Backend.AI API Documentation

Release 1.0

Lablup Inc.

Mar 02, 2020

CONCEPTS

Latest API version: v4.20190315

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

It hosts 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

ONE

TABLE OF CONTENTS

1.1 Key Concepts

Here we describe the key concepts that are required to understand and follow this documentation.

Fig. 1.1: The diagram of a typical multi-node Backend.AI server architecture

Fig. 1.1 shows a brief Backend.AI server-side architecture where the components are what you need to install and configure.

Each border-connected group of components is intended to be run on the same server, but you may split them into multiple servers or merge different groups into a single server as you need. For example, you can run separate servers for the nginx reverse-proxy and the Backend.AI manager or run both on a single server. In the [[development setup]], all these components run on a single PC such as your laptop.

1.1.1 Manager and Agents

Backend.AI manager is the central governor of the cluster. It accepts user requests, creates/destroys the kernels, and routes code execution requests to appropriate agents and kernels. It also collects the output of kernels and responds the users with them.

Backend.AI agent is a small daemon installed onto individual worker servers to control them. It manages and monitors the lifecycle of kernel containers, and also mediates the input/output of kernels. Each agent also reports the resource capacity and status of its server, so that the manager can assign new kernels on idle servers to load balance.

1.1.2 Kernels (aka Compute Sessions)

Backend.AI spawns compute sessions as the form of containers upon user API requests. Each compute session may have one or more containers (distributed across different nodes), and we call those member containers "kernels". Such multi-container sessions are for distributed and parallel computation at large scales. The agent automatically pulls and updates the kernel images if needed.

1.1.3 Cluster Networking

The primary networking requirements are:

- The manager server (the HTTPS 443 port) should be exposed to the public Internet or the network that your client can access.
- The manager, agents, and all other database/storage servers should reside at the same local private network where any traffic between them are transparently allowed.
- For high-volume big-data processing, you may want to separate the network for the storage using a secondary network interface on each server, such as Infiniband and RoCE adaptors.

1.1.4 Databases

Redis and PostgreSQL are used to keep track of liveness of agents and compute sessions (which may be composed of one or more kernels). They also store user metadata such as keypairs and resource usage statistics.

1.1.5 Configuration Management

Most cluster-level configurations are stored in an etcd server or cluster. The etcd server is also used for service discovery; when new agents boot up they register themselves to the cluster manager via etcd. For production deployments, we recommend to use an etcd cluster composed of odd (3 or higher) number of nodes to keep high availability.

1.1.6 Virtual Folders

Fig. 1.2: A conceptual diagram of virtual folders when using two NFS servers as vfolder hosts

As shown in Fig. 1.2, Backend.AI abstracts network storages as "virtual folder", which provides a cloud-like private file storage to individual users. The users may create their own (one or more) virtual folders to store data files, libraries, and program codes. Each vfolder (virtual folder) is created under a designated storage mount (called "vfolder hosts"). Virtual folders are mounted into compute session containers at `/home/work/{name}` so that user programs have access to the virtual folder contents like a local directory. As of Backend.AI v18.12, users may also share their own virtual folders with other users in differentiated permissions such as read-only and read-write.

A Backend.AI cluster setup may use any filesystem that provides a local mount point at each node (including the manager and agents) given that the filesystem contents are synchronized across all nodes. The only requirement is that the local mount-point must be same across all cluster nodes (e.g., `/mnt/vfroot/mynfs`). Common setups may use a centralized network storage (served via NFS or SMB), but for more scalability, one might want to use distributed file systems such as CephFS and GlusterFS, or Alluxio that provides fast in-memory cache while backed by another storage server/service such as AWS S3.

For a single-node setup, you may simply use an empty local directory.

1.2 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.

1.2.1 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.

1.2.2 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.

1.3 FAQ

vs. Notebooks

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

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, Azure Functions	Stateless, light-weight functions	Serverless, zero-management

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.)

1.4 Quickstart Guides

1.4.1 Install from Source

This is the recommended way to install on most setups, for both development and production.

Note: For production deployments, we also recommend pinning specific releases when cloning or updating source repositories.

Setting Up Manager and Agent (single node)

Prerequisites

For a standard installation:

- Ubuntu 16.04+ / CentOS 7.4+ / macOS 10.12+
  - For Linux: `sudo` with access to the package manager (`apt-get` or `yum`)
  - For macOS: homebrew with the latest Xcode Command Line tools.
- bash
- git

To enable CUDA (only supported in Ubuntu or CentOS):

- CUDA 9.0 or later (with compatible NVIDIA driver)
- nvidia-docker 1.0 or 2.0

Running the Installer

Clone the meta repository first. For the best result, clone the branch of this repo that matches with the target server branch you want to install. Inside the cloned working copy, `scripts/install-dev.sh` is the automatic single-node installation script.

It provides the following options (check with `--help`):

- `--python-version`: The Python version to install.
- `--install-path`: The target directory where individual Backend.AI components are installed together as subdirectories.
- `--server-branch`: The branch/tag used for the manager, agent, and common components.
- `--client-branch`: The branch/tag used for the client-py component.
- `--enable-cuda`: If specified, the installer will install the open-source version of CUDA plugin for the agent.
- `--cuda-branch`: The branch/tag used for the CUDA plugin.

With default options, the script will install a source-based single-node Backend.AI cluster as follows:

- The installer tries to install pyenv, the designated Python version, docker-compose, and a few libraries (e.g., libsnappy) automatically after checking their availability. If it encounters an error during installation, it will show manual instructions and stop.
- It creates a set of Docker containers for Redis 5, PostgreSQL 9.6, and etcd 3.3 via docker-compose, with the default credentials: The Redis and etcd is configured without authentication and PostgreSQL uses `postgres` / `develove`. We call these containers as "halfstack".
- `./backend.ai-dev/{component}` where components are manager, agent, common, client, and a few others, using separate virtualenvs. They are all installed as "editable" so modifying the cloned sources takes effects immediately.
- For convenience, when `cd`-ing into individual component directories, pyenv will activate the virtualenv automatically for supported shells. This is configured via `pyenv local` command during installation.
- The default vfolder mount point is `./backend.ai/vfolder` and the default vfolder host is `local`.
- The installer automatically populates the example fixtures (in the `sample-configs` directory of the manager repository) for during the database initialization.
- It automatically updates the list of available Backend.AI kernel images from the public Docker Hub. It also pulls a few frequently used images such as the base Python image.
- The manager and agent are *NOT* daemonized. You must run them by running `scripts/run-with-halfstack.sh python -m ...` inside each component's source clones. Those wrapper scripts configure environment variables suitable for the default halfstack containers.

