StreamSets Microservice Pipeline Example

Table of Contents

Introduction

StreamSets Data Collector allows pipelines to be deployed as microservices. Not surprisingly, these are referred to as Microservice Pipelines. This document describes an example microservice pipeline that returns `true` or `false` if a number posted to it is a prime number or not. We'll deploy the `is_prime` microservice as a Data Collector Job, and then call the microservice from `curl` as well as from a traditional Data Collector pipeline, where the microservice response will be used to enrich records flowing through the pipeline. And then finally, we'll demonstrate how a pipeline can call the microservice in a loop until a prime number is found and then exit.

Prerequisites

- The microservice pipeline deployed in this example is configured to use Data Collector as an API Gateway with Gateway Authentication so the caller must have StreamSets Platform API Credentials.

- The microservice in this example will be exposed on Data Collector's default port, typically port 18630. Firewall rules must allow clients of the microservice access to the Data Collector host on this port. Production microservices are typically fronted by a load balancer with multiple back-end Data Collector instances providing HA and failover for the microservice.

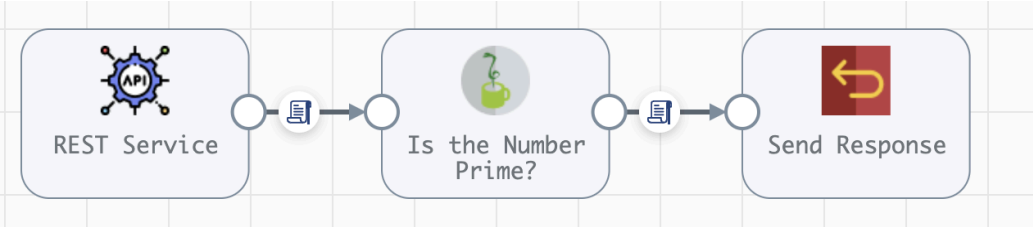
Import the Pipelines

- Download the pipeline archive as a zip file from here (click the page's download link)
- Import the pipeline as a pipeline archive (docs on how to do that are here).

You should see the following three pipelines have been added to your pipeline list.

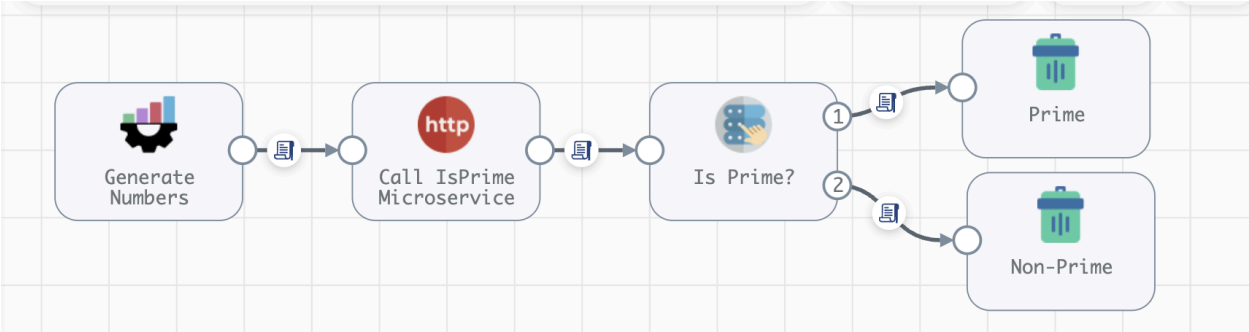
IsPrimeMicroservice

Here is the `IsPrimeMicroservice` Data Collector pipeline:



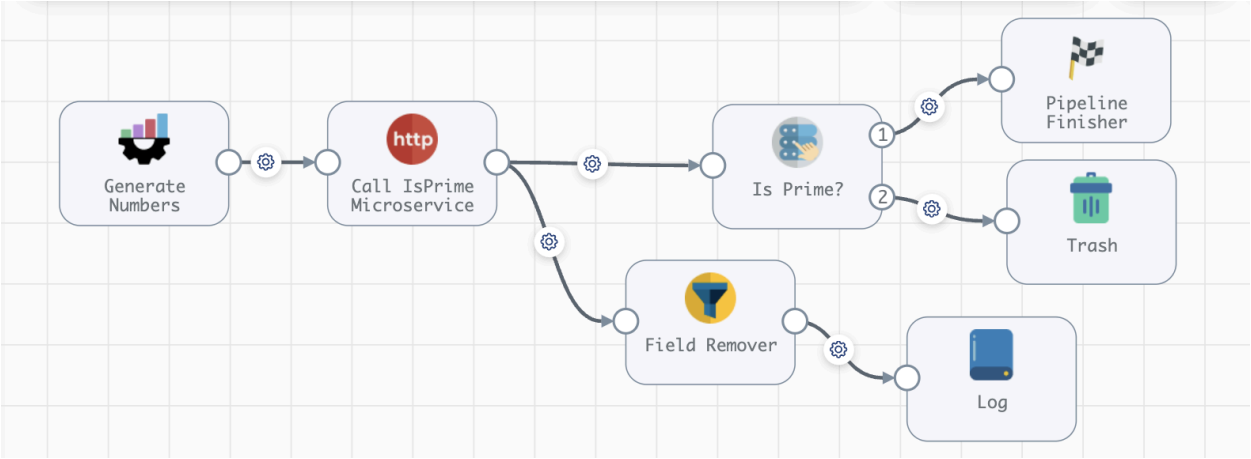
IsPrimeMicroserviceConsumer

Here is the `IsPrimeMicroserviceConsuer` Data Collector pipeline that calls the `IsPrimeMicroservice` to enrich records (before discarding them):



