

# JR: Quality Random Data from the Command line

**Ugo Landini - Staff Solutions Engineer** 

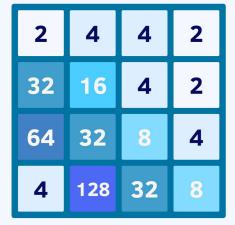
**Last updated: 28/06/23** 

**2**\$48

SCORE 1512 BEST 6056

Scoreboard

New Game



**HOW TO PLAY:** Use your **arrow keys** to move the tiles. When two tiles with the same number touch, they **merge into one!** 

**NOTE:** This game is the powered by <u>Confluent Cloud</u>. You can recreate this demo following <u>self-paced workshop</u>.

Demo by Gianluca Natali. Based on 2048 by Gabriele Cirulli.



https://gianlucanatali.github.io/streaming-games/index.html

#### > whoami



apiVersion: confluent/v1

kind: senior solutions engineer

metadata:

name: ugo landini

nick: ugol

email: ugo@confluent.io, ugo.landini@gmail.com

namespace: confluent

annotations: apache/committer, oss lover, distributed geek

site: <a href="https://ugol.io">https://ugol.io</a>

labels:

family: dad of two

prev\_companies: sun microsystems, vmware, red hat

spec:

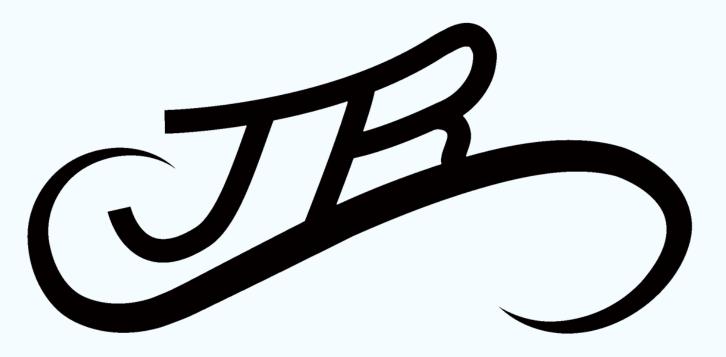
replicas: 1 containers:

- image: github.com/ugol:latest



# > apropos jr





## > apropos jr



- **J**son **R**andom generator
- Just another Random generator
- Similar to JQ, which is one of the tools I use most <a href="https://stedolan.github.io/jg/">https://stedolan.github.io/jg/</a>
- ...



## > apropos jr

\*

- Json Random (
- Just another R
- Similar to JQ, v <u>https://stedolar</u>
- ..





### > history | grep jr



- Had to generate traffic for a customer, on the fly, with just an example of a json
- They asked how much this stuff would be compressed by the producer, which obviously varies with:
  - different algorithms
  - different throughput
  - different batching kafka configuration
  - o can't use a single json to do that, would be compressed **too much**
- Existing tools couldn't easily answer this question, and for sure not in a 5 minutes time frame, for example:
  - Datagen with custom objects is complex to setup
  - Managed **Datagen** on Confluent Cloud can't use custom objects and can't compress

#### > history | grep jr



```
"VLAN": "DELTA",
"IPV4_SRC_ADDR": "10.1.41.98",
"IPV4 DST ADDR": "10.1.137.141",
"IN_BYTES": 1220,
"FIRST_SWITCHED": 1681984281,
"LAST SWITCHED": 1682975009,
"L4_SRC_PORT": 81,
"L4_DST_PORT": 80,
"TCP FLAGS": 0,
"PROTOCOL": 1,
"SRC_TOS": 211,
"SRC_AS": 4,
"DST AS": 1,
"L7_PR0T0": 443,
"L7_PROTO_NAME": "ICMP",
"L7_PROTO_CATEGORY": "Application"
```

Copyright 2021, Confluent, Inc. All rights reser

#### > history | grep jr

Copyright 2021, Co



```
"VLAN": "{{randoms "ALPHA|BETA|GAMMA|DELTA"}}",
"IPV4_SRC_ADDR": "{{ip "10.1.0.0/16"}}",
"IPV4_DST_ADDR": "{{ip "10.1.0.0/16"}}",
"IN_BYTES": {{integer 1000 2000}},
"FIRST_SWITCHED": {{unix_time_stamp 60}},
"LAST_SWITCHED": {{unix_time_stamp 10}},
"L4_SRC_PORT": {{ip_known_port}},
"L4_DST_PORT": {{ip_known_port}},
"TCP_FLAGS": 0,
"PROTOCOL": {{integer 0 5}},
"SRC_TOS": {{integer 128 255}},
"SRC_AS": {{integer 0 5}},
"DST_AS": {{integer 0 2}},
"L7_PROTO": {{ip_known_port}},
"L7_PROTO_NAME": "{{ip_known_protocol}}}",
"L7_PROTO_CATEGORY": "{{randoms "Network|Application|Transport|Session"}}"
```

## > whois jr



- Is a template system, leveraging wonderful Golang text/template package
- Has a CLI but also REST APIs
- Can generate anything you could write a template for (so, not tied to json)
- Embeds a specialized fake library (no use of existing faking libraries)
- Has automatic integrity for related fields (city, zip, mobile, phone, email/company, etc)
- Can maintain integrity between objects generated (relations)
- It's been designed for Kafka, but can directly output to Elastic, Redis,
   MongoDB, S3
- Can talk to Confluent Schema Registry for Kafka, serializing in Avro/Json
   Schema

