Inferno Documentation

Release 0.1

Chango Inc.

October 21, 2016

Contents

Inferno is an open-source python map/reduce library, powered by Disco, and focused on the following.

- A **query language** for large amounts of **structured text** (csv, json, etc).
- A continuous and scheduled **map/reduce daemon** with an HTTP interface that automatically launches map/reduce jobs to handle a constant stream of incoming data.

CHAPTER 1

Questions?

Mailing list:

- https://groups.google.com/group/python-inferno

Developers:

- Tim Spurway (@t1m)
- Diana Clarke (@diana_clarke)
- Mazdak Rezvani (@mazdak)

Source/Bug Tracker:

- https://bitbucket.org/chango/inferno

CHAPTER 2

Table of Contents

2.1 Inferno Overview

2.1.1 Inferno Query Language

In its simplest form, you can think of Inferno as a query language for large amounts of structured text.

This structured text could be a CSV file, or a file containing many lines of valid JSON, etc.

people.json

```
{"first":"Homer", "last":"Simpson"}
{"first":"Manjula", "last":"Nahasapeemapetilon"}
{"first":"Herbert", "last":"Powell"}
{"first":"Ruth", "last":"Powell"}
{"first":"Bart", "last":"Simpson"}
{"first":"Apu", "last":"Nahasapeemapetilon"}
{"first":"Marge", "last":"Simpson"}
{"first":"Janey", "last":"Powell"}
{"first":"Maggie", "last":"Simpson"}
{"first":"Sanjay", "last":"Nahasapeemapetilon"}
{"first":"Lisa", "last":"Simpson"}
{"first":"Maggie", "last":"Términos"}
```

people.csv

```
first,last
Homer,Simpson
Manjula,Nahasapeemapetilon
Herbert,Powell
Ruth,Powell
Bart,Simpson
Apu,Nahasapeemapetilon
Marge,Simpson
Janey,Powell
Maggie,Simpson
Sanjay,Nahasapeemapetilon
Lisa,Simpson
Maggie,Términos
```

people.db

If you had this same data in a database, you would just use SQL to query it.

```
SELECT last_name, COUNT(*) FROM users GROUP BY last_name;

Nahasapeemapetilon, 3
Powell, 3
Simpson, 5
Términos, 1
```

Terminal

Or if the data was small enough, you might just use command line utilities.

```
diana@ubuntu:~$ awk -F ',' '{print $2}' people.csv | sort | uniq -c

3 Nahasapeemapetilon
3 Powell
5 Simpson
1 Términos
```

Inferno

But those methods don't necessarily scale when you're processing terabytes of structured text per day.

Here's what a similar query using Inferno would look like:

```
InfernoRule(
    name='last_names_json',
    source_tags=['example:chunk:users'],
    map_input_stream=chunk_json_keyset_stream,
    parts_preprocess=[count],
    key_parts=['last'],
    value_parts=['count'],
)
```

```
diana@ubuntu:~$ inferno -i names.last_names_json

last,count
Nahasapeemapetilon,3
Powell,3
Simpson,5
Términos,1
```

Don't worry about the details - we'll cover this rule in depth here. For now, all you need to know is that Inferno will yield the same results by starting a Disco map/reduce job to distributing the work across the many nodes in your cluster.

2.2 Getting Started

2.2.1 Disco

Before diving into Inferno, you'll need to have a working Disco cluster (even if it's just a one node cluster on your development machine).

This only takes a few minutes on a Linux machine, and basically just involves installing Erlang and Disco. See Installing Disco for complete instructions.

Bonus Points: A great talk about Disco and Map/Reduce.

2.2.2 Inferno

Inferno is available as a package on the Python Package Index, so you can use either pip or easy_install to install it from there.

```
diana@ubuntu:~$ sudo easy_install inferno
    -- or --
diana@ubuntu:~$ sudo pip install inferno
```

If you've got both Inferno and Disco installed, you should be able to ask Inferno for its version number:

```
diana@ubuntu:~$ inferno --version
inferno-0.1.17
```

2.2.3 Source Code

Both Disco and Inferno are open-source libraries. If you end up writing more complicated map/reduce jobs, you'll eventually need to look under the hood.

- Inferno: http://bitbucket.org/chango/inferno
- Disco: https://github.com/discoproject

2.3 Inferno Settings

2.3.1 Example Rules

The Inferno project comes with a few example rules. Tell inferno where to put those rules so that we can test your Inferno install.

```
diana@ubuntu:~$ inferno --example_rules example_rules

Created example rules dir:

        /home/diana/example_rules

                __init__.py
                names.py
                election.py
```

2.3.2 Settings File

On a development machine, the recommended place to put Inferno settings is: **~/.inferno**

1. Create a file called `.inferno` in your home directory.
2. Add your Disco master (`server`).
3. Add the example rules directory we just created (`rules_directory`).

