Apyfal Documentation

Release 1.2.3

Accelize

Oct 23, 2018

User Documentation

CHAPTER 1

Overview

Apyfal is a powerful and flexible toolkit that enables you to use FPGA[1] accelerated functions."

Some reasons to use Apyfal :

- Apyfal provides an abstraction layer to use the power of FPGA accelerated function in a hybrid multi-cloud environment.
- The configuration and the provisioning is generated for the FPGA cloud context.
- Apyfal can perform acceleration directly on cloud storage objects.
- Don't like Python ? Use the REST API and generate a client in any language.

All the accelerated functions

Apyfal provides a variety of accelerated functions.

Browse our web site AccelStore, to discover them.

Basic Python code example

Accelerator API is easy to use and only needs a few lines of codes to instantiate an accelerator and its host and process data:

```
import apyfal

# Choose and initialize an accelerator
with apyfal.Accelerator(accelerator='my_accelerator') as myaccel:

    # Start and configure accelerator
    myaccel.start()

    # Process data using FPGA accelerated function
    myaccel.process(src='/path/myfile1.dat',  dst='/path/result1.dat')
    myaccel.process(src='/path/myfile2.dat',  dst='/path/result2.dat')
```

[1] FPGA is a programmable chip that can be used as a function-specialized, high-performance accelerator.

CHAPTER 2

Features

- Configuration of cloud host environment
- Remote or local execution
- Cloud storage direct access from accelerator

Limitations in remote mode

- Max input data size is limited to 30GB when using client local file.
- Timeout will occur if a request takes more than 900s.

2.1 Installation

2.1.1 Requirements

Supported Python versions: 2.7, 3.4, 3.5, 3.6, 3.7

Python 3.5 or more is recommended.

Linux

On Linux, some extra packages are required:

- *Pip* is required.
- Depending on the Python version, host targeted and wheel format availability, a C/C++ compiler may also be required for install dependencies. In this case, `GCC` (Or another compatible compiler) & `Python-dev` packages are required.

Use the package manager of the Linux distribution to install packages.

RHEL or CentOS 7:

The EPEL repository is required to install packages.

`-dev` package suffix is renamed `-devel` on RHEL/CentOS.

Python 2.7 is the only Python version installed by default on RHEL/CentOS 7. But installation of Python 3.6 is possible.

Python 3:

```
sudo yum install gcc python36 python36-pip python36-devel -y
```

Use `python36` instead of `python` and `pip36` instead of `pip` to call Python and Pip from this point on.

Python 2:

```
sudo yum install gcc python-pip python-devel -y
```

Debian or Ubuntu:

Python 3: (*Debian 8 Jessie*/*Ubuntu 14.04 Trusty* and more)

Python 3 packages are prefixed `python3-` instead of `python-`.

```
sudo apt-get install gcc python3-pip python3-dev
```

Use `python3` instead of `python` and `pip3` instead of `pip` to call Python and Pip from this point on.

Python 2:

```
sudo apt-get install gcc python-pip python-dev
```

Windows

Python for Windows is available on the Python Website.

Depending on the Python version, the host targeted, and wheel format availability, a C/C++ compiler may also be required to install dependencies.

- See Windows Compilers on Python documentation for more information.

2.1.2 Setup

All installation is performed using PIP.

The base package with all features and AWS support can be installed with:

```
pip install apyfal
```

Some extra host type are supported as optional components.

You can also install these optional extras:

- `all`: Install all extras.
- `Alibaba`: Requirements for Alibaba.

- `AWS`: Requirements for AWS (Installed by default).
- `OpenStack`: Requirements for OpenStack.
- `OVH`: Requirements for OVH.

Example for installing the `all` extra:

```
pip install apyfal[all]
```

Example for installing the `OpenStack` + `Alibaba` extras:

```
pip install apyfal[OpenStack,Alibaba]
```

2.2 Configuration

Warning: do not use credentials on untrusted environments. You are responsible for the secure use of your account.

2.2.1 What is needed to configure an accelerator ?

Two main configuration steps must be performed before running an accelerator

Accelize account

The first part is your Accelize account details (login and password), which are required to unlock the accelerator:

- Accelize credential

Your AccelStore account also provides metering information about your accelerator use

Host configuration

Accelerator needs an host with FPGA device that needs to be configured to run.

See *Getting Started* to see examples of possible cases.

2.2.2 Accelerator configuration

The accelerator can be configured either by using the configuration file or by passing parameter information to the API directly.

Using the configuration File

You can use `accelerator.conf` file to provide parameters to run your accelerator.

For more information on the configuration file, see:

Apyfal configuration file

The configuration file is used to provide configuration information to Apyfal.

Use of this file is optional. All parameters can also be passed as arguments to Apyfal classes and functions.

All parameters or section in this file can be removed without issue. Non specified or missing parameters use default values. See your targeted accelerator's documentation for information about which parameters are needed for the accelerator and host.

The API automatically searches for "accelerator.conf" in the current working directory, or in the current user home directory.

A custom path to a configuration file can also be passed to classes or with the `APYFAL_CONFIG_FILE` environment variable.

Read the "Getting Started" documentation for examples of parameters use.

This configuration file can set any parameter as `parameter =value` in the related section, even if not specified in this example file. Note that parameter names are case sensitive.

Subsections

Some sections support *subsections*.

A subsection is a section that overrides parameters of a parent section. If a parameter is not specified in a subsection, it uses the parameter values of its parent.

Subsection have the following format `[section.subsection]` (With `[section]` meaning the parent section of this subsection).

See each section's documentation to see if subsections are supported and how to use them.

The "accelize" section

This section sets the account details for the Accelize server and the access keys created via: https://accelstore.accelize.com/user/applications

Client and Secret ID from Accelize account page.

`client_id =`

`secret_id =`

The "host" section

This section contains all the information related to the host used to deploy the accelerator.

This section support subsections based on `host_type` parameter. You can add parameters to the `[host.host_type]` subsection to override the `[host]` section parameters for the specified `host_type`.

Host type

Possible values: `Alibaba`, `AWS`, `OpenStack`, `OVH`

`host_type =`

Host AccessKey/account details for cloud hosts.

`client_id =`

`secret_id =`

Host region for cloud hosts A region that supports instances with FPGA devices is required.

Required only for: Alibaba, AWS, OpenStack, OVH

Possible values for:

- *Alibaba:* `cn-hangzhou`
- *AWS:* `eu-west-1`, `us-east-1`, `us-west-2`
- *OVH:* `GRA3`

`region =`

OpenStack Project ID

Required only for: OpenStack, OVH

`project_id =`

OpenStack Auth-URL

Required only for: OpenStack

`auth_url =`

IP address to use as to define accelerator URL.

Possible values:

- `True` = *Use private IP by default.*
- `False` = *Use public IP by default.*

`use_private_ip =`

SSL certificate. Used to:

- Allows to access accelerator using HTTPS instead of HTTP when configuring a new host.
- Allows client to verify HTTPS connection using this certificate.

`ssl_cert_crt =`

Generates a self signed certificate and save it on path specified by `ssl_cert_crt` and `ssl_cert_key` before configuring host.

Possible values: `True`, `False`

`ssl_cert_generate =`

Defining host side configuration

SSL certificate private key. Allows to access accelerator using HTTPS instead of HTTP when configuring a new host.

`ssl_cert_key =`

Configuration file to use host side.

`init_config =`

Bash script to execute on instance startup.

`init_script =`

Using a Pre-Existing Host Instance

Host `client_id` and `secret_id` are optional if one of these values is provided.

Instance ID of the cloud host instance to use.

`instance_id =`

IP address of the host to use.

`host_ip =`

Host instance stopping Behavior

Defines the way the host instance is to be stopped in case of object closure, a script end or a function stop call.