### > man jr



- You choose your template from the available templates
- You choose -n number of objects to generate at each pass
- You choose -f frequency
- You choose -d duration

```
jr template list
jr template run net_device | jq
jr template run -n 2 net_device | jq
jr template run -n 2 -f 100ms net_device | jq
jr template run -n 2 -f 100ms -d 5s net_device | jq
```

#### > man template



- There are 3 different templates to control jr
  - Key template, which defaults to null
  - Output template, which defaults to Value only: {{.V/n}}
  - Value template, which you control in two different ways
    - Embedding directly in the command line (--embedded)
    - By name (**user,net\_device**, etc) for the OOTB templates

```
jr template list
jr template show net_device
jr template show user
jr template run --key '{{key "ID" 100}}' user
jr template run --key '{{key "ID" 100}}' --outputTemplate '{{.K}} {{.V}}' net_device
jr template run --key '{{key "ID" 100}}' --embedded '{{name}} {{email}}' --kcat
```

#### > cat cli



- You have 3 resources: emitters, templates and functions
  - You can list, show and run templates
  - You can list available **functions** and test directly (--run) without writing a template. There are **127** functions at the moment, and growing
  - Emitters are a new concept: you configure different emitters all at once, with different frequency and other parameters, and then you just list/show/run the emitters with a single command

```
jr function list -c finance
jr function list card --run
jr function list regex --run
jr emitter list
jr emitter run
```

#### > man functions



- There are 127 functions at the moment, categorized as
  - People
  - Text utilities
  - Network
  - Context
  - Address
  - Finance
  - o Math
  - Phone

```
cat .jr/templates/data/it/movie
jr template run --template '{{from "movie"}}'
jr template run --locale IT --template '{{from_n "beer" 3}}'
jr template run --locale IT --template '{{from_n "actor" 15}}'
```

#### > cat automatic\_integrity



- Some functions are "smart", for example:
  - Mobile phones are generated by "inverse" regular expressions, using mobile company numbers valid for the chosen country (--locale)
  - Streets, cities, zip codes, phone prefix and more are all localizable and coherent without doing anything special
  - your work email is generated automatically using if already in the template - previously generated name, surname and company

```
jr template run --template '{{name}} {{email}}'
jr template run --template '{{name}} {{surname}} {{company}} {{email_work}}'
jr template run user | jq
jr --locale IT template run user | jq
jr --locale FR template run user | jq
```

### > echo "hello" 2>&1 >> \$LOG



- You can choose different output for jr:
  - stdout (default)
  - kafka
  - redis
  - mongo
  - elastic
  - o **s3**
- Each output needs a specific configuration
- Output can easily be extended implementing Producer interface

jr template run user -o kafka
jr template run user -o kafka -t topic\_user -a
jr template run user -o mongo

## > select \* from customers where custID='X1001';



- Relational Integrity is where most of similar tools fall. To generate "related" data, they end up having long lists of prebuilt json documents, not at all random. Basically they become equivalent to:
  - kcat -P -b localhost:9092 -t topic -K: -l prebuilt\_json.txt
- jr has two features to help with integrity
  - preload to create a bunch of events at the beginning
  - context functions, especially add\_v\_to\_list, random\_n\_v\_from\_list and random\_v\_from\_list

#### > select \* from customers where custID='X1001';



- With preload and context you can for example:
  - o generate **1000** random products all at once to a topic
  - generate 100 random customers all at once and then add 1 customer every minute
  - stream 5 random orders every 100ms by existing customers with existing products
- To test your streaming apps (KStream, ksqlDB, Flink), you definitely need relations!

```
jr function list -c context
jr template show shoe
jr template show shoe_customer
jr template show shoe_order
jr template show shoe_order
jr template show shoe_clickstream
jr emitter run
jr emitter run
```

#### > more | grep future



- We need your help!
  - Close issues if you can: <a href="https://github.com/ugol/jr/issues">https://github.com/ugol/jr/issues</a>
  - Localizations in different languages
  - Useful new functions for templates
  - Useful pre-configured emitters for complex use cases
  - New output Producers (every k/v store is a candidate)
- Pls star, watch and fork the project on Github!
  - The brew guys told us that we need a minimum of:
  - 30 forks
  - 30 watchers

  - (if you want to brew install jr!)



#### > more | grep links



- Links
  - Issues <a href="https://github.com/ugol/jr/issues">https://github.com/ugol/jr/issues</a>
  - Documentation <a href="https://jrnd.io/">https://jrnd.io/</a>
  - Blog first part:
     <a href="https://dev.to/ugol/jr-quality-random-data-from-the-command-line-part-i-5e90">https://dev.to/ugol/jr-quality-random-data-from-the-command-line-part-i-5e90</a>
  - Blog second part:
     <u>https://dev.to/ugol/jr-quality-random-data-from-the-command</u>

     -line-part-ii-3nb3
    - Blog third part: SOON

# > more | grep questions?



