
Backend.AI API Documentation

Release 20.03

Lablup Inc.

Aug 08, 2023

CONCEPTS

1	Table of Contents	3
1.1	Key Concepts	3
1.2	API Overview	5
1.3	FAQ	5
1.4	Quickstart Guides	6
1.5	Supplementary Guides	9
1.6	Upgrading from 20.03 to 20.09	15
1.7	Migrating from the Docker Hub to cr.backend.ai	15
1.8	Client SDK Libraries and Tools	16
1.9	API and Document Conventions	17
1.10	Authentication	19
1.11	Rate Limiting	22
1.12	JSON Object References	23
1.13	Introduction	30
1.14	Session Management	32
1.15	Service Ports (aka Service Proxies)	36
1.16	Code Execution (Streaming)	38
1.17	Code Execution (Query Mode)	40
1.18	Code Execution (Batch Mode)	45
1.19	Event Monitoring	48
1.20	Virtual Folders	51
1.21	Resource Presets	60
1.22	Introduction	62
1.23	Agent Monitoring	64
1.24	Scaling Group Management	65
1.25	Domain Management	67
1.26	Group Management	68
1.27	User Management	70
1.28	Image Management	71
1.29	Compute Session Monitoring	73
1.30	Virtual Folder Management	77
1.31	KeyPair Management	78
1.32	KeyPair Resource Policy Management	79
1.33	Resource Preset Management	81
1.34	Development Setup	82
1.35	Adding New Kernel Images	87
2	Indices and tables	95

Latest API version: v5.20191215

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.

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 sessions, and routes code execution requests to appropriate agents and sessions. It also collects the output of sessions 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 sessions. Each agent also reports the resource capacity and status of its server, so that the manager can assign new sessions on idle servers to load balance.

1.1.2 Compute sessions and Kernels

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 adapters.

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 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 <i>frontends</i>	Insecure host resource sharing
Backend.AI	Pluggable <i>backend</i> 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 8.0 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 / devlove`. 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

This recording has been archived

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.

This recording has been archived

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` according 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 recognized by pyenv. Any pyenv-enabled shells will automatically 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

This recording has been archived

```
$ 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 Upgrading from 20.03 to 20.09

(TODO)

1.7 Migrating from the Docker Hub to cr.backend.ai

As of November 2020, the Docker Hub has begun to limit the retention time and the rate of pulls of public images. Since Backend.AI uses a number of Docker images with variety of access frequencies, we decided to migrate to our own container registry, <https://cr.backend.ai>.

It is strongly recommended to set a maintenance period if there are active users of the Backend.AI cluster to prevent new session starts during migration. This registry migration does not affect existing running sessions, though the Docker image removal in the agent nodes can only be done after terminating all existing containers started with the old images and there will be brief disconnection of service ports as the manager requires to be restarted.

1. Update your Backend.AI installation to the latest version (manager 20.03.11 or 20.09.0b2) to get support for Harbor v2 container registries.
2. Save the following JSON snippet as `registry-config.json`.

```
{
  "config": {
    "docker": {
      "registry": {
        "cr.backend.ai": {
          "": "https://cr.backend.ai",
          "type": "harbor2",
          "project": "stable,community"
        }
      }
    }
  }
}
```

3. Run the following using the manager CLI on one of the manager nodes:

```
$ sudo systemctl stop backendai-manager # stop the manager daemon (may differ by_
↪setup)
$ backend.ai mgr etcd put-json ' registry-config.json
$ backend.ai mgr etcd rescan-images cr.backend.ai
$ sudo systemctl start backendai-manager # start the manager daemon (may differ_
↪by setup)
```

- The agents will automatically pull the images since the image references are changed even when the new images are actually same to the existing ones. It is recommended to pull the essential images by yourself in the agents to avoid long waiting times when starting sessions using the `docker pull` command in the agent nodes.
- Now the images are categorized with additional path prefix, such as `stable` and `community`. More prefixes may be introduced in the future and some prefixes may be set only available to specific set of users/user groups, with dedicated credentials.

For example, `lablup/python:3.6-ubuntu18.04` is now referred as `cr.backend.ai/stable/python:3.6-ubuntu18.04`.

- If you have configured image aliases, you need to update them manually as well, using the `backend.ai mgr etcd alias` command. This does not affect existing sessions running with old aliases.

4. Update the allowed docker registries policy for each domain using the `backend.ai mgr dbshell` command. Remove “index.docker.io” from the existing values and replace “...” below with your own domain names and additional registries.

```
SELECT name, allowed_docker_registries FROM domains; -- check the current config
UPDATE domains SET allowed_docker_registries = '{cr.backend.ai,...}' WHERE name =
↳ '...';
```

5. Now you may start new sessions using the images from the new registry.
6. After terminating all existing sessions using the old images from the Docker Hub (i.e., images whose names start with `lablup/` prefix), remove the image metadata and registry configuration using the manager CLI:

```
$ backend.ai mgr etcd delete --prefix images/index.docker.io
$ backend.ai mgr etcd delete --prefix config/docker/registry/index.docker.io
```

7. Run `docker rmi` commands to clean up the pulled images in the agent nodes. (Automatic/managed removal of images will be implemented in the future versions of Backend.AI)

1.8 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.

1.8.1 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](#)

1.8.2 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](#)

1.8.3 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](#)

1.8.4 PHP

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

1.9 API and Document Conventions

1.9.1 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.

1.9.2 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.

1.9.3 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.

1.9.4 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
<code>a</code>	The attribute <code>a</code> of the root object. (e.g., 123 at <code>{"a": 123}</code>)
<code>a.b</code>	The attribute <code>b</code> of the object <code>a</code> on the root. (e.g., 456 at <code>{"a": {"b": 456}}</code>)
<code>a.0</code>	An item in the list <code>a</code> 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 <code>{"a": [13, 57, 24, 68]}</code>)
<code>a.0.b</code>	The attribute <code>b</code> of an item in the list <code>a</code> on the root. (e.g., any of 1, 2, and 3 at <code>{"a": [{"b": 1}, {"b": 2}, {"b": 3}]}</code>)

1.9.5 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
<code>decimal</code>	Fractional numbers represented as <code>str</code> not to loose precision. (e.g., to express money amounts)
<code>slug</code>	Similar to <code>str</code> , 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.
<code>datetime</code>	ISO-8601 timestamps in <code>str</code> , e.g., <code>"YYY-mm-ddTHH:MM:SS.ffffff+HH:MM"</code> . 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).
<code>enum[*]</code>	Only allows a fixed/predefined set of possible values in the given parametrized type.

1.9.6 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"
}
```

1.10 Authentication

1.10.1 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).