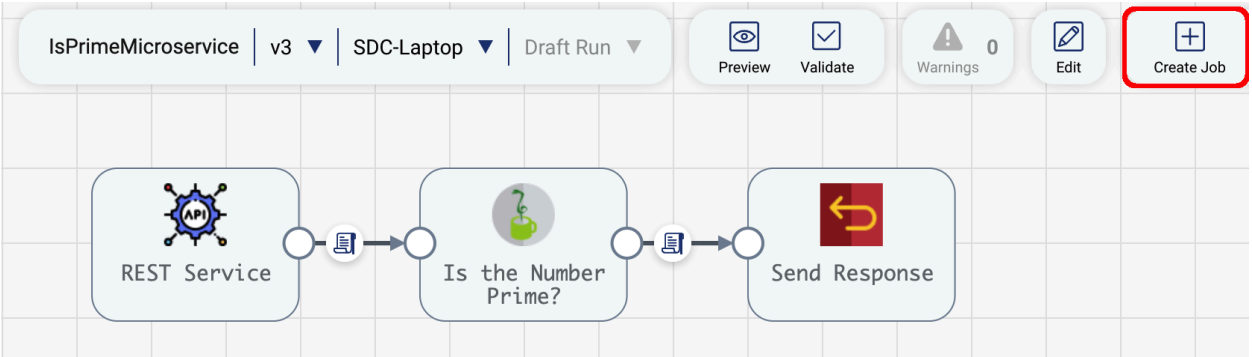
Loop until we get a prime number

Here is the `Loop until we get a prime number` Data Collector pipeline that calls the `IsPrimeMicroservice` with a series of random numbers in a loop until we get a prime number, at which point the pipeline exits:

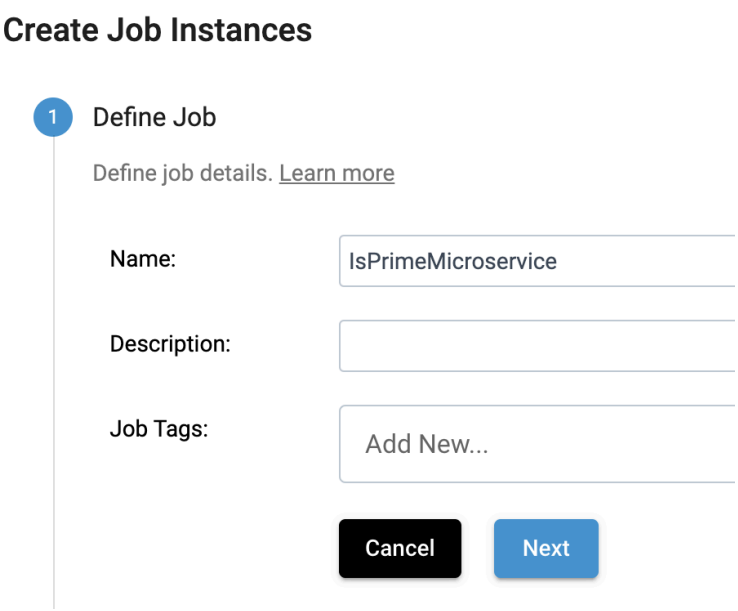


Create a Job for the IsPrimeMicroservice pipeline

Open the `IsPrimeMicroservice` pipeline and click its `Create Job` button:

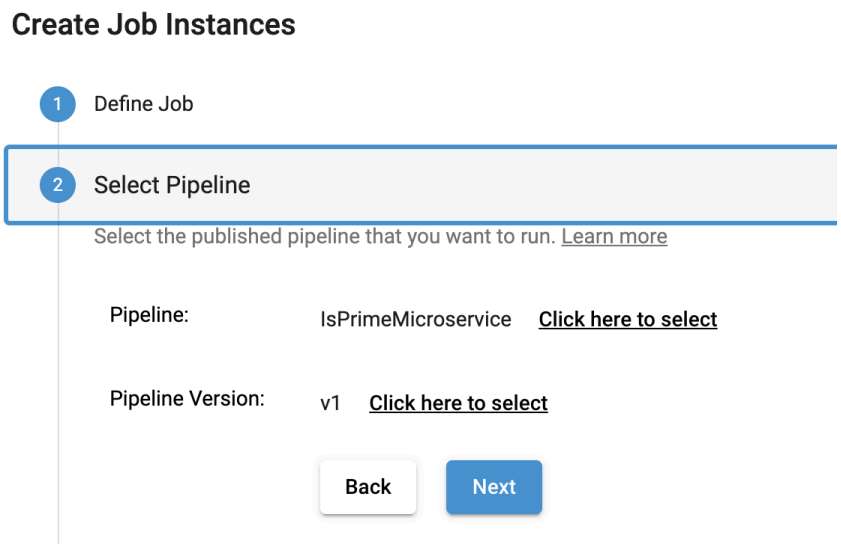


Set the Job's name. I personally like to remove the "Job for " prefix so my Job's name matches my pipeline's name, like this:



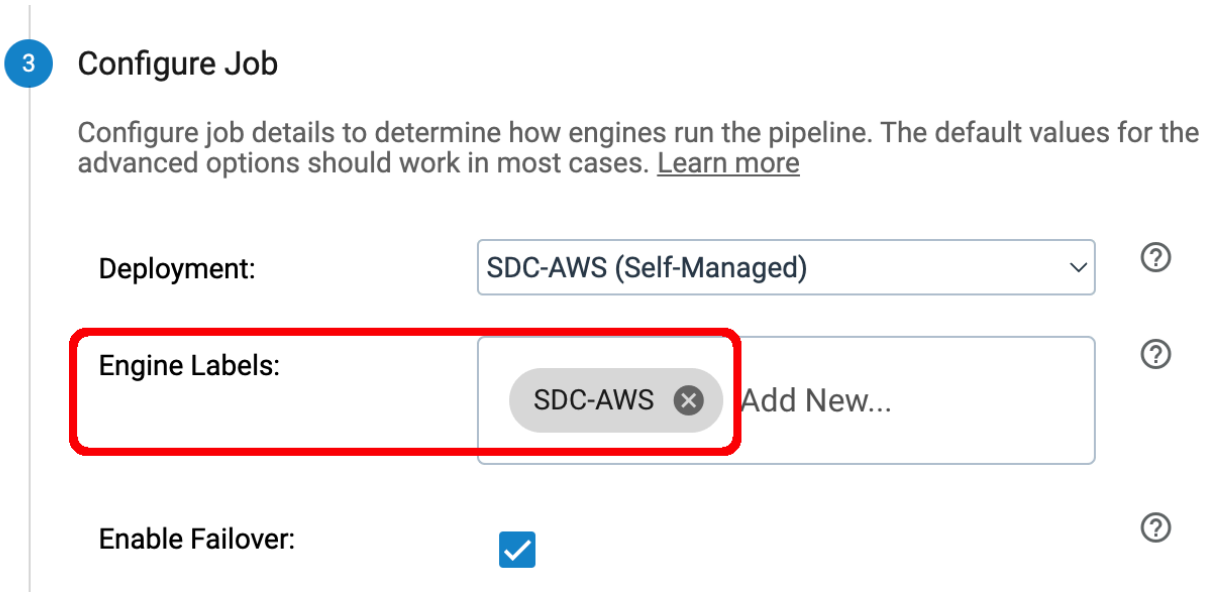
Click `Next`

The current version of the pipeline will be selected for the Job:



