Setting Up MockMotor Server for Load Testing

Quick Start Guide

Contents

You Have an API Problem, Eh?

Your performance testing team needs to run load tests against your application.

Unfortunately, the application uses backend APIs and any attempt to generate a PROD-like load to the application causes APIs to either throttle the requests, or degrade the response time, or charge your team some wild amounts.

The efficient solution here is to mock the backend APIs.

However, the default manual mode of mocking multiple transactions for many accounts is time-consuming and error prone.

This guide provides steps for **quickly** setting up SOAP, REST, or GraphQL mock services capable of supporting load tests with production-like volumes.

We're going to use MockMotor as the server for these mocks.

To make the process of recording the mocks faster, MockMotor supports a special "forward and cache" mode. Each request that doesn't have a matching mock reaction yet is first forwarded to the live API, and then a mock reaction is created based on the request data and the API response. Eventually, no traffic goes through to the live API.

In many cases, the performance team doesn't have to do any manual work except executing a load run, letting MockMotor record all the transactions.

Even when a human intervention is required, the amount of manual work is reduced significantly.

Sounds Complicated?

Worry not!

If your team doesn't have the competences or time to setup the mock services, I can do that for you on a contracting basis.

Contact me at support@mockmotor.com or text +1-416-878-5693 to discuss.

Who am I?

My name Vlad. I'm a Software Architect for more than 20 years, living in Calgary, Canada.

I worked with various flavours of IPC, starting from RPC to CORBA to EJB to SOA and, finally, to APIs.

I'm helping my clients establish effective ways to test API-based applications in the performance testing phase.

Pre-Requisites

Linux

You will need:

- An x86(*) 64-bit Linux box
- 2GB+ of free disk space and 2GB+ of free memory

> (*) MockMotor is thoroughly tested on x86 machines. However, as a Java application, it is expected to work equally well on non-x86 boxes.

Windows? Sure, but ...

You can also install MockMotor as a Windows application for evaluation purposes. MockMotor is expected to work just as well as Linux version, but it was not tested under PROD-like load under Windows.

Installation

Log into the Linux box as the user who will run MockMotor. The username is not important, so let's suppose it is *mm*.

Work Directory

Create a dedicated work directory for MockMotor. It creates a significant number of files and it's better not to mix them with files from other applications.

Then change to that work directory before continuing with the installation.

```
$ mkdir mockmotor

$ cd mockmotor
```

Installing JDK 21

MockMotor requires JDK 21 or higher to run.

Check if the box already has this JDK installed:

```
$ java -version
```

If Java is not found, or its version is below 21, you must install JDK 21 or higher first. You can install it from the package manager (check your distribution's documentation) or download it from the MockMotor site. For the latter, execute:

```
$ curl https://mockmotor.com/jdk21.tgz | tar xfz -
```

This creates a directory named jdk-21/, which contains the latest JDK-21 version tested with MockMotor.

> If curl cannot access the JDK download, check if you need to set the HTTP proxy environment variable before executing curl, e.g.:
>
> export https_proxy=<your organization's proxy>

You can smoke-test the JDK with this command:

```
$ jdk-21/bin/java -version
```

It should print something like

```
openjdk version "21.0.8" 2024-10-15
```

Installing MockMotor Admin

> MockMotor only needs an admin node to work. For horizontal scalability, you can add extra clone nodes later. For information on adding clone nodes, see the documentation.

In the planned MockMotor work directory, execute

```
$ curl https://mockmotor.com/MockMotor-latest.tgz | tar xfz -
```

After this script, the working directory should have MockMotor JAR and shell scripts like below

```
283       Nov 13 06:57 mockmotor.config.xml.template
157127612 Nov 13 06:57 mockmotor.jar
2379      Nov 13 06:57 startMockMotor.sh
476       Nov 13 06:57 stopMockMotor.sh
166       Nov 13 06:57 updateMockMotor.sh
```

Setting Custom Ports if Required

By default, the MockMotor console listens on HTTP port 7080 and HTTPS port 7081. Port 7082 is used for clustering if more than one instance is configured.

> You can skip this section if you're fine with the default ports.

Clone the Configuration Template

First, copy the configuration file template to the configuration file:

```
cp mockmotor.config.xml.template mockmotor.config.xml
```

Update the Ports

Then, using your editor of choice, update the mockmotor.config.xml to have the desired port number. For example, below, the ports are moved to the 17xxx range:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<config xmlns="http://mockmotor.com/config">
    <dataDirectory>.</dataDirectory>
    <clusterPort>17082</clusterPort>
    <httpPort>17080</httpPort>
    <httpsPort>17081</httpsPort>
    <logsDirectory>.</logsDirectory>
</config>
```

Starting the Admin Node

To start the admin node, execute

