Artifact: Serverledge: Decentralized Function-as-a-Service for the Edge-Cloud Continuum

Gabriele Russo Russo*, Tiziana Mannucci*, Valeria Cardellini* and Francesco Lo Presti*
* *University of Rome Tor Vergata, Rome 00133, Italy*
Email: {russo.russo, cardellini}@ing.uniroma2.it, tiziana.mannucci@alumni.uniroma2.eu, lopresti@info.uniroma2.it

***Abstract*—Serverledge is a research prototype to run serverless functions in the Edge-Cloud Continuum. This artifact guide illustrates the minimal steps required to obtain, configure and run Serverledge. We also explain how to access the evaluation results presented in the paper and generate the associated figures.**

***Index Terms*—serverless, edge computing, offloading**

I. Introduction

Serverledge [1] has been implemented in Go and its source code is available as a GitHub repository[1], also archived in Zenodo [2]. The project comprises four key components:

- the core `serverledge` software, to be deployed in every node and delivering core function management and execution capabilities;
- `serverledge-cli`, a command-line interface (CLI) to easily interact with Serverledge nodes;
- `lb`, a simple load balancer to dispatch function invocation requests to multiple Serverledge nodes;
- function *runtime container images*: scripts to build Docker container images to run functions written in different languages (e.g., Python, NodeJS).

We provide instructions to run Serverledge on a single server: although the execution is related to a simple scenario, it allows the user to test the main features of Serverledge, including function invocation and offloading.

In the remainder of this section, we describe Serverledge prerequisites. For convenience, we also provide a script[2] to automatically install Serverledge and its dependencies on Ubuntu 22.04. It can be downloaded and run as follows:

```
$ wget -O deploy.sh \
https://bit.ly/serverledge2204 \
&& bash deploy.sh
```

After successfully executing the script, it is possible to immediately skip to Sec. II.

Preliminary Steps

In order to use the artifact, the following hardware and software requirements must be satisfied:

- Linux-based OS, on a x86-64 machine[3]
- Go[4] 1.19
- GNU Make
- Docker[5] 20.x, with permissions to create containers[6].

Serverledge itself can be downloaded from Zenodo [2] or GitHub[7].

II. Running Serverledge in the Local Host

Hereafter, we will assume that a terminal has been opened in the project root directory.

Compiling. The project can be compiled running:

```
$ make
```

Generated executable files will be in the `bin/` directory.

Starting Etcd. Serverledge uses Etcd to implement the *Global Registry* component. The repository contains a script to quickly run a local Etcd server using Docker:

```
$ bash scripts/start-etcd.sh
```

Starting Serverledge. We can start a Serverledge node on the local host with a default configuration running:

```
$ bin/serverledge
```

If there are no issues, the process will initialize the node and start listening on port 1323 for HTTP requests. Therefore, we need to launch a new terminal to proceed with the next steps.

III. Creating and Invoking Functions

The repository contains example functions in the `examples/` directory. Among them, `isprime.py` implements a trivial primality check as a Serverledge function, written in Python. The source code includes the *handler* function, which is called upon function invocation and receives user-specified parameters. We must register functions in Serverledge specifying their source file, the

[1] https://github.com/grussorusso/serverledge
[2] https://gist.github.com/grussorusso/1c1cca879616a038a99641ce8097a81f
[3] Actually, it is possible to run Serverledge on other Unix-like systems (e.g., macOS) or architectures (e.g., ARM). However, the project documentation and scripts currently target x86-64 Linux hosts only.
[4] https://go.dev/dl/
[5] https://docs.docker.com/engine/install/
[6] https://docs.docker.com/engine/install/linux-postinstall/#manage-docker-as-a-non-root-user
[7] https://github.com/grussorusso/serverledge/archive/refs/tags/v1.0.0-percom23.zip

amount of memory they require, the runtime environment (e.g., Python) and the identification of the handler function. Using `serverledge-cli`, we register an *isprime* function as follows:

```
$ bin/serverledge-cli create \
--function isprime --memory 256 \
--src examples/isprime.py \
--runtime python310 \
--handler "isprime.handler"
```

