Backend.AI Documentation

Release 1.0

Joongi Kim

Aug 08, 2023

User Manuals

Latest API version: v3.20170615 (beta)

Backend.AI is a hassle-free backend for AI programming and service. It runs arbitrary user codes safely in resource-constrained environments, using Docker and our own sandbox wrapper.

Backend.AI supports various programming languages and runtimes, such as Python 2/3, R, PHP, C/C++, Java, Javascript, Julia, Octave, Haskell, Lua and NodeJS, as well as AI-oriented libraries such as TensorFlow, Keras, Caffe, and MXNet.

CHAPTER 1

FAQ

1.1 vs. Notebooks

Product	Role	Problem and Solution
Apache Zeppelin, Jupyter Notebook	Notebook-style document + code *front-ends*	Insecure host resource sharing
Backend.AI	Pluggable *back-end* to any front-ends	Built for multi-tenancy: scalable and better isolation

1.2 vs. Orchestration Frameworks

Product	Target	Value
Amazon ECS, Kubernetes	Long-running service daemons	Load balancing, fault tolerance, incremental deployment
Backend.AI	Stateful compute sessions	Low-cost high-density computation
Amazon Lambda	Stateless, light-weight functions	Serverless, zero-management

1.3 vs. Big-data and AI Frameworks

Product	Role	Problem and Solution
TensorFlow, Apache Spark, Apache Hive	Computation runtime	Difficult to install, configure, and operate
Amazon ML, Azure ML, GCP ML	Managed MLaaS	Still complicated for scientists, too restrictive for engineers
Backend.AI	Host of computation runtimes	Pre-configured, versioned, reproducible, customizable (open-source)

(All product names and trade-marks are the properties of their respective owners.)

CHAPTER 2

Table of Contents

2.1 API Overview

Backend.AI API v3 consists of two parts: User APIs and Admin APIs.

Warning: APIv3 breaks backward compatibility a lot, and we will primarily support v3 after June 2017. Please upgrade your clients immediately.

2.1.1 API KeyPair Registration

For managed, best-experience service, you may register to our cloud version of Backend.AI API service instead of installing it to your own machines. Simply create an account at cloud.backend.ai and generate a new API keypair. You may also use social accounts for log-ins such as Twitter, Facebook, and GitHub.

An API keypair is composed of a 20-characters access key (`AKIA...`) and a 40-characters secret key, in a similar form to AWS access keys.

Currently, the service is BETA: it is free of charge but each user is limited to have only one keypair and have up to 5 concurrent kernel sessions for a given keypair. Keep you eyes on further announcements for upgraded paid plans.

2.1.2 Accessing Admin APIs

The admin APIs require a special keypair with the admin privilege:

- The public cloud service (`api.backend.ai`): It currently does *not* offer any admin privileges to the end-users, as its functionality is already available via our management console at cloud.backend.ai.
- On-premise installation: You will get an auto-generated admin keypair during installation.

2.2 Python Client Library

We provide an official Python client library that abstracts the low-level HTTP REST APIs via a function-based interface.

2.2.1 Requirements

Python 3.6 or higher is required. You can download its official installer from python.org, or use a 3rd-party package/version manager such as homebrew, miniconda, or pyenv. It works on Linux, macOS, and Windows.

2.2.2 Installation

We recommend to create a virtual environment for isolated, unobtrusive installation of the library.

```
$ python3 -m venv venv-backend-ai
$ source venv-backend-ai/bin/activate
(venv-backend-ai) $
```

Then install the client library from PyPI.

```
(venv-backend-ai) $ pip install -U pip wheel setuptools
(venv-backend-ai) $ pip install sorna-client
```

2.2.3 Configuration

Set your API keypair as environment variables:

```
(venv-backend-ai) $ export BACKEND_ACCESS_KEY=AKIA...
(venv-backend-ai) $ export BACKEND_SECRET_KEY=...
```

The run Python in the virtual environment and check if your credentials are valid:

```
>>> from sorna.request import Request
>>> request = Request('GET', '/authorize', {'echo': 'test'})
>>> request.sign()
>>> response = request.send()
>>> response.status
200
>>> response.json()
{'authorized': 'yes', 'echo': 'test'}
```

2.3 Overview

Welcome to the Backend.AI Installation!

2.3.1 Guides

Server Installation Guides

- Server Concepts – Key concepts to know before diving into the installation process

- Install on Clouds – How to install a working cluster on public IaaS clouds
- Install on Premise – How to install a working cluster on your server farm
- *Development Setup* – How to install all components as editable packages on your PC
- *Demo Setup* – `docker-compose up -d` and that's all!

Supplemental Guides

- Install Docker – How to install docker
- Install Python via pyenv – Install pyenv to install multiple Python versions side-by-side
- Install Monitoring and Logging Tools – Add third-party monitoring and logging tools (optional)
- Prepare Databases for Manager – Prepare database for complex manager managements and tasks
- Install CUDA – Install CUDA to use Nvidia GPUs

Developer Guides

- Version Management and Upgrades

2.4 Demo Setup

This meta-repository provides a docker-compose configuration to fire up a single-node Backend.AI cluster running on your PC (http://localhost:8081).

2.4.1 Prerequisites

- All: install Docker 17.06 or later with docker-compose v1.21 or later
- Linux users: change "docker.for.mac.localhost" in docker-compose.yml to "172.17.0.1"

Notes

- This demo setup does *not* support GPUs.

2.4.2 All you have to do

- Clone the repository
- Check out the prerequisites above
- `docker-compose up -d`
  - For Windows, `docker-compose -f docker-compose.win-demo.yml up -d`
- Pull some kernel images to try out

Pulling kernel images

Pull the images on your host Docker daemon like:

```
$ docker pull lablup/kernel-python:latest
$ docker pull lablup/kernel-python-tensorflow:latest-dense
$ docker pull lablup/kernel-c:latest
```

By default this demo cluster already has metadata/alias information for all publicly available Backend.AI kernels, so you don't have to manually register the pulled kernel information to the cluster but only have to *pull* those you want to try out.

Using Clients

To access this local cluster, set the following configurations to your favoriate Backend.AI client:

```
$ export BACKEND_ENDPOINT="http://localhost:8081"
$ export BACKEND_ACCESS_KEY="AKIAIOSFODNN7EXAMPLE"
$ export BACKEND_SECRET_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
```

With our official Python client, you can do:

```
$ backend.ai run python -c "print('hello world')"
✓ Session 9c737d84724173354fa10445d0b35fe0 is ready.
hello world
✓ Finished. (exit code = 0)

$ backend.ai run python-tensorflow:latest-dense -c "import tensorflow as tf; print(tf.
↪__version__)"
✓ Session 950713741d5ed43a191704f2cd375ff0 is ready.
1.5.0
✓ Finished. (exit code = 0)
```

WARNING: This demo configuration is highly insecure. DO NOT USE in production!

2.4.3 FAQ

- When launching a kernel, it says "Service Unavailable"!
  - Each image has different default resource requirements and your Docker daemon may have a too small amount of resources. For example, TensorFlow images require 8 GiB or more RAM for your Docker daemon.
  - Or, you might have launched 30 kernel sessions already, which is the default limit for this demo setup.
- What does the "dense" tag mean in the TensorFlow kernel images?
  - Images with "dense" tags are optimized for shared multi-tenancy environments. There is no difference in functionalities.

2.5 Development Setup

Currently Backend.AI is developed and tested under only *NIX-compatible platforms (Linux or macOS).

2.5.1 Method 1: Automatic Installation

For the ease of on-boarding developer experience, we provide an automated script that installs all server-side components in editable states with just one command.

Prerequisites

Install the followings accordingly to your host operating system.

- pyenv and pyenv-virtualenv
- docker
- docker-compose

Note: In some cases, locale conflicts between the terminal client and the remote host may cause encoding errors when installing Backend.AI components due to Unicode characters in README files. Please keep correct locale configurations to prevent such errors.

Warning: In macOS, Homebrew offers its own pyenv and pyenv-virtualenv packages but we *do not* recommend using them! Updating those packages and cleaning up via Homebrew will break your virtual environments as each version uses different physical directories.

Our installer script will try to install pyenv automatically if not installed, but we *do* recommend installing them by yourself as it may interfere with your shell configurations.

Running the script

```
$ wget https://raw.githubusercontent.com/lablup/backend.ai/master/scripts/install-dev.
↪sh
$ chmod +x ./install-dev.sh
$ ./install-dev.sh
```

Note: The script may ask your root password in the middle to run sudo in Linux.

This installs a set of Backend.AI server-side components in the `backend.ai-dev` directory under the current working directory.

Inside the directory, there are `manager`, `agent`, `common` and a few other auxiliary directories. You can directly modify the source codes inside them and re-launch the gateway and agent. The `common` directory is shared by `manager` and `agent` so just editing sources there takes effects in the next launches of the gateway and agent.

At the end of execution, the script will show several command examples about launching the gateway and agent. It also displays a unique random key called "environment ID" to distinguish a particular execution of this script so that repeated execution does not corrupt your existing setups.

By default, the script pulls the docker images for our standard Python kernel and TensorFlow CPU-only kernel. To try out other images, you have to pull them manually afterwards.

The script provides a set of command-line options. Check out them using `-h` / `--help` option.