1.10.2 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 <code>owner_access_key</code> 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 <code>application/json</code>
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.yyyymmdd 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

1.10.3 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

1.10.4 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
x-sorna-version:v2.20170215
e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855
```

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>
```

1.10.5 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."
}
```

1.11 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.

1.12 JSON Object References

1.12.1 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 <code>index</code> is ignored.
index	int	The page number to show, zero-based.

1.12.2 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.

1.12.3 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	date	The timestamp when the keypair was created.

1.12.4 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 <code>true</code> to <code>false</code> , existing running sessions continue to run but any requests to create new sessions are refused. (default: <code>true</code>)
concurrency	int	The maximum number of concurrent sessions allowed for this keypair. (default: 5)
ML. clusterSize	int	Sets the number of instances clustered together when launching new machine learning sessions. (default: 1)
ML. instanceMemory	int (MiB)	Sets the memory limit of each instance in the cluster launched for new machine learning sessions. (default: 8)

The enterprise edition offers the following additional properties:

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

1.12.5 Service Port Object

Key	Type	Description
name	slug	The name of service provided by the container. See also: <i>Terminal Emulation</i>
protocol	str	The type of network protocol used by the container service.

1.12.6 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 <code>null</code> , it skips the build step. If this field is a constant string <code>"*"</code> , 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 compiles them into <code>./main</code> executable, with commonly used C/link flags: <code>"-pthread -lm -lrt -ldl"</code> .
exec	str	The bash command to execute the main program. If this is not present, an empty string, or <code>null</code> , the server only performs the build step and <code>options.buildLog</code> is assumed to be <code>true</code> (the given value is ignored).
clean	str	The bash command to clean the intermediate files produced during the build phase. The clean step comes <i>before</i> the build step if specified so that the build step can (re)start fresh. If the field is not present, an empty string, or <code>null</code> , it skips the clean step. Unlike the build and exec command, the default for <code>"*"</code> 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"
}
```

1.12.7 Execution Result Object

Key	Type	Description
<code>runId</code>	string	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.
<code>status</code>	enum	One of "continued", "waiting-input", "finished", "clean-finished", "build-finished", or "exec-timeout". See more details at Code Execution Model .
<code>exitCode</code>	integer 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 <i>without</i> global context support, <code>exitCode</code> 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.
<code>console</code>	list of object	A list of Console Item Object .
<code>options</code>	object	An object containing extra display options. If there is no options indicated by the kernel, this field is null. When <code>result.status</code> is "waiting-input", it has a boolean field <code>is_password</code> so that you could use different types of text boxes for user inputs.
<code>files</code>	list of string	A list of Execution Result File Object that represents files generated in <code>/home/work/.output</code> directory of the container during the code execution.

1.12.8 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 .

1.12.9 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.

1.12.10 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	int (Byte)	Currently not used field.
io_write_bytes	int (Byte)	Currently not used field.

1.12.11 Creation Config Object

Key	Type	Description
environ	object	Additional object specifying additional environment variables. The values must be strings.
mounts	list	An optional list of the name of virtual folders that belongs to the current API key. These virtual folders are mounted under <code>/home/work</code> . For example, if the virtual folder name is <code>abc</code> , you can access it on <code>/home/work/abc</code> . 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 <code>/home/work</code> . You may mount up to 5 folders for each session.
clusterSize	int	The number of instances bundled for this session.
resources	Resource Slot 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.

1.12.12 Resource Slot Object

Key	Type	Description
cpu	str int	The number of CPU cores.
mem	str int	<p>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 <i>k</i>, <i>m</i>, <i>g</i>, <i>t</i>, <i>p</i>, <i>e</i>, <i>z</i>, and <i>y</i>, 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.</p> <div style="border: 1px solid black; padding: 10px; margin: 10px 0;"> <p>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 <code>Number.MAX_SAFE_INTEGER</code> 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.</p> </div>
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	<p>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”.</p> <p>For “count” slots, you may put arbitrary positive real number there, but fractions may be truncated depending on the plugin implementation.</p> <p>For “bytes” slots, its interpretation and representation follows that of the <code>mem</code> field.</p>

1.12.13 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 <code>/resource/*</code> REST APIs.
shared_memory	int (Byte)	The pre-configured shared memory size. Client can send humanized strings like ‘2g’, ‘128m’, ‘534773760’, etc, and they will be automatically converted into bytes.

1.12.14 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.

1.12.15 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, <code>null</code> .
group	UUID	The group ID if the owner of this item is a group vfolder. Otherwise, <code>null</code> .
type	enum	The owner type of vfolder. One of "user" or "group".

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

1.12.16 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	date	The date and time when the folder is created.
last_used	date	The date and time when the folder is last used.
user	UUID	The user ID if the owner of this item is a user. Otherwise, <code>null</code> .
group	UUID	The group ID if the owner of this item is a group. Otherwise, <code>null</code> .
type	enum	The owner type of vfolder. One of "user" or "group".

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

1.12.17 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.

1.12.18 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.

Key	Type	Description
content	str	The retrieved content (multi-line string) of fstab.
node	str	The node type, either "agent" or "manager."
node_id	str	The node's unique ID.

New in version v4.20190615.

1.13 Introduction

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

1.13.1 Code Execution Model

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

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

To distinguish different runs which may be overlapped, the client must provide the same **run ID** to all **execute** calls during a single run. The run ID should be unique for each run and can be an arbitrary random string. If the run ID is not provided by the client at the first execute call of a run, the API server will assign a random one and inform it to the client via the first response. Normally, if two or more runs are overlapped, they are processed in a FIFO order using

an internal queue. But they may be processed in parallel if the kernel runtime supports parallel processing. Note that the API server may raise a timeout error and cancel the run if the waiting time exceeds a certain limit.

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

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

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

Depending on the value of `status` field in the returned *Execution Result Object*, the client must perform another subsequent **execute** call with appropriate arguments or stop. Fig. 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.

1.13.2 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 `
`, 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.

1.14 Session Management

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

1.14.1 Creating Session

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

Creates a new 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 <code>lang</code> field when <code>image</code> 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 <i>without</i> creating a new session if a session with the same ID and the same image already exists and not terminated. In this case <code>config</code> options are <i>ignored</i> . If set false but a session with the same ID and image exists, the manager returns an error: "session 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 session is already running and you are okay to reuse it.
201 Created	The session 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
sessId	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	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 sessions. Changed in version v4.20190615.
created	bool	True if the session is freshly created.

Example:

```

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

```

1.14.2 Getting Session Information

- URI: `/session/:id`
- Method: GET

Retrieves information about a 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
<code>:id</code>	slug	The session ID.

Response

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

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

1.14.3 Destroying Session

- URI: `/session/:id`
- Method: DELETE

Terminates a session.

Parameters

Parameter	Type	Description
<code>:id</code>	slug	The session ID.

Response

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

Key	Type	Description
stats	object	The <i>Container Stats Object</i> of the kernel when deleted.

1.14.4 Restarting Session

- URI: /session/:id
- Method: PATCH

Restarts a session. The idle time of the session 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 session ID.

Response

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

1.15 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, offering 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.

1.15.1 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
<code>:id</code>	slug	The kernel ID.
<code>:service</code>	slug	The service name to connect.

1.15.2 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
<code>:id</code>	slug	The kernel ID.
<code>:service</code>	slug	The service name to connect.

1.16 Code Execution (Streaming)

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

1.16.1 Code Execution

- URI: `/stream/session/: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.

