Computing with Charity Engine

DRAFT - WORK IN PROGRESS | Services are in active development and are subject to change.

Contents

1. Overview

Charity Engine provides a compute service running on volunteer devices. The system allows one to run compute jobs, executing standard Docker images from Docker Hub, custom docker containers or a set of proprietary applications for an additional fee.

Compute jobs can range from a single small job to a massively parallel compute job spanning hundreds of thousands of nodes. All nodes have a limited network access allowing one to retrieve input data from the files submitted during job creation or internet resources that are available at HTTP(S) websites.

Resources are provisioned in a familiar “instance type” system, where various compute nodes are made available in uniform sets of instances, differing in CPU capacity, memory and disk availability.

To start computing, one must obtain credentials to the system and submit jobs through one of the supported interfaces. The interfaces tailor to several of the use cases, such as manual submission of jobs through a graphical web interface (useful for testing or submitting small quantities of jobs), integration to the existing systems via a Remote API, or using Remote CLI to get massively parallel executions with already-known tools such as GNU Parallel.

You will need your docker image name/URL or a proprietary application name, your input files and the command line to execute within the executed environment. The command line can be as simple as executing an application with a docker image, a command to execute the first input file, or a more involved script that handles the logic of your job.

2. Interfaces

- Remote API
    - This API is provided for integration of Charity Engine as a backend service, and summarizes functions for managing the entire lifecycle of jobs submitted to the network.
- Remote CLI
    - The Remote CLI provides a means to run jobs on Charity Engine compute resources using simple command line tools. (e.g. custom scripts or tools such as GNU Parallel). This interface is geared toward running large batches of jobs, but can be used anytime a command line approach is more appropriate than a web API.
        - Standalone command-line interface
        - GNU Parallel (using *Parallel* allows you to manage batches of work using just a single command line tool)
- Ethereum
    - Smart Contract
    - Dapp UI
- Web
    - A web-based GUI is available here.
- Custom Arrangements
    - contact@charityengine.com

3. Applications

Charity Engine supports several types of applications:

- Docker images available on Docker Hub (e.g. `docker:python:3.7`)
- Custom Docker images available anywhere online (e.g. `docker:image-name https://example.com/file` for 64bit docker images, or `docker-x86:image-name https://example.com/file` for 32bit docker images)
- Proprietary applications deployed directly on the Charity Engine network (e.g. `charityengine:wolframengine`). The AppStore library can be viewed here.

4. Input + Output Files

Once the computation environment runs, the input files provided during job completion are downloaded from the internet and are made available for the applications on the compute nodes in the `/local/input` virtual disk location.

If the input files are marked as cacheable, they might get cached on the compute node (*see Interface documentation for details), and subsequent executions will skip the download and use the cached file instead. Caching is done based on the file URL. If input file caching is used, one should assume that files are immutable and use new URLs for any new version of the file that is being created to ensure consistent behavior.

The output files of computations are expected to be written into `/local/output` virtual disk location. The URLs to the output files or the output files themselves are then made available in the output file section in all of the interfaces.

Some applications may not have options to change locations of their input/output files, or modifications to the locations are not desirable, for example, to minimize testing required when such changes are introduced. It is however possible to use simple shell tools to move or copy files to their final locations, for example, by specifying the application command line as follows:

```
containerized_app --param example; cp output_file /local/output/output_file
```

5. Distributed Storage

Distributed Storage is a networked file storage system which can be used to host large quantities of data, with data persistently maintained and replicated across multiple devices distributed worldwide. Access to this network is possible using all of the Charity Engine interfaces such as the Remote API, Remote CLI, and Smart Contracts, as well as industry standard IPFS client applications.

6. Networking

Network access through HTTP (port 80) and HTTPS (port 443) protocols is allowed. The network speeds may vary based on the node location and the network/system load. Network latency is artificially throttled to several seconds per request, but multiple parallel requests are allowed and recommended.

Each compute reservation receives a limited amount of free bandwidth. Once the allocated bandwidth is exhausted, any further network communication will incur additional fees.

Additionally, network requests can be routed through the Charity Engine Distributed Proxy Service which uses the power and unique, geographically-dispersed nature of our network to allow web requests to originate from locations around the world.

7. Instance Types

Initial service implementation defines three instance types (see below), where the first number denotes the number of available vCPUs and the second number denotes the amount of available RAM (in GiB).

General Purpose Computing - Linux *(*at launch)*	vCPU	RAM
C.2x2	2	2
C.2x4	2	4
C.4x4	4	4

GPU Computing (Nvidia) - Linux *(*at launch)*	GPU Model	GPU Count
N.1070x1	GeForce 1070	1
N.1070x2	GeForce 1070	2
N.1070x4	GeForce 1070	4
N.1070x8	GeForce 1070	8
N.1080x1	GeForce 1080	1
N.1080x2	GeForce 1080	2
N.1080x4	GeForce 1080	4
N.1080x8	GeForce 1080	8
N.T4x1	Tesla T4	1
N.T4x2	Tesla T4	2
N.T4x4	Tesla T4	4

N.T4x8	Tesla T4	8
N.V100x1	Tesla V100	1
N.V100x2	Tesla V100	2
N.V100x4	Tesla V100	4
N.V100x8	Tesla V100	8

7.1. Special Feature Flags

Workloads may require specific hardware and software features beyond those in the basic instance specification above. This can be requested by use of flags in the instance request interface. Note: use of such flags may {1} reduce available inventory, and {2} be subject to additional charges.

Currently supported flags are:

- `Tensorflow` - Instance CPU with AVX support
- `AVX` - Instance CPU with AVX support
- `Driver:nnn.nn` - GPU driver version for GPU instances, such as 440.33.01

Flags are case insensitive.

7.2. Custom Arrangements

Custom arrangements may be possible; contact us if currently available instance types or feature-sets are not suitable for your workloads.

8. Support

(*link to forums)