Possible values:

- `term` = *Instance will be deleted.*
- `stop` = *Instance will be stopped and can be restarted.*
- `keep` = *Instance will stay in running mode.*

Optionally, override the following default values:

- *term: if a new instance has been started*
- *keep: if using a pre-existing instance*

`stop_mode =`

Overriding default host environment value

All of these parameters are optional.

Prefix to add before instance name

`host_name_prefix =`

Host Key pair to use

`key_pair =`

Host Security group to use

`security_group =`

Authorization role (AWS IAM role/Alibaba RAM role). Generated to allow an instance to load FPGA bitstream and access to storage

Only for: Alibaba, AWS

`role =`

Authorization policy (AWS IAM policy/Alibaba RAM policy). Generated to allow an instance to load FPGA bitstream and access to storage

Only for: Alibaba, AWS

`policy =`

The "configuration" section

This section enables *specific configuration parameters* for the accelerator configuration step to be passed. These parameters will act as default values. Any parameter passed to the "start" method will override the values defined here.

Read your targeted accelerator's documentation for information about the possible specific parameter values.

You can add parameters to the `[configuration.accelerator]` subsection to override the `[configuration]` section parameters for the specified accelerator.

Examples of parameters use (From some existing accelerators):

nbBytes = 1024 type= "sha1"

This section also enables parameters that use the *JSON parameters file* format to be passed using the following `parameters` value:

parameters =

The "process" section

This section works exactly like the "configuration" section but for *specific process parameters*.

The "security" section

This section configure Apyfal client/host communication security options

Directories that can be processed remotely on host using `host://` Apyfal storage URL (default to `~/shared`). Support multiple paths (On per line):

Only available host side: See `init_config` in `host` section to pass a configuration file to host.

`authorized_host_dirs =`

The "storage" section

This section contains all the information related Cloud storage.

Cloud storage configuration is not shared between client and host. A proper configuration file needs to be passed to the host with all required storage parameters to allow storage access from `process` and `start` methods. See `init_config` in `host` section to pass a configuration file to host.

This section support subsections based on `storage_type`. You can add parameters to the `[storage.storage_type]` subsection to override the `[storage]` section parameters for the specified `storage_type`.

Theses subsections also support subsection to handle multiple storage using the same storage_type, in this case the subsection name is `[storage.storage_type.name]`

If an `host` section or sub-section is defined for a `host_type` equal to the `storage_type`, parameters are get from this section if not found in the storage section. This allow to not repeat definition of access key or other parameters in both sections.

Disable TLS/SSL/HTTPS. If True (default) disables TLS/SSL/HTTPS for transfer. This can improve performance, but makes connection insecure.

Possible values: `True`, `False`

`unsecure =`

This file is automatically loaded by the API if found in the current working directory or current user home directory. A custom path can also be passed as an argument to the API.

`accelerator.conf example file`.

Passing Parameters to Apyfal

The use of the configuration file is not mandatory; all parameters can be passed directly to the API as arguments. Please read the API documentation for more information.

See *Apyfal API* for more information.

If both the configuration file and arguments are used to configure an accelerator, configuration by arguments override configuration file values.

2.3 Getting Started

This section explains how to use Apyfal with Python to run accelerators.

All of these examples require you to first install the Apyfal and to have an Accelize account (the `accelize_client_id` and `accelize_secret_id` parameters in following examples).

See *Installation* and *Configuration* for more information.

You also need the name of the accelerator you want to use (The `accelerator` parameter in following example)

See AccelStore for more information.

The examples below use configuration by arguments for clarity, but you can also set them using the configuration file.

You can enable the Apyfal logger to see more details about each step that's running. This is particularly useful for when running tests or going through examples:

```
import apyfal
apyfal.get_logger(True)
```

2.3.1 Running an accelerator remotely on a cloud instance host

This tutorial will describe how to create a simple accelerator and process a file using a Cloud Service Provider (*CSP*) as a host.

The parameters required in this case may depend on the CSP used, but will always include:

- `host_type`: CSP name
- `region`: CSP region name (a region that supports FPGA is required).
- `client_id` and `secret_id`: CSP account details

See *apyfal.host* for information about potential parameters of the targeted CSP.

See your CSP documentation for information about how to obtain these values.

```
# Import the accelerator module.
import apyfal

# Choose an accelerator to use and configure it.
with apyfal.Accelerator(
```

(continues on next page)

(continued from previous page)

```
        # Accelerator parameters
        accelerator='my_accelerator',
        # host parameters
        host_type='my_provider', region='my_region',
        client_id='my_client_id', secret_id='my_secret_id',
        # Accelize parameters
        accelize_client_id='my_accelize_client_id',
        accelize_secret_id='my_accelize_secret_id') as myaccel:

    # Start the accelerator:
    # A new cloud instance will be created and your account details passed
    # to Accelerator as host
    # Note: This step can take some minutes, depending on the CSP
    myaccel.start()

    # Process data:
    # Define which data to process and where they should be stored.
    myaccel.process(src='/path/myfile1.dat', dst='/path/result1.dat')
    myaccel.process(src='/path/myfile2.dat', dst='/path/result2.dat')
    # ... It is possible to process any number of data

# The accelerator is automatically closed  on "with" exit.
# In this case, the default stop_mode ('term') is used:
# the previously created host will be deleted and all its content lost.
```

Keeping host running

Starting a host takes a long time, so it may be a good idea to keep it running for later use.

You can do this using the `stop_mode` parameter.

Depending on your CSP, additional fees may apply based on the host running time. Don't forget to terminate your cloud instance after use.

```
import apyfal

with apyfal.Accelerator(
        accelerator='my_accelerator',
        host_type='my_provider', region='my_region',
        client_id='my_client_id', secret_id='my_secret_id',
        accelize_client_id='my_accelize_client_id',
        accelize_secret_id='my_accelize_secret_id') as myaccel:

    # We can start the accelerator in &quot;keep&quot; stop mode to keep the
    # host running
    myaccel.start(stop_mode='keep')

    myaccel.process(src='/path/myfile.dat', dst='/path/result.dat')

    # We can get and store the host IP and instance ID for later use
    my_host_instance_id = myaccel.host.instance_id
    my_host_ip = myaccel.host.host_ip

# This time the host is not deleted and will stay running when the
# accelerator is closed.
```

Reusing an Existing Host

With host instance ID and full host access

With `instance_id`, depending on your CSP, you can reuse an already existing host without providing the `client_id` and `secret_id`.

An accelerator started with `instance_id` keeps control of the host and can stop it at any time.

```
import apyfal

# We select the host to use on Accelerator instantiation
# with its instance ID stored previously
with apyfal.Accelerator(
        accelerator='my_accelerator',
        host_type='my_provider', region='my_region',
        # Use 'instance_id' and removed 'client_id' and 'secret_id'
        instance_id='my_host_instance_id',
        accelize_client_id='my_accelize_client_id',
        accelize_secret_id='my_accelize_secret_id') as myaccel:

    myaccel.start()

    myaccel.process(src='/path/myfile.dat', dst='/path/result.dat')
```

With Host IP with Accelerator-Only Access

With `host_ip`, you can reuse an already existing host without providing any other host information.

An accelerator started with `host_ip` has no control over the host and can't stop it.

```
import apyfal

# We also can select the host to use on Accelerator instantiation
# with its IP address stored previously
with apyfal.Accelerator(
        accelerator='my_accelerator',
        # Use 'host_ip' and removed any other host parameter
        host_ip='my_host_ip',
        accelize_client_id='my_accelize_client_id',
        accelize_secret_id='my_accelize_secret_id') as myaccel:

    myaccel.start()

    myaccel.process(src='/path/myfile.dat', dst='/path/result.dat')
```

2.3.2 Running an Accelerator Locally

This tutorial describes using an accelerator locally on an already-configured FPGA host.