1.16.2 Terminal Emulation

- URI: `/stream/session/: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
<code>:id</code>	slug	The session ID.
<code>:service</code>	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 session 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 session 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 session applies some special formatting to distinguish them (e.g., make all stderr outputs 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>"
}
```

1.17 Code Execution (Query Mode)

1.17.1 Executing Snippet

- URI: /session/:id
- Method: POST

Executes a snippet of user code using the specified session. Each execution request to a same 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 guarantee the correct execution order of multiple interdependent requests. When the session is terminated or restarted, all such volatile states vanish.

Parameters

Parameter	Type	Description
:id	slug	The session 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 session.
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 session has responded with the execution result. The response body contains a JSON object as described below.

Fields	Type	Values
result	object	<i>Execution Result Object.</i>

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 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
  }
}
```

1.17.2 Auto-completion

- URI: /session/:id/complete
- Method: POST

Parameters

Parameter	Type	Description
:id	slug	The session 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 session has responded with the execution result. The response body contains a JSON object as described below.

Fields	Type	Values
result	list	An ordered list containing the possible auto-completion matches as strings. This may be empty if the current session 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
```

1.17.3 Interrupt

- URI: `/session/:id/interrupt`
- Method: POST

Parameters

Parameter	Type	Description
<code>:id</code>	slug	The session ID.

Response

HTTP Status Code	Description
204 No Content	Sent the interrupt signal to the session. Note that this does <i>not</i> guarantee the effectiveness of the interruption.

1.18 Code Execution (Batch Mode)

Some sessions 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.

1.18.1 Uploading files

- URI: `/session/:id/upload`
- Method: POST

Parameters

Upload files to the 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 sessions 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 session 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 exceeds the size limit or there are too many files.

1.18.2 Executing with Build Step

- URI: /session/:id
- Method: POST

Parameters

Parameter	Type	Description
:id	slug	The session ID.
mode	enum	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	obj	Batch Execution Query Object .

Example:

```
{
  "mode": "batch",
  "options": "{batch-execution-query-object}",
  "runId": "af9185c5fb0eachb2"
}
```

Response

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

Fields	Type	Values
result	object	Execution Result Object .

1.18.3 Listing Files

Once files are uploaded to the 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 session.

- URI: `/session/:id/files`
- Method: GET

Parameters

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

Response

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

Fields	Type	Values
<code>files</code>	str	Stringified json containing list of files.
<code>folder_path</code>	str	Absolute path inside session.
<code>errors</code>	str	Any errors occurred during scanning the specified path.

1.18.4 Downloading Files

Download files from your compute session.

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

- URI: `/session/:id/download`
- Method: GET

Parameters

Parameter	Type	Description
<code>:id</code>	slug	The session ID.
<code>files</code>	list[str]	File paths inside the session container to download. (maximum 5 files at once)

Response

HTTP Status Code	Description
200 OK	Success.

1.19 Event Monitoring

1.19.1 Session Lifecycle Events

- URI: `/events/session`
- Method: GET

Provides a continuous message-by-message JSON object stream of session 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.

Changed in version v5.20191215: The URI is changed from `/stream/session/_/events` to `/events/session`.

Parameters

Parameter	Type	Description
<code>sessionId</code>	slug	The session ID to monitor the lifecycle events. If set <code>"*"</code> , the API will stream events from all sessions visible to the client depending on the client's role and permissions.
<code>ownerAccessKey</code>	Key	(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.
<code>group</code>	str	The group name to filter the lifecycle events. If set <code>"*"</code> , the API will stream events from all sessions 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
session_prepared	The session is just scheduled from the job queue and got an agent resource allocation.
session_pulling	The session begins pulling the session image (usually from a Docker registry) to the scheduled agent.
session_created	The session is being created as containers (or other entities in different agent backends).
session_started	The session becomes ready to execute codes.
session_terminated	The session has terminated.

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

```
const sse = new EventSource('/events/session', {
  withCredentials: true,
});
sse.addEventListener('session_started', (e) => {
  console.log('session_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 session-terminated events. Only meaningful for batch-type sessions. Either one of: "UNDEFINED", "SUCCESS", "FAILURE"

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

1.19.2 Background Task Progress Events

- URI: /events/background-task
- Method: GET for server-side events

New in version v5.20191215.

Parameters

Parameter	Type	Description
taskId	UUID	The background task ID to monitor the progress and completion.

Responses

The response is a continuous stream of UTF-8 text lines following `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
task_updated	Updates for the progress. This can be generated many times during the background task execution.
task_done	The background task is successfully completed.
task_failed	The background task has failed. Check the <code>message</code> field and/or query the error logs API for error details.
task_cancelled	The background task is cancelled in the middle. Usually this means that the server is being shutdown for maintenance.
server_close	This event indicates explicit server-initiated close of the event monitoring connection, which is raised just after the background task is either done/failed/cancelled. The client should not reconnect because there is nothing more to monitor about the given task.

The event data (per-line JSON objects) include the following fields:

Field Name	Type	Description
task_id	str	The background task ID.
current_progress	int	The current progress value. Only meaningful for <code>task_update</code> events. If <code>total_progress</code> is zero, this value should be ignored.
total_progress	int	The total progress count. Only meaningful for <code>task_update</code> events. The scale may be an arbitrary positive integer. If the total count is not defined, this may be zero.
message	str	An optional human-readable message indicating what the task is doing. It may be <code>null</code> . For example, it may contain the name of agent or scaling group being worked on for image preload/unload APIs.

Check out *[the session lifecycle events API](#)* for example client-side Javascript implementations to handle `text/event-stream` responses.

If you make the request for the tasks already finished, it may return either “404 Not Found” (the result is expired or the task ID is invalid) or a single event which is one of `task_done`, `task_fail`, or `task_cancel` followed by immediate response disconnection. Currently, the results for finished tasks may be archived up to one day (24 hours).

1.20 Virtual Folders

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

You can mount virtual folders when creating new 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.

1.20.1 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	A list of <i>Virtual Folder List Item Object</i> .

Example:

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

1.20.2 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	The list of available virtual folder hosts.

Example:

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

1.20.3 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"
}
```

1.20.4 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	<i>Virtual Folder Item Object.</i>

1.20.5 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
<code>name</code>	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.

1.20.6 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
<code>:name</code>	<code>str</code>	The human-readable name of the virtual folder.
<code>path</code>	<code>str</code>	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 <i>Virtual Folder File Object</i>

1.20.7 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 Code	Status	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.

1.20.8 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.

1.20.9 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.

1.20.10 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
<code>:name</code>	<code>str</code>	The human-readable name of the virtual folder.
<code>files</code>	<code>list[str]</code>	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)	<code>multipart</code>	The gzipped content of files in the mixed multipart format.

1.20.11 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
<code>:name</code>	<code>str</code>	The human-readable name of the virtual folder.
<code>files</code>	<code>list[str]</code>	File paths inside the virtual folder to delete.
<code>recursive</code>	<code>bool</code>	Recursive option to delete folders if set to True. The default is False.

Response

HTTP Code	Status	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.

1.20.12 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	A list of <i>Virtual Folder Invitation Object</i> .

1.20.13 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
<code>:name</code>	str	The human-readable name of the virtual folder.
<code>perm</code>	str	The permission to grant to invitee.
<code>user_ids</code>	list[str]	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	A list of invited user IDs.

1.20.14 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.

1.20.15 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.

1.21 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.

1.21.1 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	The list of <i>Resource Preset Object</i>

1.21.2 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
<code>keypair_limits</code>	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".
<code>keypair_usage</code>	Resource Slot Object	The amount of total resource slots used by the current access key.
<code>keypair_remaining</code>	Resource Slot Object	The amount of total resource slots remaining for the current access key. It may contain infinity values as the string "Infinity".
<code>scaling_group_remaining</code>	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.
<code>presets</code>	list of Resource Preset Object	The list of Resource Preset Object , but with an extra boolean field <code>allocatable</code> which indicates if the given resource slot is actually allocatable considering the keypair's resource limits and the scaling group's current usage.

1.22 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 such as [GraphiQL](#).

1.22.1 Basics of GraphQL

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

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

Common Object Types

`ResourceLimit` represents a range (min, max) of specific resource slot (`key`). The max value may be the string constant “Infinity” if not specified.