```
./startMockMotor.sh
```

The script will use the JDK configured in JAVA_HOME, the one in the PATH, or the one under the jdk-21/ directory.

> If no suitable JDK is found, the script asks to set the JAVA_HOME:
>
> ERROR: No java was found; define JAVA_HOME pointing to Java 21 or newer.

If successful, the MockMotor admin prints a few log messages from the log and lists the possible console URLs.

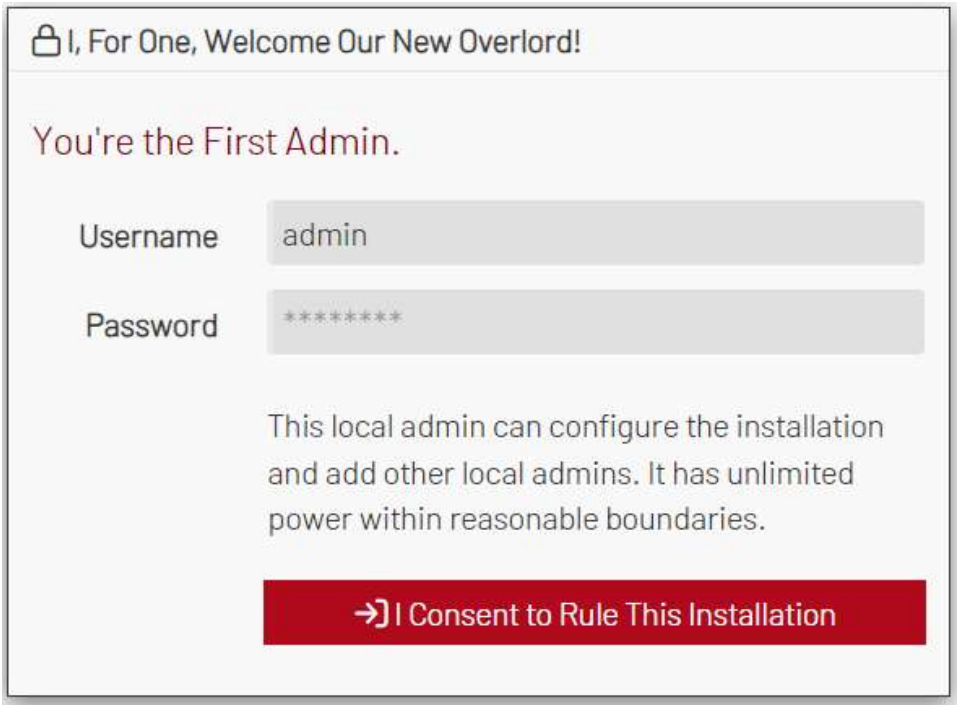
```
2024-12-11 05:59:11.099 INFO Started
ServerConnector@4b7e96a{HTTP/1.1,[http/1.1]}{0.0.0.0:17080}
2024-12-11 05:59:11.589 INFO Started
ServerConnector@5aa6202e{SSL,[ssl, http/1.1]}{0.0.0.0:17081}