Note: To install multiple instances of development environments using this script, you need to run the script at different working directories because the `backend.ai-dev` directory name is fixed.

Also, you cannot run multiple gateways and agents from different environments at the same time because docker container in different environments use the same TCP ports of the host system. Use `docker-compose` command to stop the current environment and start another to switch between environments. Please do not forget to specify `-p <ENVID>` option to `docker-compose` commands to distinguish different environments.

Resetting the environment

```
$ wget https://raw.githubusercontent.com/lablup/backend.ai/master/scripts/delete-dev.
↪sh
$ chmod +x ./delete-dev.sh
$ ./delete-dev.sh --env <ENVID>
```

Note: The script may ask your root password in the middle to run sudo in Linux.

This will purge all docker resources related to the given environment ID and the `backend.ai-dev` directory under the current working directory.

The script provides a set of command-line options. Check out them using `-h` / `--help` option.

Warning: Be aware that this script force-removes, without any warning, all contents of the `backend.ai-dev` directory, which may contain your own modifications that is not yet pushed to a remote git repository.

2.5.2 Method 2: Manual Installation

Requirement packages

- PostgreSQL: 9.6
- etcd: v3.3.9
- redis: latest

Prepare containers for external daemons

First install an appropriate version of Docker (later than 2017.03 version) and docker-compose (later than 1.21). Check out the Install Docker guide.

Note: In this guide, `$WORKSPACE` means the absolute path to an arbitrary working directory in your system.

To copy-and-paste commands in this guide, set `WORKSPACE` environment variable.

The directory structure would look like after finishing this guide:

- **`$WORKSPACE`**
  - backend.ai
  - backend.ai-manager

  - backend.ai-agent
  - backend.ai-common
  - backend.ai-client-py

```
$ cd $WORKSPACE
$ git clone https://github.com/lablup/backend.ai
$ cd backend.ai
$ docker-compose -f docker-compose.halfstack.yml up -d
$ docker ps  # you should see 3 containers running
```

This recording has been archived

This will create and start PostgreSQL, Redis, and a single-instance etcd containers. Note that PostgreSQL and Redis uses non-default ports by default (5442 and 6389 instead of 5432 and 6379) to prevent conflicts with other application development environments.

Prepare Python 3.6+

Check out Install Python via pyenv for instructions.

Create the following virtualenvs: `venv-manager`, `venv-agent`, `venv-common`, and `venv-client`.

This recording has been archived

Prepare dependent libraries

Install `snappy` (brew on macOS), `libsnappy-dev` (Debian-likes), or `libsnappy-devel` (RHEL-likes) system package depending on your environment.

Prepare server-side source clones

This recording has been archived

Clone the Backend.AI source codes.

```
$ cd $WORKSPACE
$ git clone https://github.com/lablup/backend.ai-manager
$ git clone https://github.com/lablup/backend.ai-agent
$ git clone https://github.com/lablup/backend.ai-common
```

Inside each directory, install the sources as editable packages.

Note: Editable packages makes Python to apply any changes of the source code in git clones immediately when importing the installed packages.

```
$ cd $WORKSPACE/backend.ai-manager
$ pyenv local venv-manager
$ pip install -U -r requirements-dev.txt
```

```
$ cd $WORKSPACE/backend.ai-agent
$ pyenv local venv-agent
$ pip install -U -r requirements-dev.txt
```

```
$ cd $WORKSPACE/backend.ai-common
$ pyenv local venv-common
$ pip install -U -r requirements-dev.txt
```

(Optional) Symlink backend.ai-common in the manager and agent directories to the cloned source

If you do this, your changes in the source code of the backend.ai-common directory will be reflected immediately to the manager and agent. You should install backend.ai-common dependencies into `venv-manager` and `venv-agent` as well, but this is already done in the previous step.

```
$ cd "$(pyenv prefix venv-manager)/src"
$ mv backend.ai-common backend.ai-common-backup
$ ln -s "$WORKSPACE/backend.ai-common" backend.ai-common
```

```
$ cd "$(pyenv prefix venv-agent)/src"
$ mv backend.ai-common backend.ai-common-backup
$ ln -s "$WORKSPACE/backend.ai-common" backend.ai-common
```

Initialize databases and load fixtures

Check out the Prepare Databases for Manager guide.

Prepare Kernel Images

You need to pull the kernel container images first to actually spawn compute sessions. The kernel images here must have the tags specified in image-metadata.yml file.

```
$ docker pull lablup/kernel-python:3.6-debian
```

For the full list of publicly available kernels, check out the kernels repository.

NOTE: You need to restart your agent if you pull images after starting the agent.

Setting Linux capabilities to Python (Linux-only)

To allow Backend.AI to collect sysfs/cgroup resource usage statistics, the Python executable must have the following Linux capabilities (to run without "root"): `CAP_SYS_ADMIN`, `CAP_SYS_PTRACE`, and `CAP_DAC_OVERRIDE`. You may use the following command to set them to the current virtualenv's Python executable.

```
$ sudo setcap cap_sys_ptrace,cap_sys_admin,cap_dac_override+eip $(readlink -f $(pyenv␣
↪which python))
```

Running daemons from cloned sources

```
$ cd $WORKSPACE/backend.ai-manager
$ ./scripts/run-with-halfstack.sh python -m ai.backend.gateway.server --service-
↪port=8081 --debug
```

Note that through options, PostgreSQL and Redis ports set above for development environment are used. You may change other options to match your environment and personal configurations. (Check out `-h` / `--help`)

```
$ cd $WORKSPACE/backend.ai-agent
$ mkdir -p scratches  # used as in-container scratch "home" directories
$ ./scripts/run-with-halfstack.sh python -m ai.backend.agent.server --scratch-
↪root=`pwd`/scratches --debug --idle-timeout 30
```

※ The role of `run-with-halfstack.sh` script is to set appropriate environment variables so that the manager/agent daemons use the halfstack docker containers.

Prepare client-side source clones

This recording has been archived

```
$ cd $WORKSPACE
$ git clone https://github.com/lablup/backend.ai-client-py
```

```
$ cd $WORKSPACE/backend.ai-client-py
$ pyenv local venv-client
$ pip install -U -r requirements-dev.txt
```

Inside `venv-client`, now you can use the `backend.ai` command for testing and debugging.

2.5.3 Verifying Installation

Write a shell script (e.g., `env_local.sh`) like below to easily switch the API endpoint and credentials for testing:

```
#! /bin/sh
export BACKEND_ENDPOINT=http://127.0.0.1:8081/
export BACKEND_ACCESS_KEY=AKIAIOSFODNN7EXAMPLE
export BACKEND_SECRET_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

Load this script (e.g., `source env_local.sh`) before you run the client against your server-side installation.

Now you can do `backend.ai ps` to confirm if there are no sessions running and run the hello-world:

```
$ cd $WORKSPACE/backend.ai-client-py
$ source env_local.sh  # check above
$ backend.ai run python -c 'print("hello")'
```

2.6 API and Document Conventions

2.6.1 HTTP Methods

We use the standard HTTP/1.1 methods (RFC-2616), such as `GET`, `POST`, `PUT`, `PATCH` and `DELETE`, with some additions from WebDAV (RFC-3253) such as `REPORT` method to send JSON objects in request bodies with `GET` semantics.

If your client runs under a restrictive environment that only allows a subset of above methods, you may use the universal `POST` method with an extra HTTP header like `X-Method-Override: REPORT`, so that the Backend.AI gateway can recognize the intended HTTP method.

2.6.2 Parameters in URI and JSON Request Body

The parameters with *colon prefixes* (e.g., `:id`) are part of the URI path and must be encoded using a proper URI-compatible encoding schemes such as `encodeURIComponent(value)` in Javascript and `urllib.parse.quote(value, safe='~()*!.\'')` in Python 3+.

Other parameters should be set as a key-value pair of the JSON object in the HTTP request body. The API server accepts both UTF-8 encoded bytes and standard-compliant Unicode-escaped strings in the body.

2.6.3 HTTP Status Codes and JSON Response Body

The API responses always contain a root JSON object, regardless of success or failures.

For successful responses (HTTP status 2xx), the root object has a varying set of key-value pairs depending on the API.

For failures (HTTP status 4xx/5xx), the root object contains at least two keys: `type` which uniquely identifies the failure reason as an URI and `title` for human-readable error messages. Some failures may return extra structured information as additional key-value pairs. We use RFC 7807-style problem detail description returned in JSON of the response body.

2.6.4 JSON Field Notation

Dot-separated field names means a nested object. If the field name is a pure integer, it means a list item.

Example	Meaning
`a`	The attribute `a` of the root object. (e.g., `123` at `{"a": 123}`)
`a.b`	The attribute `b` of the object `a` on the root. (e.g., `456` at `{"a": {"b": 456}}`)
`a.0`	An item in the list `a` on the root. `0` means an arbitrary array index, not the specific item at index zero. (e.g., any of `13`, `57`, `24`, and `68` at `{"a": [13, 57, 24, 68]}`)
`a.0.b`	The attribute `b` of an item in the list `a` on the root. (e.g., any of `1`, `2`, and `3` at `{"a": [{"b": 1}, {"b": 2}, {"b": 3}]}`)

2.6.5 JSON Value Types

This documentation uses a type annotation style similar to Python's typing module, but with minor intuitive differences such as lower-cased generic type names and wildcard as asterisk `*` instead of `Any`.

The common types are `array` (JSON array), `object` (JSON object), `int` (integer-only subset of JSON number), `str` (JSON string), and `bool` (JSON `true` or `false`). `tuple` and `list` are aliases to `array`. Optional values may be omitted or set to `null`.

We also define several custom types:

Type	Description
`decimal`	Fractional numbers represented as `str` not to loose precision. (e.g., to express money amounts)
`slug`	Similar to `str`, but the values should contain only alpha-numeric characters, hyphens, and underscores. Also, hyphens and underscores should have at least one alphanumeric neighbor as well as cannot become the prefix or suffix.
`datetime`	ISO-8601 timestamps in `str`, e.g., `"YYY-mm-ddTHH:MM:SS.ffffff+HH:MM"`. It may include an optional timezone information. If timezone is not included, the value is assumed to be UTC. The sub-seconds parts has at most 6 digits (micro-seconds).
`enum[*]`	Only allows a fixed/predefined set of possible values in the given parametrized type.

2.6.6 API Versioning

A version string of the Backend.AI API uses two parts: a major revision (prefixed with `v`) and minor release dates after a dot following the major revision. For example, `v23.20250101` indicates a 23rd major revision with a minor release at January 1st in 2025.

We keep backward compatibility between minor releases within the same major version. Therefore, all API query URLs are prefixed with the major revision, such as `/v2/kernel/create`. Minor releases may introduce new parameters and response fields but no URL changes. Accessing unsupported major revision returns HTTP 404 Not Found.

A client must specify the API version in the HTTP request header named `X-BackendAI-Version`. To check the latest minor release date of a specific major revision, try a GET query to the URL with only the major revision part (e.g., `/v2`). The API server will return a JSON string in the response body containing the full version. When querying the API version, you do not have to specify the authorization header and the rate-limiting is enforced per the client IP address. Check out more details about *Authentication* and *Rate Limiting*.

Example version check response body:

```
{
   "version": "v2.20170315"
}
```

2.7 Authentication

2.7.1 Access Tokens and Secret Key

To make requests to the API server, a client needs to get a pair of an access token and a secret key as sepcified in /gsg/registration. The server uses access tokens to identify each client and secret keys to verify integrity of API requests as well as to authenticate clients.

Warning: For security reasons (to avoid exposition of your API access key and secret keys to arbitrary Internet users), we highly recommend to setup a server-side proxy to our API service if you are building a public-facing front-end service using Backend.AI.

For local deployments, you may create a master dummy pair in the configuration (TODO).

2.7.2 Common Structure of API Requests

HTTP Headers	Values
Method	`GET` / `REPORT` / `POST` / `PUT` / `PATCH` / `DELETE`
`Content-Type`	Always should be `application/json`
`Authorization`	Signature information generated as the section *Signing API Requests* describes.
`Date`	The date/time of the request formatted in RFC 8022 or ISO 8601. If no timezone is specified, UTC is assumed. The deviation with the server-side clock must be within 15-minutes.
`X-BackendAI-Date`	Same as `Date`. May be omitted if `Date` is present.
`X-BackendAI-Version`	`vX.yyymmdd` where `X` is the major version and `yyyymmdd` is the minor release date of the specified API version. (e.g., 20160915)
`X-BackendAI-Client-Token`	An optional, client-generated random string to allow the server to distinguish repeated duplicate requests. It is important to keep idempotent semantics with multiple retries for intermittent failures. (Not implemented yet)
Body	JSON-encoded request parameters

2.7.3 Common Structure of API Responses

HTTP Headers	Values
Status code	API-specific HTTP-standard status codes. Responses commonly used throughout all APIs include 200, 201, 2014, 400, 401, 403, 404, 429, and 500, but not limited to.
`Content-Type`	`application/json` and its variants (e.g., `application/problem+json` for errors)
`Link`	Web link headers specified as in RFC 5988. Only optionally used when returning a collection of objects.
`X-RateLimit-*`	The rate-limiting information (see *Rate Limiting*).
Body	JSON-encoded results

2.7.4 Signing API Requests

Each API request must be signed with a signature. First, the client should generate a signing key derived from its API secret key and a string to sign by canonicalizing the HTTP request.

Generating a signing key

Here is a Python code that derives the signing key from the secret key. The key is nestedly signed against the current date (without time) and the API endpoint address.

```
import hashlib, hmac
from datetime import datetime

SECRET_KEY = b'abc...'

def sign(key, msg):
  return hmac.new(key, msg, hashlib.sha256).digest()

def get_sign_key():
  t = datetime.utcnow()
```

(continues on next page)

(continued from previous page)

```
    k1 = sign(SECRET_KEY, t.strftime('%Y%m%d').encode('utf8'))
    k2 = sign(k1, b'your.sorna.api.endpoint')
    return k2
```

Generating a string to sign

The string to sign is generated from the following request-related values:

- HTTP Method (uppercase)
- URI including query strings
- The value of `Date` (or `X-BackendAI-Date` if `Date` is not present) formatted in ISO 8601 (`YYYYmmddTHHMMSSZ`) using the UTC timezone.
- The canonicalized header/value pair of `Host`
- The canonicalized header/value pair of `Content-Type`
- The canonicalized header/value pair of `X-BackendAI-Version`
- The hex-encoded hash value of body as-is. The hash function must be same to the one given in the `Authorization` header (e.g., SHA256).

To generate a string to sign, the client should join the above values using the newline (`"\n"`, ASCII 10) character. All non-ASCII strings must be encoded with UTF-8. To canonicalize a pair of HTTP header/value, first trim all leading/trailing whitespace characters (`"\n"`, `"\r"`, `" "`, `"\t"`; or ASCII 10, 13, 32, 9) of its value, and join the lowercased header name and the value with a single colon (`":"`, ASCII 58) character.

The success example in *Example Requests and Responses* makes a string to sign as follows (where the newlines are `"\n"`):

```
GET
/v2
20160930T01:23:45Z
host:your.sorna.api.endpoint
content-type:application/json
x-sorna-version:v2.20170215
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
```

In this example, the hash value `e3b0c4...` is generated from an empty string using the SHA256 hash function since there is no body for GET requests.

Then, the client should calculate the signature using the derived signing key and the generated string with the hash function, as follows:

```
import hashlib, hmac

str_to_sign = 'GET\n/v2...'
sign_key = get_sign_key()  # see "Generating a signing key"
m = hmac.new(sign_key, str_to_sign.encode('utf8'), hashlib.sha256)
signature = m.hexdigest()
```

Attaching the signature

Finally, the client now should construct the following HTTP `Authorization` header:

```
Authorization: BackendAI signMethod=HMAC-SHA256, credential=<access-key>:<signature>
```

2.7.5 Example Requests and Responses

For the examples here, we use a dummy access key and secret key:

- Example access key: `AKIAIOSFODNN7EXAMPLE`
- Example secret key: `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`

Success example for checking the latest API version

```
GET /v2 HTTP/1.1
Host: your.sorna.api.endpoint
Date: 20160930T01:23:45Z
Authorization: BackendAI signMethod=HMAC-SHA256,␣
↪credential=AKIAIOSFODNN7EXAMPLE:022ae894b4ecce097bea6eca9a97c41cd17e8aff545800cd696112cc387059cf
Content-Type: application/json
X-BackendAI-Version: v2.20170215
```

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Language: en
Content-Length: 31
X-RateLimit-Limit: 2000
X-RateLimit-Remaining: 1999
X-RateLimit-Reset: 897065

{
   "version": "v2.20170215"
}
```

Failure example with a missing authorization header

```
GET /v2/kernel/create HTTP/1.1
Host: your.sorna.api.endpoint
Content-Type: application/json
X-BackendAI-Date: 20160930T01:23:45Z
X-BackendAI-Version: v2.20170215
```

```
HTTP/1.1 401 Unauthorized
Content-Type: application/problem+json
Content-Language: en
Content-Length: 139
X-RateLimit-Limit: 2000
X-RateLimit-Remaining: 1998
X-RateLimit-Reset: 834821

{
   "type": "https://sorna.io/problems/unauthorized",
   "title": "Unauthorized access",
   "detail": "Authorization header is missing."
}
```

2.8 Rate Limiting

The API server imposes a rate limit to prevent clients from overloading the server. The limit is applied to the last *N* minutes at ANY moment (*N* is 15 minutes by default).

For public non-authorized APIs such as version checks, the server uses the client's IP address seen by the server to impose rate limits. Due to this, please keep in mind that large-scale NAT-based deployments may encounter the rate limits sooner than expected. For authorized APIs, it uses the access key in the authorization header to impose rate limits. The rate limit includes both all successful and failed requests.

Upon a valid request, the HTTP response contains the following header fields to help the clients flow-control their requests.

HTTP Headers	Values
`X-RateLimit-Limit`	The maximum allowed number of requests during the rate-limit window.
`X-RateLimit-Remaining`	The number of further allowed requests left for the moment.
`X-RateLimit-Window`	The constant value representing the window size in seconds. (e.g., 900 means 15 minutes) Changed in version v3.20170615: Deprecated `X-RateLimit-Reset` and transitional `X-Retry-After` as we have implemented a rolling counter that measures last 15 minutes API call counts at any moment.