Verifying the Installation

Run the manager and agent as follows in their respective component directories:

- manager:

```
$ cd backend.ai-dev/manager
$ scripts/run-with-halfstack.sh python -m ai.backend.gateway.server
```

  By default, it listens on the localhost's 8080 port using the plain-text HTTP.

- agent:

```
$ cd backend.ai-dev/agent
$ scripts/run-with-halfstack.sh python -m ai.backend.agent.server \
      --scratch-root=$(pwd)/scratches
```

Note: The manager and agent may be executed without the root privilege on both Linux and macOS. In Linux, the installer sets extra capability bits to the Python executable so that the agent can manage cgroups and access the Docker daemon.

If all is well, they will say "started" or "serving at ...". You can also check their CLI options using `--help` option to change service IP and ports or enable the debug mode.

To run a "hello world" example, you first need to configure the client using the following script:

```
# env-local-admin.sh
export BACKEND_ENDPOINT=http://127.0.0.1:8080
export BACKEND_ACCESS_KEY=AKIAIOSFODNN7EXAMPLE
export BACKEND_SECRET_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

And then run the following inside the client directory. If you see similar console logs, your installation is now working:

```
$ cd backend.ai-dev/client-py
$ source env-local-admin.sh
$ backend.ai run --rm -c 'print("hello world")' python:3.6-ubuntu18.04
 Session token prefix: fb05c73953
✓ [0] Session fb05c73953 is ready.
hello world
✓ [0] Execution finished. (exit code = 0)
✓ [0] Cleaned up the session.
```

Setting Up Additional Agents (multi-node)

Updating Manager Configuration for Multi-Nodes

Verifying the Installation

1.4.2 Install from Package (Enterprise Edition)

This is for enterprise customers who need self-contained prebuilt packages for private clusters.

Prerequisites

For a standard installation:

- Ubuntu 16.04+ / CentOS 7.4+
- sudo
- bash
- git

To enable CUDA:

- CUDA 9.0 or later (with compatible NVIDIA driver)

- nvidia-docker 1.0 or 2.0

Running the Installer

Verifying the Installation

1.5 Supplementary Guides

1.5.1 Install Docker

For platform-specific instructions, please consult the docker official documentation.

Alternative way of docker installation on Linux (Ubuntu, CentOS, . . .)

```
$ curl -fsSL https://get.docker.io | sh
```

type your password to install docker.

Run docker commands without sudo (required)

By default, you need sudo to execute docker commands. To do so without sudo, add yourself to the system `docker` group.

```
$ sudo usermod -aG docker $USER
```

It will work after restarting your login session.

Install docker-compose (only for development/single-server setup)

You need to install docker-compose separately. Check out the official documentation.

Install nvidia-docker (only for GPU-enabled agents)

Check out the official repository for instructions.

On the clouds, we highly recommend using vendor-provided GPU-optimized instance types (e.g., p2/p3 series on AWS) and GPU-optimized virtual machine images which include ready-to-use CUDA drivers and configurations.

Since Backend.AI's kernel container images ship all the necessary libraries and 3rd-party computation packages, you may choose the light-weight "base" image (e.g., Amazon Deep Learning *Base* AMI) instead of full-featured images (e.g., Amazon Deep Learning Conda AMI).

1.5.2 Manually install CUDA at on-premise GPU servers

Please search for this topic on the Internet, as Linux distributions often provide their own driver packages and optimized method to install CUDA.

To download the driver and CUDA toolkit directly from NVIDIA, visit here.

1.5.3 Let Backend.AI to utilize GPUs

If an agent server has properly configured nvidia-docker (ref: [[Install Docker]]) with working host-side drivers and the agent's Docker daemon has GPU-enabled kernel images, there is *nothing* to do special. Backend.AI tracks the GPU capacity just like CPU cores and RAM, and uses that information to schedule and assign GPU-enabled kernels.

We highly recommend pyenv to install multiple Python versions side-by-side, which does not interfere with system-default Pythons.

1.5.4 Install dependencies for building Python

Ubuntu

```
$ sudo apt-get update -y
$ sudo apt-get dist-upgrade -y
$ sudo apt-get install -y \
> build-essential git-core                                    # for generic C/C++␣
↪builds
> libreadline-dev libsqlite3-dev libssl-dev libbz2-dev tk-dev  # for Python builds
> libzmq3-dev libsnappy-dev                                     # for Backend.AI␣
↪dependency builds
```

CentOS / RHEL

(TODO)

1.5.5 Install pyenv

NOTE: Change `~/.profile` accroding to your shell/system (e.g., `~/.bashrc`, `~/.bash_profile`, `~/.zshrc`, ...) – whichever loaded at startup of your shell!

```
$ git clone https://github.com/pyenv/pyenv.git ~/.pyenv
...
$ echo 'export PYENV_ROOT="$HOME/.pyenv"' >> ~/.profile
$ echo 'export PATH="$PYENV_ROOT/bin:$PATH"' >> ~/.profile
$ echo 'eval "$(pyenv init -)"' >> ~/.profile
$ exec $SHELL -l
$ pyenv  # check installation
pyenv 1.2.0-6-g9619e6b
Usage: pyenv <command> [<args>]
```

(continues on next page)

(continued from previous page)

```
Some useful pyenv commands are:
   ...
```

1.5.6 Install pyenv's virtualenv plugin

```
$ git clone https://github.com/pyenv/pyenv-virtualenv.git ~/.pyenv/plugins/pyenv-
↪virtualenv
...
$ echo 'eval "$(pyenv virtualenv-init -)"' >> ~/.profile
$ exec $SHELL -l
$ pyenv virtualenv  # check installation
pyenv-virtualenv: no virtualenv name given.
```

1.5.7 Install Python via pyenv

Install Python 3.6 latest version.

Warning: Currently Python 3.7 is not supported yet.

```
$ pyenv install 3.6.6
```

1.5.8 Create a virtualenv using a specific Python version

Change `myvenv` to specific names required in other guide pages.

```
$ pyenv virtualenv 3.6.6 myvenv
```

1.5.9 Activate the virtualenv for the current shell

```
$ pyenv shell myvenv
```

1.5.10 Activate the virtualenv when your shell goes into a directory

```
$ cd some-directory
$ pyenv local myvenv
```

Note: *pyenv local* creates a hidden *.python-version* file at each directory specifying the Python version/virtualenv recongnized by pyenv. Any pyenv-enabled shells will automagically activate/deactivate this version/virtualenv when going in/out such directories.

1.5.11 Install monitoring and logging tools

The Backend.AI can use several 3rd-party monitoring and logging services. Using them is completely optional.

Guide variables

Prepare the values of the following variables before working with this page and replace their occurrences with the values when you follow the guide.

Name	Description
`{DDAPIKEY}`	>The Datadog API key
`{DDAPPKEY}`	The Datadog application key
`{SENTRYURL}`	The private Sentry report URL

Install Datadog agent

Datadog is a 3rd-party service to monitor the server resource usage.