To access the MockMotor console, point your browser to one of the
below URLs

  http://192.168.50.167:17080/console
  http://127.0.0.1:17080/console
```

Creating Instance Admin User

Navigate to the console URL most likely accessible from your machine.

> MockMotor presents a self-signed certificate, and your browser will not be happy about it, warning that the connection is not secure. Politely ignore the warning and continue.

MockMotor will ask you to create an instance administrator. The default username is admin, but it can be changed to anything.



Set the password and continue.

The instance is up and running now.

Repointing the Application to Mocks

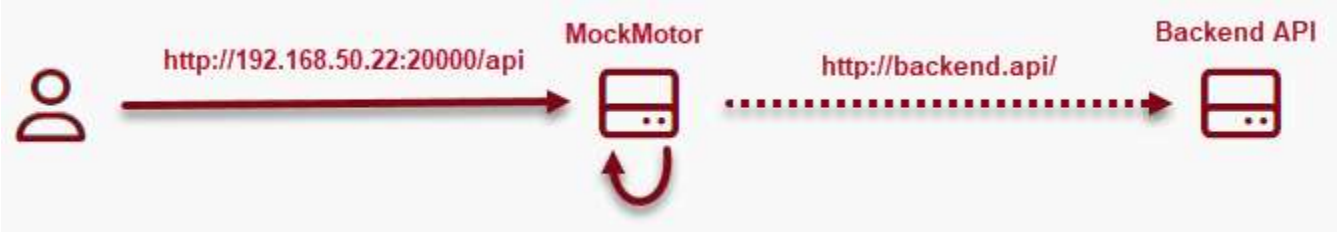
The fastest way to set up a mock is to forward requests to a live service and record the transactions as mock reactions.

Before the repointing, your application points directly to the backend API.

After the repointing, your application points to MockMotor mock service. That mock service has a forwarding reaction that points to the backend API.

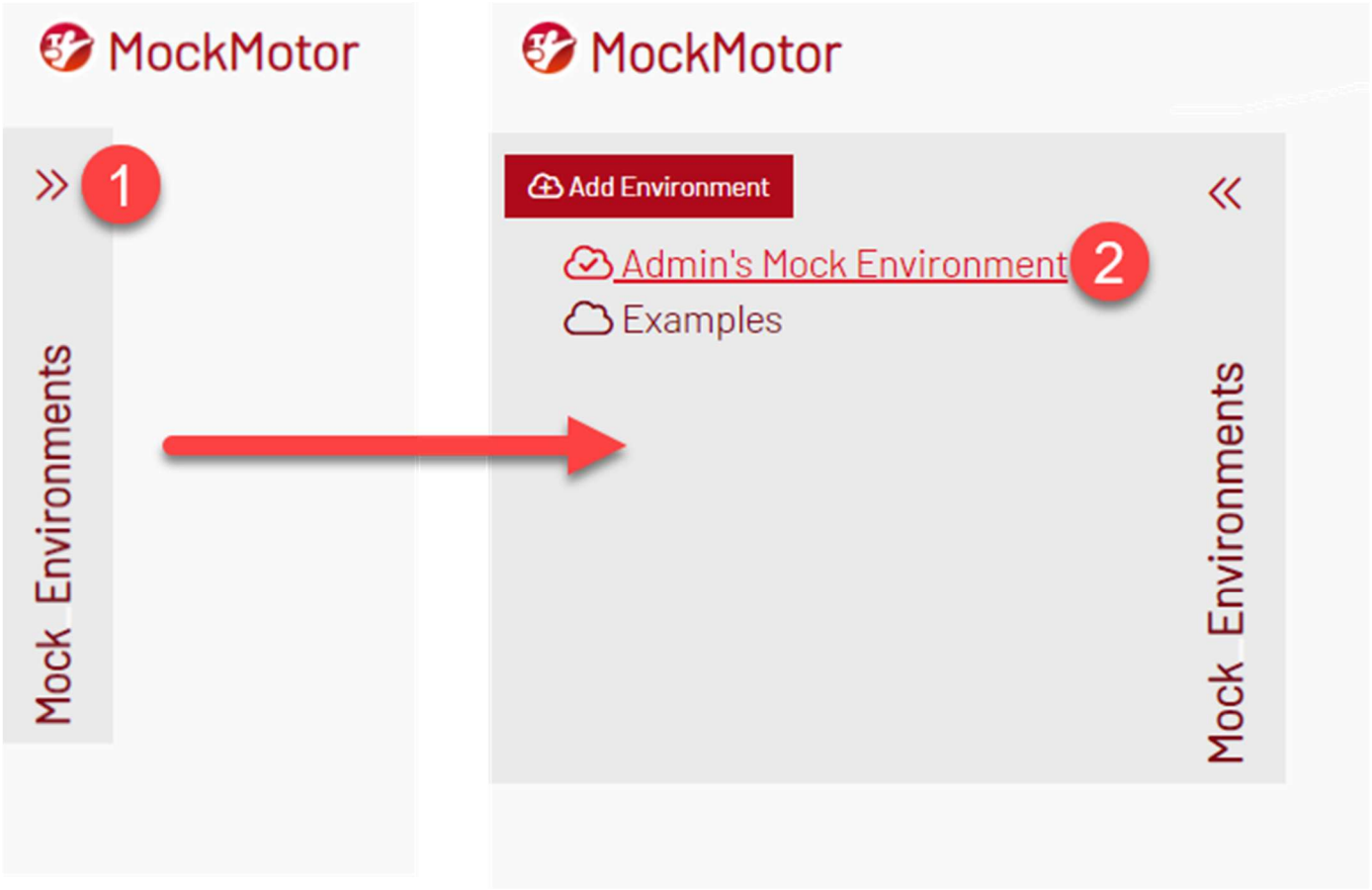