When the limit is exceeded, further API calls will get HTTP 429 "Too Many Requests". If the client seems to be DDoS-ing, the server may block the client forever without prior notice.

2.9 JSON Object References

2.9.1 Paging Query Object

It describes how many items to fetch for object listing APIs. If `index` exceeds the number of pages calculated by the server, an empty list is returned.

Key	Type	Description
`size`	`int`	The number of items per page. If set zero or this object is entirely omitted, all items are returned and `index` is ignored.
`index`	`int`	The page number to show, zero-based.

2.9.2 Paging Info Object

It contains the paging information based on the paging query object in the request.

Key	Type	Description
`pages`	`int`	The number of total pages.
`count`	`int`	The number of all items.

2.9.3 KeyPair Item Object

Key	Type	Description
`accessKey`	`slug`	The access key part.
`isActive`	`bool`	Indicates if the keypair is active or not.
`totalQueries`	`int`	The number of queries done via this keypair. It may have a stale value.
`created`	`datetime`	The timestamp when the keypair was created.

2.9.4 KeyPair Properties Object

Key	Type	Description
`isActive`	`bool`	Indicates if the keypair is activated or not. If not activated, all authentication using the keypair returns 401 Unauthorized. When changed from `true` to `false`, existing running kernel sessions continue to run but any requests to create new kernel sessions are refused. (default: `true`)
`concurrecy`	`int`	The maximum number of concurrent kernel sessions allowed for this keypair. (default: `5`)
`ML. clusterSize`	`int`	Sets the number of instances clustered together when launching new machine learning kernel sessions. (default: `1`)
`ML. instanceMemory`	`int` (MiB)	Sets the memory limit of each instance in the cluster launched for new machine learning kernel sessions. (default: `8`)

The enterprise edition offers the following additional properties:

Key	Type	Description
`cost. automatic`	`bool`	If set `true`, enables automatic cost optimization (BETA). With supported kernel types, it automatically suspends or resize the kernel sessions not to exceed the configured cost limit per day. (default: `false`)
`cost. dailyLimit`	`str`	The string representation of money amount as decimals. The currency is fixed to USD. (default: `"50.00"`)

2.9.5 Batch Execution Query Object

Key	Type	Description
`build`	`str`	The bash command to build the main program from the given uploaded files. If this field is not present, an empty string or `null`, it skips the build step. If this field is a constant string `"*"`, it will use a default build script provided by the kernel. For example, the C kernel's default Makefile adds all C source files under the working directory and copmiles them into `./main` executable, with commonly used C/link flags: `"-pthread -lm -lrt -ldl"`.
`exec`	`str`	The bash command to execute the main program. If this is not present, an empty string, or `null`, the server only performs the build step and `options.buildLog` is assumed to be `true` (the given value is ignored).

Note: A client can distinguish whether the current output is from the build phase or the execution phase by whether it has received `build-finished` status or not.

Note: All shell commands are by default executed under `/home/work`. The common environment is:

```
TERM=xterm
LANG=C.UTF-8
SHELL=/bin/bash
USER=work
HOME=/home/work
```

but individual kernels may have additional environment settings.

Warning: The shell does NOT have access to sudo or the root privilege. Though, some kernels may allow installation of language-specific packages in the user directory.

Also, your build script and the main program is executed inside Backend.AI Jail, meaning that some system calls are blocked by our policy. Since `ptrace` syscall is blocked, you cannot use native debuggers such as gdb.

This limitation, however, is subject to change in the future.

Example:

```
{
  "build": "gcc -Wall main.c -o main -lrt -lz",
  "exec": "./main"
}
```

2.9.6 Execution Result Object

Key	Type	Description
`runId`	`str`	The user-provided run identifier. If the user has NOT provided it, this will be set by the API server upon the first execute API call. In that case, the client should use it for the subsequent execute API calls during the same run.
`status`	`enum[str]`	One of `"continued"`, `"waiting-input"`, `"finished"`, or `"build-finished"`. See more details at *Code Execution Model*.
`exitCode`	`int | null`	The exit code of the last process. This field has a valid value only when the `status` is `"finished"` or `"build-finished"`. Otherwise it is set to `null`. For batch-mode kernels and query-mode kernels *without* global context support, `exitCode` is the return code of the last executed child process in the kernel. In the execution step of a batch mode run, this is always 127 (a UNIX shell common practice for "command not found") when the build step has failed. For query-mode kernels with global context support, this value is always zero, regardless of whether the user code has caused an exception or not. A negative value (which cannot happen with normal process termination) indicates a Backend.AI-side error.
`console`	`list[ tuple[ enum[str], * ] ]`	Contains a list of console output items. Each item is a pair of the item type (`enum[str]`) and its value (`*`). See more details at *Handling Console Output*.
`options`	`object`	An object containing extra display options. If there is no options indicated by the kernel, this field is `null`. When `result.status` is `"waiting-input"`, it has a boolean field `is_password` so that you could use different types of text boxes for user inputs.

2.9.7 Kernel Session Item Object

Key	Type	Description
`id`	`slug`	The kernel session ID.
`type`	`str`	The kernel type (typically the name of runtime or programming lanauge).
`status`	`enum[str]`	One of `"preparing"`, `"building”`, `"running"`, `"restarting"`, `"resizing"`, `"success"`, `"error"`, `"terminating"`, `"suspended"`.
`statusInfo`	`str`	An optional message related to the current status. (e.g., error information)
`age`	`int` (msec)	The time elapsed since the kernel has started.
`execTime`	`int` (msec)	The time taken for execution. Excludes the time taken for being suspended, restarting, and resizing.
`numQueriesExecuted`	`int`	The total number of queries executed after start-up.
`memoryUsed`	`int` (MiB)	The amount of memory currently used (sum of all resident-set size across instances). It may show a stale value.
`cpuUtil`	`int` (%)	The current CPU utilization (sum of all used cores across instances, hence may exceed 100%). It may show a stale value. Changed in version v3.20170615: This had been separated into multiple credit-based fields, but that was never implemented properly. We has changed it to represent more intuitive value.
`config`	`object`	*Creation Config Object* specified when created.

2.9.8 Creation Config Object

Key	Type	Description
`environ`	`object`	A dictionary object specifying additional environment variables. The values must be strings.
`mounts`	`list[str]`	An optional list of the name of virtual folders that belongs to the current API key. These virtual folders are mounted under `/home/work`. For example, if the virtual folder name is `abc`, you can access it on `/home/work/abc`. If the name contains a colon in the middle, the second part of the string indicates the alias location in the kernel’s file system which is relative to `/home/work`. You may mount up to 5 folders for each kernel session.
`clusterSize`	`int`	The number of instances bundled for this session.
`instanceMemory`	`int` (MiB)	The maximum memory allowed per instance. The value is capped by the per-kernel image limit. Additional charges may apply on the public API service.
`instanceCores`	`int`	The number of CPU cores. The value is capped by the per-kernel image limit. Additional charges may apply on the public API service.
`instanceGPUs`	`float`	The fraction of GPU devices (1.0 means a whole device). The value is capped by the per-kernel image limit. Additional charges may apply on the public API service.

2.9.9 Virtual Folder Item Object

Key	Type	Description
`name`	`str`	The human readable name set when created.
`id`	`slug`	The unique ID of the folder. Use this when making API requests referring this folder.
`linked`	`bool`	Indicates if this folder is linked to an external service. (enterprise edition only)
`usedSize`	`int` (MiB)	The sum of the size of files in this folder.
`numFiles`	`int`	The number of files in this folder.
`maxSize`	`int` (MiB)	The maximum size of this folder.
`created`	`datetime`	The date and time when the folder is created.

2.10 Introduction

Backend.AI User API is for running instant compute sessions at scale in clouds or on-premise clusters.

2.10.1 Code Execution Model

The core of the user API is the **execute** call which allows clients to execute user-provided codes in isolated **compute sessions** (aka **kernels**). Each session is managed by a **kernel runtime**, whose implementation is language-specific. A runtime is often a containerized daemon that interacts with the Backend.AI agent via our internal ZeroMQ protocol. In some cases, kernel runtimes may be just proxies to other code execution services instead of actual executor daemons.

Inside each compute session, a client may perform multiple **runs**. Each run is for executing different code snippets (**the query mode**) or different sets of source files (**the batch mode**). The client often has to call the **execute** API *multiple times* to finish a single run. It is completely legal to mix query-mode runs and batch-mode runs inside the same session, given that the kernel runtime supports both modes.

To distinguish different runs which may be overlapped, the client must provide the same **run ID** to all **execute** calls during a single run. The run ID should be unique for each run and can be an arbitrary random string. If the run ID is not provided by the client at the first execute call of a run, the API server will assign a random one and inform it to the client via the first response. Normally, if two or more runs are overlapped, they are processed in a FIFO order using an internal queue. But they may be processed in parallel if the kernel runtime supports parallel processing. Note that the API server may raise a timeout error and cancel the run if the waiting time exceeds a certain limit.

In the query mode, usually the runtime context (e.g., global variables) is preserved for next subsequent runs, but this is not guaranteed by the API itself—it's up to the kernel runtime implementation.