When you're done, it should look something like the following.

```
diana@ubuntu:~$ cat /home/diana/.inferno
server: localhost
rules_directory: /home/diana/example_rules
```

Note: The other place inferno looks for these settings is inside **/etc/inferno/settings.yaml**

2.4 Example 1 - Count Last Names

The canonical map/reduce example: **count** the occurrences of words in a document. In this case, we'll count the occurrences of last names in a data file containing lines of json.

2.4.1 Input

Here's the input data:

```
diana@ubuntu:~$ cat data.txt
{"first":"Homer", "last":"Simpson"}
{"first":"Manjula", "last":"Nahasapeemapetilon"}
{"first":"Herbert", "last":"Powell"}
{"first":"Ruth", "last":"Powell"}
{"first":"Bart", "last":"Simpson"}
{"first":"Apu", "last":"Nahasapeemapetilon"}
{"first":"Marge", "last":"Simpson"}
{"first":"Janey", "last":"Powell"}
{"first":"Maggie", "last":"Simpson"}
{"first":"Sanjay", "last":"Nahasapeemapetilon"}
{"first":"Lisa", "last":"Simpson"}
{"first":"Maggie", "last":"Términos"}
```

2.4.2 Inferno Rule

And here's the Inferno rule (`inferno/example_rules/names.py`):

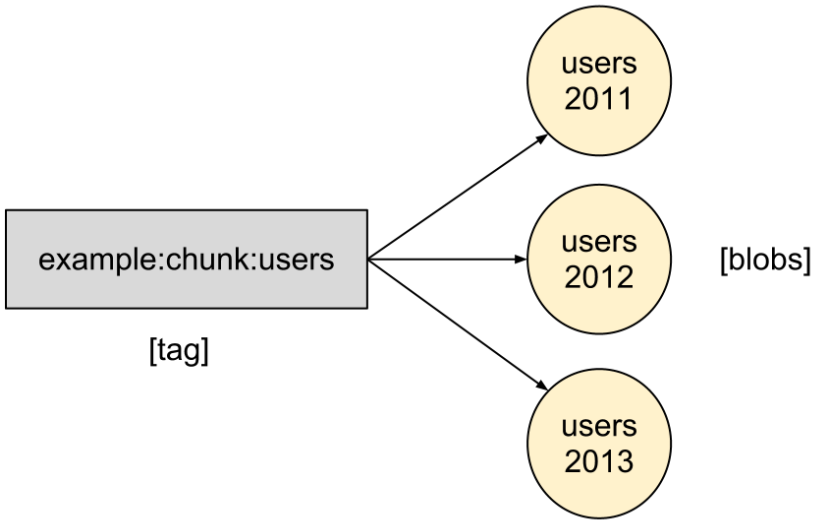
```
def count(parts, params):
    parts['count'] = 1
    yield parts


InfernoRule(
    name='last_names_json',
    source_tags=['example:chunk:users'],
    map_input_stream=chunk_json_keyset_stream,
    parts_preprocess=[count],
    key_parts=['last'],
    value_parts=['count'],
)
```

2.4.3 DDFS

The first step is to place the data file in Disco's Distributed Filesystem (DDFS). Once placed in DDFS, a file is referred to by Disco as a **blob**.

DDFS is a tag-based filesystem. Instead of organizing files into directories, you **tag** a collection of blobs with a **tag_name** for lookup later.

In this case, we'll be tagging our data file as **example:chunk:users**.



Make sure Disco is running:

```
diana@ubuntu:~$ disco start
Master ubuntu:8989 started
```

Place the input data in DDFS:

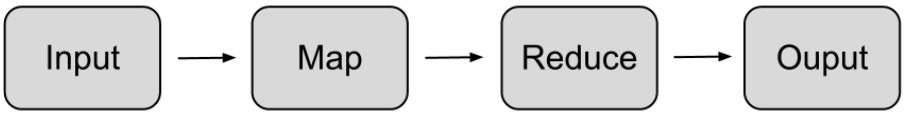
```
diana@ubuntu:~$ ddfs chunk example:chunk:users ./data.txt
created: disco://localhost/ddfs/vol0/blob/99/data_txt-0$533-406a9-e50
```

Verify that the data is in DDFS:

```
diana@ubuntu:~$ ddfs xcat example:chunk:users | head -2
{"first":"Homer", "last":"Simpson"}
{"first":"Manjula", "last":"Nahasapeemapetilon"}
```

2.4.4 Map Reduce

For the purpose of this introductory example, think of an Inferno map/reduce job as a series of four steps, where the output of each step is used as the input to the next.