> If your application's configuration permits, use the plain HTTP URL for the mock service. That will save you the hassle of configuring the trust between your application and MockMotor.

MockMotor records the transactions, and gradually the traffic to the backend API disappears. Eventually all requests are served from mocks.



Begin to Mock the Services

Let's navigate to the admin's mock environment. Click on the slider on the left to open the sidebar. Then click on the "Admin's Mock Environment" link.



Let's see how it can be done for common service types.

REST API Forward & Mock

Let's mock a currency conversion REST API as an example.

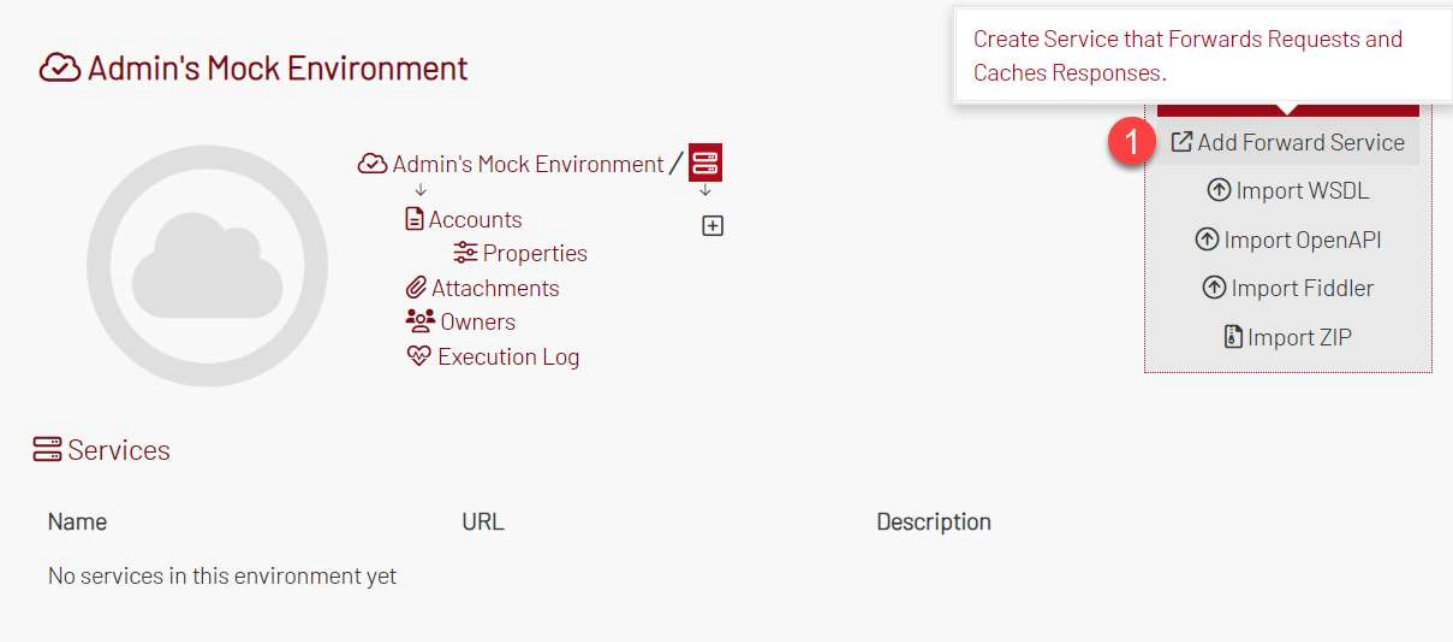
> The service documentation is here: https://currency.getgeoapi.com/documentation/

Like most public APIs, its costs grow rapidly with the request volume, and a typical load test generates high volumes.

Reviewing the documentation reveals that the key parameters for conversion are "format," "from," "to," and "amount."

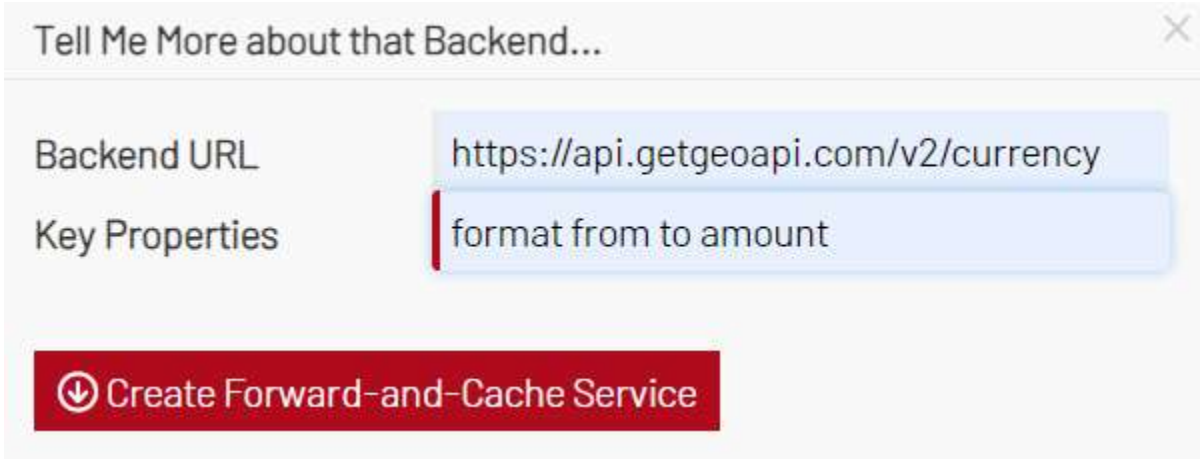
Let's create a mock service that calls the API and caches the responses.

On the environment page, click the Add Forward Service button.