Requirements

An already-configured host is required to use this feature.

You can easily create a cloud instance using *Apyfal* and keep the host running using the `stop_mode='keep'`; parameter. See above for more information.

Don't forget to terminate the cloud instance after use to avoid additional fees.

You connect to your host using SSH:

- `key_pair` is the key pair name that can be obtained with `myaccel.host.key_pair`. The related private key file in `.pem` format is generally stored in the `.ssh` sub folder of user home.
- `host_ip` is the IP address of the instance and can be obtained with `myaccel.host.host_ip`.

Linux:

```
ssh -Yt -i ~/.ssh/${key_pair}.pem centos@${host_ip}
```

Windows:

On Windows, you can use Putty to connect with SSH. The private key file needs to be in `.ppk` format (`puttygen.exe`, supplied with Putty, can convert `.pem` to `.ppk`).

```
putty.exe -ssh centos@%host_ip% 22 -i %userprofile%\.ssh\%key_pair%.ppk
```

Running Apyfal

Running Apyfal in this case is straightforward as the accelerator is preconfigured:

- By default, the `accelize_client_id` and `accelize_secret_id` values are those used when creating an instance. You can change them by passing other values.
- `accelerator` value is the one used when creating an instance and cannot be changed.
- Host related arguments are not required and don't have any effect (`stop_mode`, `host_ip`, etc)

```
import apyfal

with apyfal.Accelerator() as myaccel:

    myaccel.start()

    myaccel.process(src='/path/myfile.dat', dst='/path/result.dat')
```

2.3.3 Configuring accelerators

Some accelerators require configuration before being run. An accelerator is configured using the `start` and `process` methods.

Configuration step: the `start` method

Parameters passed to `start` apply to every `process` calls that follows.

You can call `start` again to change parameters.

The `start` parameters is divided in two parts:

- The `src` argument: Some accelerators may require a data to run. Read the accelerator documentation to see the data to use.

- The `**parameters` argument(s): Parameters are *specific configuration parameters* that are passed as keyword arguments. See the accelerator documentation for more information about possible *specific configuration parameters*. Any value passed to this argument overrides the default configuration values.

```
import apyfal

with apyfal.Accelerator(accelerator='my_accelerator') as myaccel:

    # The parameters are passed to "start" to configure the accelerator;
    # theses parameters are:
    # - src: The path to "src1.dat" data.
    # - parameter1, parameter2: Keywords parameters are passed to
    #   "**parameters" arguments.
    myaccel.start(src='/path/src1.dat',
                  parameter1='my_parameter_1', parameter2='my_parameter_2')

    # Every "process" call after start uses the previously specified
    # parameters to perform processing
    myaccel.process(src='/path/myfile1.dat', dst='/path/result1.dat')
    myaccel.process(src='/path/myfile2.dat', dst='/path/result2.dat')
    # ...

    # It is possible to re-call "start" method with other parameters
    myaccel.start(src='/path/src2.dat')

    # The following "process" will use new parameters.
    myaccel.process(src='/path/myfile3.dat', dst='/path/result3.dat')
    # ...
```

Process step: the `process` method

Parameters passed to `process` applies only to this `process` call.

The `process` method accept the following arguments:

- `src`: Input data. Check the accelerator documentation to see if an input data is required.
- `dst`: Output data. Check the accelerator documentation to see if an output data is required.
- The `**parameters` argument(s): Parameters are *specific configuration parameters* that are passed as keyword arguments. See the accelerator documentation for more information about possible *specific configuration parameters*. Any value passed to this argument overrides the default configuration values.

```
import apyfal

with apyfal.Accelerator(accelerator='my_accelerator') as myaccel:
    myaccel.start()

    # The parameters are passed to "process" to configure it;
    # theses parameters are:
    # - parameter1, parameter2: Keywords parameters are passed to
    #   "**parameters" arguments.
    myaccel.process(src='/path/myfile1.dat', dst='/path/result1.dat',
                    parameter1='my_parameter_1', parameter2='my_parameter_2')
```

The `process` method wait result from accelerator before return.

The `process_submit` is a non blocking asynchronous equivalent of `process`. This method returns a `concurrent.futures.Future` object to handle the result and can be used to request multiple processing tasks

in parallel to reduce the data transfer and network overhead.

Note that the hardware accelerated processing itself is exclusive and take no benefit of the use of parallels tasks.

```
import apyfal

data_list = ['/path/myfile1', '/path/myfile2', '/path/myfile3']

with apyfal.Accelerator(accelerator='my_accelerator') as myaccel:
    myaccel.start()

    # Submit asynchronous processing tasks for a list of data
    futures = [myaccel.process_submit(src=my_data)
               for my_data in data_list]

    # All Processing tasks are performed in parallel.
    # It is now possible to wait and get results from "Future" objects.
    results = [future.result() for future in futures]
```

A `process_map` function also exists to submit directly iterables to process.

```
import apyfal

data_list = ['/path/myfile1', '/path/myfile2', '/path/myfile3']

with apyfal.Accelerator(accelerator='my_accelerator') as myaccel:
    myaccel.start()

    # This performs the previous example in only one line
    results = myaccel.process_map(srcs=data_list)
```

2.3.4 Metering information

Using Accelerators consumes "coins" based on the amount of processed data. You can access your metering information via your AccelStore account.

2.4 Advanced use

2.4.1 Storage

Apyfal storage provides the ability to use cloud storage services and other storage as a source and target for an accelerator.

Using this feature to handle files provides some advantages:

- When using Apyfal to remotely control an accelerator, all file transfers are performed on the host directly.
- Apyfal storage uses simple URLs like `str` to define files.
- Apyfal storage can also be used to handle basic copy or open operations on storage services files.

Apyfal storage URL format

Apyfal storage works with extended URLs that support extra schemes.

Apyfal use the standard URL format `scheme://path`.

Basic schemes

Apyfal storage supports the following basic schemes.

- `file`: Local file on file system (`file` scheme is assumed if no scheme provided). Example: `file:///home/user/myfile` or `/home/user/myfile`
- `http`/`https`: File available on HTTP/HTTPS. Example: `http://www.accelize.com/file` or `https://www.accelize.com/file`

Cloud storage schemes

Apyfal storage supports extra schemes for cloud storage services. In this case, the scheme is the service name or the provider name.

Cloud storage use buckets (or containers) to hold data. The bucket name needs to be specified just before the file path in URL.

See *apyfal.storage* for information available storage services and the scheme to use.

Example:

- `s3://my_bucket/my_file`: File with `my_file` key on AWS S3 `my_bucket` bucket.
- `ovh://my_container/my_file`: File with `my_file` name on OVH Object Store `my_container` container.

The `host` scheme

The `host` scheme is similar to the `file` scheme, but is only available when using Apyfal to control the accelerator remotely.

In this case:

- `file` represents a client-side file that needs to be transferred to/from host.
- `host` represents a host-side file that can be used directly.

For security reason, `host` scheme is restricted to a whitelisted list of directories on host. This list can be modified, host side, using the `authorized_host_dirs` of the `security` section in the configuration file. The only default authorized directory is `~/shared`.

Using storage with Accelerator

`apyfal.Accelerator` has native Apyfal storage URL support for file parameters:

```
import apyfal

with apyfal.Accelerator(accelerator='my_accelerator') as myaccel:

    myaccel.start(src='my_storage://src')

    myaccel.process(src='my_storage://src', dst='my_storage://dst')
```

Basic storage operations

Apyfal storage provides some basic file operation functions to easily manipulate storage files:

- `apyfal.storage.open`: Open a file as file-like object. Like builtin `open`.
- `apyfal.storage.copy`: Copy a file between two URL. Like `shutil.copy`.