Fig. 2.1: The state diagram of a "run" with the **execute** API.

The **execute** API accepts 4 arguments: `mode`, `runId`, `code`, and `options` (`opts`). It returns an *Execution Result Object* encoded as JSON.

Depending on the value of `status` field in the returned *Execution Result Object*, the client must perform another subsequent **execute** call with appropriate arguments or stop. Fig. 2.1 shows all possible states and transitions between them via the `status` field value.

If `status` is `"finished"`, the client should stop.

If `status` is `"continued"`, the client should make another **execute** API call with the `code` field set to an empty string and the `mode` field set to `"continue"`. Continuation happens when the user code runs longer than a few seconds to allow the client to show its progress, or when it requires extra step to finish the run cycle.

If `status` is `"build-finished"` (this happens at the batch-mode only), the client should make the same continuation call. All outputs prior to this status return are from the build program and all future outputs are from the executed program built. Note that even when the `exitCode` value is non-zero (failed), the client must continue once again to complete the run cycle.

If `status` is `"waiting-input"`, you should make another **execute** API call with the `code` field set to the user-input text and the `mode` field set to `"input"`. This happens when the user code calls interactive `input()` functions. Until you send the user input, the current run is blocked. You may use modal dialogs or other input forms (e.g., HTML input) to retrieve user inputs. When the server receives the user input, the kernel's `input()` returns the given value. Note that each kernel runtime may provide different ways to trigger this interactive input cycle or may not provide at all.

When each call returns, the `console` field in the *Execution Result Object* have the console logs captured since the last previous call. Check out the following section for details.

2.10.2 Handling Console Output

The console output consists of a list of tuple pairs of item type and item data. The item type is one of `"stdout"`, `"stderr"`, `"media"`, `"html"`, or `"log"`.

When the item type is `"stdout"` or `"stderr"`, the item data is the standard I/O stream outputs as (non-escaped) UTF-8 string. The total length of either streams is limited to 524,288 Unicode characters per each **execute** API call; all excessive outputs are truncated. The stderr often includes language-specific tracebacks of (unhandled) exceptions or errors occurred in the user code. If the user code generates a mixture of stdout and stderr, the print ordering is preserved and each contiguous block of stdout/stderr becomes a separate item in the console output list so that the client user can reconstruct the same console output by sequentially rendering the items.

Note: The text in the stdout/stderr item may contain arbitrary terminal control sequences such as ANSI color codes and cursor/line manipulations. It is the user's job to strip out them or implement some sort of terminal emulation.

Tip: Since the console texts are *not* escaped, the client user should take care of rendering and escaping depending on the UI implementation. For example, use `<pre>` element, replace newlines with `<br>`, or apply `white-space: pre` CSS style when rendering as HTML. An easy way to do escape the text safely is to use `insertAdjacentText()` DOM API.

When the item type is `"media"`, the item data is a pair of the MIME type and the content data. If the MIME type is text-based (e.g., `"text/plain"`) or XML-based (e.g., `"image/svg+xml"`), the content is just a string that represent the content. Otherwise, the data is encoded as a data URI format (RFC 2397). You may use backend.ai-media library to handle this field in Javascript on web-browsers.

When the item type is `"html"`, the item data is a partial HTML document string, such as a table to show tabular data. If you are implementing a web-based front-end, you may use it directly to the standard DOM API, for instance, `consoleElem.insertAdjacentHTML(value, "beforeend")`.

When the item type is `"log"`, the item data is a 4-tuple of the log level, the timestamp in the ISO 8601 format, the logger name and the log message string. The log level may be one of `"debug"`, `"info"`, `"warning"`, `"error"`, or `"fatal"`. You may use different colors/formatting by the log level when printing the log message. Not every kernel runtime supports this rich logging facility.

2.11 Kernel Management

Here are the API calls to create and manage compute sessions.

2.11.1 Creating Kernel Session

- URI: `/v2/kernel/` (`/v2/kernel/create` also works for legacy)
- Method: `POST`

Creates a kernel session if there is no existing (running) kernel with the same `clientSessionToken`. If there is an existing session and it has the same `lang`, no new session is created but the API returns successfully. In this case, `config` options are *ignored* and the `created` field in the response is set `false` (otherwise it's `true`). If there is an existing session but with a different `lang`, then the API server returns an error.

Parameters

Parameter	Type	Description
`lang`	`str`	The kernel runtime type, usually in the form of the language name and its version tag connected with a colon. (e.g., `"python:latest"`)
`clientSessionToken`	`str`	Client-provided session token which can contain ASCII alphabets, numbers, and hyphens in the middle. The length must be between 4 to 64 characters inclusively. It is useful for aliasing the session with a human-friendly name. There can exist only one running session with the same token at a time, but you can reuse the same token if previous session has been terminated.
`config`	`object`	An optional *Creation Config Object* to specify extra kernel configuration.

Example:

```
{
  "lang": "python:3.6",
  "clientSessionToken": "EXAMPLE:STRING",
  "config": {
    "clusterSize": 1,
    "instanceMemory": 51240,
    "environ": {
      "MYCONFIG": "XXX",
    },
    "mounts": [
      "mydata",
      "mypkgs:.local/lib/python3.6/site-packages"
    ],
  }
}
```

Response

HTTP Status Code	Description
201 Created	The kernel is successfully created.
406 Not acceptable	The requested resource limits exceed the server's own limits.

Fields	Type	Values
`kernelId`	`slug`	The kernel ID used for later API calls.
`created`	`bool`	True if the kernel is freshly created.

Example:

```
{
  "kernelId": "TSSJT2Z4SnmQhxjWMnJljg",
  "created": true
}
```

2.11.2 Getting Kernel Information

- URI: `/v2/kernel/:id`
- Method: `GET`

Retrieves information about a kernel session. For performance reasons, the returned information may not be real-time; usually they are updated every a few seconds in the server-side.

Parameters

Parameter	Type	Description
`:id`	`slug`	The kernel ID.

Response

HTTP Status Code	Description
200 OK	The information is successfully returned.
404 Not Found	There is no such kernel.

Fields	Type	Values
`item`	`object`	*Kernel Session Item Object*.

2.11.3 Destroying Kernel Session

- URI: `/v2/kernel/:id`
- Method: `DELETE`

Terminates a kernel session.

Parameters

Parameter	Type	Description
`:id`	`slug`	The kernel ID.

Response

HTTP Status Code	Description
204 No Content	The kernel is successfully destroyed.
404 Not Found	There is no such kernel.

2.11.4 Restarting Kernel Session

- URI: `/v2/kernel/:id`
- Method: `PATCH`

Restarts a kernel session. The idle time of the kernel will be reset, but other properties such as the age and CPU credit will continue to accumulate. All global states such as global variables and modules imports are also reset.

Parameters

Parameter	Type	Description
`:id`	`slug`	The kernel ID.

Response

HTTP Status Code	Description
204 No Content	The kernel is successfully restarted.
404 Not Found	There is no such kernel.

2.12 Code Execution (Query Mode)

2.12.1 Executing Snippet

- URI: `/v2/kernel/:id`
- Method: `POST`

Executes a snippet of user code using the specified kernel session. Each execution request to a same kernel session may have side-effects to subsequent executions. For instance, setting a global variable in a request and reading the variable in another request is completely legal. It is the job of the user (or the front-end) to gaurantee the correct execution order of multiple interdependent requests. When the kernel session is terminated or restarted, all such volatile states vanish.

Parameters

Parameter	Type	Description
`:id`	`slug`	The kernel ID.
`mode`	`str`	A constant string `"query"`.
`code`	`str`	A string of user-written code. All non-ASCII data must be encoded in UTF-8 or any format acceptable by the kernel.
`runId`	`str`	A string of client-side unique identifier for this particular run. For more details about the concept of a run, see *Code Execution Model*. If not given, the API server will assign a random one in the first response and the client must use it for the same run afterwards.

Example:

```
{
  "type": "query",
  "code": "print('Hello, world!')",
  "runId": "5facbf2f2697c1b7"
}
```

Response

HTTP Status Code	Description
200 OK	The kernel has responded with the execution result. The response body contains a JSON object as described below.

Fields	Type	Values
`result`	`object`	*Execution Result Object*.

Note: Even when the user code raises exceptions, such queries are treated as successful execution. i.e., The failure of this API means that our API subsystem had errors, not the user codes.

Warning: If the user code tries to breach the system, causes crashs (e.g., segmentation fault), or runs too long (timeout), the kernel session is automatically terminated. In such cases, you will get incomplete console logs with `"finished"` status earlier than expected. Depending on situation, the `result.stderr` may also contain specific error information.

Here we demonstrate a few example returns when various Python codes are executed.

Example: Simple return.

```
print("Hello, world!")
```

```
{
  "result": {
    "runId": "5facbf2f2697c1b7",
    "status": "finished",
    "console": [
```

(continues on next page)

(continued from previous page)

```
      ["stdout", "Hello, world!\n"]
    ],
    "options": null
  }
}
```

Example: Runtime error.

```
a = 123
print('what happens now?')
a = a / 0
```

```
{
  "result": {
    "runId": "5facbf2f2697c1b7",
    "status": "finished",
    "console": [
      ["stdout", "what happens now?\n"],
      ["stderr", "Traceback (most recent call last):\n  File \"<input>\", line 3, in
↪<module>\nZeroDivisionError: division by zero"],
    ],
    "options": null
  }
}
```

Example: Multimedia output.

