.head subHeader<>"_error"] <>
    subHeader
```

Standard field exports

```
instance (Csv.DefaultOrdered a
         ,Csv.ToNamedRecord a)
      =>  Csv.DefaultOrdered (Either String a) where
  headerOrder (_ :: Either String a) =
    Csv.header [Data.Vector.head subHeader<>"_error"] <>
    subHeader

instance (Csv.DefaultOrdered                    a
         ,Csv.ToNamedRecord                     a)
      =>  Csv.ToNamedRecord (Either String a) where
```

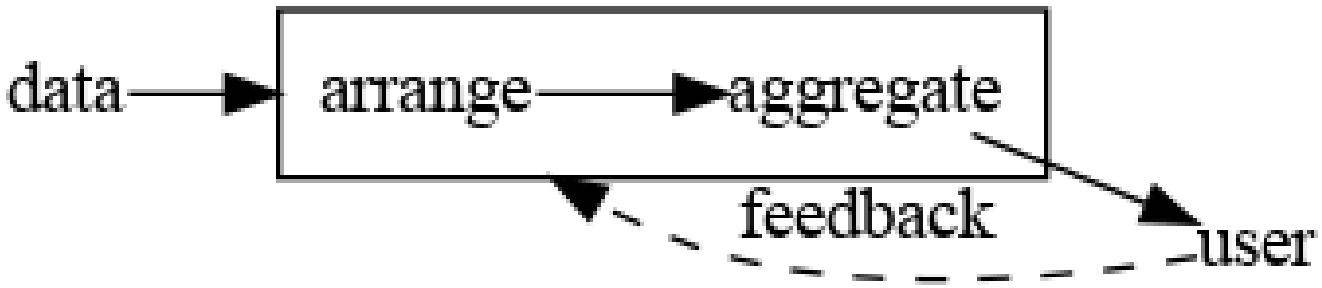
Standard field exports

```
instance (Csv.DefaultOrdered a
         ,Csv.ToNamedRecord a)
      =>  Csv.DefaultOrdered (Either String a) where
  headerOrder (_ :: Either String a) =
    Csv.header [Data.Vector.head subHeader<>"_error"] <>
    subHeader

instance (Csv.DefaultOrdered                    a
         ,Csv.ToNamedRecord                     a)
      =>  Csv.ToNamedRecord (Either String a) where
```

- You may use `cassava-generic`.

Aggregation as key to feedback



More on aggregations

RECORD	ALL	URL	Parameter	cURL	#Calls
backblaze	96%	100%	100%	96%	27
office	3%	86%	100%	3%	1494
cloudflare	86%	100%	87%	97%	488
transferwise	34%	98%	96%	37%	81
gitlab	98%	99%	100%	98%	2436
linode	13%	100%	13%	98%	196
azure	17%	99%	100%	17%	5461
site24x7	92%	99%	100%	93%	385
digitalocean	100%	100%	100%	100%	224
dropbox	94%	100%	100%	94%	234
mailgun	100%	100%	100%	100%	81
vultr	100%	100%	100%	100%	155

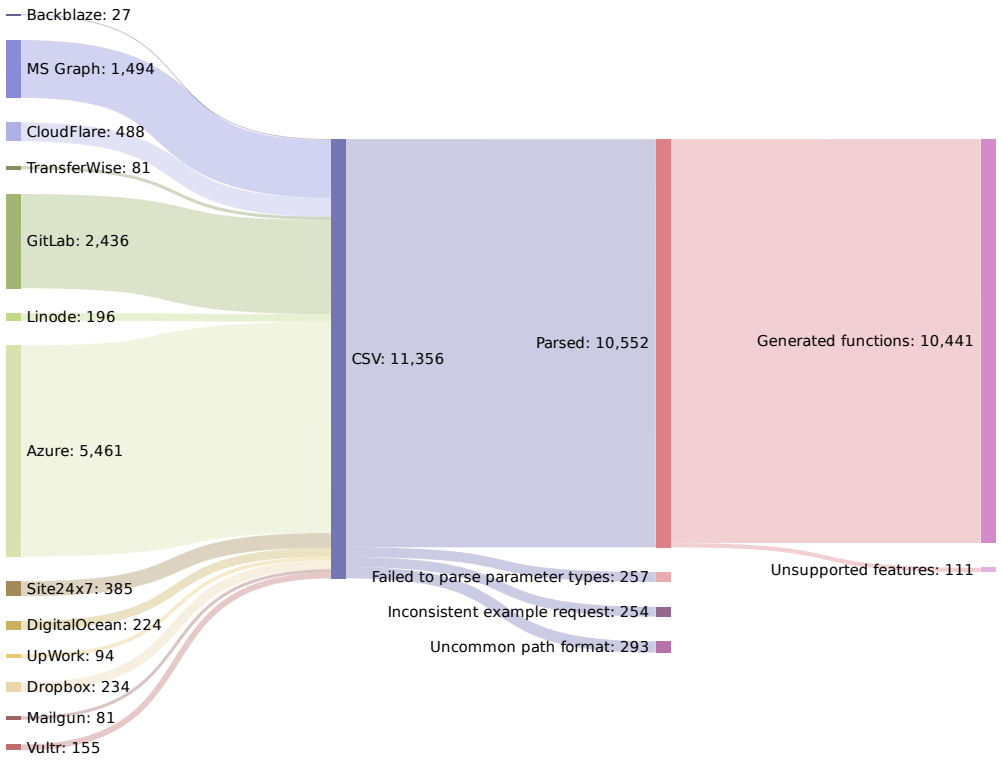
Aggregation and sort by size

RECORD	ALL	URL	Parameter	cURL	#Calls
azure	17%	99%	100%	17%	5461
gitlab	98%	99%	100%	98%	2436
office	3%	86%	100%	3%	1494
cloudflare	86%	100%	87%	97%	488
site24x7	92%	99%	100%	93%	385
dropbox	94%	100%	100%	94%	234
digitalocean	100%	100%	100%	100%	224
linode	13%	100%	13%	98%	196
vultr	100%	100%	100%	100%	155
transferwise	34%	98%	96%	37%	81
mailgun	100%	100%	100%	100%	81
backblaze	96%	100%	100%	96%	27
ibmcloud	0%	0%	0%	0%	0

Aggregation and sort by quality

RECORD	ALL	URL	Parameter	cURL	#Calls
digitalocean	100%	100%	100%	100%	224
mailgun	100%	100%	100%	100%	81
vultr	100%	100%	100%	100%	155
backblaze	96%	100%	100%	96%	27
dropbox	94%	100%	100%	94%	234
site24x7	92%	99%	100%	93%	385
cloudflare	86%	100%	87%	97%	488
transferwise	34%	98%	96%	37%	81
gitlab	98%	99%	100%	98%	2436
linode	13%	100%	13%	98%	196
azure	17%	99%	100%	17%	5461
office	3%	86%	100%	3%	1494
ibmcloud	0%	0%	0%	0%	0

Pipeline overview



Agile core principles

1. Individuals and interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following the plan

Agile *data pipeline* core principles

1. *Quality of observations* over fast iterations, continuous processes
2. Working *pipelines and aggregations* over schemas
3. *Analysis feedback* over expectations
4. *Revealing data* over following the process

Observations on **narrow subdatasets**

- Aggregation is key to feedback
- Analyze intermediate data using standard formats
- Never *discard* records, just *tag* them as errors
- Tagging with errors allows to analyze co-occurrence of problem
- Optimize for observations beside iteration speed
- Keep unique *source id* with each record (lesson from cloud monitoring)

References