```
$ DD_API_KEY={DDAPIKEY} bash -c "$(curl -L https://raw.githubusercontent.com/DataDog/
↪dd-agent/master/packaging/datadog-agent/source/install_agent.sh)"
```

Install Raven (Sentry client)

Raven is the official client package name of Sentry, which reports detailed contextual information such as stack and package versions when an unhandled exception occurs.

```
$ pip install "raven>=6.1"
```

1.5.12 Prepare Database for Manager

Guide variables

Prepare the values of the following variables before working with this page and replace their occurrences with the values when you follow the guide.

Name	Description
`{NS}`	The etcd namespace
`{ETCDADDR}`	The etcd cluster address (`{ETCDHOST}:{ETCDPORT}`, `localhost:8120` for development setup)
`{DBADDR}`	The PostgreSQL server address (`{DBHOST}:{DBPORT}`, `localhost:8100` for development setup)
`{DBUSER}`	The database username (e.g., `postgres` for development setup)
`{DBPASS}`	The database password (e.g., `develove` for development setup)
`{STRGMOUNT}`	The path to a directory that the manager and all agents share together (e.g., a network-shared storage mountpoint). Note that the path must be same across all the nodes that run the manager and agents. Development setup: Use an arbitrary empty directory where Docker containers can also mount as volumes — e.g., Docker for Mac requires explicit configuration for mountable parent folders.

Load initial etcd data


```
$ cd backend.ai-manager
```

Copy `sample-configs/image-metadata.yml` and `sample-configs/image-aliases.yml` and edit according to your setup.

```
$ cp sample-configs/image-metadata.yml image-metadata.yml
$ cp sample-configs/image-aliases.yml image-aliases.yml
```

By default you can pull the images listed in the sample via `docker pull lablup/kernel-xxxx:tag`(e.g. `docker pull lablup/kernel-python-tensorflow:latest` for the latest tensorflow) as they are hosted on the public Docker registry.

Load image registry metadata

(Instead of manually specifying environment variables, you may use `scripts/run-with-halfstack.sh` script in a development setup.)

```
$ BACKEND_NAMESPACE={NS} BACKEND_ETCD_ADDR={ETCDADDR} \
> python -m ai.backend.manager.cli etcd update-images \
>        -f image-metadata.yml
```

Load image aliases

```
$ BACKEND_NAMESPACE={NS} BACKEND_ETCD_ADDR={ETCDADDR} \
> python -m ai.backend.manager.cli etcd update-aliases \
>        -f image-aliases.yml
```

Set the default storage mount for virtual folders

```
$ BACKEND_NAMESPACE={NS} BACKEND_ETCD_ADDR={ETCDADDR} \
> python -m ai.backend.manager.cli etcd put \
>        volumes/_mount {STRGMOUNT}
```

Database Setup

Create a new database

In docker-compose based configurations, you may skip this step.

```
$ psql -h {DBHOST} -p {DBPORT} -U {DBUSER}
```

```
postgres=# CREATE DATABASE backend;
postgres=# \q
```

Install database schema

Backend.AI uses alembic to manage database schema and its migration during version upgrades. First, localize the sample config:

```
$ cp alembic.ini.sample alembic.ini
```

Modify the line where `sqlalchemy.url` is set. You may use the following shell command: (ensure that special characters in your password are properly escaped)

```
$ sed -i'' -e 's!^sqlalchemy.url = .*$!sqlalchemy.url = postgresql://{DBUSER}:{DBPASS}
↪@{DBHOST}:{DBPORT}/backend!' alembic.ini
```

```
$ python -m ai.backend.manager.cli schema oneshot head
```

example execution result

```
201x-xx-xx xx:xx:xx INFO alembic.runtime.migration [MainProcess] Context impl␣
↪PostgresqlImpl.
201x-xx-xx xx:xx:xx INFO alembic.runtime.migration [MainProcess] Will assume␣
↪transactional DDL.
201x-xx-xx xx:xx:xx INFO ai.backend.manager.cli.dbschema [MainProcess] Detected a␣
↪fresh new database.
201x-xx-xx xx:xx:xx INFO ai.backend.manager.cli.dbschema [MainProcess] Creating␣
↪tables...
201x-xx-xx xx:xx:xx INFO ai.backend.manager.cli.dbschema [MainProcess] Stamping␣
↪alembic version to head...
INFO  [alembic.runtime.migration] Context impl PostgresqlImpl.
INFO  [alembic.runtime.migration] Will assume transactional DDL.
INFO  [alembic.runtime.migration] Running stamp_revision  -> f9971fbb34d9
```

NOTE: All sub-commands under "schema" uses alembic.ini to establish database connections.

Load initial fixtures

Edit `ai/backend/manager/models/fixtures.py` so that you have a randomized admin keypair.

Then pour it to the database:

```
$ python -m ai.backend.manager.cli \
>   --db-addr={DBHOST}:{DBPORT} \
>   --db-user={DBUSER} \
>   --db-password={DBPASS}
>   --db-name=backend \
>   fixture populate example_keypair
```

example execution result

```
201x-xx-xx xx:xx:xx INFO ai.backend.manager.cli.fixture [MainProcess] populating␣
↪fixture 'example_keypair'
```

1.6 Client SDK Libraries and Tools

We provide official client SDKs for popular programming languages that abstract the low-level REST/GraphQL APIs via functional and object-oriented interfaces.

1.6.1 Python

Python is the most extensively supported client programming language. The SDK also includes the official command-line interface.

- Documentation for Backend.AI Client SDK for Python
- Source repository for Backend.AI Client SDK for Python

1.6.2 Javascript

The Javascript SDK is for writing client apps on both NodeJS and web browsers. It is also used for our Atom/VSCode plugins.

- Documentation for Backend.AI Client SDK for Javascript (under construction)
- Source repository for Backend.AI Client SDK for Javascript

1.6.3 Java

The Java SDK is used for implementing our IntelliJ/PyCharm plugins.

- Documentation for Backend.AI Client SDK for Java (under construction)
- Source repository for Backend.AI Client SDK for Java

1.6.4 PHP

- Documentation for Backend.AI Client SDK for PHP (under construction)
- Source repository for Backend.AI Client SDK for PHP (under construction)

1.7 API and Document Conventions

1.7.1 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.

1.7.2 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.

1.7.3 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.

1.7.4 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}]}`)

1.7.5 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.

1.7.6 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.

Changed in version v3.20170615: Version prefix in API queries are deprecated. (Yet still supported currently) For example, now users should call `/kernel/create` rather than `/v2/kernel/create`.

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"
}
```

1.8 Authentication

1.8.1 Access Tokens and Secret Key

To make requests to the API server, a client needs to have a pair of an API access key and a secret key. You may get one from our cloud service or from the administrator of your Backend.AI cluster.