Input

The input step of an Inferno map/reduce job is responsible for parsing and readying the input data for the map step.

If you're using Inferno's built in **keyset** map/reduce functionality, this step mostly amounts to transforming your CSV or JSON input into python dictionaries.

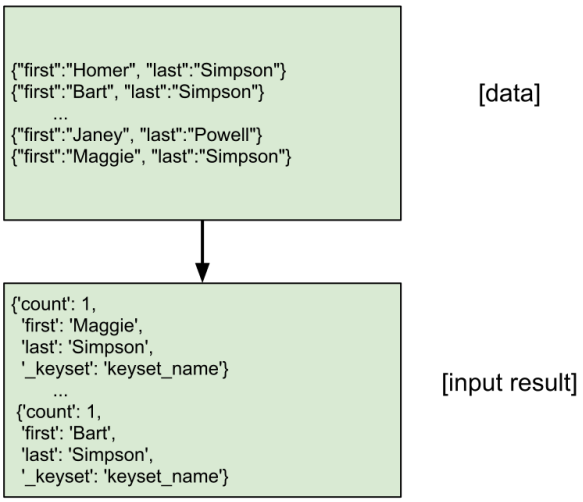
The default Inferno input reader is **chunk_csv_keyset_stream**, which is intended for CSV data that was placed in DDFS using the `ddfs chunk` command.

If the input data is lines of JSON, you would instead set the **map_input_stream** to use the **chunk_json_keyset_stream** reader in your Inferno rule.

The input reader will process all DDFS tags that are prefixed with the tag names defined in **source_tags** of your Inferno rule.

```
InfernoRule(
    name='last_names_json',
    source_tags=['example:chunk:users'],
    map_input_stream=chunk_json_keyset_stream,
    parts_preprocess=[count],
    key_parts=['last'],
    value_parts=['count'],
)
```

Example data transition during the **input** step:



click to enlarge

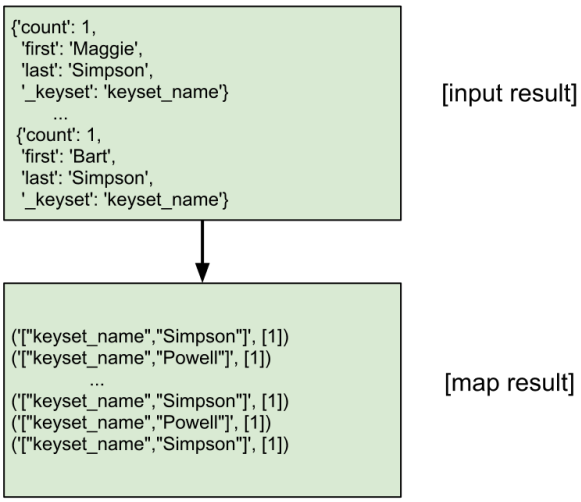
Map

The map step of an Inferno map/reduce job is responsible for extracting the relevant key and value parts from the incoming python dictionaries and yielding one, none, or many of them for further processing in the reduce step.

Inferno's default **map_function** is the **keyset_map**. You define the relevant key and value parts by declaring **key_parts** and **value_parts** in your Inferno rule.

```
InfernoRule(
    name='last_names_json',
    source_tags=['example:chunk:users'],
    map_input_stream=chunk_json_keyset_stream,
    parts_preprocess=[count],
    key_parts=['last'],
    value_parts=['count'],
)
```

Example data transition during the **map** step:



click to enlarge

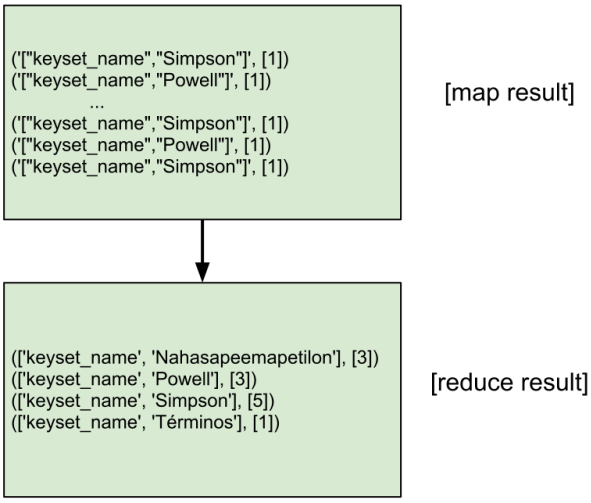
Reduce

The reduce step of an Inferno map/reduce job is responsible for summarizing the results of your map/reduce query.