Media outputs are also mixed with other console outputs according to their execution order.

```
import matplotlib.pyplot as plt
a = [1,2]
b = [3,4]
print('plotting simple line graph')
plt.plot(a, b)
plt.show()
print('done')
```

```
{
  "result": {
    "runId": "5facbf2f2697c1b7",
    "status": "finished",
    "console": [
      ["stdout", "plotting simple line graph\n"],
      ["media", ["image/svg+xml", "<?xml version=\"1.0\" ..."]],
      ["stdout", "done\n"]
    ],
    "options": null
  }
}
```

Example: Continuation results.

```
import time
for i in range(5):
    print(f"Tick {i+1}")
    time.sleep(1)
print("done")
```

```
{
  "result": {
    "runId": "5facbf2f2697c1b7",
    "status": "continued",
    "console": [
      ["stdout", "Tick 1\nTick 2\n"]
    ],
    "options": null
  }
}
```

Here you should make another API query with the empty `code` field.

```
{
  "result": {
    "runId": "5facbf2f2697c1b7",
    "status": "continued",
    "console": [
      ["stdout", "Tick 3\nTick 4\n"]
    ],
    "options": null
  }
}
```

Again.

```
{
  "result": {
    "runId": "5facbf2f2697c1b7",
    "status": "finished",
    "console": [
      ["stdout", "Tick 5\ndone\n"],
    ],
    "options": null
  }
}
```

Example: User input.

```
print("What is your name?")
name = input(">> ")
print(f"Hello, {name}!")
```

```
{
  "result": {
    "runId": "5facbf2f2697c1b7",
    "status": "waiting-input",
    "console": [
      ["stdout", "What is your name?\n>> "]
    ],
    "options": {
      "is_password": false
    }
  }
}
```

You should make another API query with the `code` field filled with the user input.

```
{
  "result": {
    "runId": "5facbf2f2697c1b7",
    "status": "finished",
    "console": [
      ["stdout", "Hello, Lablup!\n"]
    ],
    "options": null
  }
}
```

2.12.2 Auto-completion

- URI: `/v2/kernel/:id/complete`
- Method: `POST`

Parameters

Parameter	Type	Description
`:id`	`slug`	The kernel ID.
`code`	`str`	A string containing the code until the current cursor position.
`options.post`	`str`	A string containing the code after the current cursor position.
`options.line`	`str`	A string containing the content of the current line.
`options.row`	`int`	An integer indicating the line number (0-based) of the cursor.
`options.col`	`int`	An integer indicating the column number (0-based) in the current line of the cursor.

Example:

```
{
  "code": "pri",
  "options": {
    "post": "\nprint(\"world\")\n",
    "line": "pri",
    "row": 0,
    "col": 3
  }
}
```

Response

HTTP Status Code	Description
200 OK	The kernel has responded with the execution result. The response body contains a JSON object as described below.

Fields	Type	Values
`result`	`list[str]`	An ordered list containing the possible auto-completion matches as strings. This may be empty if the current kernel does not implement auto-completion or no matches have been found. Selecting a match and merging it into the code text are up to the front-end implementation.

Example:

```
{
  "result": [
    "print",
    "printf"
  ]
}
```

2.12.3 Interrupt

- URI: `/v2/kernel/:id/interrupt`
- Method: `POST`

Parameters

Parameter	Type	Description
`:id`	`slug`	The kernel ID.

Response

HTTP Status Code	Description
204 No Content	Sent the interrupt signal to the kernel. Note that this does *not* guarantee the effectiveness of the interruption.

2.13 Code Exectuion and Monitoring (Streaming Mode)

The streaming mode provides a direct web-based terminal access to kernel containers.

2.13.1 Terminal Emulation

- URI: `/v2/stream/kernel/:id/pty`
- Method: GET upgraded to WebSockets

This endpoint provides a duplex continuous stream of JSON objects via the native WebSocket. Although WebSocket supports binary streams, we currently rely on TEXT messages only conveying JSON payloads to avoid quirks in typed array support in Javascript across different browsers.

Note: We do *not* provide any legacy WebSocket emulation interfaces such as socket.io or SockJS. You need to set up your own proxy if you want to support legacy browser users.

Parameters

Parameter	Type	Description
`:id`	`slug`	The kernel ID.

Client-to-Server Protocol

The endpoint accepts the following four types of input messages.

Standard input stream

All ASCII (and UTF-8) inputs must be encoded as base64 strings. The characters may include control characters as well.

```
{
  "type": "stdin",
  "chars": "<base64-encoded-raw-characters>"
}
```

Terminal resize

Set the terminal size to the given number of rows and columns. You should calculate them by yourself.

For instance, for web-browsers, you may do a simple math by measuring the width and height of a temporarily created, invisible HTML element with the (monospace) font styles same to the terminal container element that contains only a single ASCII character.

```
{
  "type": "resize",
  "rows": 25,
  "cols": 80
}
```

Ping

Use this to keep the kernel alive (preventing it from auto-terminated by idle timeouts) by sending pings periodically while the user-side browser is open.

```
{
  "type": "ping",
}
```

Restart

Use this to restart the kernel without affecting the working directory and usage counts. Useful when your foreground terminal program does not respond for whatever reasons.

```
{
  "type": "restart",
}
```

Server-to-Client Protocol

Standard output/error stream

Since the terminal is an output device, all stdout/stderr outputs are merged into a single stream as we see in real terminals. This means there is no way to distinguish stdout and stderr in the client-side, unless your kernel applies some special formatting to distinguish them (e.g., make all stderr otuputs red).

The terminal output is compatible with xterm (including 256-color support).

```
{
  "type": "out",
  "data": "<base64-encoded-raw-characters>"
}
```

Server-side errors

```
{
  "type": "error",
  "data": "<human-readable-message>"
}
```

2.13.2 Executing Snippet via WebSocket

- URI: `/v2/stream/kernel/:id/ws`
- Method: GET upgraded to WebSockets

This API function is read-only — meaning that you cannot send any data to this URI.

Warning: This API is not implemented yet.

Note: There is timeout enforced in the server-side but you may need to adjust defaults in your client-side WebSocket library.

Parameters

Parameter	Type	Description
`:id`	`slug`	The kernel ID.

Responses

Field Name	Value
`name`	The name of an event as a string. May be one of: `"terminated"`, `"restarted"`
`reason`	The reason for the event as a canonicalized string such as `"out-of-memory"`, `"bad-action"`, and `"execution-timeout"`.

Example:

```json
{
  "name": "terminated",
  "reason": "execution-timeout"
}
```

2.13.3 Rate limiting

The streaming mode uses the same rate limiting policy as other APIs use. The limitation only applies to all client-generated messages including the initial WebSocket connection handshake but except stdin type messages such as individual keystrokes in the terminal. Server-generated messages are also exempted from rate limiting.

2.13.4 Usage metrics

The streaming mode uses the same method that the query mode uses to measure the usage metrics such as the memory and CPU time used.

2.14 Code Execution (Batch Mode)

Some kernels provide the batch mode, which offers an explicit build step required for multi-module programs or compiled programming languages. In this mode, you first upload files in prior to execution.

2.14.1 Uploading files

- URI: `/v2/kernel/:id/upload`
- Method: `POST`

Parameters

Upload files to the kernel session. You may upload multiple files at once using multi-part form-data encoding in the request body (RFC 1867/2388). The uploaded files are placed under `/home/work` directory (which is the home directory for all kernels by default), and existing files are always overwritten. If the filename has a directory part, non-existing directories will be auto-created. The path may be either absolute or relative, but only sub-directories under `/home/work` is allowed to be created.

Hint: This API is for uploading frequently-changing source files in prior to batch-mode execution. All files uploaded via this API is deleted when the kernel terminates. Use *virtual folders* to store and access larger, persistent, static data and library files for your codes.

Warning: You cannot upload files to mounted virtual folders using this API directly. However, you may copy/move the generated files to virtual folders in your build script or the main program for later uses.

There are several limits on this API:

The maximum size of each file	1 MiB
The number of files per upload request	20

Response

HTTP Status Code	Description
200 OK	The kernel has responded with the execution result. The response body contains a JSON object as described below.
400 Bad Request	Returned when one of the uploaded file exeeds the size limit or there are too many files.

2.14.2 Executing with Build Step

- URI: `/v2/kernel/:id`
- Method: `POST`

Parameters

Parameter	Type	Description
`:id`	`slug`	The kernel ID.
`mode`	`enum[batch]`	A constant string `"batch"`.
`code`	`str`	Must be an empty string `""`.
`runId`	`str`	A string of client-side unique identifier for this particular run. For more details about the concept of a run, see *Code Execution Model*. If not given, the API server will assign a random one in the first response and the client must use it for the same run afterwards.
`options`	`object`	*Batch Execution Query Object*.

Example:

```
{
  "type": "batch",
  "options": "{batch-execution-query-object}",
  "runId": "af9185c5fb0eacb2"
}
```

Response

HTTP Status Code	Description
200 OK	The kernel has responded with the execution result. The response body contains a JSON object as described below.