Click `Next`

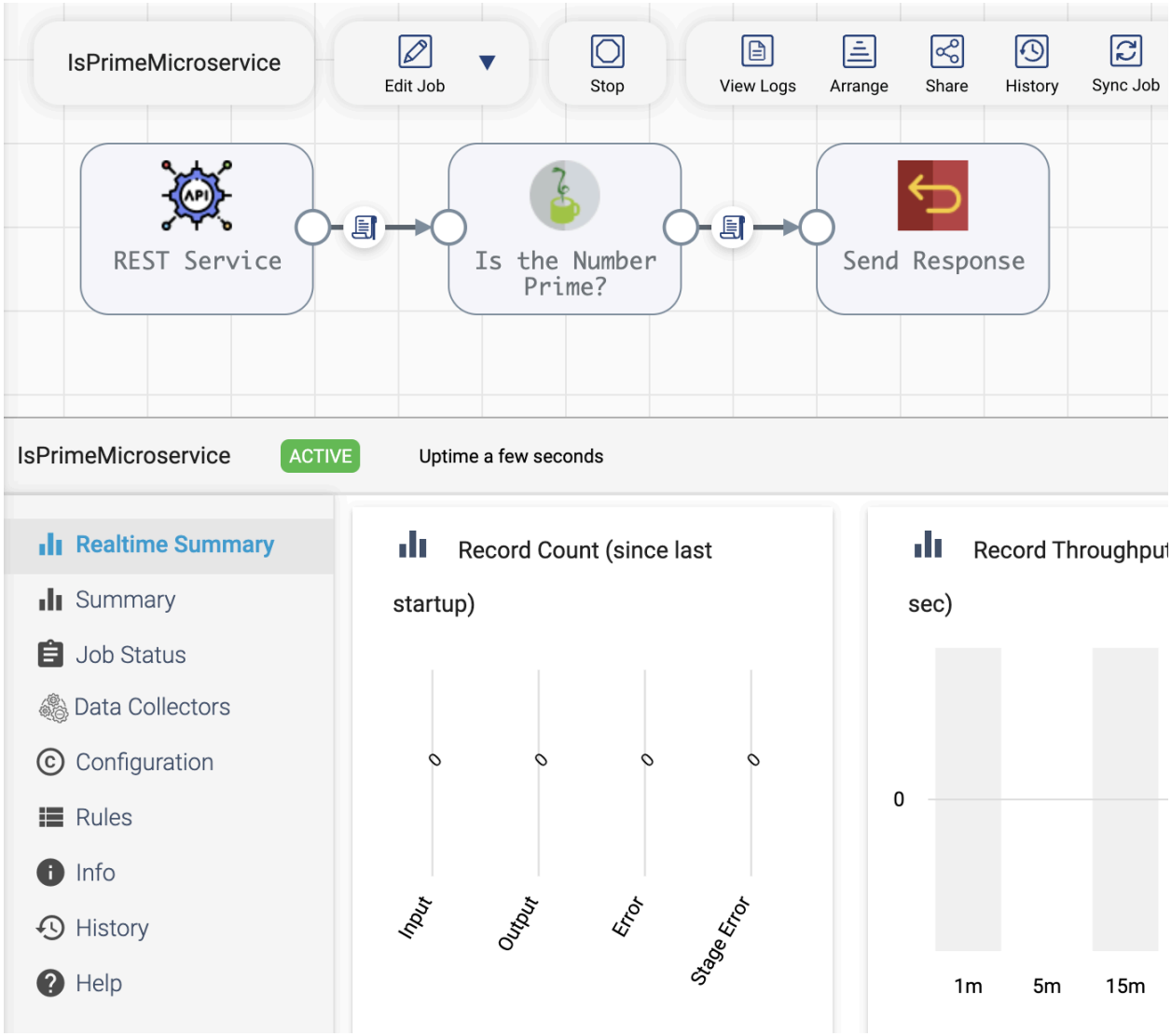
Set your Data Collector Engine Label:



Click `Save & Next`

Then click `Start & Monitor Job`

You should see the monitoring view for the `IsPrimeMicroservice` that has not yet received any requests:



Call the microservice using curl

The URL for the microservice is:

```
http://<Data Collector Host>:18630/rest/v1/gateway/is_prime
```

In a terminal session, set and export your CRED_ID and CRED_TOKEN from your API Credentials:

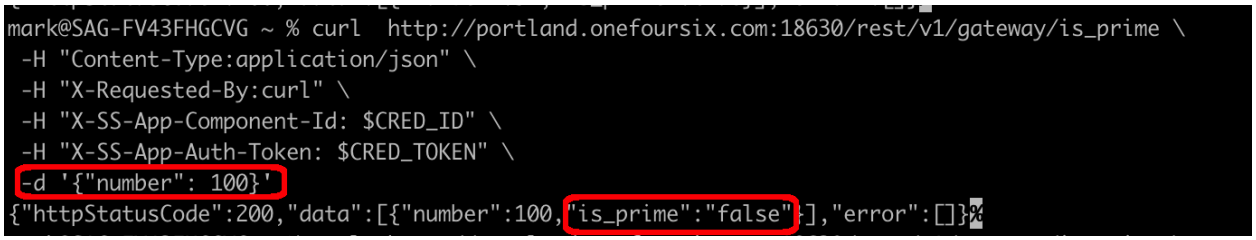
```
$ export CRED_ID=50f6eca6-eaca…

$ export CRED_TOKEN=eyJ0eXAiOi…
```