Inferno's default **reduce_function** is the **keyset_reduce**. It will sum the value parts yielded by the map step, grouped by the key parts.

In this example, we're only summing one value (the `count`). You can define and sum many value parts, as you'll see here in the next example.

Example data transition during the **reduce** step:



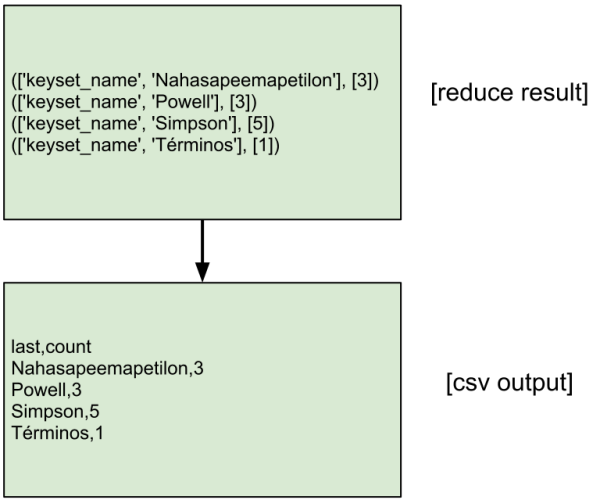
click to enlarge

Output

Unless you create and specify your own **result_processor**, Inferno defaults to the **keyset_result** processor which simply uses a CSV writer to print the results from the reduce step to standard output.

Other common result processor use cases include: populating a cache, persisting to a database, writing back to DDFS or DiscoDB, etc.

Example data transition during the **output** step:



click to enlarge

2.4.5 Execution

Run the last name counting job:

```
diana@ubuntu:~$ inferno -i names.last_names_json
2012-03-09 Processing tags: ['example:chunk:users']
2012-03-09 Started job last_names_json@533:40914:c355f processing 1 blobs
2012-03-09 Done waiting for job last_names_json@533:40914:c355f
2012-03-09 Finished job job last_names_json@533:40914:c355f
```

The output:

```
last,count
Nahasapeemapetilon,3
Powell,3
Simpson,5
Términos,1
```

2.5 Example 2 - Campaign Finance

In this example we'll introduce a few more key Inferno concepts:

- Inferno rules with **multiple keysets**
- **field_transforms**: input data casting
- **parts_preprocess**: a pre-map hook
- **parts_postprocess**: a post-reduce hook
- **column_mappings**: rename columns post-reduce

2.5.1 Inferno Rule

An Inferno rule to query the 2012 presidential campaign finance data. (`inferno/example_rules/election.py`):

```
import re

from inferno.lib.rule import InfernoRule
from inferno.lib.rule import Keyset
from inferno.lib.rule import chunk_csv_stream

# an example field_transform
def alphanumeric(val):
    return re.sub(r'\W+', ' ', val).strip().lower()


# an example parts_preprocess that modifies the map input
def count(parts, params):
    parts['count'] = 1
    yield parts


# an example parts_preprocess that filters the map input
def candidate_filter(parts, params):
    active = [
        'P20002721',  # Santorum, Rick
        'P60003654',  # Gingrich, Newt
        'P80000748',  # Paul, Ron
        'P80003338',  # Obama, Barack
        'P80003353',  # Romney, Mitt
    ]
    if parts['cand_id'] in active:
        yield parts


# an example parts_postprocess that filters the reduce output
def occupation_count_filter(parts, params):
    if parts['count_occupation_candidate'] > 1000:
        yield parts


RULES = [
    InfernoRule(
        name='presidential_2012',
        source_tags=['gov:chunk:presidential_campaign_finance'],
        map_input_stream=chunk_csv_stream,
        partitions=1,
        field_transforms={
            'cand_nm':alphanumeric,
            'contbr_occupation':alphanumeric,
        },
        parts_preprocess=[candidate_filter, count],
        parts_postprocess=[occupation_count_filter],
        csv_fields=(
            'cmte_id', 'cand_id', 'cand_nm', 'contbr_nm', 'contbr_city',
            'contbr_st', 'contbr_zip', 'contbr_employer', 'contbr_occupation',
            'contb_receipt_amt', 'contb_receipt_dt', 'receipt_desc',
            'memo_cd', 'memo_text', 'form_tp', 'file_num',
```

```
        ),
        csv_dialect='excel',
        keysets={
            'by_candidate':Keyset(
                key_parts=['cand_nm'],
                value_parts=['count', 'contb_receipt_amt'],
                column_mappings={
                    'cand_nm': 'candidate',
                    'contb_receipt_amt': 'amount',
                },
             ),
            'by_occupation':Keyset(
                key_parts=['contbr_occupation', 'cand_nm'],
                value_parts=['count', 'contb_receipt_amt'],
                column_mappings={
                    'count': 'count_occupation_candidate',
                    'cand_nm': 'candidate',
                    'contb_receipt_amt': 'amount',
                    'contbr_occupation': 'occupation',
                },
             )
        }
    )
]
```