The following example shows some possible file operations:

```
import apyfal.storage

# Open file as text for reading
with apyfal.storage.open('my_storage://my_file', 'rt') as file:
    text = file.read()

# Open file as binary for writing
with apyfal.storage.open('my_storage://my_file', 'wb') as file:
    file.write(b'binary_data')

# Copy file from storage to local file system
copy('my_storage://my_file', 'my_file')

# Copy file from local file system to storage
copy('my_file', 'my_storage://my_file')

# Copy file between storage
copy('my_storage://my_file', 'my_other_storage://my_file')

# Download a file from internet to storage
copy('http://www.accelize.com/file', 'my_storage://my_file')
```

Mount extra storage services

Cloud storage services use a login and password to secure access and can't be accessed without them.

By default, storage services that are already configured as host are automatically mounted with same parameters.

But, in other cases, these services need to be mounted before use. Each storage needs a unique `storage_type` that will be used to mount it and to access it with a URL.

This can be done using the `apyfal.storage.mount` function or with the configuration file.

See *apyfal.storage* for information on possible parameters for the targeted storage.

The following examples show the registration of the `my_storage` storage type. This storage needs the following parameters to be mounted: `client_id` and `secret_id`.

With mount function

The registration of `my_storage` storage is performed as follows.

```
import apyfal.storage

# Mount "my_provider.my_bucket" storage
apyfal.storage.mount(storage_type='my_storage',
                     client_id='my_client_id', secret_id='my_secret_id')
```

With configuration file

The registration of `my_storage` storage is performed by adding a `storage` subsection to the configuration file containing storage parameters.

```
[storage.my_storage]
client_id  = my_client_id
secret_id  = my_secret_id
```

See *Configuration* for more information on the configuration file.

2.4.2 JSON configuration files

This section provides more information about parameter use than that described in the *Getting Started*.

JSON is not the recommanded way to send parameters to Apyfal

Apyfal provides JSON support to allow compatibility with JSON used with Accelerator CLI.

The low-level accelerator API that runs on the FPGA host works with parameters files.

These files are JSON files that have the following format:

```
{
    "app": {
        "specific":{
        # Specific parameters as key value pairs.
        }
    }
}
```

See the accelerator documentation for specific parameters values.

Using `**parameters` argument with JSON parameters files

The `**parameters` argument passed to the `start` and `process` methods, as described previously, can also be used to pass *JSON parameters files*. In this case, `**parameters` is used as `parameters=`

Assuming `parameters.json` is the JSON parameters files:

- To pass the `parameters.json` file, simply pass its path: `parameters='/path/parameters.json'`.
- To pass the `parameters.json` content as JSON `str` literal: `parameters=parameters_json_content`.
- To pass the `dict` equivalent of `parameters.json`: `parameters=parameters_json_content_as_dict`.

`parameters=` can be used with classical `**parameters` keywords arguments. In this case, the keywords arguments override values already existing in the dict passed to `parameters=`.

```
import apyfal

with apyfal.Accelerator(accelerator='my_accelerator') as myaccel:
    myaccel.start()

    # Example of passing the parameter JSON file and keyword arguments
    myaccel.process(src='/path/myfile1.dat', dst='/path/result1.dat',
                    # Passing Path to JSON file to "parameters="
                    parameters='/path/parameters.json',
```

(continues on next page)

(continued from previous page)

```
                    # Passing keywords arguments
                    parameter1='my_parameter_1', parameter2='my_parameter_2')
```

Using JSON parameters files with the configuration file

JSON parameters files can also be defined directly in `accelerator.conf`. Parameters in configuration files will act as default values and will be overridden by any parameter passed directly to the `start` and `process` methods.

See *Configuration* for more information.

2.4.3 Security

This chapter detail security configuration in Apyfal.

Apyfal configuration file

The Apyfal configuration file may contain some access keys. This file needs to be stored and exchanged with care to not compromises theses access keys.

Cloud hosts security configuration

By default, Apyfal configures some cloud instances settings to improves security.

SSH key pair

By default, Apyfal generates an SSH key pair to give user access to the instance using SSH. The private key file is automatically added in PEM format to the `.ssh` directory of your user home.

A custom key pair can be used using the `key_pair` argument.

An example of SSH connexion is given in *Getting Started*

Security groups

Security groups are like firewalls for cloud instance and are used to allow only a limited range of IP address to connect to the host using a limited range of ports.

By default, Apyfal creates a security group that allows only the machine used to generate the host instance to access it using SSH and HTTP/HTTPS (ports `22`, `80` and `443`).

A custom security group can be used using the `security_group` argument.

Before host instantiation, the range of ports can be modified in the `ALLOW_PORTS` host class attribute.

Security groups rules can also be modified using the CSP console.

Host local storage

Apyfal allows to process file that are stored locally on host instance. By default, this is limited to an unique directory for security reason.

This behavior can be change with the `authorized_host_dirs` parameter of the `security` section in the host side configuration file.

See *Storage* for more information.

SSL certificate

The SSL/TLS certificate allows to access host over HTTPS instead of HTTP.

Certificate selection and generation

Apyfal allows to define a certificate to use, or to generates automatically a certificate.

Using an user defined certificate: A certificate can to be passed to the host creation using `ssl_cert_crt` and `ssl_cert_key` parameters.

Generating a self signed wildcard certificate: `ssl_cert_generate` parameter can be used to generate a self signed certificate.

If `ssl_cert_generate` is used without `ssl_cert_crt` and `ssl_cert_key`, a generic certificate is generated in the `.ssh` directory of the user home. If the generic certificate exists, it is reused and not regenerated.

Certificate verification

The certificate needs to be verified to provides security.

Classical DNS host name based certificate This kind of certificate is verified as usual using a Certificate Authority.

Self signed wildcard certificate Certificate Authority cannot verify this kind of certificate., but Apyfal can verify the connexion against a specified certificate.

Once a client have the `ssl_cert_crt` specified or a generic certificate is found in the `.ssh` directory, connections are verified using this certificate.

Host name is not verified when using a wildcard certificate.

Disabling HTTPS and using HTTP

If `ssl_cert_crt` is specified or if a generic certificate is found in the `.ssh` directory, HTTPS is enabled.

`ssl_cert_crt` can be set to `False` to disable HTTPS.

Usage scenarios

Host has a DNS host name: Use a Certificate Authority to generate a certificate for this DNS host name and define `ssl_cert_crt` and `ssl_cert_key` to this certificate files.

This is the most secure and recommended way, but it require to bind hosts IP address to an host name.

Host has no DNS host name and is shared between its creator and other users: Creator generate a self signed certificate by defining `ssl_cert_crt`, `ssl_cert_key` and `ssl_cert_generate` and share the public certificate file with other users. Others user set `ssl_cert_crt` to this certificate file to enable connection verification with it.

Host has no DNS host name and is used only by its creator for a single session: Creator generate a generic self signed certificate by defining only `ssl_cert_generate`.

Using HTTP to improve data transfer speed between client and host. Set `ssl_cert_generate` to `False` to be sure using HTTP instead of HTTPS.

Note that with this option, transferred data and Accelize access keys are publicly visible to anyone on the network.

Host side configuration file

The `init_config` parameter allows to pass Apyfal configuration file to host instance, this can be used to configure some parameters on host.

On this parameter use, always transfer a cleaned up configuration file to your instance to avoid the risk of compromises your access keys.

CSP specific parameters

Some CSP provides more security options, refer to each host class for more information.

Cloud storage security configuration

Some CSP that provides both computes and storage services allows to configure host instance to access storage. By default, Apyfal allows access to all storage to accelerator cloud instance.

But, this can be modified, refers to each host class for more information.

2.4.4 Managing multiple accelerators

This chapter explain how to manage multiples accelerators.