Fields	Type	Values
`result`	`object`	*Execution Result Object*.

2.14.3 Listing Files

Once files are uploaded to the kernel session or generated during the execution of the code, there is a need to identify what files actually are in the current session. In this case, use this API to get the list of files of your compute sesison.

- URI: `/v2/kernel/:id/files`
- Method: `GET`

Parameters

Parameter	Type	Description
`:id`	`slug`	The kernel ID.
`path`	`str`	Path inside the session (default: `/home/work`).

Response

HTTP Status Code	Description
200 OK	Success.
404 Not Found	There is no such path.

Fields	Type	Values
`files`	`str`	Stringified json containing list of files.
`folder_path`	`str`	Absolute path inside kernel session.

2.14.4 Downloading Files

Download files from your compute session.

The response contents are multiparts with tarfile binaries. Post-processing, such as unpacking and save them, should be handled by the client.

- URI: `/v2/kernel/:id/download`
- Method: `GET`

Parameters

Parameter	Type	Description
`:id`	`slug`	The kernel ID.
`files`	`list` of `str`	File paths inside the virtual folder to download.

Response

HTTP Status Code	Description
200 OK	Success.

2.15 Virtual Folders

Virtual folders provide access to shared, persistent, and reused files across different kernel sessions.

You can mount virtual folders when creating new kernel sessions, and use them like a plain directory on the local filesystem. Of course, reads/writes to virtual folder contents may have degraded performance compared to the main scratch directory (usually `/home/work` in most kernels) as internally it uses a networked file system.

Note: Currently the total size of a virtual folder is limited to 1 GiB and the number of files is limited to 1,000 files during public beta, but these limits are subject to change in the future.

2.15.1 Listing Virtual Folders

Retruns the list of virtual folders created by the current keypair.

- URI: `/v2/folders`
- Method: `GET`

Parameters

Parameter	Type	Description
`paging`	`object`	*Paging Query Object*.

Response

HTTP Status Code	Description
200 OK	Success.

Fields	Type	Values
`paging`	`object`	*Paging Info Object*.
`items`	`list[object]`	A list of *Virtual Folder Item Object*.

2.15.2 Creating a virtual folder

- URI: `/v2/folders/create`
- Method: `POST`

Creates a virtual folder associated with the current API key.

Parameters

Parameter	Type	Description
`tagName`	`str`	The human-readable name of the virtual folder.

Example:

```
{
  "tagName": "My Data",
}
```

Response

HTTP Status Code	Description
201 Created	The kernel is successfully created.
400 Bad Request	The name is malformed or duplicate with your existing virtual folders.
406 Not acceptable	You have exceeded internal limits of virtual folders. (e.g., the maximum number of folders you can have.)

Fields	Type	Values
`folderId`	`slug`	The unique folder ID used for later API calls.

Example:

```
{
  "folderId": "oyU2WOYRYmjCGuKoSkiJ7H2rlN4"
}
```

2.15.3 Getting Virtual Folder Information

- URI: `/v2/folders/:id`
- Method: `GET`

Retrieves information about a virtual folder. For performance reasons, the returned information may not be real-time; usually they are updated every a few seconds in the server-side.

Parameters

Parameter	Type	Description
`:id`	`slug`	The virtual folder ID.

Response

HTTP Status Code	Description
200 OK	The information is successfully returned.
404 Not Found	There is no such folder.

Fields	Type	Values
`item`	`object`	*Virtual Folder Item Object*.

2.15.4 Deleting Virtual Folder

- URI: `/v2/folders/:id`
- Method: `DELETE`

This immediately deletes all contents of the given virtual folder and makes the folder unavailable for future mounts.

Danger: If there are running kernels that have mounted the deleted virtual folder, those kernels are likely to break!

Warning: There is NO way to get back the contents once this API is invoked.

Parameters

Parameter	Description
`:id`	The virtual folder ID.

Response

HTTP Status Code	Description
204 No Content	The folder is successfully destroyed.
404 Not Found	There is no such folder.

2.15.5 Listing Files in Virtual Folder

Returns the list of files in a virtual folder associated with current keypair.

- URI: `/v2/folders/:id/files`
- Method: `GET`

Parameters

Parameter	Type	Description
`:id`	`slug`	The virtual folder ID.
`path`	`str`	Path inside the virtual folder (default: root).

Response

HTTP Status Code	Description
200 OK	Success.
404 Not Found	There is no such path.

Fields	Type	Values
`files`	`str`	Stringified json containing list of files.
`folder_path`	`str`	Absolute path inside the virtual folder.

2.15.6 Uploading Files to Virtual Folder

Upload local files to a virtual folder associated with current keypair.

- URI: `/v2/folders/:id/upload`
- Method: `POST`

Warning: If a file with the same name already exists in the virtual folder, it will be overwritten without warning.

Parameters

Parameter	Type	Description
`:id`	`slug`	The virtual folder ID.
Request content	`list` of `aiohttp.web.FileField_`	List of file objects to upload.

Response

HTTP Status Code	Description
201 Created	Success.

2.15.7 Downloading Files from Virtual Folder

Download files from a virtual folder associated with the current keypair.

The response contents are streamed as gzipped binaries (`Content-Encoding: gzip`). Post-processing, such as reading by chunk or unpacking the binaries, should be handled by the client.

- URI: `/v2/folders/:id/download`
- Method: `GET`

Parameters

Parameter	Type	Description
`:id`	`slug`	The virtual folder ID.
`files`	`list` of `str`	File paths inside the virtual folder to download.

Response

HTTP Status Code	Description
200 OK	Success.
404 Not Found	File not found.

2.15.8 Deleting Files in Virtual Folder

This deletes files inside a virtual folder.

Warning: There is NO way to get back the files once this API is invoked.

- URI: `/v2/folders/:id/delete_files`
- Method: `DELETE`

Parameters

Parameter	Type	Description
`:id`	`slug`	The virtual folder ID.
`files`	`list` of `str`	File paths inside the virtual folder to delete.

Response

HTTP Status Code	Description
200 OK	Success.

2.16 Introduction

Backend.AI's Admin API is for developing in-house management consoles.

There are two modes of operation:

1. Full admin access: you can query all information of all users. It requires a privileged keypair.
2. Restricted owner access: you can query only your own information. The server processes your request in this mode if you use your own plain keypair.

Warning: The Admin API *only* accepts authenticated requests.

Tip: To test and debug with the Admin API easily, try the proxy mode of the official Python client. It provides an insecure (non-SSL, non-authenticated) local HTTP proxy where all the required authorization headers are attached from the client configuration. Using this you do not have to add any custom header configurations to your favorite API development tools.

2.16.1 Basics of GraphQL

The Admin API uses a single GraphQL endpoint for both queries and mutations.

```
https://api.sorna.io/v3/admin/graphql
```

For more information about GraphQL concepts and syntax, please visit the following site(s):

- GraphQL official website

HTTP Request Convention

A client must use the `POST` HTTP method. The server accepts a JSON-encoded body with an object containing two fields: `query` and `variables`, pretty much like other GraphQL server implementations.

Warning: Currently the API gateway does not support schema discovery which is often used by API development tools such as Insomnia and GraphiQL.

Field Naming Convention

We do *NOT* automatically camel-case our field names. All field names follow the underscore style, which is common in the Python world as our server-side framework uses Python.

Pagination Convention

GraphQL itself does not enforce how to pass pagination information when querying multiple objects of the same type.

We use a de-facto standard pagination convention as described below:

TODO

Custom Scalar Types

- `UUID`: A hexademically formatted (8-4-4-4-12 alphanumeric characters connected via single hyphens) UUID values represented as `String`
- `DateTime`: An ISO-8601 formatted date-time value represented as `String`

Authentication

The admin API shares the same authentication method of the user API.

Versioning

As we use GraphQL, there is no explicit versioning. You can use any version prefix in the endpoint URL, from `v1` to `vN` where `N` is the latest major API version.

2.17 KeyPair Management

2.17.1 Full Admin

Query Schema

```
type KeyPair {
  access_key: String
  secret_key: String
  is_active: Boolean
  is_admin: Boolean
  resource_policy: String
  created_at: DateTime
  last_used: DateTime
  concurrency_limit: Int
  concurrency_used: Int
  rate_limit: Int
  num_queries: Int
  vfolders: [VirtualFolder]
  compute_sessions(status: String): [ComputeSession]
}

type root {
  ...
  keypair(access_key: String): KeyPair
  keypairs(user_id: Int!, is_active: Boolean): [KeyPair]
}
```

Mutation Schema

```
input KeyPairInput {
  is_active: Boolean
  resource_policy: String
  concurrency_limit: Int
```

(continues on next page)

(continued from previous page)

```
  rate_limit: Int
}

type CreateKeyPair {
  ok: Boolean
  msg: String
  keypair: KeyPair
}

type ModifyKeyPair {
  ok: Boolean
  msg: String
}

type DeleteKeyPair {
  ok: Boolean
  msg: String
}

type root {
  ...
  create_keypair(user_id: Int!, props: KeyPairInput!): CreateKeyPair
  modify_keypair(access_key: String!, props: KeyPairInput!): ModifyKeyPair
  delete_keypair(access_key: String!): DeleteKeyPair
}
```

2.17.2 Restricted Owner Access

Query Schema

It shares the same `KeyPair` type, but you cannot use `user_id` argument in the root query because the client can only query the keypair that is being used to make this API query. Also the returned value is always a single object.

```
type root {
  ...
  keypair(): KeyPair!
}
```

Mutation Schema

There is no mutations available.