The server uses the API keys 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).

1.8.2 Common Structure of API Requests

HTTP Headers	Values
Method	`GET` / `REPORT` / `POST` / `PUT` / `PATCH` / `DELETE`
Query String	If your access key has the administrator privilege, your client may optionally specify other user's access key as the `owner_access_key` parameter of the URL query string (in addition to other API-specific ones if any) to change the scope of access key applied to access and manipulation of keypair-specific resources such as kernels and vfolders. New in version v4.20190315.
`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

1.8.3 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

1.8.4 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()
  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
```

(continues on next page)

(continued from previous page)

```
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>
```

1.8.5 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."
}
```

1.9 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.

1.10 JSON Object References

1.10.1 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.

1.10.2 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.

1.10.3 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.

1.10.4 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"`)

1.10.5 Service Port Object

Key	Type	Description
`name`	`slug`	The name of service provided by the container. See also: *Terminal Emulation*
`protocol`	`str`	The type of network protocol used by the container service.

1.10.6 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).
`clean`	`str`	The bash command to clean the intermediate files produced during the build phase. The clean step comes *before* the build step if specified so that the build step can (re)start fresh. If the field is not present, an empty string, or `null`, it skips the clean step. Unlike the build and exec command, the default for `"*"` is do-nothing to prevent deletion of other files unrelated to the build by bugs or mistakes.

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"
}
```

1.10.7 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"`, `"clean-finished"`, `"build-finished"`, or `"exec-timeout"`. 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"`, `"clean-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[object]`	A list of *Console Item Object*.
`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.
`files`	`list[object]`	A list of *Execution Result File Object* that represents files generated in `/home/work/.output` directory of the container during the code execution .

1.10.8 Console Item Object

Key	Type	Description
(root)	`[enum, *]`	A tuple of the item type and the item content. The type may be `"stdout"`, `"stderr"`, and others. See more details at *Handling Console Output*.

1.10.9 Execution Result File Object

Key	Type	Description
`name`	`str`	The name of a created file after execution.
`url`	`str`	The URL of a create file uploaded to AWS S3.

1.10.10 Container Stats Object

Key	Type	Description
`cpu_used`	`int` (msec)	The total time the kernel was running.
`mem_max_bytes`	`int` (Byte)	The maximum memory usage.
`mem_cur_bytes`	`int` (Byte)	The current memory usage.
`net_rx_bytes`	`int` (Byte)	The total amount of received data through network.
`net_tx_bytes`	`int` (Byte)	The total amount of transmitted data through network.
`io_read_bytes`	`int` (Byte)	The total amount of received data from IO.
`io_write_bytes`	`int` (Byte)	The total amount of transmitted data to IO.
`io_max_scratch_size`	`int` (Byte)	Currently not used field.
`io_write_bytes`	`int` (Byte)	Currently not used field.

1.10.11 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.
`resources`	*ResourceSlot Object*	The resource slot specification for each container in this session. New in version v4.20190315.
`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. Deprecated since version v4.20190315.
`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. Deprecated since version v4.20190315.
`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. Deprecated since version v4.20190315.

1.10.12 Resource Slot Object

Key	Type	Description
`cpu`	`str` | `int`	The number of CPU cores.
`mem`	`str` | `int`	The amount of main memory in bytes. When the slot object is used as an input to an API, it may be represented as binary numbers using the binary scale suffixes such as *k*, *m*, *g*, *t*, *p*, *e*, *z*, and *y*, e.g., "512m", "512M", "512MiB", "64g", "64G", "64GiB", etc. When the slot object is used as an output of an API, this field is **always** represented in the unscaled number of bytes as strings. **Warning:** When parsing this field as JSON, you must check whether your JSON library or the programming language supports large integers. For instance, most modern Javascript engines support up to $2^{53} - 1$ (8 PiB – 1) which is often defined as the `Number.MAX_SAFE_INTEGER` constant. Otherwise you need to use a third-party big number calculation library. To prevent unexpected side-effects, Backend.AI always returns this field as a string.
`cuda.device`	`str` | `int`	The number of CUDA devices. Only available when the server is configured to use the CUDA agent plugin.
`cuda.shares`	`str`	The virtual share of CUDA devices represented as fractional decimals. Only available when the server is configured to use the CUDA agent plugin with the fractional allocation mode (enterprise edition only).
`tpu.device`	`str` | `int`	The number of TPU devices. Only available when the server is configured to use the TPU agent plugin (cloud edition only).
(others)	`str`	More resource slot types may be available depending on the server configuration and agent plugins. There are two types for an arbitrary slot: "count" (the default) and "bytes". For "count" slots, you may put arbitrary positive real number there, but fractions may be truncated depending on the plugin implementation. For "bytes" slots, its interpretation and representation follows that of the `mem` field.

1.10.13 Resource Preset Object

Key	Type	Description
`name`	`str`	The name of this preset.
`resource_slots`	*Resource Slot Object*	The pre-configured combination of resource slots. If it contains slot types that are not currently used/activated in the cluster, they will be removed when returned via `/resource/*` REST APIs.

1.10.14 Virtual Folder Creation Result Object

Key	Type	Description
`id`	UUID	An internally used unique identifier of the created vfolder. Currently it has no use in the client-side.
`name`	str	The name of created vfolder, as the client has given.
`host`	str	The host name where the vfolder is created.
`user`	UUID	The user who has the ownership of this vfolder.
`group`	UUID	The group who is the owner of this vfolder.

New in version v4.20190615: `user` and `group` fields.

1.10.15 Virtual Folder List Item Object

Key	Type	Description
`name`	str	The human readable name set when created.
`id`	slug	The unique ID of the folder.
`host`	str	The host name where this folder is located.
`is_owner`	bool	True if the client user is the owner of this folder. False if the folder is shared from a group or another user.
`permission`	enum	The requested user's permission for this folder. (One of "ro", "rw", and "wd" which represents read-only, read-write, and write-delete respectively. Currently "rw" and "wd" has no difference.)
`user`	UUID	The user ID if the owner of this item is a user vfolder. Otherwise, `null`.
`group`	UUID	The group ID if the owner of this item is a group vfolder. Otherwise, `null`.
`type`	enum	The owner type of vfolder. One of "user" or "group".

New in version v4.20190615: `user`, `group`, and `type` fields.

1.10.16 Virtual Folder Item Object

Key	Type	Description
`name`	str	The human readable name set when created.
`id`	UUID	The unique ID of the folder.
`host`	str	The host name where this folder is located.
`is_owner`	bool	True if the client user is the owner of this folder. False if the folder is shared from a group or another user.
`num_files`	int	The number of files in this folder.
`permission`	enum	The requested user's permission for this folder.
`created_at`	datetime	The date and time when the folder is created.
`last_used`	datetime	The date adn time when the folder is last used.
`user`	UUID	The user ID if the owner of this item is a user. Otherwise, `null`.
`group`	UUID	The group ID if the owner of this item is a group. Otherwise, `null`.
`type`	enum	The owner type of vfolder. One of "user" or "group".

New in version v4.20190615: `user`, `group`, and `type` fields.

1.10.17 Virtual Folder File Object

Key	Type	Description
`filename`	`str`	The filename.
`mode`	`int`	The file's mode (permission) bits as an integer.
`size`	`int`	The file's size.
`ctime`	`int`	The timestamp when the file is created.
`mtime`	`int`	The timestamp when the file is last modified.
`atime`	`int`	The timestamp when the file is last accessed.

1.10.18 Virtual Folder Invitation Object

Key	Type	Description
`id`	`UUID`	The unique ID of the invitation. Use this when making API requests referring this invitation.
`inviter`	`str`	The inviter's user ID (email) of the invitation.
`permission`	`str`	The permission that the invited user will have.
`state`	`str`	The current state of the invitation.
`vfolder_id`	`UUID`	The unique ID of the vfolder where the user is invited.
`vfolder_name`	`str`	The name of the vfolder where the user is invited.

New in version v4.20190615.

1.11 Introduction

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

1.11.1 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.

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

Fig. 1.3: The state diagram of a "run" with the **execute** API.

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. 1.3 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 `"clean-finished"` or `"build-finished"` (this happens at the batch-mode only), the client should make the same continuation call. Since cleanup is performed before every build, the client will always receive `"build-finished"` after `"clean-finished"` status. All outputs prior to `"build-finished"` 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 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.

1.11.2 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.

1.12 Kernel Management

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

1.12.1 Creating Kernel Session

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

Creates a new kernel session or returns an existing session, depending on the parameters.

Parameters

Parameter	Type	Description
`image`	`str`	The kernel runtime type in the form of the Docker image name and tag. For legacy, the API also recognizes the `lang` field when `image` is not present. Changed in version v4.20190315.
`clientSessionToken`	`str`	A client-provided session token, which must be unique among the currently non-terminated sessions owned by the requesting access key. Clients may reuse the token if the previous session with the same token has been terminated. It may 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.
`enqueueOnly`	`bool`	*(optional)* If set true, the API returns immediately after queueing the session creation request to the scheduler. Otherwise, the manager will wait until the session gets started actually. (default: `false`) New in version v4.20190615.
`maxWaitSeconds`	`int`	*(optional)* Set the maximum duration to wait until the session starts after queued, in seconds. If zero, the manager will wait indefinitely. (default: `0`) New in version v4.20190615.
`reuseIfExists`	`bool`	*(optional)* If set true, the API returns *without* creating a new session if a session with the same ID and the same image already exists and not terminated. In this case `config` options are *ignored*. If set false but a session with the same ID and image exists, the manager returns an error: "kernel already exists". (default: `true`) New in version v4.20190615.
`group`	`str`	*(optional)* The name of a user group (aka "project") to launch the session within. (default: `"default"`) New in version v4.20190615.
`domain`	`str`	*(optional)* The name of a domain to launch the session within (default: `"default"`) New in version v4.20190615.
`config`	`object`	*(optional)* A *Creation Config Object* to specify kernel configuration including resource requirements. If not given, the kernel is created with the minimum required resource slots defined by the target image.
`tag`	`str`	*(optional)* A per-session, user-provided tag for administrators to keep track of additional information of each session, such as which sessions are from which users.

Example:

```
{
  "image": "python:3.6-ubuntu18.04",
  "clientSessionToken": "mysession-01",
  "enqueueOnly": false,
  "maxWaitSeconds": 0,
  "reuseIfExists": true,
  "domain": "default",
  "group": "default",
  "config": {
    "clusterSize": 1,
    "environ": {
      "MYCONFIG": "XXX",
    },
    "mounts": ["mydata", "mypkgs"],
    "resources": {
      "cpu": "2",
```

(continues on next page)

(continued from previous page)

```
      "mem": "4g",
      "cuda.devices": "1",
    }
  },
  "tag": "example-tag"
}
```

Response

HTTP Status Code	Description
200 OK	The kernel is already running and you are okay to reuse it.
201 Created	The kernel is successfully created.
401 Invalid API parameters	There are invalid or malformed values in the API parameters.
406 Not acceptable	The requested resource limits exceed the server's own limits.

Fields	Type	Values
`kernelId`	`slug`	The session ID used for later API calls, which is same to the value of `clientSessionToken`. This will be random-generated by the server if `clientSessionToken` is not provided.
`status`	`str`	The status of the created kernel. This is always `"PENDING"` if `enqueueOnly` is set true. In other cases, it may be either `"RUNNING"` (normal case), `"ERROR"`, or even `"TERMINATED"` depending on what happens during session startup. New in version v4.20190615.
`servicePorts`	`list[object]`	The list of *Service Port Object*. This field becomes an empty list if `enqueueOnly` is set true, because the final service ports are determined when the session becomes ready after scheduling. --- **Note:** In most cases the service ports are same to what specified in the image metadata, but the agent may add shared services for all kernels. --- Changed in version v4.20190615.
`created`	`bool`	True if the kernel is freshly created.

Example:

```
{
  "kernelId": "mysession-01",
  "status": "RUNNING",
  "servicePorts": [
    {"name": "jupyter", "protocol": "http"},
    {"name": "tensorboard", "protocol": "http"}
  ],
  "created": true
}
```

1.12.2 Getting Kernel Information

- URI: `/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.

Key	Type	Description
`lang`	`str`	The kernel's programming language
`age`	`int` (msec)	The time elapsed since the kernel has started.
`memoryLimit`	`int` (KiB)	The memory limit of the kernel in KiB.
`numQueriesExecuted`	`int`	The number of times the kernel has been accessed.
`cpuCreditUsed`	`int` (msec)	The total time the kernel was running.

1.12.3 Destroying Kernel Session

- URI: `/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.

Key	Type	Description
`stats`	`object`	The *Container Stats Object* of the kernel when deleted.

1.12.4 Restarting Kernel Session

- URI: `/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.

1.13 Service Ports (aka Service Proxies)

The service ports API provides WebSocket-based authenticated and encrypted tunnels to network-facing services ("container services") provided by the kernel container. The main advantage of this feature is that all application-specific network traffic are wrapped as a standard WebSocket API (no need to open extra ports of the manager). It also hides the container from the client and the client from the container, offerring an extra level of security.

Fig. 1.4: The diagram showing how tunneling of TCP connections via WebSockets works.

As Fig. 1.4 shows, all TCP traffic to a container service could be sent to a WebSocket connection to the following API endpoints. A single WebSocket connection corresponds to a single TCP connection to the service, and there may be multiple concurrent WebSocket connections to represent multiple TCP connections to the service. It is the client's responsibility to accept arbitrary TCP connections from users (e.g., web browsers) with proper authorization for multi-user setups and wrap those as WebSocket connections to the following APIs.

When the first connection is initiated, the Backend.AI Agent running the designated kernel container signals the kernel runner daemon in the container to start the designated service. It shortly waits for the in-container port opening and

then delivers the first packet to the service. After initialization, all WebSocket payloads are delivered back and forth just like normal TCP packets. Note that the WebSocket message type must be `BINARY`.

The container service will see the packets from the manager and it never knows the real origin of packets unless the service-level protocol enforces to state such client-side information. Likewise, the client never knows the container's IP address (though the port numbers are included in *service port objects* returned by *the session creation API*).

Note: Currently non-TCP (e.g., UDP) services are not supported.

1.13.1 Service Proxy (HTTP)

- URI: `/stream/kernel/:id/httpproxy?app=:service`
- Method: `GET` upgraded to WebSockets

The service proxy API allows clients to directly connect to service daemons running *inside* compute sessions, such as Jupyter and TensorBoard.

The service name should be taken from the list of *service port objects* returned by *the session creation API*.

New in version v4.20181215.

Parameters

Parameter	Type	Description
`:id`	`slug`	The kernel ID.
`:service`	`slug`	The service name to connect.

1.13.2 Service Proxy (TCP)

- URI: `/stream/kernel/:id/tcpproxy?app=:service`
- Method: `GET` upgraded to WebSockets

This is the TCP version of service proxy, so that client users can connect to native services running inside compute sessions, such as SSH.

The service name should be taken from the list of *service port objects* returned by *the session creation API*.

New in version v4.20181215.

Parameters

Parameter	Type	Description
`:id`	`slug`	The kernel ID.
`:service`	`slug`	The service name to connect.

1.14 Code Execution (Streaming)

The streaming mode provides a lightweight and interactive method to connect with the kernel containers.

1.14.1 Code Execution

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

This is a real-time streaming version of *Code Execution (Batch Mode)* and *Code Execution (Query Mode)* which uses long polling via HTTP.

(under construction)

New in version v4.20181215.

1.14.2 Terminal Emulation

- URI: `/stream/kernel/:id/pty?app=:service`
- 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.

The service name should be taken from the list of *service port objects* returned by *the session creation API*.

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.

Changed in version v4.20181215: Added the `service` query parameter.

Parameters

Parameter	Type	Description
`:id`	`slug`	The kernel ID.
`:service`	`slug`	The service name to connect.

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>"
}
```

1.15 Code Execution (Query Mode)

1.15.1 Executing Snippet

- URI: `/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:

```
{
  "mode": "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 crashes (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": [
      ["stdout", "Hello, world!\n"]
    ],
    "options": null
  }
}
```

Example: Runtime error.

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

```
{
  "result": {
```

(continues on next page)

(continued from previous page)

```
    "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
  }
}
```

1.15.2 Auto-completion

- URI: `/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"
```

(continues on next page)

(continued from previous page)

```
  ]
}
```

1.15.3 Interrupt

- URI: `/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.

1.16 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.

1.16.1 Uploading files

- URI: `/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
204 OK	Success.
400 Bad Request	Returned when one of the uploaded file exeeds the size limit or there are too many files.

1.16.2 Executing with Build Step

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

Parameters

Parameter	Type	Description
`:id`	`slug`	The kernel ID.
`mode`	`enum[str]`	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:

```
{
  "mode": "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*.

1.16.3 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: `/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.
`errors`	`str`	Any errors occurred during scanning the specified path.

1.16.4 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: `/kernel/:id/download`
- Method: `GET`

Parameters

Parameter	Type	Description
`:id`	`slug`	The kernel ID.
`files`	`list[str]`	File paths inside the kernel container to download. (maximum 5 files at once)

Response

HTTP Status Code	Description
200 OK	Success.

1.17 Event Monitoring

1.17.1 Kernel Lifecycle Events

- URI: `/stream/kernel/_/events`
- Method: `GET`

Provides a continuous message-by-message JSON object stream of kernel lifecycles. It uses HTML5 Server-Sent Events (SSE). Browser-based clients may use the EventSource API for convenience.

New in version v4.20190615: First properly implemented in this version, deprecating prior unimplemented interfaces.

Parameters

Parameter	Type	Description
`sessionId`	`slug`	The session ID to monitor the lifecycle events. If set `"*"`, the API will stream events from all kernels visible to the client depending on the client's role and permissions.
`ownerAccessKey`	`str`	*(optional)* The access key of the owner of the specified session, since different access keys (users) may share a same session ID for different session instances. You can specify this only when the client is either a domain admin or a superadmin.
`group`	`str`	The group name to filter the lifecycle events. If set `"*"`, the API will stream events from all kernels visible to the client depending on the client's role and permissions.

Responses

The response is a continuous stream of UTF-8 text lines following the `text/event-stream` format. Each event is composed of the event type and data, where the data part is encoded as JSON.

Possible event names (more events may be added in the future):

Event Name	Description
`kernel_preparing`	The session is just scheduled from the job queue and got an agent resource allocation.
`kernel_pulling`	The session begins pulling the kernel image (usually from a Docker registry) to the scheduled agent.
`kernel_creating`	The session is being created as containers (or other entities in different agent backends).
`kernel_started`	The session becomes ready to execute codes.
`kernel_terminated`	The session has terminated.