The Accelerator iterator

The accelerator iterator allows to iterate over all already running accelerators. It can found every host defined in the configuration file.

The iterated accelerators can normally performs all operations provided by the accelerator class.

The following example shows how to stop all existing accelerators:

```
import apyfal

for accelerator in apyfal.iter_accelerators():
    accelerator.stop('term')
```

The iterator also provides filters to select accelerators to iterates based on any accelerator property.

This example shows how to list all accelerator IP address of a specific host type.

```
import apyfal

addresses = [accelerator.public_ip for accelerator
             in apyfal.iter_accelerators(host_type='my_provider')]
```

The Accelerator pool executor

The `apyfal.AcceleratorPoolExecutor` is an object inspired by `concurrent.futures` that allows to submit processing task to a pool of multiple accelerator.

It works like the `apyfal.Accelerator` in asynchronous mode and provides the same `process_submit` and `process_map` methods. The difference is the use of a pool of accelerator instead of only one accelerator.

Tasks are submitted between accelerators in order to balance the load.

All accelerators in a pool are identical and are created using the same parameters.

Unlike the single accelerator, the pool executor allows to perform the hardware accelerated processing in parallel.

```
import apyfal

data_list = ['/path/myfile1', '/path/myfile2', '/path/myfile3']

# Instantiates all accelerators in parallel
with apyfal.AcceleratorPoolExecutor(accelerator='my_accelerator') as executor:

    # Starts all accelerators in parallel with same parameters
    executor.start()

    # Submits tasks between to the accelerator pools
    results = executor.process_map(srcs=data_list)
```

2.4.5 Starting a custom host

This section explain how to customize the Accelerator host to run.

Overriding specific host parameters

Some hosts provides the ability to override some parameters of the lower level libraries used to run them.

For each host class, overriding parameters are represented by arguments of the `dict` type with a name that finish by `kwargs`.

The content of theses dict are specific to each case and their description can be found in the target library documentation.

Example with AWS EC2. The AWS host provides some `dict` arguments to customize the `boto3` client used to run AWS instances:

- `boto3_session_kwargs`: Overrides arguments of the boto3.session.Session class.
- `boto3_client_kwargs`: Overrides arguments of the boto3.session.Session.client method.
- `boto3_create_instances_kwargs`: Overrides arguments of the boto3.session.Session.resource('ec2').create_instances method.

With this it is by example possible to change billing options when creating the instance. Example with the use of AWS spot instance:

```
import apyfal

 # Based on AWS documentation, create a dict containing arguments to pass
 # to the "create_instances" method.
 use_spot_instance = {
     'InstanceMarketOptions': {
         'MarketType': 'spot',
         'SpotOptions': {
             'MaxPrice': '0.75',
             'SpotInstanceType': 'persistent',
             'InstanceInterruptionBehavior': 'stop'
         }
     }
 }

# On Accelerator instantiation, pass the dict to the
# "boto3_create_instances_kwargs" argument.
with apyfal.Accelerator(
        accelerator='my_accelerator', host_type='AWS',
        boto3_create_instances_kwargs=use_spot_instance) as myaccel:

    # Continue the use of the accelerator as normal.
    myaccel.start()
```

Starting host manually

The following explain how start an host without the use of Apyfal.

This is not the recommended way to instantiate an accelerator, but, this allow to use a more custom host configuration.

Getting host requirements

The host configuration differ depending the target host. It is possible to get the configuration to use with Apyfal.

```
import apyfal

# Load Apyfal "accelerator.conf" configuration file
config = apyfal.configuration.Configuration()

# Configuration file needs at least Accelize access keys, if not present in
# file, it is possible to add it programmatically:
config['accelize']['client_id'] = 'my_client_id'
config['accelize']['secret_id'] = 'my_secret_id'

# Get configuration from Accelize server
host_config = config.get_host_requirements(
    host_type='my_provider', accelerator='my_accelerator')
```

The `host_config` is a dictionary that contain all required information to configure the host. Depending the `host_type`, this dictionary may contain sub-dictionaries representing sub-categories like the CSP `region`.

Following values may be found inside it (Depending the `host_type`):

- `image`: Host virtual machine image to use on CSP.

- `instancetype`: The type/flavor of instance to use on CSP.
- `fpgaimage`: The FPGA device bitstream image. This value may be ignored and will be configured automatically in following steps.

Instantiating host

The host can now be instantiated and started using previously retrieved parameters. This step is different for each `host_type` and will not be explained here. Read your host provider documentation for more information.

Configuring accelerator on host

Once the host is started and ready to use, connect to it using SSH:

```
ssh -Yt -i ~/.ssh/${key_pair}.pem centos@${host_ip}
```

Create the `/home/centos/accelerator.conf` file and complete it with at least:

- `client_id` and `secret_id` in `accelize` section
- `host_type` and `region` in `host` section.

Then, run Apyfal CLI to set initial configuration of the FPGA device:

```
apyfal create --accelerator my_accelerator
apyfal start
```

The Accelerator is now ready to use.

2.5 FAQ and Troubleshooting

This page list some frequently asked questions and troubleshooting.

If you encounter technical issues not listed here, don't hesitate to open an issue on GitHub. For any other question, please contact Accelize.

2.5.1 Apyfal installation

I get errors when trying to install Apyfal with Pip, Host to fix this ? Theses kind of errors may appear on some outdated Python environment. Update Python environment is the recommended way to fix theses issues.

Multiple answer are available:

Using a dedicated virtual environment:

This allows to have a separate environment for each use.

To install and use a virtual environments, refer to following documentations:

- On any Python versions with Pipenv (Equivalent to `pip` + `venv` in one command).
- On Python 3 with venv standard library module.
- On Python 2 with Virtualenv module.

Updating all dependencies packages on current environment:

Firstly upgrade Pip to the last version

```
python -m pip install -U pip
```

Then, when installing (or updating) Apyfal add following arguments to the `pip install` command:

- `--upgrade --upgrade-strategy eager`: Upgrade all dependencies to the last version.
- `--ignore-installed`: Completely reinstall all packages. To use in case the previous arguments are not sufficient.

2.5.2 Cloud service providers

AWS: How to create access keys ? See "Managing Access Keys for IAM Users" on AWS documentation.

AWS: How to run more Accelerator instances on AWS ? Request an adjustment of the limit of AWS EC2 F1 instances you can launch to AWS support (0 by default).

OVH: How to create access keys ? See "Configure user access to Horizon" on OVH documentation.

2.6 Apyfal API

This section describes the `apyfal` Python package API.

Apyfal

Copyright 2018 Accelize

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

class apyfal.**Accelerator**(*accelerator=None*, *config=None*, *accelize_client_id=None*, *accelize_secret_id=None*, *host_type=None*, *host_ip=None*, *stop_mode=None*, *prefer_self_hosted=None*, ***host_kwargs*)

This class provides the full accelerator features by handling Accelerator and its host.

Parameters

- **accelerator** (*str*) – Name of the accelerator to initialize, to know the accelerator list please visit "https://accelstore.accelize.com".
- **config** (apyfal.configuration.Configuration, *path-like object or file-like object*) – If not set, will search it in current working directory, in current user "home" folder. If none found, will use default configuration values. Path-like object can be path, URL or cloud object URL.
- **accelize_client_id** (*str*) – Accelize Client ID. Client ID is part of the access key generated on "https:/accelstore.accelize.com/user/applications".
- **accelize_secret_id** (*str*) – Accelize Secret ID. Secret ID come with xlz_client_id.
- **host_type** (*str*) – Type of host to use.

