

Evaluating Prometheus Knowledge in Interviews

Brian Brazil
Founder

Who am I?

- One of the developers of Prometheus
- Author of Prometheus: Up&Running
- Primary author of Reliable Insights blog

You may have heard of me :)

Looking Back..

Two years I gave a lightning talk on "An Exploration of the Formal Properties of PromQL" demonstrating that PromQL was Turing Complete via Conway's Life.

Last year I gave a talk on "Rule 110 for Prometheus", a simpler demonstration of the above.

Filtering

With the growth of the ecosystem you may be looking to hire an employee with existing experience to assist you on your Prometheus journey.

How can you filter the wheat from the chaff?

CV

You could look at someone's experience with Prometheus

- Are they a Prometheus developer?
- Have they written a book?
- Do they have a well known blog?
- Have they demonstrated esoteric PromQL knowledge?

But the technology is fairly new, and this is a high bar. No one has 10+ years of experience with Prometheus after all!

Low Pass Filter

Rather than looking for a unicorn, how about instead throwing away applicants that are pretty obviously faking it until they make it.

From there you can more deeply consider the remaining applicants.

Enter FizzBuzz.

FizzBuzz

Write a program that prints the numbers from 1 to 100. But for multiples of three print “Fizz” instead of the number and for the multiples of five print “Buzz”. For numbers which are multiples of both three and five print “FizzBuzz”.

Source: <http://wiki.c2.com/?FizzBuzzTest>

FizzBuzz for Prometheus

FizzBuzz is a trivial programming task, that an experienced programmer should have no problem coding up.

We could use this to evaluate PromQL knowledge, and thus get an idea if someone has at least a very basic understanding of Prometheus.

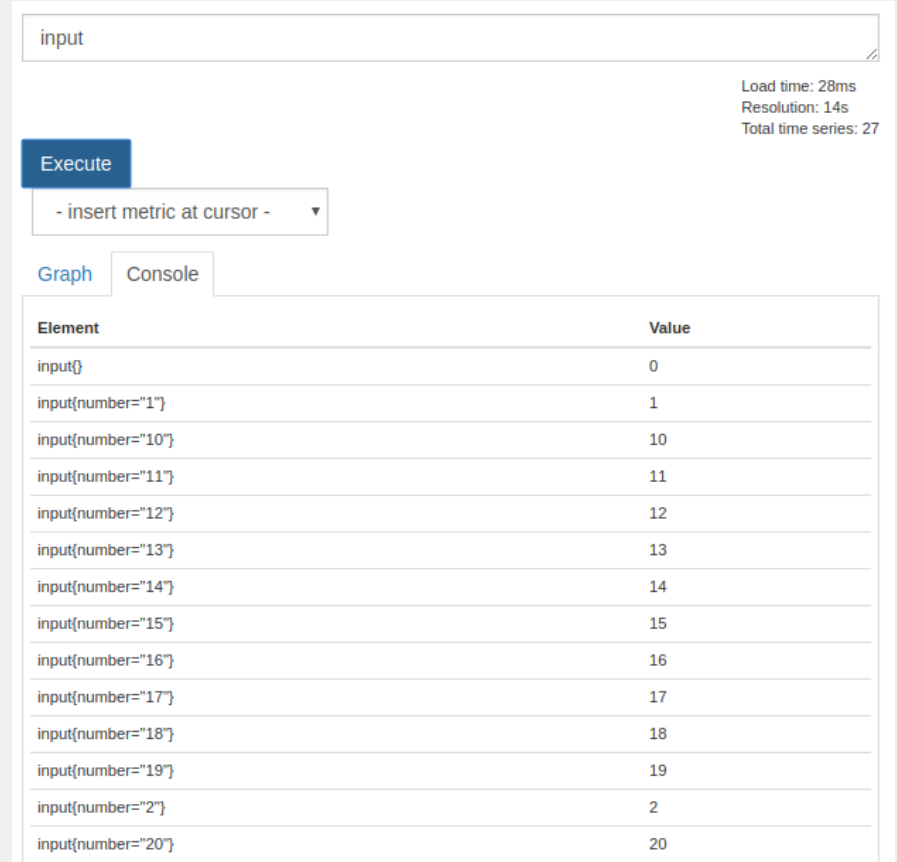
Let's do it then!