```
type ResourceLimit {
  key: String
  min: String
  max: String
}
```

`KVPair` is used to represent a mapping data structure with arbitrary (runtime-determined) key-value pairs, in contrast to other data types in GraphQL which have a set of predefined static fields.

```
type KVPair {
  key: String
  value: String
}
```

Pagination Convention

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

We use a pagination convention as described below:

```
interface Item {
  id: UUID
  # other fields are defined by concrete types
}

interface PaginatedList(
  offset: Integer!,
  limit: Integer!,
  # some concrete types define ordering customization fields:
  #   order_key: String,
  #   order_asc: Boolean,
  # other optional filter condition may be added by concrete types
) {
  total_count: Integer
  items: [Item]
}
```

`offset` and `limit` are interpreted as SQL’s `offset` and `limit` clauses. For the first page, set the `offset` to zero and the `limit` to the page size. The `items` field may contain from zero up to `limit` items. Use `total_count` field to determine how many pages are there. Fields that support pagination is suffixed with `_list` in our schema.

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`
- `BigInt`: GraphQL's integer is officially 32-bits only, so we define a "big integer" type which can represent from -9007199254740991 ($-2^{53}+1$) to 9007199254740991 ($2^{53}-1$) (or, $\pm(8 \text{ PiB} - 1 \text{ byte})$). This range is regarded as a "safe" (i.e., can be compared without loosing precision) integer range in most Javascript implementations which represent numbers in the IEEE-754 double (64-bit) format.
- `JSONString`: It contains a stringified JSON value, whereas the whole query result is already a JSON object. A client must parse the value *again* to get an object representation.

Authentication

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

Versioning

As we use GraphQL, there is no explicit versioning. Check out the descriptions for each API for its own version history.

1.23 Agent Monitoring

1.23.1 Query Schema

```
type Agent {
  id: ID
  status: String
  status_changed: DateTime
  region: String
  scaling_group: String
  available_slots: JSONString # ResourceSlot
  occupied_slots: JSONString # ResourceSlot
  addr: String
  first_contact: DateTime
  lost_at: DateTime
  live_stat: JSONString
  version: String
  compute_plugins: JSONString
  compute_containers(status: String): [ComputeContainer]

  # legacy fields
  mem_slots: Int
  cpu_slots: Float
  gpu_slots: Float
  tpu_slots: Float
  used_mem_slots: Int
  used_cpu_slots: Float
  used_gpu_slots: Float
  used_tpu_slots: Float
}
```

(continues on next page)

(continued from previous page)

```

    cpu_cur_pct: Float
    mem_cur_bytes: Float
}

type Query {
  agent_list(
    limit: Int!,
    offset: Int!
    order_key: String,
    order_asc: Boolean,
    scaling_group: String,
    status: String,
  ): PaginatedList[Agent]
}

```

1.24 Scaling Group Management

1.24.1 Query Schema

```

type ScalingGroup {
  name: String
  description: String
  is_active: Boolean
  created_at: DateTime
  driver: String
  driver_opts: JSONString
  scheduler: String
  scheduler_opts: JSONString
}

type Query {
  scaling_group(name: String): ScalingGroup
  scaling_groups(name: String, is_active: Boolean): [ScalingGroup]
  scaling_groups_for_domain(domain: String!, is_active: Boolean): [ScalingGroup]
  scaling_groups_for_user_group(user_group: String!, is_active: Boolean): ↵
  ↵ [ScalingGroup]
  scaling_groups_for_keypair(access_key: String!, is_active: Boolean): [ScalingGroup]
}

```

1.24.2 Mutation Schema

```

input ScalingGroupInput {
  description: String
  is_active: Boolean
  driver: String!
  driver_opts: JSONString
  scheduler: String!
  scheduler_opts: JSONString
}

input ModifyScalingGroupInput {

```

(continues on next page)

(continued from previous page)

```
    description: String
    is_active: Boolean
    driver: String
    driver_opts: JSONString
    scheduler: String
    scheduler_opts: JSONString
}

type CreateScalingGroup {
    ok: Boolean
    msg: String
    scaling_group: ScalingGroup
}

type ModifyScalingGroup {
    ok: Boolean
    msg: String
}

type DeleteScalingGroup {
    ok: Boolean
    msg: String
}

type AssociateScalingGroupWithDomain {
    ok: Boolean
    msg: String
}

type AssociateScalingGroupWithKeyPair {
    ok: Boolean
    msg: String
}

type AssociateScalingGroupWithUserGroup {
    ok: Boolean
    msg: String
}

type DisassociateAllScalingGroupsWithDomain {
    ok: Boolean
    msg: String
}

type DisassociateAllScalingGroupsWithGroup {
    ok: Boolean
    msg: String
}

type DisassociateScalingGroupWithDomain {
    ok: Boolean
    msg: String
}

type DisassociateScalingGroupWithKeyPair {
    ok: Boolean
    msg: String
}
```

(continues on next page)

(continued from previous page)

```

}

type DisassociateScalingGroupWithUserGroup {
  ok: Boolean
  msg: String
}

type Mutation {
  create_scaling_group(name: String!, props: ScalingGroupInput!): CreateScalingGroup
  modify_scaling_group(name: String!, props: ModifyScalingGroupInput!): ␣
  ↪ModifyScalingGroup
  delete_scaling_group(name: String!): DeleteScalingGroup
  associate_scaling_group_with_domain(domain: String!, scaling_group: String!): ␣
  ↪AssociateScalingGroupWithDomain
  associate_scaling_group_with_user_group(scaling_group: String!, user_group: String!
  ↪): AssociateScalingGroupWithUserGroup
  associate_scaling_group_with_keypair(access_key: String!, scaling_group: String!): ␣
  ↪AssociateScalingGroupWithKeyPair
  disassociate_scaling_group_with_domain(domain: String!, scaling_group: String!): ␣
  ↪DisassociateScalingGroupWithDomain
  disassociate_scaling_group_with_user_group(scaling_group: String!, user_group: ␣
  ↪String!): DisassociateScalingGroupWithUserGroup
  disassociate_scaling_group_with_keypair(access_key: String!, scaling_group: String!
  ↪): DisassociateScalingGroupWithKeyPair
  disassociate_all_scaling_groups_with_domain(domain: String!): ␣
  ↪DisassociateAllScalingGroupsWithDomain
  disassociate_all_scaling_groups_with_group(user_group: String!): ␣
  ↪DisassociateAllScalingGroupsWithGroup
}

```

1.25 Domain Management

1.25.1 Query Schema