- **host_ip** (*str*) – IP or URL address of an already existing host to use. If not specified, create a new host.
- **stop_mode** (*str or int*) – Host stop mode. Default to 'term' if new host, or 'keep' if already existing host. See "apyfal.host.Host.stop_mode" property for more information and possible values.
- **prefer_self_hosted** (*bool*) – Default to True. If True, and current machine is an accelerator host, operates this accelerator instead of instantiating a new accelerator host. If current machine is not an accelerator host, instantiates a new host normally.
- **host_kwargs** – Keyword arguments related to specific host. See targeted host class to see full list of arguments.

client
Accelerator client.

Returns Accelerator client

Return type *apyfal.client.AcceleratorClient*

host
Accelerator host.

Returns Host

Return type apyfal.host.Host subclass

process(*src=None*, *dst=None*, *info_dict=None*, ***parameters*)
Processes with accelerator.

Parameters

- **src** (*path-like object or file-like object*) – Source data to process. Path-like object can be path, URL or cloud object URL.
- **dst** (*path-like object or file-like object*) – Processed data destination. Path-like object can be path, URL or cloud object URL.
- **parameters** (*path-like object, str or dict*) – Accelerator process specific parameters Can also be a full process parameters dictionary (Or JSON equivalent as str literal) Parameters dictionary override default configuration values, individuals specific parameters overrides parameters dictionary values. Take a look to accelerator documentation for more information on possible parameters. Path-like object can be path, URL or cloud object URL.
- **info_dict** (*dict or None*) – If a dict passed, this dict is updated with extra information from current operation.

Returns Result from process operation, depending used accelerator.

process_map(*srcs=None*, *dsts=None*, *timeout=None*, *info_list=None*, ***parameters*)
Map process execution on multiples files.

Parameters

- **srcs** (*iterable of path-like object or file-like object*) – Iterable of input data to process. Must be an iterable of "src" parameters of the "process" method. Path-like object can be path, URL or cloud object URL.
- **dsts** (*iterable of path-like object or file-like object*) – Iterable of output data. Must be an iterable of "dst" parameters of the "process" method. Path-like object can be path, URL or cloud object URL.

- **timeout** (*float*) – The maximum number of seconds to wait. If None, then there is no limit on the wait time.
- **parameters** (*path-like object, str or dict*) – Accelerator process specific parameters Can also be a full process parameters dictionary (Or JSON equivalent as str literal) Parameters dictionary override default configuration values, individuals specific parameters overrides parameters dictionary values. Take a look to accelerator documentation for more information on possible parameters. Path-like object can be path, URL or cloud object URL.
- **info_list** (*list*) – If a list passed, this list is updated with "info_dict" extra information dicts for each process operation.

Returns Results.

Return type generator

Raises `concurrent.futures.TimeoutError` – "timeout" reached on at least one task.

process_running_count
Return number of asynchronous process tasks running or pending.

Returns count.

Return type int

process_submit(*src=None*, *dst=None*, *info_dict=None*, ***parameters*)
Schedules the process operation to be executed and returns a Future object representing the execution.

See "apyfal.Accelerator.process"

Parameters

- **src** (*path-like object or file-like object*) – Source data to process. Path-like object can be path, URL or cloud object URL.
- **dst** (*path-like object or file-like object*) – Processed data destination. Path-like object can be path, URL or cloud object URL.
- **parameters** (*path-like object, str or dict*) – Accelerator process specific parameters Can also be a full process parameters dictionary (Or JSON equivalent as str literal) Parameters dictionary override default configuration values, individuals specific parameters overrides parameters dictionary values. Take a look to accelerator documentation for more information on possible parameters. Path-like object can be path, URL or cloud object URL.
- **info_dict** (*dict or None*) – If a dict passed, this dict is updated with extra information from current operation. The dict will be updated on task completion.

Returns

Future object representing execution. See "apyfal.Accelerator.process" method for "Future.result()" content.

Return type concurrent.futures.Future

start(*stop_mode=None*, *src=None*, *info_dict=None*, *host_env=None*, *reload=None*, *reset=None*, ***parameters*)
Starts and/or configure an accelerator.

Parameters

- **stop_mode** (*str or int*) – Host stop mode. If not None, override current "stop_mode" value. See "apyfal.host.Host.stop_mode" property for more information and possible values.

- **src** (*path-like object or file-like object*) – Depending on the accelerator, a configuration data need to be loaded before a process can be run. Path-like object can be path, URL or cloud object URL.
- **parameters** (*str, path-like object or dict*) – Accelerator configuration specific parameters Can also be a full configuration parameters dictionary (Or JSON equivalent as str literal or apyfal.storage URL to file) Parameters dictionary override default configuration values, individuals specific parameters overrides parameters dictionary values. Take a look to accelerator documentation for more information on possible parameters. Path-like object can be path, URL or cloud object URL.
- **reload** (*bool*) – Force reload of FPGA bitstream.
- **reset** (*bool*) – Force reset of FPGA logic.
- **info_dict** (*dict or None*) – If a dict passed, this dict is updated with extra information from operation.
- **host_env** (*dict*) – Overrides Accelerator "env".

stop(*stop_mode=None*, *info_dict=None*)

Stop accelerator session and accelerator host depending of the parameters

Parameters

- **stop_mode** (*str or int*) – Host stop mode. If not None, override current "stop_mode" value. See "apyfal.host.Host.stop_mode" property for more information and possible values.
- **info_dict** (*dict or None*) – If a dict passed, this dict is updated with extra information from current operation.

class apyfal.**AcceleratorPoolExecutor**(*accelerator=None*, *config=None*, *accelize_client_id=None*, *accelize_secret_id=None*, *host_type=None*, *stop_mode='term'*, *workers_count=4*, ***host_kwargs*)

An executor that uses a pool of workers_count identically configured accelerator to execute calls asynchronously.

This class provides the full accelerator features by handling Accelerator and its host.

Parameters

- **accelerator** (*str*) – Name of the accelerator to initialize, to know the accelerator list please visit "https://accelstore.accelize.com".
- **config** (apyfal.configuration.Configuration, *path-like object or file-like object*) – If not set, will search it in current working directory, in current user "home" folder. If none found, will use default configuration values. Path-like object can be path, URL or cloud object URL.
- **accelize_client_id** (*str*) – Accelize Client ID. Client ID is part of the access key generated on "https:/accelstore.accelize.com/user/applications".
- **accelize_secret_id** (*str*) – Accelize Secret ID. Secret ID come with xlz_client_id.
- **host_type** (*str*) – Type of host to use.
- **stop_mode** (*str or int*) – Host stop mode. Default to 'term' if new host, or 'keep' if already existing host. See "apyfal.host.Host.stop_mode" property for more information and possible values.
- **workers_count** (*int*) – Number of accelerator workers.

- **host_kwargs** – Keyword arguments related to specific host. See targeted host class to see full list of arguments.

accelerators
Accelerator workers.

Returns Accelerators

Return type list of apyfal.Accelerator

clients
Accelerator workers clients.

Returns Clients

Return type list of apyfal.client.AcceleratorClient

hosts
Accelerator workers hosts.

Returns Hosts

Return type list of apyfal.host.Host subclass

process_map(*srcs=None*, *dsts=None*, *timeout=None*, *info_list=None*, ***parameters*)
Map process execution on multiples files.

Parameters

- **srcs** (*iterable of path-like object or file-like object*) – Iterable of input data to process. Must be an iterable of "src" parameters of the "process" method. Path-like object can be path, URL or cloud object URL.
- **dsts** (*iterable of path-like object or file-like object*) – Iterable of output data. Must be an iterable of "dst" parameters of the "process" method. Path-like object can be path, URL or cloud object URL.
- **timeout** (*float*) – The maximum number of seconds to wait. If None, then there is no limit on the wait time.
- **parameters** (*path-like object, str or dict*) – Accelerator process specific parameters Can also be a full process parameters dictionary (Or JSON equivalent as str literal) Parameters dictionary override default configuration values, individuals specific parameters overrides parameters dictionary values. Take a look to accelerator documentation for more information on possible parameters. Path-like object can be path, URL or cloud object URL.
- **info_list** (*list*) – If a list passed, this list is updated with "info_dict" extra information dicts for each process operation.