When using the EventSource API, you should add event listeners as follows:

```
const sse = new EventSource('/stream/kernel/_/events', {
  withCredentials: true,
});
sse.addEventListener('kernel-started', (e) => {
  console.log('kerenl-started', JSON.parse(e.data));
});
```

Note: The EventSource API must be used with the session-based authentication mode (when the endpoint is a console-server) which uses the browser cookies. Otherwise, you need to manually implement the event stream parser using the standard fetch API running against the manager server.

The event data contains a JSON string like this (more fields may be added in the future):

Field Name	Description
`sessionId`	The source session ID.
`ownerAccessKey`	The access key who owns the session.
`reason`	A short string that describes why the event happened. This may be `null` or an empty string.
`result`	Only present for `kernel-terminated` events. Only meaningful for batch-type sessions. Either one of: `"UNDEFINED"`, `"SUCCESS"`, `"FAILURE"`

```
{
  "sessionId": "mysession-01",
  "ownerAccessKey": "MYACCESSKEY",
  "reason": "self-terminated",
  "result": "SUCCESS"
}
```

1.18 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.

Also, you might share your virtual folders with other users by inviting them and granting them proper permission. Currently, there are three levels of permissions: read-only, read-write, read-write-delete. They are represented by short strings, `'ro'`, `'rw'`, `'rd'`, respectively. The owner of a virtual folder have read-write-delete permission for the folder.

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.

1.18.1 Listing Virtual Folders

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

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

Parameters

None.

Response

HTTP Status Code	Description
200 OK	Success.

Fields	Type	Values
(root)	`list[object]`	A list of *Virtual Folder List Item Object*.

Example:

```
[
  { "name": "mydata", "id": "5da5f8e163dd4b86826d6b4db2b7b71a", "...": "..." },
  { "name": "sample01", "id": "0ecfab9e608c478f98d1734b02a54774", "...": "..." },
]
```

1.18.2 Listing Virtual Folder Hosts

Returns the list of available host names where the current keypair can create new virtual folders.

New in version v4.20190315.

- URI: `/folders/_/hosts`
- Method: `GET`

Parameters

None.

Response

HTTP Status Code	Description
200 OK	Success.

Fields	Type	Values
`default`	`str`	The default virtual folder host.
`allowed`	`list[str]`	The list of available virtual folder hosts.

Example:

```
{
  "default": "nfs1",
  "allowed": ["nfs1", "nfs2", "cephfs1"]
}
```

1.18.3 Creating a Virtual Folder

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

Creates a virtual folder associated with the current API key.

Parameters

Parameter	Type	Description
`name`	`str`	The human-readable name of the virtual folder.
`host`	`str`	(optional) The name of the virtual folder host.

Example:

```
{
  "name": "My Data",
  "host": "nfs1"
}
```

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
`id`	`slug`	The unique folder ID used for later API calls.
`name`	`str`	The human-readable name of the created virtual folder.
`host`	`str`	The name of the virtual folder host where the new folder is created.

Example:

```
{
  "id": "aef1691db3354020986d6498340df13c",
  "name": "My Data",
  "host": "nfs1"
}
```

1.18.4 Getting Virtual Folder Information

- URI: `/folders/:name`
- 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
`name`	`str`	The human-readable name of the virtual folder.

Response

HTTP Status Code	Description
200 OK	The information is successfully returned.
404 Not Found	There is no such folder or you may not have proper permission to access the folder.

Fields	Type	Values
(root)	`object`	*Virtual Folder Item Object*.

1.18.5 Deleting Virtual Folder

- URI: `/folders/:name`
- 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
`name`	The human-readable name of the virtual folder.

Response

HTTP Status Code	Description
204 No Content	The folder is successfully destroyed.
404 Not Found	There is no such folder or you may not have proper permission to delete the folder.

1.18.6 Listing Files in Virtual Folder

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

- URI: `/folders/:name/files`
- Method: `GET`

Parameters

Parameter	Type	Description
`:name`	`str`	The human-readable name of the virtual folder.
`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 or you may not have proper permission to access the folder.

Fields	Type	Values
`files`	`list[object]`	List of *Virtual Folder File Object*

1.18.7 Uploading Multiple Files to Virtual Folder

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

- URI: `/folders/:name/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
`:name`	`str`	The human-readable name of the virtual folder.
(body)	`multipart`	A multi-part encoded file data which is composed of multiple occurrences of `src` field. Each part must contain a valid filename and the content type is always assumed as `application/octet-stream`.

Response

HTTP Status Code	Description
201 Created	Success.
400 Bad Request	There already exists a file with duplicated name that cannot be overwritten in the virtual folder.
404 Not Found	There is no such folder or you may not have proper permission to write into folder.

1.18.8 Creating New Directory in Virtual Folder

Create a new directory in the virtual folder associated with current keypair. this API recursively creates parent directories if they does not exist.

- URI: `/folders/:name/mkdir`
- Method: `POST`

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

Parameters

Parameter	Type	Description
`:name`	`str`	The human-readable name of the virtual folder.
`path`	`str`	The relative path of a new folder to create inside the virtual folder.

Response

HTTP Status Code	Description
201 Created	Success.
400 Bad Request	There already exists a file, not a directory, with duplicated name.
404 Not Found	There is no such folder or you may not have proper permission to write into folder.

1.18.9 Downloading Single File from Virtual Folder

Download a single file from a virtual folder associated with the current keypair. This API does not perform any encoding or compression but just outputs the raw file content as the response body, for simpler client-side implementation.

New in version v4.20190315.

- URI: `/folders/:name/download_single`
- Method: `GET`

Parameters

Parameter	Type	Description
`:name`	`str`	The human-readable name of the virtual folder.
`file`	`str`	A file path inside the virtual folder to download.

Response

HTTP Status Code	Description
200 OK	Success.
404 Not Found	File not found or you may not have proper permission to access the folder.

Fields	Type	Values
(body)	`bytes`	The content of file.

1.18.10 Downloading Multiple 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`) in a multi-part message format. Clients may detect the total download size using `X-TOTAL-PAYLOADS-LENGTH` (all upper case) HTTP header of the response in prior to reading/parsing the response body.

- URI: `/folders/:name/download`
- Method: `GET`

Parameters

Parameter	Type	Description
`:name`	`str`	The human-readable name of the virtual folder.
`files`	`list[str]`	File paths inside the virtual folder to download.

Response

HTTP Status Code	Description
200 OK	Success.
404 Not Found	File not found or you may not have proper permission to access the folder.