In the form, enter the service URL and list the key parameters.

> URL: https://api.getgeoapi.com/v2/currency
> Key Parameters: format from to amount



Submit the form, and MockMotor shows a new mock service with a forwarding reaction that will cache responses received from the API.

In your application, configure one of the service URLs provided in the mock service navigation pane instead of the real API's URL.

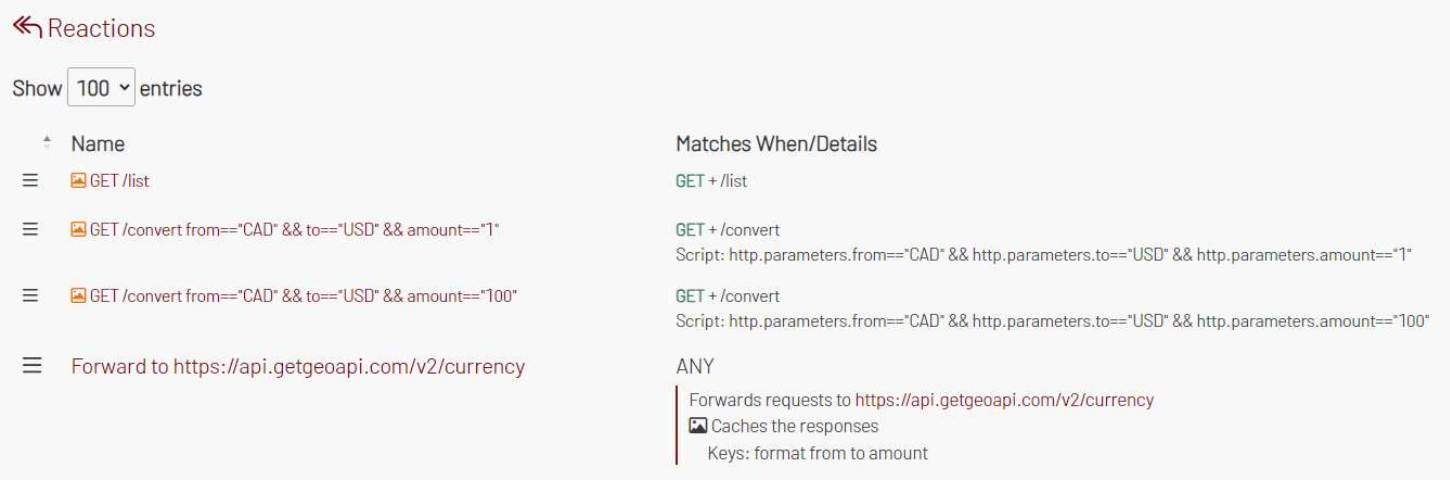
> Use a plain HTTP port to avoid extra work configuring the certificates or trust if possible.

All other properties except the URL – access key, timeouts and others – should remain the same as when the app calls the real API.

Execute a few requests from your application.

You should see new cached reactions in the list when you click the Reactions link. Inspect them. If incorrect, modify or delete them and redo the test.

The client application executed three requests to the API — one to get a list of currencies, another to convert one CAD to USD, and yet another to convert a hundred CAD to USD. All three requests were cached and, if executed again, will be served from the mocks.