Returns Results.

Return type generator

Raises `concurrent.futures.TimeoutError` – "timeout" reached on at least one task.

process_submit(*src=None*, *dst=None*, *info_dict=None*, ***parameters*)
Schedules the process operation to be executed and returns a Future object representing the execution.

See "apyfal.Accelerator.process".

Parameters

- **src** (*path-like object or file-like object*) – Source data to process. Path-like object can be path, URL or cloud object URL.

- **dst** (*path-like object or file-like object*) – Processed data destination. Path-like object can be path, URL or cloud object URL.
- **parameters** (*path-like object, str or dict*) – Accelerator process specific parameters Can also be a full process parameters dictionary (Or JSON equivalent as str literal) Parameters dictionary override default configuration values, individuals specific parameters overrides parameters dictionary values. Take a look to accelerator documentation for more information on possible parameters. Path-like object can be path, URL or cloud object URL.
- **info_dict** (*dict or None*) – If a dict passed, this dict is updated with extra information from current operation. The dict will be updated on task completion.

Returns

Future object representing execution. See "apyfal.Accelerator.process" method for "Future.result()" content.

Return type concurrent.futures.Future

start(*stop_mode=None*, *src=None*, *host_env=None*, *reload=None*, *reset=None*, *info_list=None*, ***parameters*)

Starts and/or configure all accelerators in the pool.

Parameters

- **stop_mode** (*str or int*) – Host stop mode. If not None, override current "stop_mode" value. See "apyfal.host.Host.stop_mode" property for more information and possible values.
- **src** (*path-like object or file-like object*) – Depending on the accelerator, a configuration data need to be loaded before a process can be run. Path-like object can be path, URL or cloud object URL.
- **parameters** (*str, path-like object or dict*) – Accelerator configuration specific parameters Can also be a full configuration parameters dictionary (Or JSON equivalent as str literal or apyfal.storage URL to file) Parameters dictionary override default configuration values, individuals specific parameters overrides parameters dictionary values. Take a look to accelerator documentation for more information on possible parameters. Path-like object can be path, URL or cloud object URL.
- **reload** (*bool*) – Force reload of FPGA bitstream.
- **reset** (*bool*) – Force reset of FPGA logic.
- **host_env** (*dict*) – Overrides Accelerator "env".
- **info_list** (*list*) – If a list passed, this list is updated with "info_dict" extra information dicts for each accelerator.

Returns List of "Accelerator.start" results.

Return type list

stop(*stop_mode=None*, *wait=True*, *info_list=None*)

Signal the executor that it should free any resources that it is using when the currently pending futures are done executing. Calls to process_submit() and process_map() made after shutdown will raise RuntimeError.

Stop each accelerator session and accelerator host depending of the parameters

Parameters

- **stop_mode** (*str or int*) – Host stop mode. If not None, override current "stop_mode" value. See "apyfal.host.Host.stop_mode" property for more information and possible values.
- **wait** (*bool*) – Waits stop completion before return.
- **info_list** (*list*) – If a list passed, this list is updated with "info_dict" extra information dicts for each accelerator.

Returns

List of "Accelerator.stop" results if "info_dict", else list of Futures objects.

Return type list

apyfal.**iter_accelerators**(*config=None*, *host_name_prefix=True*, ***filters*)

Iterates over all accelerators available on remote hosts.

Parameters

- **config** (apyfal.configuration.Configuration, *path-like object or file-like object*) – If not set, will search it in current working directory, in current user "home" folder. If none found, will use default configuration values. Path-like object can be path, URL or cloud object URL.
- **host_name_prefix** (*bool or str*) – If True, use "host_name_prefix" from configuration; if False don't filter by prefix; if str, uses this str as prefix
- **filters** – Arguments names are host properties to filter, values are regular expressions.

Returns Accelerators generator

Return type generator

apyfal.**get_logger**(*stdout=False*, *debug=False*)

Initialize logger

Parameters

- **stdout** (*bool*) – If True, configure logger to print on stdout, else use NullHandler
- **debug** (*bool*) – If True, set logger level to Debug and show information useful to debug.

Returns logger instance

Return type logging.Logger

2.6.1 apyfal.client

Accelerator Client

class apyfal.client.**AcceleratorClient**(*accelerator=None*, *client_type=None*, *accelize_client_id=None*, *accelize_secret_id=None*, *config=None*, ***_*)

REST accelerator client.

Parameters

- **accelerator** (*str*) – Name of the accelerator to initialize, to know the accelerator list please visit "https://accelstore.accelize.com".
- **client_type** (*str*) – Type of client.
- **accelize_client_id** (*str*) – Accelize Client ID. Client ID is part of the access key generate from "https:/accelstore.accelize.com/user/applications".

- **accelize_secret_id** (*str*) – Accelize Secret ID. Secret ID come with client_id.
- **config** (apyfal.configuration.Configuration, *path-like object or file-like object*) – If not set, will search it in current working directory, in current user "home" folder. If none found, will use default configuration values. Path-like object can be path, URL or cloud object URL.

DEFAULT_AUTHORIZED_HOST_DIRS = ['~/shared']
Default directories that can be processed remotely on host

DEFAULT_CONFIGURATION_PARAMETERS = {'app': {'reset': False, 'reload': True, 'enable
Default parameters JSON for configuration/start stage

DEFAULT_PROCESS_PARAMETERS = {'app': {'reset': False, 'enable-sw-comparison': 0, 'l
Default parameters JSON for process stage

as_tmp_file(*url*, *mode*)
Return temporary representation of a file.

Parameters

- **url** (*str*) – apyfal.storage URL of the file.
- **mode** (*str*) – Access mode. 'r' or 'w'.

Returns temporary object.

Return type str or file-like object

name
Accelerator name

Returns name

Return type str

process(*src=None*, *dst=None*, *info_dict=None*, ***parameters*)
Processes with accelerator.

Parameters

- **src** (*path-like object or file-like object*) – Source data to process. Path-like object can be path, URL or cloud object URL.
- **dst** (*path-like object or file-like object*) – Processed data destination. Path-like object can be path, URL or cloud object URL.
- **parameters** (*path-like object, str or dict*) – Accelerator process specific parameters Can also be a full process parameters dictionary (Or JSON equivalent as str literal) Parameters dictionary override default configuration values, individuals specific parameters overrides parameters dictionary values. Take a look to accelerator documentation for more information on possible parameters. Path-like object can be path, URL or cloud object URL.
- **info_dict** (*dict or None*) – If a dict passed, this dict is updated with extra information from current operation.

Returns Result from process operation, depending used accelerator.

start(*src=None*, *info_dict=None*, *host_env=None*, *reload=None*, *reset=None*, ***parameters*)
Configures accelerator.

Parameters

- **src** (*path-like object or file-like object*) – Depending on the accelerator, a configuration data need to be loaded before a process can be run. Path-like object can be path, URL or cloud object URL.
- **parameters** (*str, path-like object or dict*) – Accelerator configuration specific parameters Can also be a full configuration parameters dictionary (Or JSON equivalent as str literal or apyfal.storage URL to file) Parameters dictionary override default configuration values, individuals specific parameters overrides parameters dictionary values. Take a look to accelerator documentation for more information on possible parameters. Path-like object can be path, URL or cloud object URL.
- **reload** (*bool*) – Force reload of FPGA bitstream.
- **reset** (*bool*) – Force reset of FPGA logic.
- **info_dict** (*dict or None*) – If a dict passed, this dict is updated with extra information from current operation.
- **host_env** (*dict*) – Overrides Accelerator "env".

stop(*info_dict=None*, *full_stop=True*)
Stop accelerator.