2.5.2 Input

Make sure Disco is running:

```
diana@ubuntu:~$ disco start
Master ubuntu:8989 started
```

Get the 2012 presidential campaign finance data (from the FEC):

```
diana@ubuntu:~$ head -4 P00000001-ALL.txt
cmte_id,cand_id,cand_nm,contbr_nm,contbr_city,contbr_st,contbr_zip,contbr_employer,contbr_occupation,
C00410118,"P20002978","Bachmann, Michelle","HARVEY, WILLIAM","MOBILE","AL","366010290","RETIRED","RET
C00410118,"P20002978","Bachmann, Michelle","HARVEY, WILLIAM","MOBILE","AL","366010290","RETIRED","RET
C00410118,"P20002978","Bachmann, Michelle","BLEVINS, DARONDA","PIGGOTT","AR","724548253","NONE","RETI
```

Place the input data in Disco's Distributed Filesystem (DDFS):

```
diana@ubuntu:~$ ddfs chunk gov:chunk:presidential_campaign_finance:2012-03-19 ./P00000001-ALL.txt
created: disco://localhost/ddfs/vol0/blob/1c/P00000001-ALL_txt-0$533-86a6d-ec842
```

Verify that the data is in DDFS:

```
diana@ubuntu:~$ ddfs xcat gov:chunk:presidential_campaign_finance:2012-03-19 | head -3
C00410118,"P20002978","Bachmann, Michelle","HARVEY, WILLIAM","MOBILE","AL","366010290","RETIRED","RET
C00410118,"P20002978","Bachmann, Michelle","HARVEY, WILLIAM","MOBILE","AL","366010290","RETIRED","RET
C00410118,"P20002978","Bachmann, Michelle","BLEVINS, DARONDA","PIGGOTT","AR","724548253","NONE","RETI
```

2.5.3 Contributions by Candidate

Run the contributions by candidate job:

```
diana@ubuntu:~$ inferno -i election.presidential_2012.by_candidate
2012-03-19 Processing tags: ['gov:chunk:presidential_campaign_finance']
2012-03-19 Started job presidential_2012@533:87210:81a1b processing 1 blobs
2012-03-19 Done waiting for job presidential_2012@533:87210:81a1b
2012-03-19 Finished job presidential_2012@533:87210:81a1b
```

The output in CSV:

```
candidate,count,amount
gingrich newt,27740,9271750.98
obama barack,292400,81057578.81
paul ron,87697,15435762.37
romney mitt,58420,55427338.84
santorum rick,9382,3351439.54
```

The output as a table:

Candidate	Count	Amount
Obama Barack	292,400	$ 81,057,578.81
Romney Mitt	58,420	$ 55,427,338.84
Paul Ron	87,697	$ 15,435,762.37
Gingrich Newt	27,740	$ 9,271,750.98
Santorum Rick	9,382	$ 3,351,439.54

2.5.4 Contributions by Occupation

Run the contributions by occupation job:

```
diana@ubuntu:~$ inferno -i election.presidential_2012.by_occupation > occupations.csv
2012-03-19 Processing tags: ['gov:chunk:presidential_campaign_finance']
2012-03-19 Started job presidential_2012@533:87782:c7c98 processing 1 blobs
2012-03-19 Done waiting for job presidential_2012@533:87782:c7c98
2012-03-19 Finished job presidential_2012@533:87782:c7c98
```

The output:

```
diana@ubuntu:~$ grep retired occupations.csv
retired,gingrich newt,8810,2279602.27
retired,obama barack,74465,15086766.92
retired,paul ron,9373,1800563.88
retired,romney mitt,12798,6483596.24
retired,santorum rick,1752,421952.98
```

The output as a table:

Occupation	Candidate	Count	Amount
Retired	Obama Barack	74,465	$ 15,086,766.92
Retired	Romney Mitt	12,798	$ 6,483,596.24
Retired	Gingrich Newt	8,810	$ 2,279,602.27
Retired	Paul Ron	9,373	$ 1,800,563.88
Retired	Santorum Rick	1,752	$ 421,952.98