2.18 Compute Session Monitoring

2.18.1 Full Admin

Query Schema

```
type ComputeSession {
  sess_id: String
  id: UUID
  status: String
  status_info: String
  created_at: DateTime
  terminated_at: DateTime
  agent: String
  container_id: String
  mem_slot: Int
  cpu_slot: Int
  gpu_slot: Int
  num_queries: Int
  cpu_used: Int
  max_mem_bytes: Int
  cur_mem_bytes: Int
  net_rx_bytes: Int
  net_tx_bytes: Int
  io_read_bytes: Int
  io_write_bytes: Int
  lang: String
  workers(status: String): [ComputeWorker]
}

type ComputeWorker {
  sess_id: String
  id: UUID
  status: String
  status_info: String
  created_at: DateTime
  terminated_at: DateTime
  agent: String
  container_id: String
  mem_slot: Int
  cpu_slot: Int
  gpu_slot: Int
  num_queries: Int
  cpu_used: Int
  max_mem_bytes: Int
  cur_mem_bytes: Int
  net_rx_bytes: Int
  net_tx_bytes: Int
  io_read_bytes: Int
  io_write_bytes: Int
}

type root {
  ...
  compute_sessions(access_key: String, status: String): [ComputeSession]
  compute_workers(sess_id: String!, status: String): [ComputeWorker]
}
```

2.18.2 Restricted Owner Access

Query Schema

It shares the same `ComputeSession` and `ComputeWorker` type, but with a slightly different root query type:

```
type root {
  ...
  compute_sessions(status: String): [ComputeSession]
  compute_workers(sess_id: String!, status: String): [ComputeWorker]
}
```

2.19 Virtual Folder Management

2.19.1 Full Admin

Query Schema

```
type VirtualFolder {
  id: UUID
  host: String
  name: String
  max_files: Int
  max_size: Int
  created_at: DateTime
  last_used: DateTime
  num_files: Int
  cur_size: Int
}

type rootQuery {
  ...
  vfolders(access_key: String): [VirtualFolder]
}
```

2.19.2 Restricted Owner Access

Query Schema

It shares the same `VirtualFolder` type, but you cannot use `access_key` argument in the root query.

```
type root {
  ...
  vfolders(): [VirtualFolder]
}
```

2.20 Statistics

2.20.1 Full Admin

Query Schema

TODO

2.20.2 Restricted Owner Access

Query Schema

TODO

2.21 Adding New REPL Kernels

2.21.1 Architecture Overview

Inside containers, each kernel is a simple daemon process that accepts user code snippets and replies with its execution results via TCP-based ZeroMQ connections. The rationale to use ZeroMQ is: 1) it is message-based; we do not have to concern the message boundaries and encodings, 2) it automatically reconnects when the connection is lost due to network failures or packet losses, 3) it is one of the most universally supported networking library in various programming languages.

A kernel should offer the *query mode* and/or the *PTY mode*. The TCP port 2001 is reserved for the query mode whereas 2002 and 2003 are reserved for the PTY mode (stdin and stdout combined with stderr).

2.21.2 Ingredients of Kernel Images

A kernel is a Docker image with the following format:

- `Dockerfile`
  - `WORKDIR /home/work`: this path is used to mount an external directory so that the agent can access files generated by user codes.
  - `CMD` must be set to the main program.
  - Required Labels
    * `io.sorna.maxcores`: *N* (the number of CPU cores recommended for this kernel)
    * `io.sorna.maxmem`: *M* (the memory size in a human-readable bytes recommended for this kernel, `128m` (128 MBytes) for example)
    * `io.sorna.timeout`: *T* (the maximum seconds allowed to execute a single query)
    * Above limits are used as default settings by Backend.AI Agent, but the agents may enforce lower limits due to the service policy. Backend.AI Gateway may refer these information for load balancing and scheduling.
    * `io.sorna.mode`: `query`, `pty`, or `query+pty`
  - Optional Labels
    * `io.sorna.envs.corecount`: a comma-separated string of environment variable names which will be set to the number of assigned CPU cores by the agent. (e.g., `JULIA_CPU_CORES`, `OPENBLAS_NUM_THREADS`)
    * `io.sorna.nvidia.enabled`: `yes` or `no` (if yes, Backend.AI Agent attaches an NVIDIA CUDA GPU device with a driver volume. You must use nvidia-docker images as base of your Dockerfile.)

  * io.sorna.extra_volumes: a comma-separated string of extra volume mounts (volume name and path inside container separated by a colon), such as deep learning sample data sets (e.g., sample-data:/home/work/samples,extra-data:/home/work/extra). Note that we allow only read-only mounts. The available list of extra volumes depends on your Backend.AI Agent setup; there is no standard or predefined ones. If you want to add a new one, use docker volume commands. When designated volumes do not exist in the agent's host, the agent silently skips mounting them.
  * io.sorna.features: a comma-separated string keywords indicating available features of this kernel.

Keyword	Feature
media.images	Generates images (PNG, JPG, and SVG) without uploading into AWS S3.
media.svgplot	Generates plots in SVG.
media.drawing	Generates animated vector graphics which can be rendered by sorna-media Javascript library
media.audio	Generates audio signal streams. (not implemented)

- The main program that implements the query mode and/or the PTY mode (see below).
  - We strongly recommend to create a normal user instead of using root for the main program.
  - The main program should be wrapped with jail, like:

```
#! /bin/bash
exec /home/sorna/jail default `which lua` /home/sorna/run.lua
```

    The first argument to jail is the policy name and the second and laters are the absolute path of the main program with its arguments. To customize the jail policy, *see below*.

  - jail and intra-jail must be copied into the kernel image.
- Other auxilliary files used in Dockerfile or the main program. (e.g., Python and package installation scripts)

2.21.3 Writing Query Mode Kernels

Most kernels fall into this category. You just write a simple blocking loop that receives a input code message and send a output result message via a ZeroMQ REP socket listening on port 2001. All complicated stuffs such as multiplexing multiple user requests and container management is done by Backend.AI Agent.

The input is a ZeroMQ's multipart message with two payloads. The first payload should contain a unique identifier for the code snippet (usually a hash of it), but currently it is ignored (reserved for future caching implementations). The second payload should contain a UTF-8 encoded source code string.

The reply is a ZeroMQ's multipart message with a single payload, containing a UTF-8 encoded string of the following JSON object:

```
{
    "stdout": "hello world!",
    "stderr": "oops!",
    "exceptions": [
        ["exception-name", ["arg1", "arg2"], false, null]
    ],
    "media": [
        ["image/png", "data:image/base64,...."]
```

(continues on next page)

(continued from previous page)

```
    ],
    "options": {
        "upload_output_files": true
    }
}
```

Each item in `exceptions` is an array composed of four items: exception name, exception arguments (optional), a boolean indicating if the exception is raised outside the user code (mostly false), and a traceback string (optional).

Each item in `media` is an array of two items: MIME-type and the data string. Specific formats are defined and handled by the Backend.AI Media module.

The `options` field may present optionally. If `upload_output_files` is true (default), then the agent uploads the files generated by user code in the working directory (`/home/work`) to AWS S3 bucket and make their URLs available in the front-end.

2.21.4 Writing PTY Mode Kernels

If you want to allow users to have real-time interactions with your kernel using web-based terminals, you should implement the PTY mode as well. A good example is our “git” kernel runner.

The key concept is separation of the “outer” daemon and the “inner” target program (e.g., a shell). The outer daemon should wrap the inner program inside a pseudo-tty. As the outer daemon is completely hidden in terminal interaction by the end-users, the programming language may differ from the inner program. The challenge is that you need to implement piping of ZeroMQ sockets from/to pseudo-tty file descriptors. It is up to you how you implement the outer daemon, but if you choose Python for it, we recommend to use asyncio or similar event loop libraries such as tornado and Twisted to mulitplex sockets and file descriptors for both input/output directions. When piping the messages, the outer daemon should not apply any specific transformation; it should send and receive all raw data/control byte sequences transparently because the front-end (e.g., terminal.js) is responsible for interpreting them. Currently we use PUB/SUB ZeroMQ socket types but this may change later.

Optionally, you may run the query-mode loop side-by-side. For example, our git kernel supports terminal resizing and pinging commands as the query-mode inputs. There is no fixed specification for such commands yet, but the current CodeOnWeb uses the followings:

- `%resize <rows> <cols>`: resize the pseudo-tty’s terminal to fit with the web terminal element in user browsers.
- `%ping`: just a no-op command to prevent kernel idle timeouts while the web terminal is open in user browsers.

A best practice (not mandatory but recommended) for PTY mode kernels is to automatically respawn the inner program if it terminates (e.g., the user has exited the shell) so that the users are not locked in a “blank screen” terminal.

2.21.5 Writing Custom Jail Policies

Implement the jail policy interface in Go and ebmed it inside your jail build. Please give a look to existing jail policies as good references.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search