The REST API mock is ready and can handle high loads without increasing the API bill.

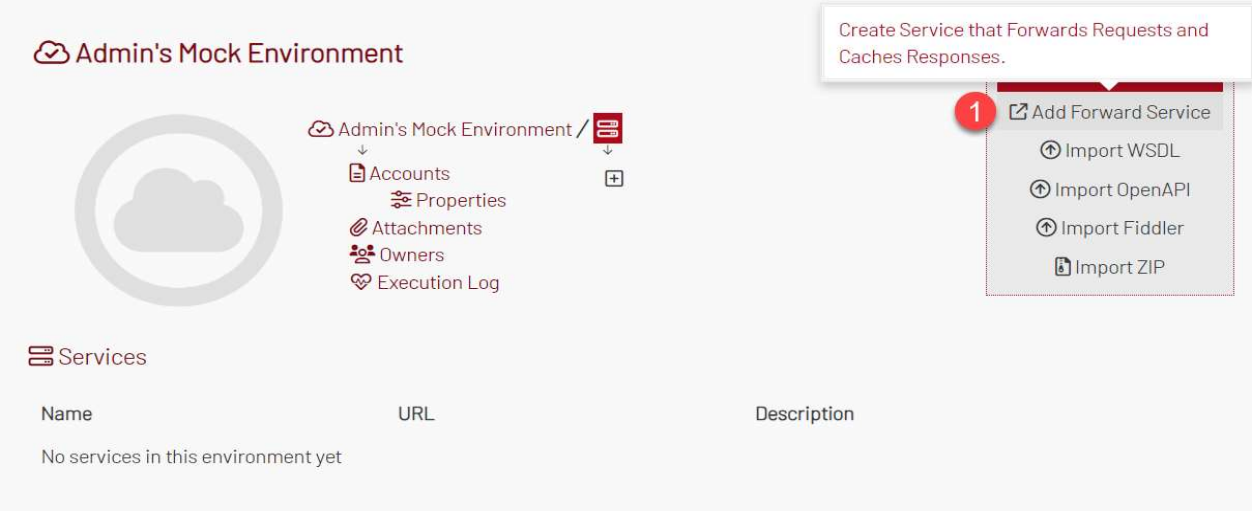
GraphQL API Forward & Mock

MockMotor can cache GraphQL services as well. Let's mock an example GraphQL service named GraphQLZero.

> The service documentation is here: https://graphqlzero.almansi.me/

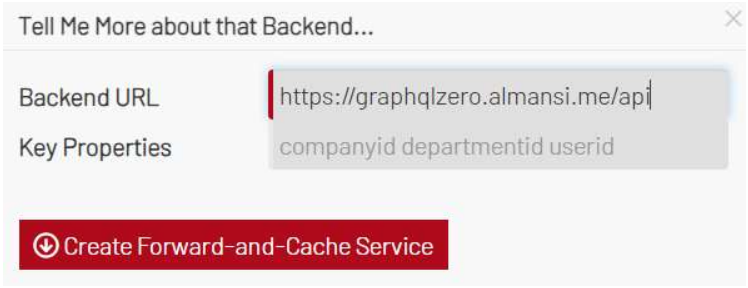
For GraphQL services, unlike REST and SOAP, MockMotor doesn't require the specification of key parameters. Instead, it detects them automatically from the query and argument names.

On the environment page, click the Add Forward Service button.

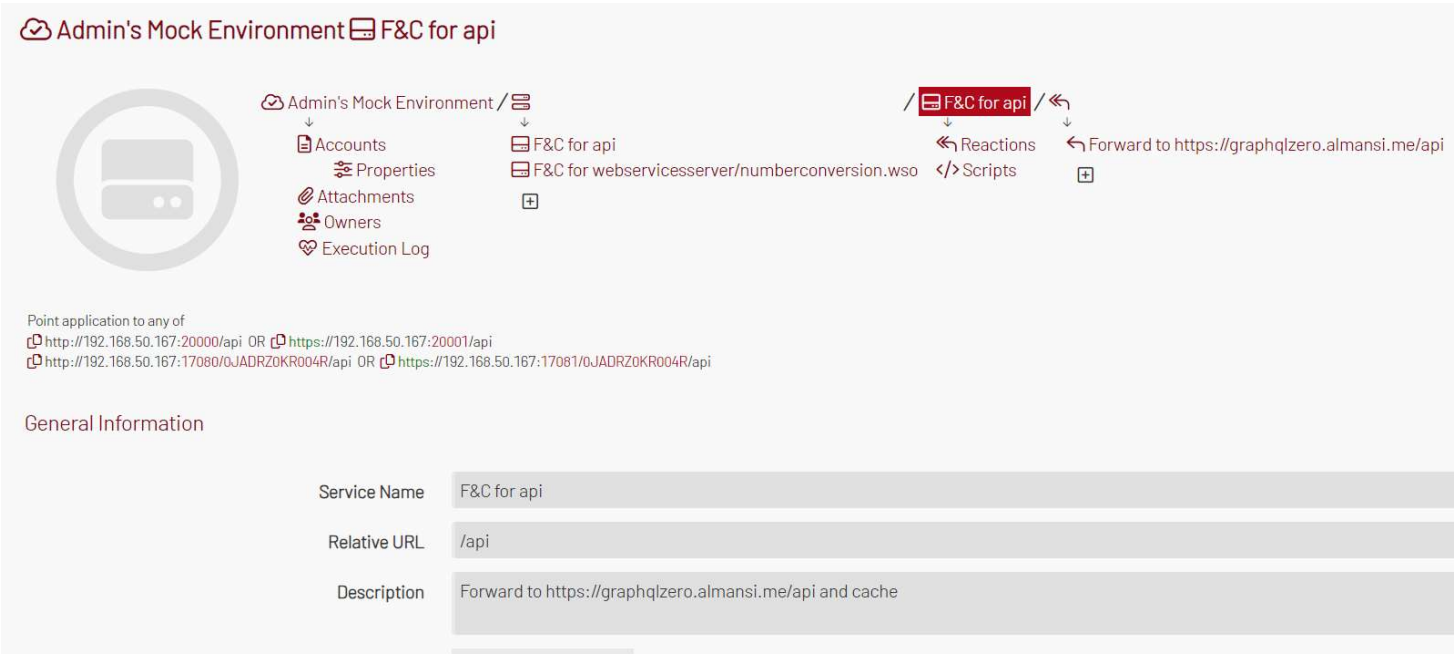


Enter the service URL into the form field. Leave the "Key Properties" field unfilled.

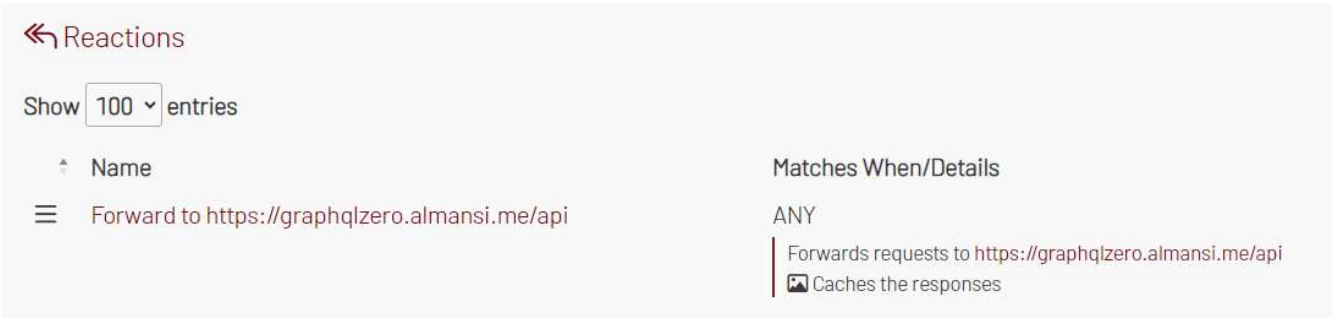
> URL: https://graphqlzero.almansi.me/api
> Key Parameters: <leave empty>



Click on “Create ...” button, and MockMotor creates a forwarding mock service for the provided URL.

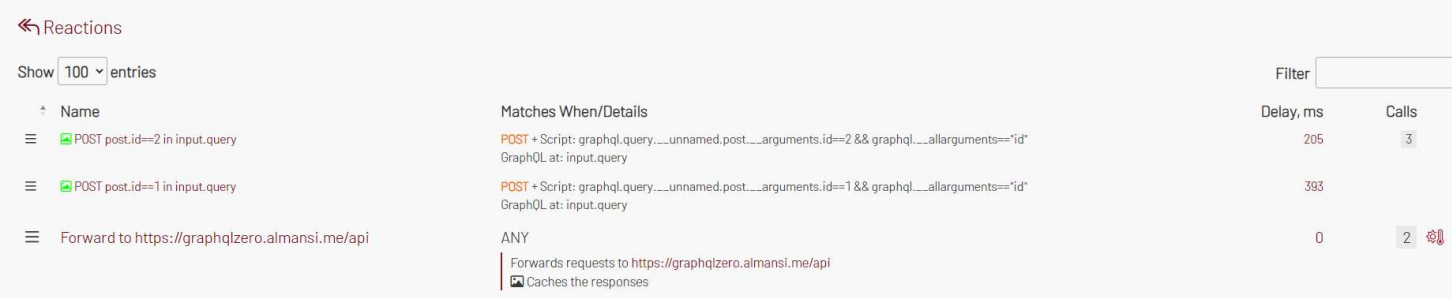


The service has one reaction, which forwards the requests to the GraphQL backend and records the transactions as cached reactions.