2.6 Inferno Daemon

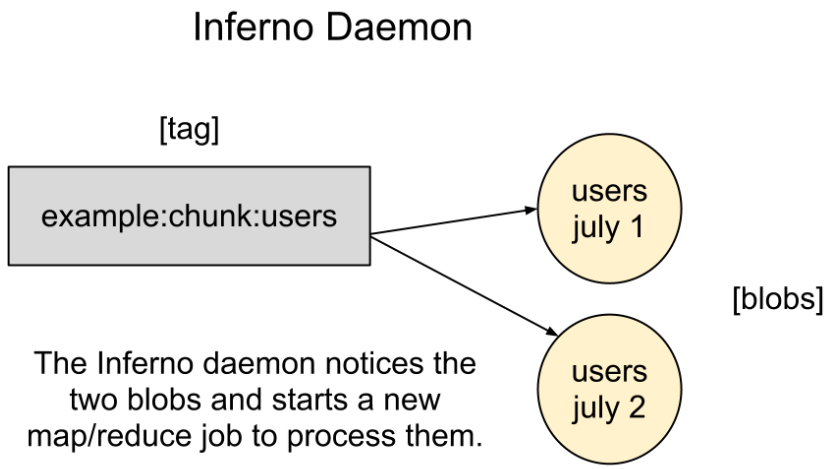
2.6.1 Immediate Mode

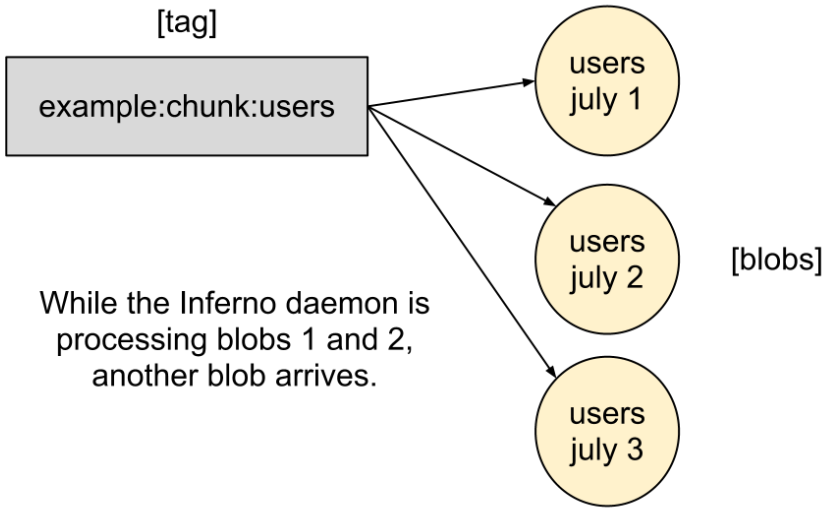
Up until now, the examples have all used Inferno's **immediate mode** (-i option). That is, they execute exactly one map/reduce job and then exit.

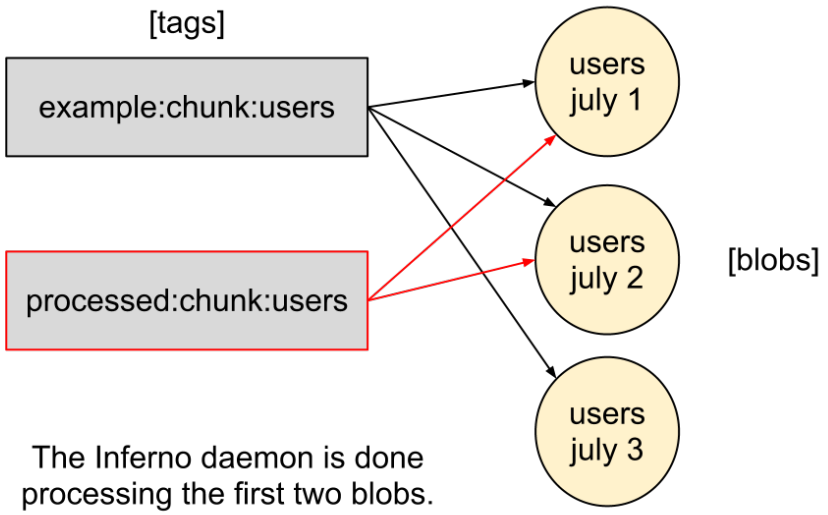
```
diana@ubuntu:~$ inferno -i names.last_names_json
last,count
Nahasapeemapetilon,3
Powell,3
Simpson,5
Términos,1
```