```

type Domain {
  name: String
  description: String
  is_active: Boolean
  created_at: DateTime
  modified_at: DateTime
  total_resource_slots: JSONString # ResourceSlot
  allowed_vfolder_hosts: [String]
  allowed_docker_registries: [String]
  integration_id: String
  scaling_groups: [String]
}

type Query {
  domain(name: String): Domain
  domains(is_active: Boolean): [Domain]
}

```

1.25.2 Mutation Schema

```
input DomainInput {
  description: String
  is_active: Boolean
  total_resource_slots: JSONString # ResourceSlot
  allowed_vfolder_hosts: [String]
  allowed_docker_registries: [String]
  integration_id: String
}

input ModifyDomainInput {
  name: String
  description: String
  is_active: Boolean
  total_resource_slots: JSONString # ResourceSlot
  allowed_vfolder_hosts: [String]
  allowed_docker_registries: [String]
  integration_id: String
}

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

type ModifyDomain {
  ok: Boolean
  msg: String
}

type DeleteDomain {
  ok: Boolean
  msg: String
}

type Mutation {
  create_domain(name: String!, props: DomainInput!): CreateDomain
  modify_domain(name: String!, props: ModifyDomainInput!): ModifyDomain
  delete_domain(name: String!): DeleteDomain
}
```

1.26 Group Management

1.26.1 Query Schema

```
type Group {
  id: UUID
  name: String
  description: String
  is_active: Boolean
  created_at: DateTime
  modified_at: DateTime
}
```

(continues on next page)

(continued from previous page)

```

    domain_name: String
    total_resource_slots: JSONString # ResourceSlot
    allowed_vfolder_hosts: [String]
    integration_id: String
    scaling_groups: [String]
}

type Query {
  group(id: String!): Group
  groups(domain_name: String, is_active: Boolean): [Group]
}

```

1.26.2 Mutation Schema

```

input GroupInput {
  description: String
  is_active: Boolean
  domain_name: String!
  total_resource_slots: JSONString # ResourceSlot
  allowed_vfolder_hosts: [String]
  integration_id: String
}

input ModifyGroupInput {
  name: String
  description: String
  is_active: Boolean
  domain_name: String
  total_resource_slots: JSONString # ResourceSlot
  user_update_mode: String
  user_uuids: [String]
  allowed_vfolder_hosts: [String]
  integration_id: String
}

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

type ModifyGroup {
  ok: Boolean
  msg: String
}

type DeleteGroup {
  ok: Boolean
  msg: String
}

type Mutation {
  create_group(name: String!, props: GroupInput!): CreateGroup
  modify_group(name: String!, props: ModifyGroupInput!): ModifyGroup
  delete_group(name: String!): DeleteGroup
}

```

(continues on next page)

(continued from previous page)

```
}
```

1.27 User Management

1.27.1 Query Schema

```
type User {
  uuid: UUID
  username: String
  email: String
  password: String
  need_password_change: Boolean
  full_name: String
  description: String
  is_active: Boolean
  created_at: DateTime
  domain_name: String
  role: String
  groups: [UserGroup]
}

type UserGroup {  # shorthand reference to Group
  id: UUID
  name: String
}

type Query {
  user(domain_name: String, email: String): User
  user_from_uuid(domain_name: String, user_id: String): User
  users(domain_name: String, group_id: String, is_active: Boolean): [User]
}
```

1.27.2 Mutation Schema

```
input UserInput {
  username: String!
  password: String!
  need_password_change: Boolean!
  full_name: String
  description: String
  is_active: Boolean
  domain_name: String!
  role: String
  group_ids: [String]
}

input ModifyUserInput {
  username: String
  password: String
  need_password_change: Boolean
  full_name: String
```

(continues on next page)

(continued from previous page)

```

description: String
is_active: Boolean
domain_name: String
role: String
group_ids: [String]
}

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

type ModifyUser {
  ok: Boolean
  msg: String
  user: User
}

type DeleteUser {
  ok: Boolean
  msg: String
}

type Mutation {
  create_user(email: String!, props: UserInput!): CreateUser
  modify_user(email: String!, props: ModifyUserInput!): ModifyUser
  delete_user(email: String!): DeleteUser
}

```

1.28 Image Management

1.28.1 Query Schema

```

type Image {
  name: String
  humanized_name: String
  tag: String
  registry: String
  digest: String
  labels: [KeyValuePair]
  aliases: [String]
  size_bytes: BigInt
  resource_limits: [ResourceLimit]
  supported_accelerators: [String]
  installed: Boolean
  installed_agents: [String] # super-admin only
}

```

```

type Query {
  image(reference: String!): Image

  images(

```

(continues on next page)

(continued from previous page)

```

    is_installed: Boolean,
    is_operation: Boolean,
    domain: String,          # only settable by super-admins
    group: String,
    scaling_group: String,   # null to take union of all agents from allowed scaling_
↪groups
  ): [Image]
}

```

The image list is automatically filtered by: 1) the allowed docker registries of the current user's domain, 2) whether at least one agent in the union of all agents from the allowed scaling groups for the current user's group has the image or not. The second condition applies only when the value of `group` is given explicitly. If `scaling_group` is not null, then only the agents in the given scaling group are checked for image availability instead of taking the union of all agents from the allowed scaling groups.

If the requesting user is a super-admin, clients may set the filter conditions as they want. In this case, setting no conditions works like v19.09 and prior versions.

New in version v5.20191215: `domain`, `group`, and `scaling_group` filters are added to the `images` root query field.

Changed in version v5.20191215: `images` query returns the images currently usable by the requesting user as described above. Previously, it returned all etcd-registered images.

1.28.2 Mutation Schema

```

type RescanImages {
  ok: Boolean
  msg: String
  task_id: String
}

type PreloadImage {
  ok: Boolean
  msg: String
  task_id: String
}

type UnloadImage {
  ok: Boolean
  msg: String
  task_id: String
}

type ForgetImage {
  ok: Boolean
  msg: String
}

type AliasImage {
  ok: Boolean
  msg: String
}

type DealiasImage {

```

(continues on next page)

(continued from previous page)

```

    ok: Boolean
    msg: String
}

type Mutation {
  rescan_images(registry: String!): RescanImages
  preload_image(reference: String!, target_agents: String!): PreloadImage
  unload_image(reference: String!, target_agents: String!): UnloadImage
  forget_image(reference: String!): ForgetImage
  alias_image(alias: String!, target: String!): AliasImage
  dealias_image(alias: String!): DealiasImage
}

```

All these mutations are only allowed for super-admins.

The query parameter `target_agents` takes a special expression to indicate a set of agents.

The mutations that returns `task_id` may take an arbitrarily long time to complete. This means that getting the response does not necessarily mean that the requested task is complete. To monitor the progress and actual completion, clients should use [the background task API](#) using the `task_id` value.

New in version v5.20191215: `forget_image`, `preload_image` and `unload_image` are added to the root mutation.

Changed in version v5.20191215: `rescan_images` now returns immediately and its completion must be monitored using the new background task API.

1.29 Compute Session Monitoring

As of Backend.AI v20.03, compute sessions are composed of one or more containers, while interactions with sessions only occur with the *master* container when using REST APIs. The GraphQL API allows users and admins to check details of sessions and their belonging containers.

Changed in version v5.20191215.

1.29.1 Query Schema

`ComputeSession` provides information about the whole session, including user-requested parameters when creating sessions.