In the mock service navigation pane, fine the mock service URLs. They are under “Point application to any of” header. Repoint your application to one of the provided URLs.

Once we point our client application to the mock service URL, we can execute the GraphQL requests. The responses are then cached as the mock reactions.



As you can see from the calls’ count against the first cached reaction, the new requests with the same parameters (the same query or mutation and the same arguments) are served from the mocks and no longer sent to the real service.

SOAP API Forward & Mock

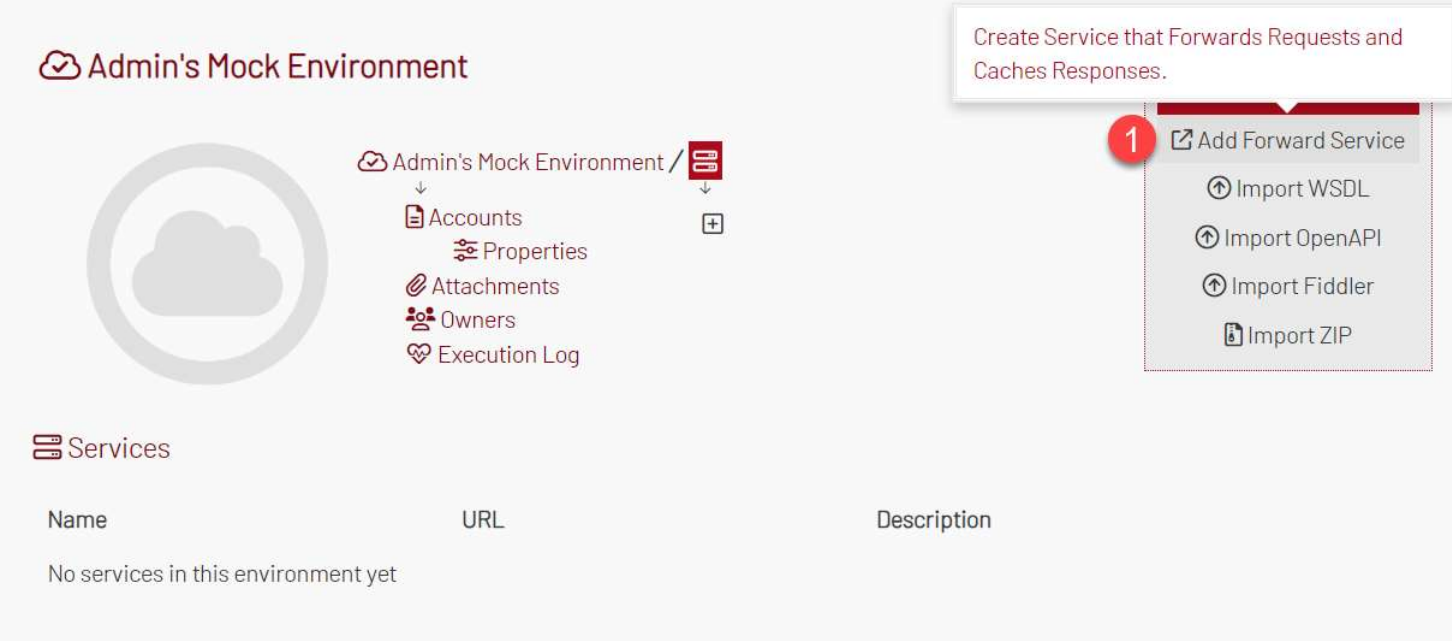
Let's mock a number-to-words conversion SOAP API as an example.

> The service documentation is here:
> https://www.dataaccess.com/webservicesserver/numberconversion.wso

This example service is a free API, so calling it does not cost anything. However, like for many other free services, a throttle control kicks in when the requests-per-second value gets too high, interfering with the load test.

Let's make sure our load tests do not get throttled.

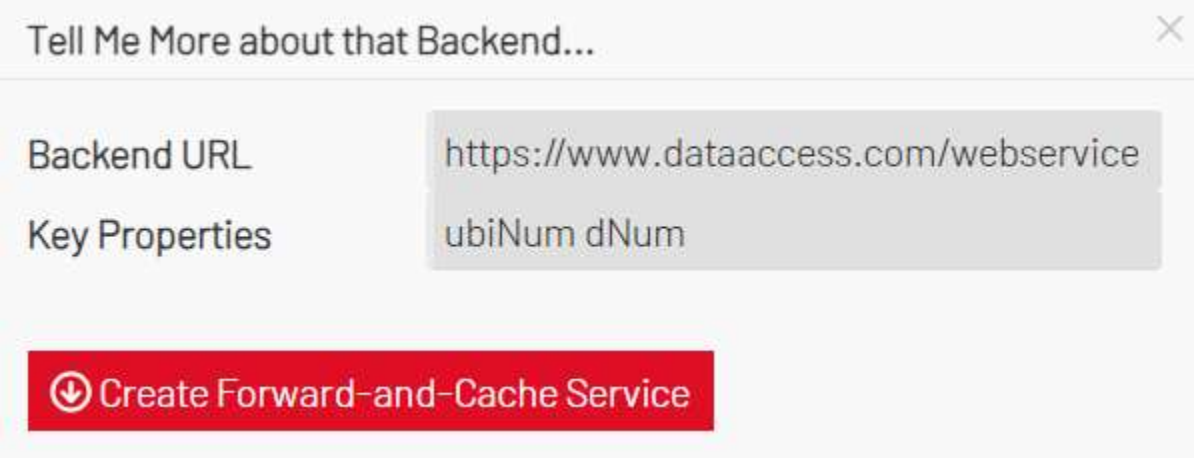
On the environment page, click the Add Forward Service button.



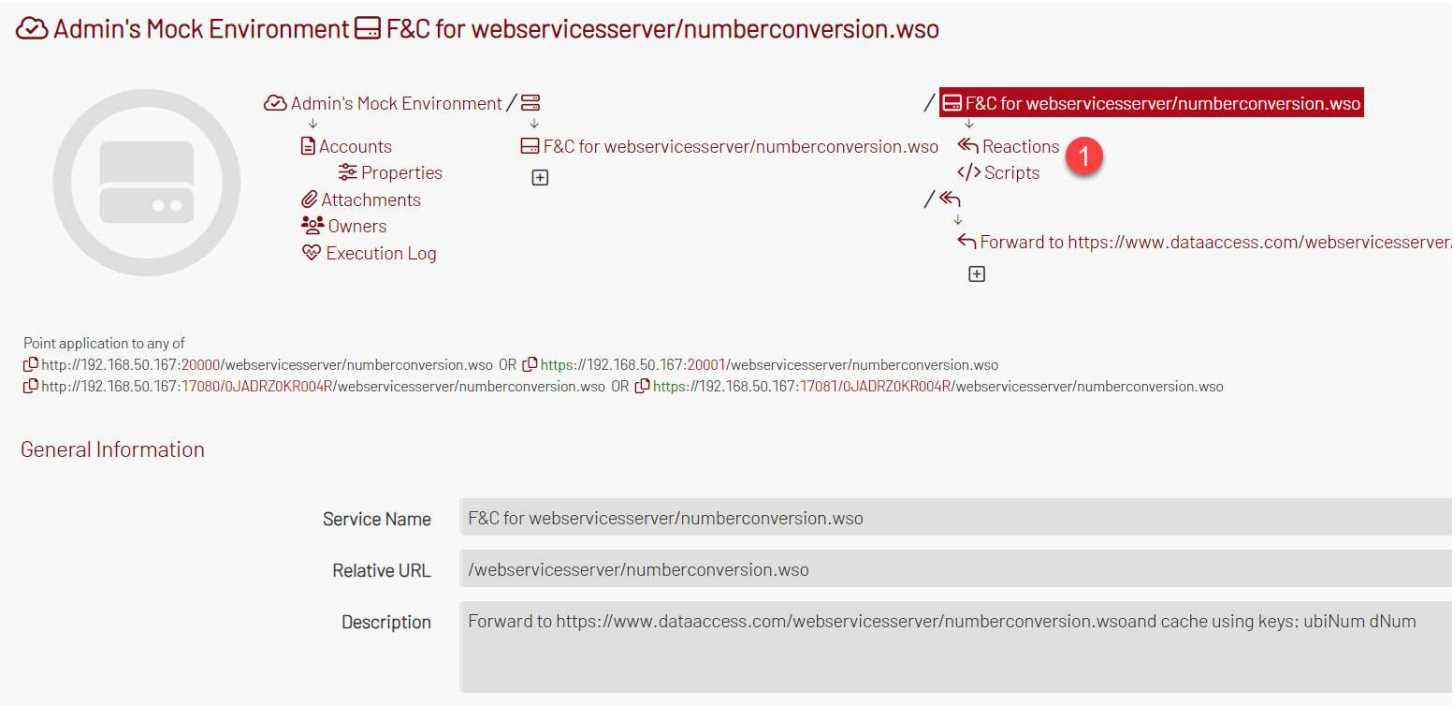