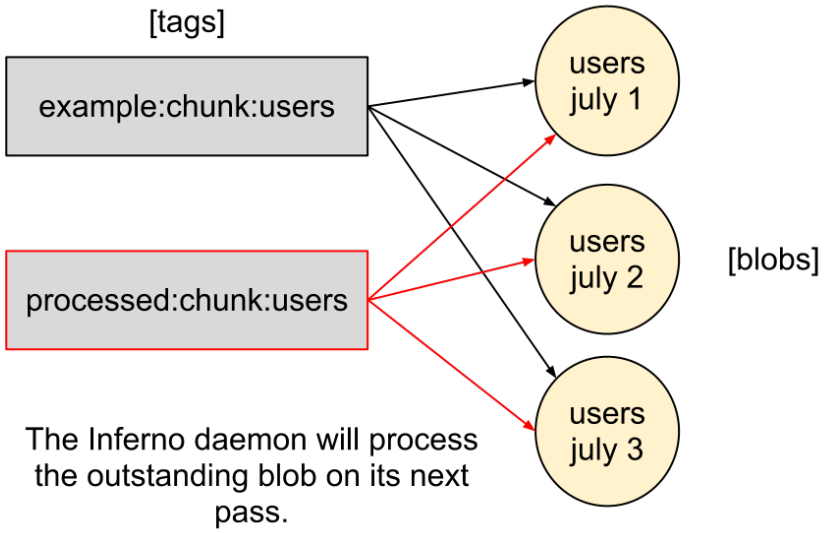
2.6.2 Daemon Mode

You can also run Inferno in **daemon mode**. The Inferno daemon will continuously monitor the blobs in DDFS and launch new map/reduce jobs to process the incoming blobs as the minimum blobs counts are met.




[tag]
example:chunk:users
users
july 1
users
july 2
users
july 3
[blobs]
While the Inferno daemon is
processing blobs 1 and 2,
another blob arrives.


[tags]
example:chunk:users
processed:chunk:users
users
july 1
users
july 2
users
july 3
[blobs]
The Inferno daemon is done
processing the first two blobs.



2.6.3 Example Daemon Logs

Here's the Inferno daemon in action. Notice that it skips the first **automatic rule** because the minimum blob count wasn't met. The next automatic rules's blob count was met, so the Inferno daemon processes those blobs and then persists the results to a data warehouse.

```
diana@ubuntu:~$ sudo start inferno
2012-03-27 31664 [inferno.lib.daemon] Starting Inferno...

...

2012-03-27 31694 [inferno.lib.job] Processing tags:['incoming:server01:chunk:task']
2012-03-27 31694 [inferno.lib.job] Skipping job task_stats_daily: 8 blobs required, have only 0

...

2012-03-27 31739 [inferno.lib.job] Processing tags:['incoming:server01:chunk:user']
2012-03-27 31739 [inferno.lib.job] Started job user_stats@534:d6c58:d5dcb processing 1209 blobs
2012-03-27 31739 [inferno.lib.job] Done waiting for job user_stats@534:d6c58:d5dcb
2012-03-27 31739 [rules.core.database] user_stats@534:d6c58:d5dcb: Saving user_stats_daily data in /t
2012-03-27 31739 [rules.core.database] user_stats@534:d6c58:d5dcb: Finished processing 240811902 line
2012-03-27 31739 [inferno.lib.archiver] Archived 1209 blobs to processed:server01:chunk:user_stats:20
```

2.7 Inferno Keysets

2.7.1 parts_preprocess

Part pre-processors are typically used to filter, expand, or modify the data before sending it to the map step.

The **parts_preprocess** functions are called before the **field_transforms** to ready the data for the **map_function**.

Note that a `parts_preprocess` function always takes `parts` and `params`, and must `yield` one, none, or many parts.

Example parts_preprocess:

```
def count(parts, params):
    parts['count'] = 1
    yield parts
```

```
def geo_filter(parts, params):
    if parts['country_code'] in params.geo_codes:
        yield parts
```

```
def insert_country_region(parts, params):
    record = params.geo_ip.record_by_addr(str(parts['ip']))
    parts['country_code'] = record['country_code']
    parts['region'] = record['region']
    yield parts
```

```
def slice_phrase(parts, params):
    terms = parts['phrase'].strip().lower().split(' ')
    terms_size = len(terms)
    for index, term in enumerate(terms):
        for inner_index in xrange(index, terms_size):
            slice_val = ' '.join(terms[index:inner_index + 1]).strip()
            parts_copy = parts.copy()
            parts_copy['slice'] = slice_val
            yield parts_copy
```

```
 InfernoRule(
     name='some_rule_name',
     source_tags=['some:ddfs:tag'],
     parts_preprocess=[
         insert_country_region,
         geo_filter,
         slice_phrase,
         count
     ],
     key_parts=['country_code', 'region', 'slice'],
     value_parts=['count'],
 ),
```

2.7.2 field_transforms

Field transforms are typically used to cast data from one type to another, or otherwise prepare the input for the map step.