Fields	Type	Values
(body)	`multipart`	The gzipped content of files in the mixed multipart format.

1.18.11 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: `/folders/:name/delete_files`
- Method: `DELETE`

Parameters

Parameter	Type	Description
`:name`	`str`	The human-readable name of the virtual folder.
`files`	`list[str]`	File paths inside the virtual folder to delete.
`recursive`	`bool`	Recursive option to delete folders if set to True. The default is False.

Response

HTTP Status Code	Description
200 OK	Success.
400 Bad Request	You tried to delete a folder without setting recursive option as True.
404 Not Found	There is no such folder or you may not have proper permission to delete the file in the folder.

1.18.12 Listing Invitations for Virtual Folder

Returns the list of pending invitations that requested user received.

- URI: `/folders/invitations/list`
- Method: `GET`

Parameters

This API does not need any parameter.

Response

HTTP Status Code	Description
200 OK	Success.

Fields	Type	Values
`invitations`	`list[object]`	A list of *Virtual Folder Invitation Object*.

1.18.13 Creating an Invitation

Invite other users to share a virtual folder with proper permissions. If a user is already invited, then this API does not create a new invitation or update the permission of the existing invitation.

- URI: `/folders/:name/invite`
- Method: `POST`

Parameters

Parameter	Type	Description
`:name`	`str`	The human-readable name of the virtual folder.
`perm`	`str`	The permission to grant to invitee.
`user_ids`	`list[slug]`	A list of user IDs to invite.

Response

HTTP Status Code	Description
200 OK	Success.
400 Bad Request	No invitee is given.
404 Not Found	There is no invitation.

Fields	Type	Values
`invited_ids`	`list[str]`	A list of invited user IDs.

1.18.14 Accepting an Invitation

Accept an invitation and receive permission to a virtual folder as in the invitation.

- URI: `/folders/invitations/accept`
- Method: `POST`

Parameters

Parameter	Type	Description
`inv_id`	`slug`	The unique invitation ID.
`inv_ak`	`bool`	The access key of invitee.

Response

HTTP Status Code	Description
200 OK	Success.
400 Bad Request	The name of the target virtual folder is duplicate with your existing virtual folders.
404 Not Found	There is no such invitation.

Fields	Type	Values
`msg`	`str`	Detail message for the invitation acceptance.

1.18.15 Rejecting an Invitation

Reject an invitation.

- URI: `/folders/invitations/delete`
- Method: `DELETE`

Parameters

Parameter	Type	Description
`inv_id`	`slug`	The unique invitation ID.

Response

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

Fields	Type	Values
`msg`	`str`	Detail message for the invitation deletion.

1.19 Resource Presets

Resource presets provide a simple storage for pre-configured resource slots and a dynamic checker for allocatability of given presets before actually calling the kernel creation API.

To add/modify/delete resource presets, you need to use the admin GraphQL API.

New in version v4.20190315.

1.19.1 Listing Resource Presets

Returns the list of admin-configured resource presets.

- URI: `/resource/presets`
- Method: `GET`

Parameters

None.

Response

HTTP Status Code	Description
200 OK	The preset list is returned.

Fields	Type	Values
`presets`	`list[object]`	The list of *Resource Preset Object*

1.19.2 Checking Allocatability of Resource Presets

Returns current keypair and scaling-group's resource limits in addition to the list of admin-configured resource presets. It also checks the allocatability of the resource presets and adds `allocatable` boolean field to each preset item.

- URI: `/resource/check-presets`
- Method: `POST`

Parameters

None.

Response

HTTP Status Code	Description
200 OK	The preset list is returned.
401 Unauthorized	The client is not authorized.

Fields	Type	Values
`keypair_limits`	*Resource Slot Object*	The maximum amount of total resource slots allowed for the current access key. It may contain infinity values as the string "Infinity".
`keypair_using`	*Resource Slot Object*	The amount of total resource slots used by the current access key.
`keypair_remaining`	*Resource Slot Object*	The amount of total resource slots remaining for the current access key. It may contain infinity values as the string "Infinity".
`scaling_group_remaining`	*Resource Slot Object*	The amount of total resource slots remaining for the current scaling group. It may contain infinity values as the string "Infinity" if the server is configured for auto-scaling.
`presets`	`list(tuple)`	The list of *Resource Preset Object*, but with an extra boolean field `allocatable` which indicates if the given resource slot is actually allocatable considering the keypair's resrouce limits and the scaling group's current usage.

1.20 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.

1.20.1 Basics of GraphQL

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

```
https://api.backend.ai/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.

1.21 KeyPair Management

1.21.1 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
  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
}
```

1.21.2 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.

1.22 Compute Session Monitoring

1.22.1 Full Admin

Query Schema

```
type ComputeSession {
  sess_id: String
  id: UUID
  role: String
  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
  mem_max_bytes: Int
  mem_cur_bytes: Int
  net_rx_bytes: Int
  net_tx_bytes: Int
  io_read_bytes: Int
  io_write_bytes: Int
  lang: String
  tag: String
  workers(status: String): [ComputeWorker]
}

type ComputeWorker {
  sess_id: String
  id: UUID
  role: String
  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
  mem_max_bytes: Int
  mem_cur_bytes: Int
  net_rx_bytes: Int
  net_tx_bytes: Int
  io_read_bytes: Int
  io_write_bytes: Int
}

type root {
```

(continues on next page)

(continued from previous page)

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

1.22.2 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]
}
```

1.23 Virtual Folder Management

1.23.1 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]
}
```

1.23.2 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]
}
```

1.24 Statistics

1.24.1 Full Admin

Query Schema

TODO

1.24.2 Restricted Owner Access

Query Schema

TODO

1.25 Development Setup

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

1.25.1 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.

1.25.2 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 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`.

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

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


```
$ 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.

1.25.3 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")'
```

1.26 Adding New REPL Kernels

1.26.1 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).

1.26.2 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
    - `ai.backend.maxcores`: *N* (the number of CPU cores recommended for this kernel)
    - `ai.backend.maxmem`: *M* (the memory size in a human-readable bytes recommended for this kernel, `128m` (128 MBytes) for example)
    - `ai.backend.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.
    - `ai.backend.mode`: `query`, `pty`, or `query+pty`
  - Optional Labels
    - `ai.backend.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`)
    - `ai.backend.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.)
    - `ai.backend.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.
    - `ai.backend.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/backend.ai/jail default `which lua` /home/backend.ai/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)

1.26.3 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,...."]
    ],
    "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.

1.26.4 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.

1.26.5 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

TWO

INDICES AND TABLES

- genindex
- modindex
- search