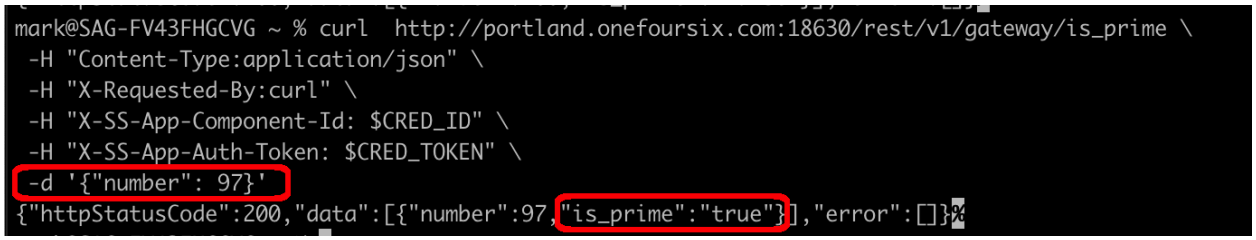
Then call the microservice using `curl` like this, passing in a composite number like 100. Use the Data Collector's hostname in the URL:

```
$ curl  http://<Data Collector Host>:18630/rest/v1/gateway/is_prime \
 -H "Content-Type:application/json" \
 -H "X-Requested-By:curl" \
 -H "X-SS-App-Component-Id: $CRED_ID" \
 -H "X-SS-App-Auth-Token: $CRED_TOKEN" \
 -d '{"number": 100}'
```

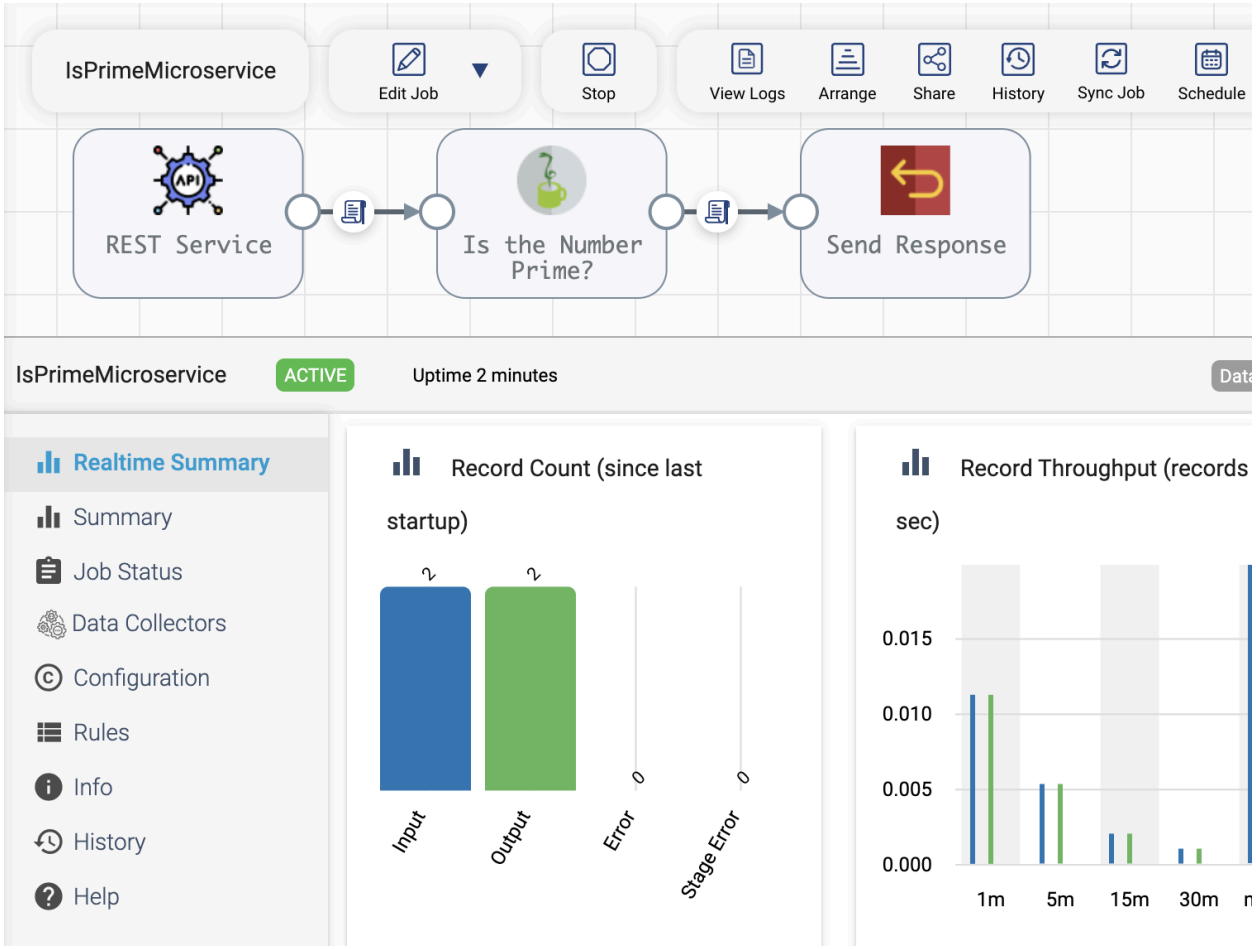
You should receive a response that 100 is not prime:



Repeat the command with a prime number:

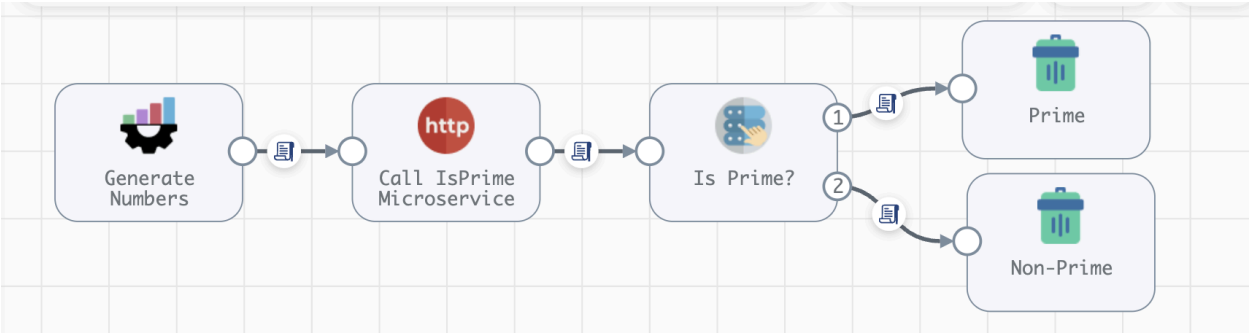


You should also see the microservice has handled two requests:

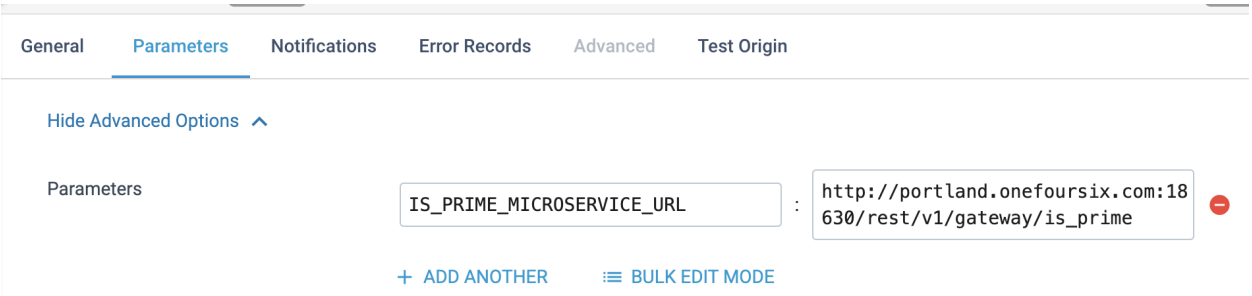


Call the microservice from a Data Collector Pipeline

Open the `IsPrimeMicroserviceConsumer` pipeline in the pipeline editor.



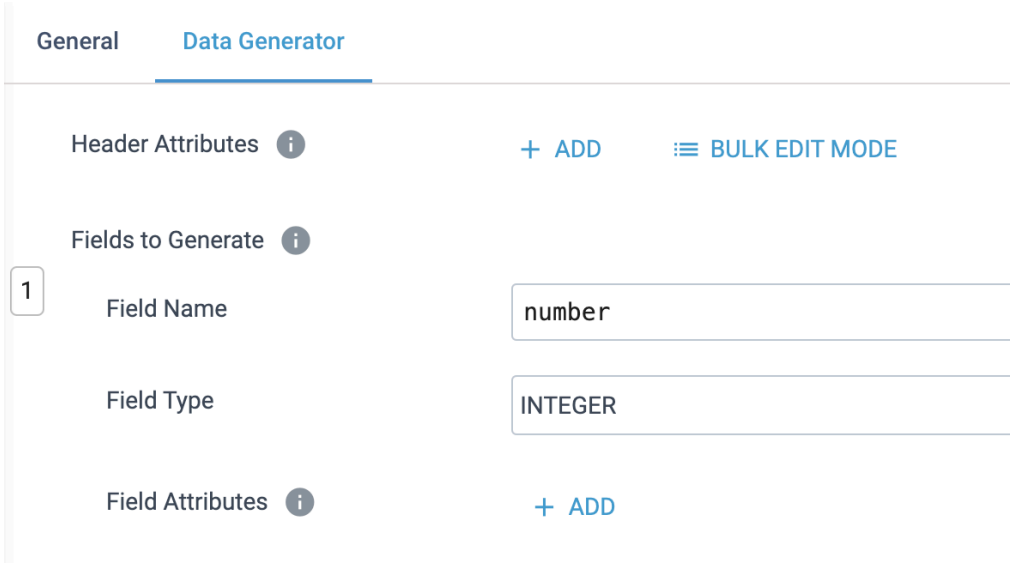
Edit the pipeline's parameters and set the parameter IS_PRIME_MICROSERVICE_URL to the URL of the the microservice:



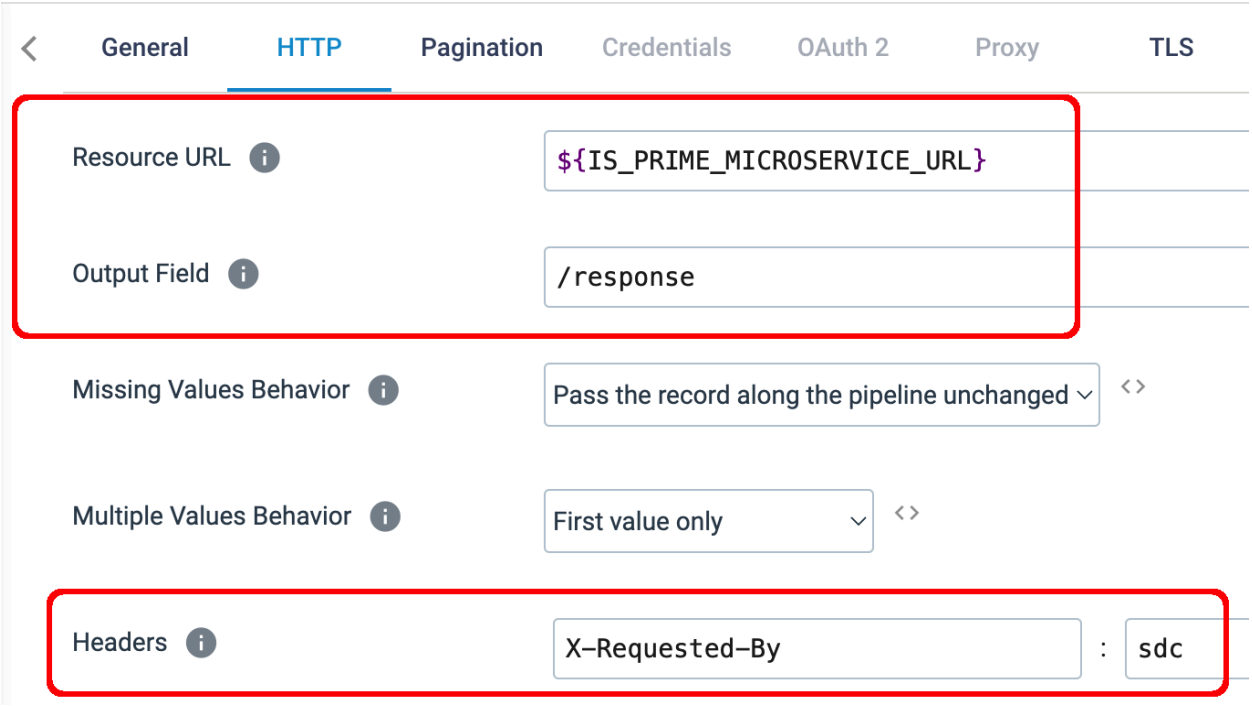
In my environment the value is:

```
http://portland.onefoursix.com:18630/rest/v1/gateway/is_prime
```