FizzBuzz is a trivial problem, but let me share my solution.

Input Data

We'll need some input data for the for loop.

We can build it up with a recording rule.



The screenshot shows a data recording interface. At the top, there is an input field containing the text "input". To the right of the input field, the following statistics are displayed: "Load time: 28ms", "Resolution: 14s", and "Total time series: 27". Below the input field is a blue "Execute" button. Underneath the button is a dropdown menu with the text "- insert metric at cursor -". Below the dropdown are two tabs: "Graph" and "Console". The "Console" tab is active, displaying a table of recorded data. The table has two columns: "Element" and "Value". The data points are as follows:

Element	Value
input{}	0
input{number="1"}	1
input{number="10"}	10
input{number="11"}	11
input{number="12"}	12
input{number="13"}	13
input{number="14"}	14
input{number="15"}	15
input{number="16"}	16
input{number="17"}	17
input{number="18"}	18
input{number="19"}	19
input{number="2"}	2
input{number="20"}	20

Input Recording Rule

```
record: input
expr: >
  (
    input
  or
    count_values("number", (input + 1) <= 100)
    * on() group_left
    max(input) + 1
  or
    vector(0)
  )
```

Output

From there we need to filter based on the values.

PromQL has a modulus operator, and `label_replace` can do strings, so this isn't hard.

Output Recording Rule

```
record: output
expr: >
  (
    label_replace(input % 15 == 0, "output", "FizzBuzz", "", "")
  or on (number)
    label_replace(input % 5 == 0, "output", "Buzz", "", "")
  or on (number)
    label_replace(input % 3 == 0, "output", "Fizz", "", "")
  or on (number)
    label_replace(input, "output", "$1", "number", "(.*)")
  )
```

Output Result

The answer is right, but out of order.

That can be fixed.

Element	Value
{output="FizzBuzz"}	0
{number="15",output="FizzBuzz"}	0
{number="30",output="FizzBuzz"}	0
{number="45",output="FizzBuzz"}	0
{number="60",output="FizzBuzz"}	0
{number="75",output="FizzBuzz"}	0
{number="90",output="FizzBuzz"}	0
{number="10",output="Buzz"}	0
{number="100",output="Buzz"}	0
{number="20",output="Buzz"}	0
{number="25",output="Buzz"}	0
{number="35",output="Buzz"}	0
{number="40",output="Buzz"}	0
{number="5",output="Buzz"}	0
{number="50",output="Buzz"}	0
{number="55",output="Buzz"}	0
{number="65",output="Buzz"}	0
{number="70",output="Buzz"}	0

Output Recording Rule - v2

```
record: output
expr: >
  (
    label_replace(input % 15 == 0, "output", "FizzBuzz", "", "")
  or on (number)
    label_replace(input % 5 == 0, "output", "Buzz", "", "")
  or on (number)
    label_replace(input % 3 == 0, "output", "Fizz", "", "")
  or on (number)
    label_replace(input, "output", "$1", "number", "(.*)")
  ) * 0 + on(number) group_left input > 0
```

Output Result v2 and Sorted

The values now match in input number, so we can do `sort(output)`!

Element	Value
<code>output(number="1",output="1")</code>	1
<code>output(number="2",output="2")</code>	2
<code>output(number="3",output="Fizz")</code>	3
<code>output(number="4",output="4")</code>	4
<code>output(number="5",output="Buzz")</code>	5
<code>output(number="6",output="Fizz")</code>	6
<code>output(number="7",output="7")</code>	7
<code>output(number="8",output="8")</code>	8
<code>output(number="9",output="Fizz")</code>	9
<code>output(number="10",output="Buzz")</code>	10
<code>output(number="11",output="11")</code>	11
<code>output(number="12",output="Fizz")</code>	12
<code>output(number="13",output="13")</code>	13
<code>output(number="14",output="14")</code>	14
<code>output(number="15",output="FizzBuzz")</code>	15
<code>output(number="16",output="16")</code>	16

Resources

Book: <http://shop.oreilly.com/product/0636920147343.do>

Robust Perception Blog: www.robustperception.io/blog

Queries: prometheus@robustperception.io