Reviewing the documentation, we can see that the key parameters for this service are "ubiNum" for number-to-text conversion and "dNum" for dollars-to-text conversion.

Fill in the form and provide the service URL and the key request parameters.

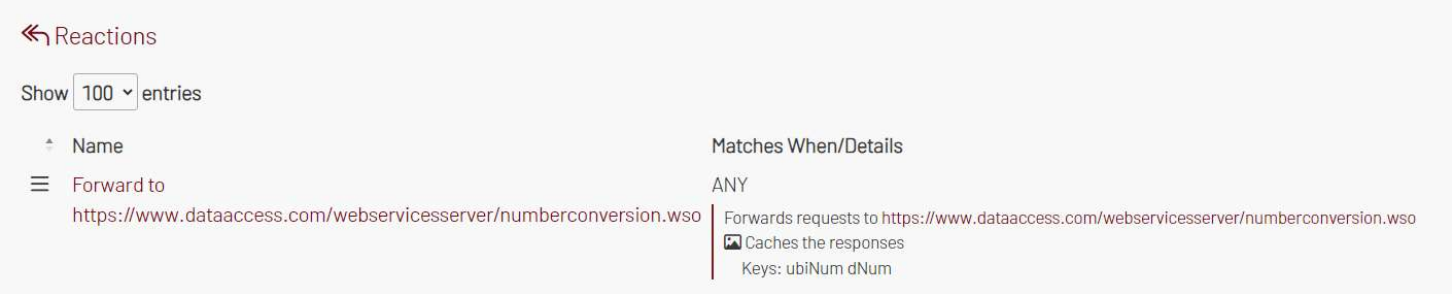
URL: https://www.dataaccess.com/webservicesserver/numberconversion.wso
Key Parameters: ubiNum dNum



Click on the “Create...” button. MockMotor creates a mock service with a forwarding reaction that sends all requests to the SOAP API but then caches the responses.



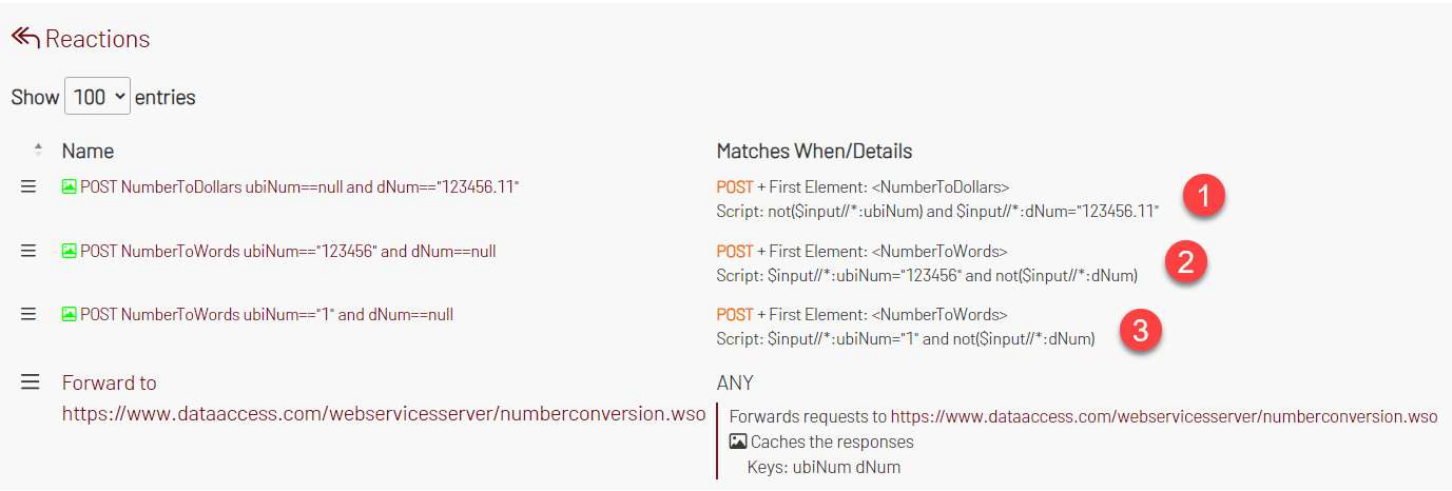
You should see the forwarding reaction when you click on the “Reactions” link.



Point your application to one of the URLs listed under the mock service.

> I suggest you use the plain HTTP URLs to avoid configuring the certificate trust.

If you execute a few requests from your application, you should see new cached reactions created.



The requests with repeating SOAP parameters are now served from mocks.

Questions?

I'm available to help you in installing the MockMotor instance.

Email me at: support@mockmotor.com

Text or WhatsApp me at: +1-416-878-5693