```

type ComputeSession {
  # identity and type
  id: UUID
  name: String
  type: String
  id: UUID
  tag: String

  # image
  image: String
  registry: String
  cluster_template: String # reserved for future release

  # ownership

```

(continues on next page)

(continued from previous page)

```

domain_name: String
group_name: String
group_id: UUID
user_email: String
user_id: UUID
access_key: String
created_user_email: String # reserved for future release
created_user_uuid: UUID    # reserved for future release

# status
status: String
status_changed: DateTime
status_info: String
created_at: DateTime
terminated_at: DateTime
startup_command: String
result: String

# resources
resource_opts: JSONString
scaling_group: String
service_ports: JSONString # only available in master
mounts: List[String]      # shared by all kernels
occupied_slots: JSONString # ResourceSlot; sum of belonging containers

# statistics
num_queries: BigInt

# owned containers (aka kernels)
containers: List[ComputeContainer] # full list of owned containers

# pipeline relations
dependencies: List[ComputeSession] # full list of dependency sessions
}

```

The sessions may be queried one by one using `compute_session` field on the root query schema, or as a paginated list using `compute_session_list`.

```

type Query {
  compute_session(
    id: UUID!,
  ): ComputeSession

  compute_session_list(
    limit: Int!,
    offset: Int!,
    order_key: String,
    order_asc: Boolean,
    domain_name: String, # super-admin can query sessions in any domain
    group_id: String,    # domain-admins can query sessions in any group
    access_key: String,  # admins can query sessions of other users
    status: String,
  ): PaginatedList[ComputeSession]
}

```

`ComputeContainer` provides information about individual containers that belongs to the given session. Note that the client must assume that `id` is different from `container_id`, because agents may be configured to use non-

Docker backends.

Note: The container ID in the GraphQL queries and REST APIs are *different* from the actual Docker container ID. The Docker container IDs can be queried using `container_id` field of `ComputeContainer` objects. If the agents are configured to using non-Docker-based backends, then `container_id` may also be completely arbitrary identifiers.

```
type ComputeContainer {
  # identity
  id: UUID
  role: String      # "master" is reserved, other values are defined by cluster_
  templates
  hostname: String  # used by sibling containers in the same session
  session_id: UUID

  # image
  image: String
  registry: String

  # status
  status: String
  status_changed: DateTime
  status_info: String
  created_at: DateTime
  terminated_at: DateTime

  # resources
  agent: String      # super-admin only
  container_id: String
  resource_opts: JSONString
  # NOTE: mounts are same in all containers of the same session.
  occupied_slots: JSONString # ResourceSlot

  # statistics
  live_stat: JSONString
  last_stat: JSONString
}
```

In the same way, the containers may be queried one by one using `compute_container` field on the root query schema, or as a paginated list using `compute_container_list` for a single session.

Note: The container ID of the master container of each session is same to the session ID.

```
type Query {
  compute_container(
    id: UUID!,
  ): ComputeContainer

  compute_container_list(
    limit: Int!,
    offset: Int!,
    session_id: UUID!,
    role: String,
  ): PaginatedList[ComputeContainer]
```

(continues on next page)

(continued from previous page)

```
}
```

1.29.2 Query Example

```
query(  
  $limit: Int!,  
  $offset: Int!,  
  $ak: String,  
  $status: String,  
) {  
  compute_session_list(  
    limit: $limit,  
    offset: $offset,  
    access_key: $ak,  
    status: $status,  
  ) {  
    total_count  
    items {  
      id  
      name  
      type  
      user_email  
      status  
      status_info  
      status_updated  
      containers {  
        id  
        role  
        agent  
      }  
    }  
  }  
}
```

API Parameters

Using the above GraphQL query, clients may send the following JSON object as the request:

```
{  
  "query": "...",  
  "variables": {  
    "limit": 10,  
    "offset": 0,  
    "ak": "AKIA....",  
    "status": "RUNNING"  
  }  
}
```


API Response

```
{
  "compute_session_list": {
    "total_count": 1,
    "items": [
      {
        "id": "12c45b55-ce3c-418d-9c58-223bbba307f1",
        "name": "mysession",
        "type": "interactive",
        "user_email": "user@lablup.com",
        "status": "RUNNING",
        "status_info": null,
        "status_updated": "2020-02-16T15:47:28.997335+00:00",
        "containers": [
          {
            "id": "12c45b55-ce3c-418d-9c58-223bbba307f1",
            "role": "master",
            "agent": "i-agent01"
          },
          {
            "id": "12c45b55-ce3c-418d-9c58-223bbba307f2",
            "role": "slave",
            "agent": "i-agent02"
          },
          {
            "id": "12c45b55-ce3c-418d-9c58-223bbba307f3",
            "role": "slave",
            "agent": "i-agent03"
          }
        ]
      }
    ]
  }
}
```

1.30 Virtual Folder Management

1.30.1 Query Schema

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

(continues on next page)

(continued from previous page)

```
type Query {
  vfolder_list(
    limit: Int!,
    offset: Int!,
    order_key: String,
    order_asc: Boolean,
    domain_name: String,
    group_id: String,
    access_key: String,
  ): PaginatedList[VirtualFolder]
}
```

1.31 KeyPair Management

1.31.1 Query Schema

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

type Query {
  keypair(domain_name: String, access_key: String): KeyPair
  keypairs(domain_name: String, email: String, is_active: Boolean): [KeyPair]
}
```

1.31.2 Mutation Schema

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

input ModifyKeyPairInput {
  is_active: Boolean
}
```

(continues on next page)

(continued from previous page)

```

    is_admin: 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 Mutation {
    create_keypair(props: KeyPairInput!, user_id: String!): CreateKeyPair
    modify_keypair(access_key: String!, props: ModifyKeyPairInput!): ModifyKeyPair
    delete_keypair(access_key: String!): DeleteKeyPair
}

```

1.32 KeyPair Resource Policy Management

1.32.1 Query Schema

```

type KeyPairResourcePolicy {
    name: String
    created_at: DateTime
    default_for_unspecified: String
    total_resource_slots: JSONString # ResourceSlot
    max_concurrent_sessions: Int
    max_containers_per_session: Int
    idle_timeout: BigInt
    max_vfolder_count: Int
    max_vfolder_size: BigInt
    allowed_vfolder_hosts: [String]
}

type Query {
    keypair_resource_policy(name: String): KeyPairResourcePolicy
    keypair_resource_policies(): [KeyPairResourcePolicy]
}

```

1.32.2 Mutation Schema