The **field_transforms** happen before the **map_function** is called, but after **parts_preprocess** functions.

You often see `field_transforms` like `trim_to_255` when the results of a map/reduce are persisted to a database using a custom `result_processor`.

Example field_transforms:

```
def trim_to_255(val):
    if val is not None:
        return val[:254]
    else:
        return None
```

```
def alphanumeric(val):
    return re.sub(r'\W+', ' ', val).strip().lower()
```

```
def pad_int_to_10(val):
    return '%10d' % int(val)
```

```
def to_int(val):
    try:
        return int(val)
    except:
        return 0
```

```
 InfernoRule(
     name='some_rule_name',
     source_tags=['some:ddfs:tag'],
     field_transforms={
         'key1':trim_to_255,
         'key2':alphanumeric,
         'value1':pad_int_to_10,
         'value2':to_int,
     },
     key_parts=['key1', 'key2', 'key3'],
     value_parts=['value2', 'value2', 'value3'],
 ),
```

2.7.3 parts_postprocess

The **parts_postprocess** happen after the **map_function** is called.

2.7.4 key_parts

2.7.5 value_parts

2.7.6 column_mappings

2.8 Inferno Rules

2.8.1 rule_init_function

2.8.2 map_input_stream

2.8.3 map_init_function

2.8.4 map_function

2.8.5 reduce_function

2.8.6 result_processor

2.9 Inferno Command Line Interface

```
diana@ubuntu:~$ inferno --help
usage: inferno [-h] [-v] [-s SERVER] [-e SETTINGS_FILE] [-i IMMEDIATE_RULE]
               [-y RULES_DIRECTORY] [-q] [-f] [-x] [-D] [-d] [-p]
               [-t SOURCE_TAGS] [-u SOURCE_URLS] [-r RESULT_TAG]
               [-S DAY_START] [-R DAY_RANGE] [-O DAY_OFFSET] [-P PARAMETERS]
               [-l PARAMETER_FILE] [--data-file DATA_FILE]
               [--example_rules EXAMPLE_RULES]

Inferno: a python map/reduce library powered by disco.

optional arguments:
  -h, --help            show this help message and exit
  -v, --version         show program's version number and exit
  -s SERVER, --server SERVER
                        master disco server
  -e SETTINGS_FILE, --settings SETTINGS_FILE
                        path to settings file
  -i IMMEDIATE_RULE, --immediate-rule IMMEDIATE_RULE
                        execute <module>.<rule> immediately and exit
  -y RULES_DIRECTORY, --rules-directory RULES_DIRECTORY
                        directory to search for Inferno rules
  -q, --just-query      print out the blobs of the source query and generated
                        SQL, but don't execute the rule (only useful for
                        debugging rules in immediate mode)
  -f, --force           force processing of blobs
  -x, --start-paused    start Inferno without starting any automatic rules
                        (pause mode)
  -D, --disco-debug     debug map/reduce to disco console
  -d, --debug           debug flag for inferno consumers
  -p, --profile         output disco profiling data
  -t SOURCE_TAGS, --source-tags SOURCE_TAGS
```

```
                        override the ddfs source tags
  -u SOURCE_URLS, --source-urls SOURCE_URLS
                        override the source urls
  -r RESULT_TAG, --result-tag RESULT_TAG
                        override the ddfs result tag
  -S DAY_START, --day-start DAY_START
                        override the start day for blobs
  -R DAY_RANGE, --day-range DAY_RANGE
                        override the day range for blobs
  -O DAY_OFFSET, --day-offset DAY_OFFSET
                        override the days previous to start day for blobs
  -P PARAMETERS, --parameters PARAMETERS
                        additional rule parameters (in yaml)
  -l PARAMETER_FILE, --parameter-file PARAMETER_FILE
                        additional rule parameters (in a yaml file)
  --data-file DATA_FILE
                        arbitrary data file made available to job
  --example_rules EXAMPLE_RULES
                        create example rules
```

2.10 Inferno HTTP Interface

CHAPTER 3

Indices and tables

- genindex
- modindex
- search