If the creation succeeds, the node will reply:

```
{ "Created": "isprime" }
```

We can again use the CLI to invoke the function passing the required parameter n as follows:

```
$ bin/serverledge-cli invoke -f isprime \
  -p "n:50"
```

The expected output looks as follows:[8]

```
{  "Success": true,
   "Result": "{\"IsPrime\": false}",
   "ResponseTime": 1.01616,
   "IsWarmStart": false,
   "InitTime": 1.01040, ... }
```

The output produced by the CLI includes the actual execution result within the `Result` field, along with a few metrics. For instance, the `IsWarmStart` field indicates whether a warm container has been used to execute the function or the request has experienced a cold start. In the latter case, the `InitTime` field reports the time spent initializing the container.

Repeating the command, we observe that new invocations exploit the warm container and enjoy reduced response times.

IV. Configuration

Serverledge can be configured by means of a YAML configuration file `serverledge-conf.yaml`, which can be placed either in `/etc/serverledge/`, in the user `$HOME` directory, or in the working directory where the server is started. Alternatively, you can indicate a specific configuration file when starting the node (an example is shown below).

The most relevant configuration options are listed in `docs/configuration.md`. For instance, create a file `temp-conf.yaml` in the current directory and enter the following lines to reduce the "expiration time" for idle containers:

```
container.expiration: 20
janitor.interval: 20
```

Terminate Serverledge and restart it as follows:

```
$ bin/serverledge ./temp-conf.yaml
```

Now, invoke the function as explained above and wait 30-40 seconds. Invoking again the function, we observe a cold start, as the warm container has been already terminated.

[8]The first time Serverledge is used, it may need to pull function runtime container images from Docker Hub. As such, the function invocation may require several seconds to complete.

V. Offloading

Serverledge supports function invocation offloading. In this section, we describe a proof-of-concept configuration to demonstrate the offloading mechanism without relying on a distributed deployment. For this purpose, we run 2 Serverledge nodes (i.e., Edge and Cloud nodes) on the local host, but we configure the Edge node to always offload requests to the Cloud (by doing so, we will have a single node spawning containers on the host).

After terminating any active Serverledge node, launch the two nodes using already prepared configuration files:

```
$ bin/serverledge \
   examples/local_offloading/confEdge.yaml
$ bin/serverledge\
   examples/local_offloading/confCloud.yaml
```

Now, using the instructions from Sec. III, invoke the *isprime* function. Since the Cloud node is configured to listen for requests on a non-default port, requests will be directed to the Edge node. The output will look like the following:

```
{  "Success": true,
   ...
   "OffloadLatency": 0.001413,
   "Duration": 0.006174,
   "SchedAction": "O" }
```

We verify that the request has been offloaded checking the `SchedAction` and `OffloadLatency` fields. The former signals an offloading decision through the string `'O'`, while the latter reports the additional latency caused by offloading (e.g., network latency, which is negligible in this example).

VI. Evaluation Data

The results of the experiments presented in the paper [1] are also publicly archived in Zenodo [3], along with scripts and instructions to reproduce the figures from the paper using Gnuplot.

For convenience, we also provide a Docker container image with the required software. The following command downloads the data from Zenodo, generates the plots and makes them accessible at `http://127.0.0.1:8080` as EPS and PDF files:

```
$ docker run --rm -it -p 8080:80 \
 grussorusso/serverledge-data-artifact
```

References

[1] G. Russo Russo, T. Mannucci, V. Cardellini, and F. Lo Presti, "Serverledge: Decentralized function-as-a-service for the edge-cloud continuum," in *Proceedings of IEEE International Conference on Pervasive Computing and Communications, PerCom 2023, Atlanta, GA, USA, March 13-17, 2023*. IEEE, 2023.

[2] G. Russo Russo, T. Mannucci, V. Cardellini, and F. Lo Presti, "Serverledge: Decentralized Function-as-a-Service for the Edge-Cloud Continuum," Jan. 2023. [Online]. Available: https://doi.org/10.5281/zenodo.7572481

[3] ——, "Experiment Results for "Serverledge: Decentralized Function-as-a-Service for the Edge-Cloud Continuum"," Feb. 2023. [Online]. Available: https://doi.org/10.5281/zenodo.7607376