```
input CreateKeyPairResourcePolicyInput {
  default_for_unspecified: String!
  total_resource_slots: JSONString!
  max_concurrent_sessions: Int!
  max_containers_per_session: Int!
  idle_timeout: BigInt!
  max_vfolder_count: Int!
  max_vfolder_size: BigInt!
  allowed_vfolder_hosts: [String]
}

input ModifyKeyPairResourcePolicyInput {
  default_for_unspecified: String
  total_resource_slots: JSONString
  max_concurrent_sessions: Int
  max_containers_per_session: Int
  idle_timeout: BigInt
  max_vfolder_count: Int
  max_vfolder_size: BigInt
  allowed_vfolder_hosts: [String]
}

type CreateKeyPairResourcePolicy {
  ok: Boolean
  msg: String
  resource_policy: KeyPairResourcePolicy
}

type ModifyKeyPairResourcePolicy {
  ok: Boolean
  msg: String
}

type DeleteKeyPairResourcePolicy {
  ok: Boolean
  msg: String
}

type Mutation {
  create_keypair_resource_policy(name: String!, props: ↪
↪CreateKeyPairResourcePolicyInput!): CreateKeyPairResourcePolicy
  modify_keypair_resource_policy(name: String!, props: ↪
↪ModifyKeyPairResourcePolicyInput!): ModifyKeyPairResourcePolicy
  delete_keypair_resource_policy(name: String!): DeleteKeyPairResourcePolicy
}
```

1.33 Resource Preset Management

1.33.1 Query Schema

```
type ResourcePreset {
  name: String
  resource_slots: JSONString
  shared_memory: BigInt
}

type Query {
  resource_preset(name: String!): ResourcePreset
  resource_presets(): [ResourcePreset]
}
```

1.33.2 Mutation Schema

```
input CreateResourcePresetInput {
  resource_slots: JSONString
  shared_memory: String
}

type CreateResourcePreset {
  ok: Boolean
  msg: String
  resource_preset: ResourcePreset
}

input ModifyResourcePresetInput {
  resource_slots: JSONString
  shared_memory: String
}

type ModifyResourcePreset {
  ok: Boolean
  msg: String
}

type DeleteResourcePreset {
  ok: Boolean
  msg: String
}

type Mutation {
  create_resource_preset(name: String!, props: CreateResourcePresetInput!): ↪CreateResourcePreset
  modify_resource_preset(name: String!, props: ModifyResourcePresetInput!): ↪ModifyResourcePreset
  delete_resource_preset(name: String!): DeleteResourcePreset
}
```

1.34 Development Setup

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

1.34.1 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.

1.34.2 Method 2: Manual Installation

Requirement packages

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

Prepare containers for external daemons

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

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

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

The directory structure would look like after finishing this guide:

- `$WORKSPACE`
 - `backend.ai`
 - `backend.ai-manager`
 - `backend.ai-agent`
 - `backend.ai-common`
 - `backend.ai-client-py`

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

This recording has been archived

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

Prepare Python 3.6+

Check out [Install Python via pyenv](#) for instructions.

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

This recording has been archived

Prepare dependent libraries

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

Prepare server-side source clones

This recording has been archived

Clone the Backend.AI source codes.

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

Inside each directory, install the sources as editable packages.

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

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

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

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

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

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

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

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

Initialize databases and load fixtures

Check out the *Prepare Databases for Manager* guide.

Prepare Kernel Images

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

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

For the full list of publicly available kernels, [check out the kernels repository](#).

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

Setting Linux capabilities to Python (Linux-only)

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

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

Running daemons from cloned sources

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

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

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

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

Prepare client-side source clones

This recording has been archived

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

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

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

1.34.3 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")'
```

1.35 Adding New Kernel Images

1.35.1 Overview

Backend.AI supports running Docker containers to execute user-requested computations in a resource-constrained and isolated environment. Most Docker container images can be *imported* as Backend.AI kernels with appropriate metadata annotations.

1. Prepare a Docker image based on Ubuntu 16.04/18.04, CentOS 7.6, or Alpine 3.8.
2. Create a Dockerfile that does:
 - Install the OpenSSL library in the image for the kernel runner (if not installed).
 - Add metadata labels.
 - Add service definition files.
 - Add a jail policy file.
3. Build a derivative image using the Dockerfile.
4. Upload the image to a Docker registry to use with Backend.AI.

1.35.2 Kernel Runner

Every Backend.AI kernel should run a small daemon called “kernel runner”. It communicates with the Backend.AI Agent running in the host via ZeroMQ, and manages user code execution and in-container service processes.

The kernel runner provides runtime-specific implementations for various code execution modes such as the query mode and the batch mode, compatible with a number of well-known programming languages. It also manages the process lifecycles of service-port processes.

To decouple the development and update cycles for Docker images and the Backend.AI Agent, we don’t install the kernel runner inside images. Instead, Backend.AI Agent mounts a special “krunner” volume as `/opt/backend.ai` inside containers. This volume includes a customized static build of Python. The kernel runner daemon package is mounted as one of the site packages of this Python distribution as well. The agent also uses `/opt/kernel` as the directory for mounting other self-contained single-binary utilities. This way, image authors do not have to bother with installing Python and Backend.AI specific software. All dirty jobs like volume deployment, its content updates, and mounting for new containers are automatically managed by Backend.AI Agent.

Since the customized Python build and binary utilities need to be built for specific Linux distributions, we only support Docker images built on top of Alpine 3.8+, CentOS 7+, and Ubuntu 16.04+ base images. Note that these three base distributions practically cover all commonly available Docker images.

Image Prerequisites

Currently Python does not officially support static-linking OpenSSL it depends on until [bpo-38794](#) is resolved. Therefore, All Docker images to be used as Backend.AI kernel images should have its own OpenSSL system packages, such as `libssl` or `openssl` depending on the distributions.

1.35.3 Metadata Labels

Any Docker image based on Alpine 3.8+, CentOS 7+, and Ubuntu 16.04+ become a Backend.AI kernel image if you add the following image labels:

- Required Labels
 - `ai.backend.kernelspec: 1` (this will be used for future versioning of the metadata specification)
 - `ai.backend.features`: A list of constant strings indicating which Backend.AI kernel features are available for the kernel.
 - * **batch**: Can execute user programs passed as files.
 - * **query**: Can execute user programs passed as code snippets while keeping the context across multiple executions.
 - * **uid-match**: As of 19.03, this must be specified always.
 - * **user-input**: The query/batch mode supports interactive user inputs.
 - `ai.backend.resource.min.*`: The minimum amount of resource to launch this kernel. At least, you must define the CPU core (`cpu`) and the main memory (`mem`). In the memory size values, you may use binary scale-suffixes such as `m` for MiB, `g` for GiB, etc.
 - `ai.backend.base-distro`: Either “ubuntu16.04” or “alpine3.8”. Note that Ubuntu 18.04-based kernels also need to use “ubuntu16.04” here.
 - `ai.backend.runtime-type`: The type of kernel runner to use. (One of the directories in [the `ai.backend.kernels` namespace](#).)

- * **python**: This runtime is for Python-based kernels, allowing the given Python executable accessible via the query and batch mode, also as a Jupyter kernel service.
 - * **app**: This runtime does not support code execution in the query/batch modes but just manages the service port processes. For custom kernel images with their own service ports for their main applications, this is the most frequently used runtime type for derivative images.
 - * For the full list of available runtime types, [check out the `lang_map` variable at the `ai.backend.kernels` module code](#)
- `ai.backend.runtime-path`: The path to the language runtime executable.
- Optional Labels
 - `ai.backend.service-ports`: A list of port mapping declaration strings for services supported by the image. (See the next section for details) Backend.AI manages the host-side port mapping and network tunneling via the API gateway automatically.
 - `ai.backend.envs.corecount`: A comma-separated string list of environment variable names. They are set to the number of available CPU cores to the kernel container. It allows the CPU core restriction to be enforced to legacy parallel computation libraries. (e.g., `JULIA_CPU_CORES`, `OPENBLAS_NUM_THREADS`)

1.35.4 Service Ports

As of Backend.AI v19.03, *service ports* are our preferred way to run computation workloads inside Backend.AI kernels. It provides tunneled access to Jupyter Notebooks and other daemons running in containers.