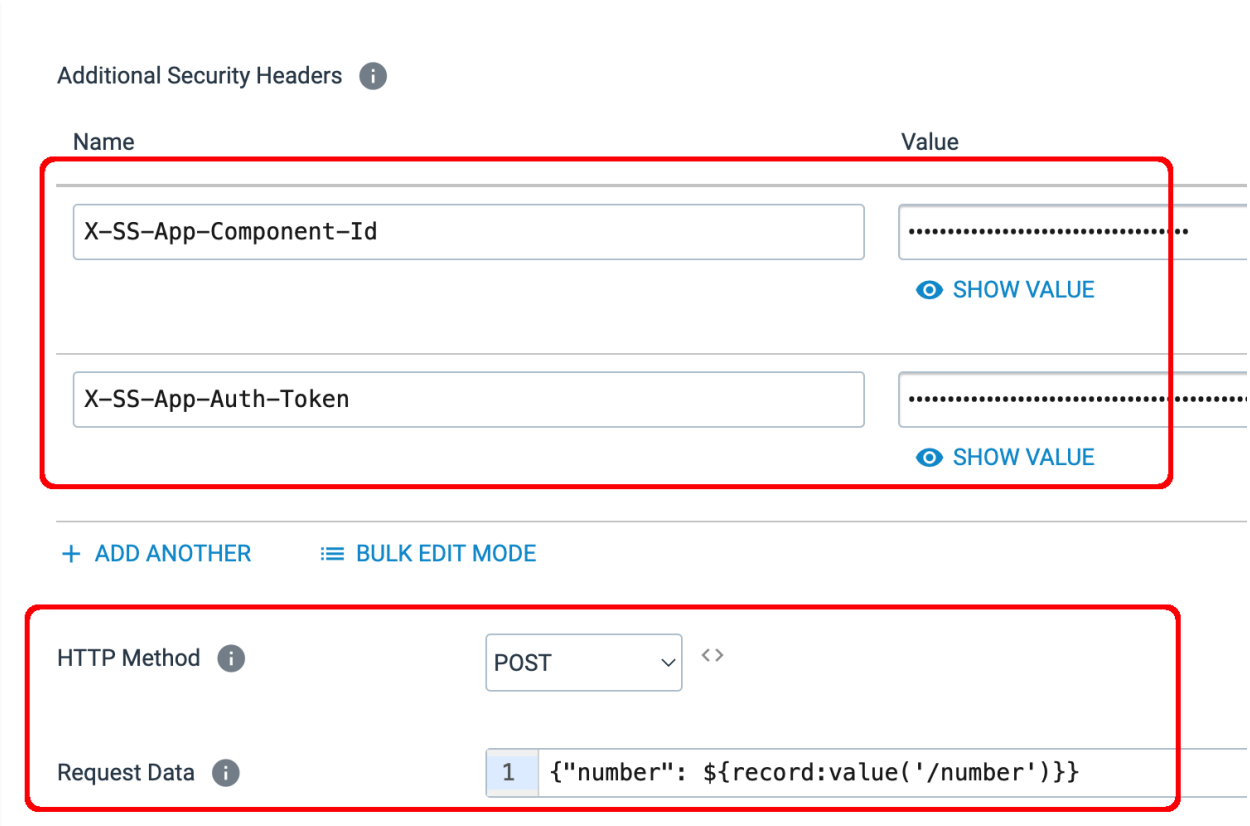
This pipeline uses a Dev Data Generator to generate records with an integer field named "number":



An HTTP Client Processor is used to call the microservice. Here is the top half of the config pane:



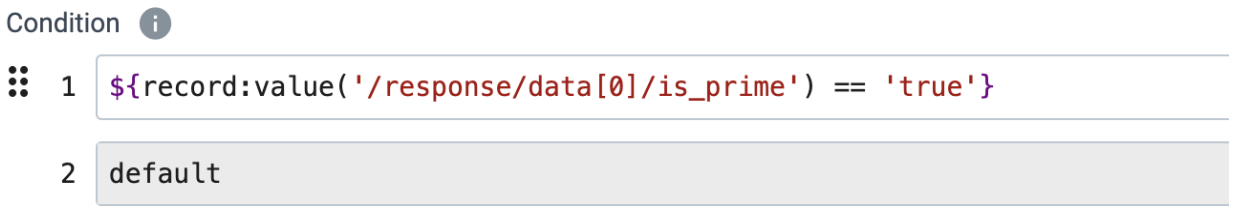
And here is the lower half of the config pane:



Note that the JSON payload posted to the microservice is constructed like this:

```
{"number": ${record:value('/number')}}
```

A Stream Selector is used to route records that are prime to the top path and non-prime records to the bottom path. Here is the routing condition in the Stream Selector:



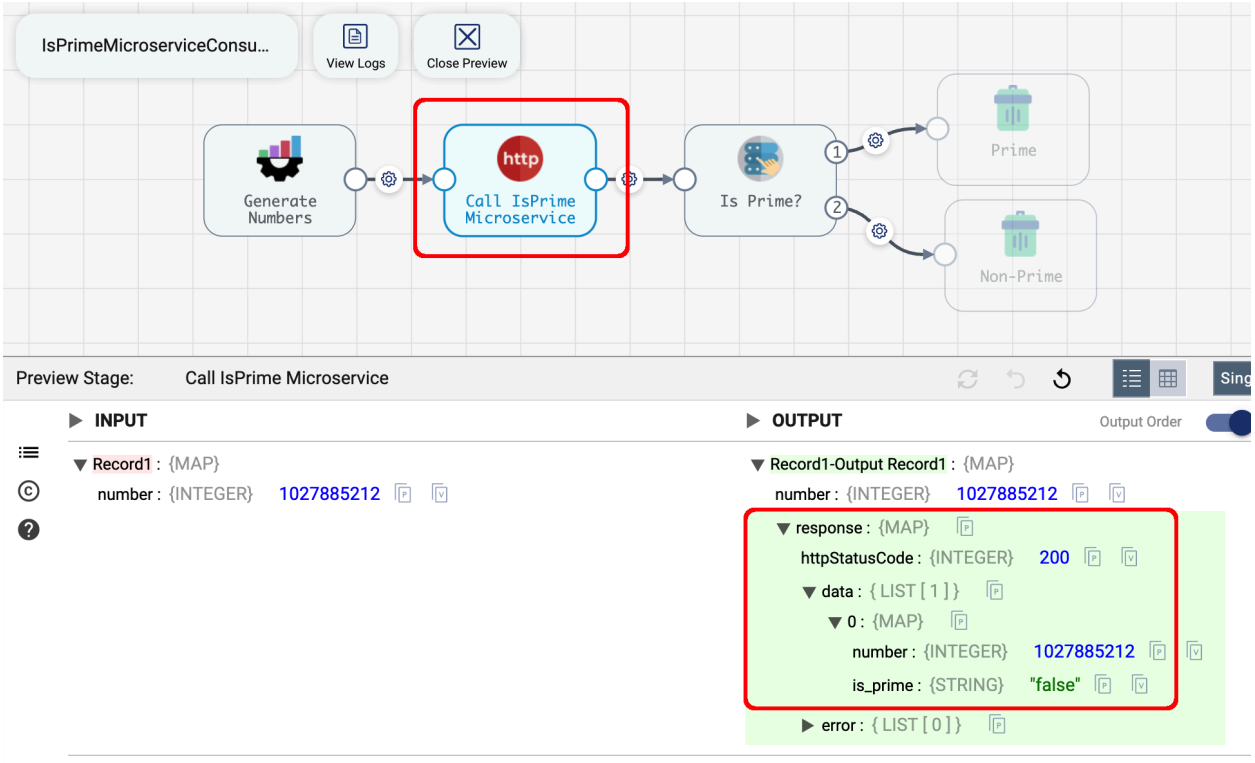
The expression is:

```
${record:value('/response/data[0]/is_prime') == 'true'}
```

A good way to see where and how to unpack the response is to use Preview mode as shown in the next section.

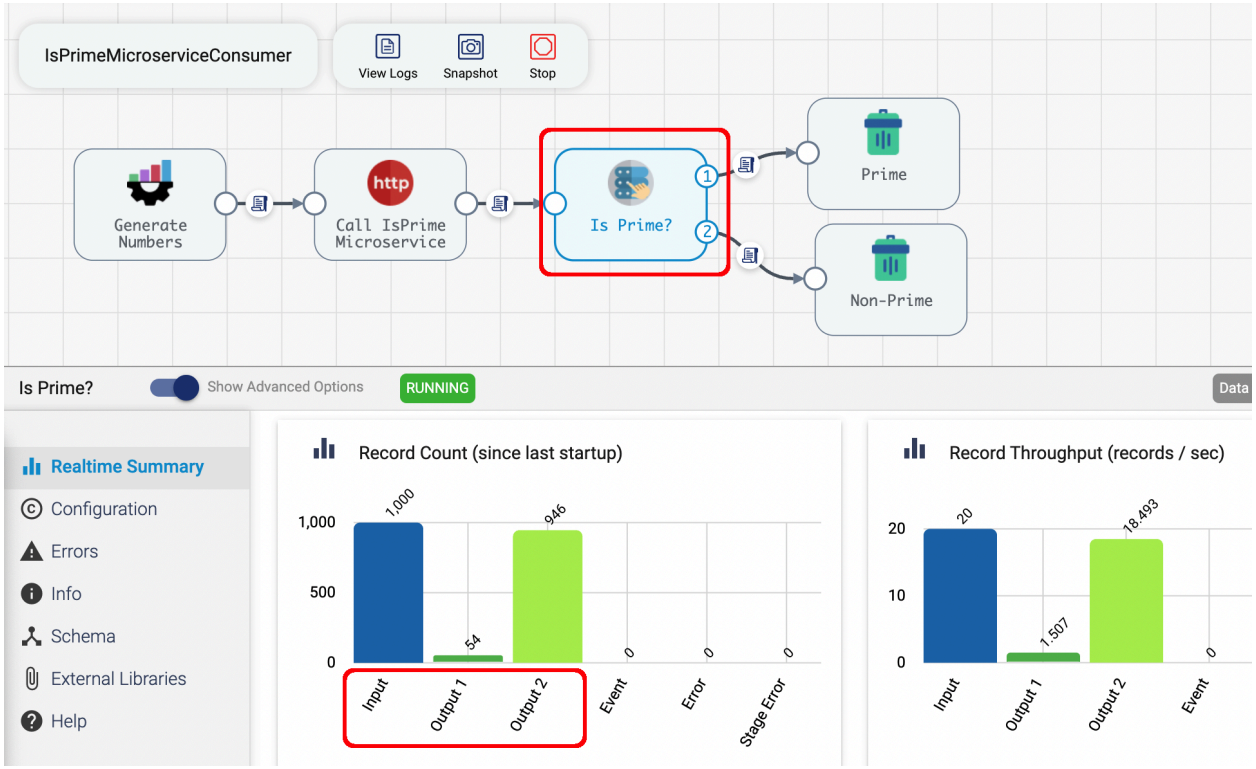
Preview the IsPrimeMicroserviceConsumer pipeline

Preview the `IsPrimeMicroserviceConsumer` pipeline to see the numbers generated and the response from the microservice:



Run the IsPrimeMicroserviceConsumer pipeline

Run the `IsPrimeMicroserviceConsumer` pipeline to see the proportion of prime vs non-prime numbers. Click on the Stream Selector to see the distribution. In my tests, after 1000 numbers were generated, I saw 54 were primes and 946 were not:



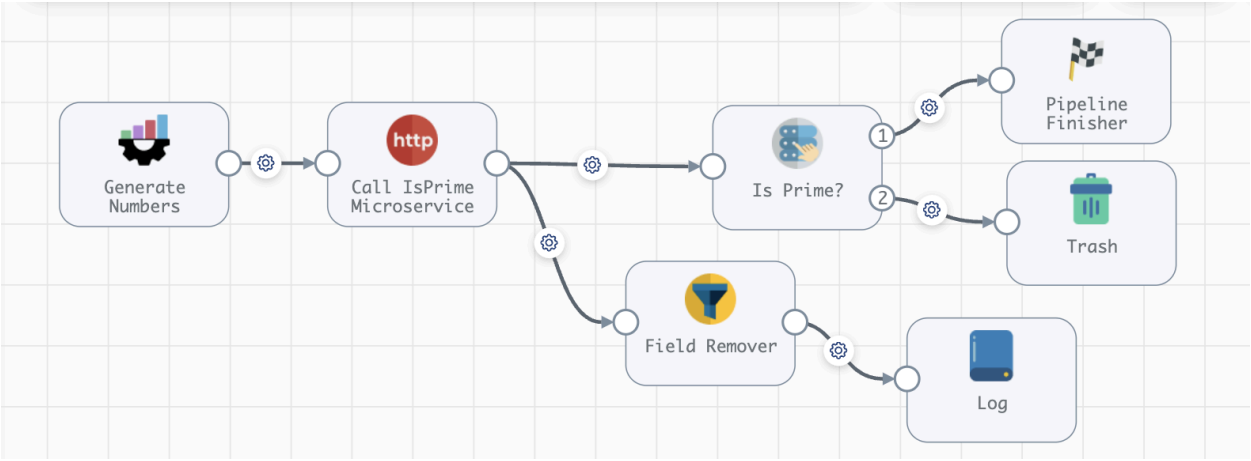
Calling a Microservice in a loop until a desired response is returned

We can use the same `IsPrimeMicroservice` to demonstrate how to call a service in a loop until a desired state is reached.

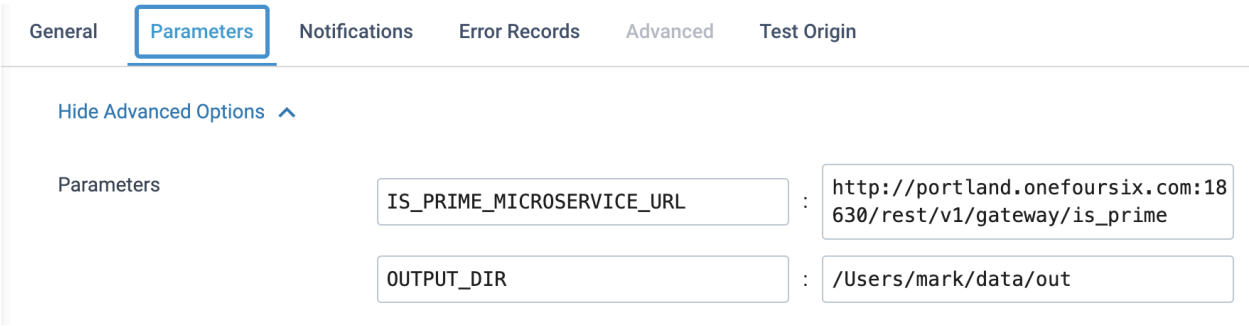
In this case, we'll generate a random integer and test if it is a prime number. If not, we'll ignore that value and generate a new random integer. We'll stay in this loop until we get a prime number, in which case we will exit the loop.

We'll write all the numbers to a file, and expect the last number written to be prime.

Here is the `Loop until we get a prime number` pipeline:



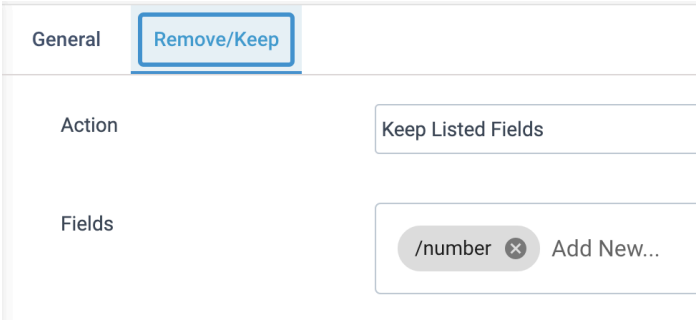
Set these parameters for the microservice URL and the output dir on the local file system for the numbers log:



A key aspect of the Generate Numbers data generator is it is configured to send only one record per batch, in order to process each record individually (set this on the General tab for the stage):

The call to the microservice is the same as in the previous pipeline.

The Field Remover keep only the number field:



The `Is Prime?` Stream Selector has the same conditions as the previous pipeline, but in this case if the first leg is true (i.e. if the number is prime), it routes the record to a pipeline finisher and the pipeline exits. If the number is non prime the event goes to the trash (i.e. it is ignored) but the pipeline is not stopped, so the origin generates the next number.

A test run shows it took 15 tries before I hit a prime number (see the next page)

Here is the file written by the pipeline:

```
{"number":-596013418}
{"number":981744798}
{"number":774886028}
{"number":1845276890}
{"number":-855017289}
{"number":-271813162}
{"number":-936348597}
{"number":-941727796}
{"number":381160663}
{"number":564360002}
{"number":-1167057438}
{"number":1076428630}
{"number":-1392352967}
{"number":1920748387}
{"number":540607961}
```

I'll curl the microservice to confirm that the last number is indeed prime:

```
mark@SAG-FV43FHGCVG ~ % curl  http://portland.onefoursix.com:18630/rest/v1/gateway/is_prime \
 -H "Content-Type:application/json" \
 -H "X-Requested-By:curl" \
 -H "X-SS-App-Component-Id: $CRED_ID" \
 -H "X-SS-App-Auth-Token: $CRED_TOKEN" \
 -d '{"number": 540607961}'
{"httpStatusCode":200,"data":[{"number":540607961,"is_prime":"true"}],"error":[]}%
```