As of Backend.AI v19.09, Backend.AI provides SSH (including SFTP and SCP) and `tttyd` (web-based xterm shell) as intrinsic services for all kernels. “Intrinsic” means that image authors do not have to do anything to support/enable the services.

As of Backend.AI v20.03, image authors may define their own service ports using service definition JSON files installed at `/etc/backend.ai/service-defs` in their images.

Port Mapping Declaration

A custom service port should define two things. First, the image label `ai.backend.service-ports` contains the port mapping declarations. Second, the service definition file which specifies how to start the service process.

A port mapping declaration is composed of three values: the service name, the protocol, and the container-side port number. The label may contain multiple port mapping declarations separated by commas, like the following example:

```
jupyter:http:8080,tensorboard:http:6006
```

The name may be an non-empty arbitrary ASCII alphanumeric string. We use the kebab-case for it. The protocol may be one of `tcp`, `http`, and `pty`, but currently most services use `http`.

Note that there are a few port numbers reserved for Backend.AI itself and intrinsic service ports. The TCP port 2000 and 2001 is reserved for the query mode, whereas 2002 and 2003 are reserved for the native pseudo-terminal mode (`stdin` and `stdout` combined with `stderr`), 2200 for the intrinsic SSH service, and 7681 for the intrinsic `tttyd` service.

Up to Backend.AI 19.09, this was the only method to define a service port for images, and the service-specific launch sequences were all hard-coded in the `ai.backend.kernel` module.

Service Definition DSL

Now the image author should define the service launch sequences using a DSL (domain-specific language). The service definitions are written as JSON files in the container's `/etc/backend.ai/service-defs` directory. The file names must be same with the name parts of the port mapping declarations.

For example, a sample service definition file for “jupyter” service (hence its filename must be `/etc/backend.ai/service-defs/jupyter.json`) looks like:

```
{
  "prestart": [
    {
      "action": "write_tempfile",
      "args": {
        "body": [
          "c.NotebookApp.allow_root = True\n",
          "c.NotebookApp.ip = \"0.0.0.0\"\n",
          "c.NotebookApp.port = {ports[0]}\n",
          "c.NotebookApp.token = \"\"\n",
          "c.FileContentsManager.delete_to_trash = False\n"
        ]
      },
      "ref": "jupyter_cfg"
    }
  ],
  "command": [
    "{runtime_path}",
    "-m", "jupyterlab",
    "--no-browser",
    "--config", "{jupyter_cfg}"
  ],
  "url_template": "http://{host}:{port}/"
}
```

A service definition is composed of three major fields: `prestart` that contains a list of prestart actions, `command` as a list of template-enabled strings, and an optional `url_template` as a template-enabled string that defines the URL presented to the end-user on CLI or used as the redirection target on GUI with wsproxy.

The “template-enabled” strings may have references to a contextual set of variables in curly braces. All the variable substitution follows the Python’s brace-style formatting syntax and rules.

Available predefined variables

There are a few predefined variables as follows:

- **ports**: A list of TCP ports used by the service. Most services have only one port. An item in the list may be referenced using bracket notation like `{ports[0]}`.
- **runtime_path**: A string representing the full path to the runtime, as specified in the `ai.backend.runtime-path` image label.

Available prestart actions

A prestart action is composed of two mandatory fields `action` and `args` (see the table below), and an optional field `ref`. The `ref` field defines a variable that stores the result of the action and can be referenced in later parts of the service definition file where the arguments are marked as “template-enabled”.

Action Name	Arguments	Return
<code>write_file</code>	<ul style="list-style-type: none"> <code>body</code>: a list of string lines (template-enabled) <code>filename</code>: a string representing the file name (template-enabled) <code>mode</code>: an optional octal number as string, representing UNIX file permission (default: “755”) <code>append</code>: an optional boolean. If set true, open the file in the appending mode. 	None
<code>write_tempfile</code>	<ul style="list-style-type: none"> <code>body</code>: a list of string line (template-enabled) <code>mode</code>: an optional octal number as string, representing UNIX file permission (default: “755”) 	The generated file path
<code>mkdir</code>	<ul style="list-style-type: none"> <code>path</code>: the directory path (template-enabled) where parent directories are auto-created 	None
<code>run_command</code>	<ul style="list-style-type: none"> <code>command</code>: the command-line argument list as passed to <code>exec syscall</code> (template-enabled) 	A dictionary with two fields: <code>out</code> and <code>err</code> which contain the console output decoded as the UTF-8 encoding
<code>log</code>	<ul style="list-style-type: none"> <code>body</code>: a string to send as kernel log (template-enabled) <code>debug</code>: a boolean to lower the logging level to <code>DEBUG</code> (default is <code>INFO</code>) 	None

Warning: `run_command` action should return quickly, otherwise the session creation latency will be increased. If you need to run a background process, you must use its own options to let it daemonize or wrap as a background shell command (`["/bin/sh", "-c", "... &"]`).

Interpretation of URL template

`url_template` field is used by the client SDK and `wsproxy` to fill up the actual URL presented to the end-user (or the end-user’s web browser as the redirection target). So its template variables are not parsed when starting the service, but they are parsed and interpolated by the clients. There are only three fixed variables: `{protocol}`, `{host}`, and `{port}`.

Here is a sample service-definition that utilizes the URL template:

```
{
  "command": [
```

(continues on next page)

(continued from previous page)

```
    "/opt/noVNC/utils/launch.sh",
    "--vnc", "localhost:5901",
    "--listen", "{ports[0]}"
  ],
  "url_template": "{protocol}://{host}:{port}/vnc.html?host={host}&port={port}&
↪password=backendai&autoconnect=true"
}
```

1.35.5 Jail Policy

(TODO: jail policy syntax and interpretation)

Adding Custom Jail Policy

To write a new policy implementation, extend the [jail policy interface](#) in Go. Embed it inside your jail build. Please give a look to existing jail policies as good references.

1.35.6 Example: An Ubuntu-based Kernel

```
FROM ubuntu:16.04

# Add commands for image customization
RUN apt-get install ...

# Backend.AI specifics
RUN apt-get install libssl
LABEL ai.backend.kernelspec=1 \
    ai.backend.resource.min.cpu=1 \
    ai.backend.resource.min.mem=256m \
    ai.backend.envs.corecount="OPENBLAS_NUM_THREADS,OMP_NUM_THREADS,NPROC" \
    ai.backend.features="batch query uid-match user-input" \
    ai.backend.base-distro="ubuntu16.04" \
    ai.backend.runtime-type="python" \
    ai.backend.runtime-path="/usr/local/bin/python" \
    ai.backend.service-ports="jupyter:http:8080"
COPY service-defs/*.json /etc/backend.ai/service-defs/
COPY policy.yml /etc/backend.ai/jail/policy.yml
```

1.35.7 Implementation details

The query mode I/O protocol

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.

The pseudo-terminal mode protocol

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 multiplex 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.

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`