Parameters

- **full_stop** (*bool*) – If True, send stop request to accelerator application. If False only clean up accelerator client environment.
- **info_dict** (*dict or None*) – If a dict passed, this dict is updated with extra information from current operation.

apyfal.client.rest

Accelerator REST client.

This client allows remote accelerator control.

class apyfal.client.rest.**RESTClient**(*accelerator=None*, *host_ip=None*, *ssl_cert_crt=None*, **args*, ***kwargs*)
Remote Accelerator OpenAPI REST client.

Parameters

- **accelerator** (*str*) – Name of the accelerator to initialize, to know the accelerator list please visit "https://accelstore.accelize.com".
- **accelize_client_id** (*str*) – Accelize Client ID. Client ID is part of the access key generate from "https:/accelstore.accelize.com/user/applications".
- **accelize_secret_id** (*str*) – Accelize Secret ID. Secret ID come with client_id.
- **host_ip** (*str*) – IP or URL address of the accelerator host.
- **ssl_cert_crt** (*path-like object or file-like object or bool*) – Public ".crt" key file of the SSL ssl_cert_key used by host to provides HTTPS. If provided, the ssl_cert_key is verified on each request. If not provided, search for a generated certificate. If False, disable HTTPS.
- **config** (apyfal.configuration.Configuration, *path-like object or file-like object*) – If not set, will search it in current working directory, in current user "home" folder. If none found, will use default configuration values. Path-like object can be path, URL or cloud object URL.

NAME = 'REST'
Client type

as_tmp_file(*url*, *mode*)
Return temporary representation of a file.

Parameters

- **url** (*str*) – apyfal.storage URL of the file.
- **mode** (*str*) – Access mode. 'r' or 'w'.

Returns temporary object.

Return type str or file-like object

name
Accelerator name

Returns name

Return type str

process(*src=None*, *dst=None*, *info_dict=None*, ***parameters*)
Processes with accelerator.

Parameters

- **src** (*path-like object or file-like object*) – Source data to process. Path-like object can be path, URL or cloud object URL.
- **dst** (*path-like object or file-like object*) – Processed data destination. Path-like object can be path, URL or cloud object URL.
- **parameters** (*path-like object, str or dict*) – Accelerator process specific parameters Can also be a full process parameters dictionary (Or JSON equivalent as str literal) Parameters dictionary override default configuration values, individuals specific parameters overrides parameters dictionary values. Take a look to accelerator documentation for more information on possible parameters. Path-like object can be path, URL or cloud object URL.
- **info_dict** (*dict or None*) – If a dict passed, this dict is updated with extra information from current operation.

Returns Result from process operation, depending used accelerator.

ssl_cert_crt
SSL Certificate of the accelerator host.

Returns Path to ssl_cert_key.

Return type str

start(*src=None*, *info_dict=None*, *host_env=None*, *reload=None*, *reset=None*, ***parameters*)
Configures accelerator.

Parameters

- **src** (*path-like object or file-like object*) – Depending on the accelerator, a configuration data need to be loaded before a process can be run. Path-like object can be path, URL or cloud object URL.
- **parameters** (*str, path-like object or dict*) – Accelerator configuration specific parameters Can also be a full configuration parameters dictionary (Or JSON equivalent as str literal or apyfal.storage URL to file) Parameters dictionary override default con-

figuration values, individuals specific parameters overrides parameters dictionary values. Take a look to accelerator documentation for more information on possible parameters. Path-like object can be path, URL or cloud object URL.

- **reload** (*bool*) – Force reload of FPGA bitstream.
- **reset** (*bool*) – Force reset of FPGA logic.
- **info_dict** (*dict or None*) – If a dict passed, this dict is updated with extra information from current operation.
- **host_env** (*dict*) – Overrides Accelerator "env".

stop(*info_dict=None*, *full_stop=True*)
Stop accelerator.

Parameters

- **full_stop** (*bool*) – If True, send stop request to accelerator application. If False only clean up accelerator client environment.
- **info_dict** (*dict or None*) – If a dict passed, this dict is updated with extra information from current operation.

url
URL of the accelerator host.

Returns URL

Return type str

apyfal.client.syscall

Accelerator system call client.

class apyfal.client.syscall.**SysCallClient**(*host_type=None*, *region=None*, **args*, ***kwargs*)
Accelerator client.

Parameters

- **accelize_client_id** (*str*) – Accelize Client ID. Client ID is part of the access key generate from "https:/accelstore.accelize.com/user/applications".
- **accelize_secret_id** (*str*) – Accelize Secret ID. Secret ID come with client_id.
- **host_type** (*str*) – Type of the current host.
- **region** (*str*) – Region of the current host.
- **config** (apyfal.configuration.Configuration, *path-like object or file-like object*) – If not set, will search it in current working directory, in current user "home" folder. If none found, will use default configuration values. Path-like object can be path, URL or cloud object URL.

APYFAL_MINIMUM_VERSION = '1.0.0'
Apyfal minimum compatible client version

NAME = 'SysCall'
Client type

as_tmp_file(*url*, *mode*)
Return temporary representation of a file.

Parameters

- **url** (*str*) – apyfal.storage URL of the file.
- **mode** (*str*) – Access mode. 'r' or 'w'.

Returns temporary object.

Return type str or file-like object

name
Accelerator name

Returns name

Return type str

process(*src=None*, *dst=None*, *info_dict=None*, ***parameters*)
Processes with accelerator.

Parameters

- **src** (*path-like object or file-like object*) – Source data to process. Path-like object can be path, URL or cloud object URL.
- **dst** (*path-like object or file-like object*) – Processed data destination. Path-like object can be path, URL or cloud object URL.
- **parameters** (*path-like object, str or dict*) – Accelerator process specific parameters Can also be a full process parameters dictionary (Or JSON equivalent as str literal) Parameters dictionary override default configuration values, individuals specific parameters overrides parameters dictionary values. Take a look to accelerator documentation for more information on possible parameters. Path-like object can be path, URL or cloud object URL.
- **info_dict** (*dict or None*) – If a dict passed, this dict is updated with extra information from current operation.

Returns Result from process operation, depending used accelerator.

start(*src=None*, *info_dict=None*, *host_env=None*, *reload=None*, *reset=None*, ***parameters*)
Configures accelerator.

Parameters

- **src** (*path-like object or file-like object*) – Depending on the accelerator, a configuration data need to be loaded before a process can be run. Path-like object can be path, URL or cloud object URL.
- **parameters** (*str, path-like object or dict*) – Accelerator configuration specific parameters Can also be a full configuration parameters dictionary (Or JSON equivalent as str literal or apyfal.storage URL to file) Parameters dictionary override default configuration values, individuals specific parameters overrides parameters dictionary values. Take a look to accelerator documentation for more information on possible parameters. Path-like object can be path, URL or cloud object URL.
- **reload** (*bool*) – Force reload of FPGA bitstream.
- **reset** (*bool*) – Force reset of FPGA logic.
- **info_dict** (*dict or None*) – If a dict passed, this dict is updated with extra information from current operation.
- **host_env** (*dict*) – Overrides Accelerator "env".

stop(*info_dict=None*, *full_stop=True*)
Stop accelerator.

Parameters

- **full_stop** (*bool*) – If True, send stop request to accelerator application. If False only clean up accelerator client environment.
- **info_dict** (*dict or None*) – If a dict passed, this dict is updated with extra information from current operation.

2.6.2 apyfal.host

FPGA Host

class apyfal.host.**Host**(*host_type=None*, *config=None*, *host_ip=None*, *stop_mode=None*, *host_name_prefix=None*, **_)
This is base class for all host classes.

This is also a factory which instantiate host subclass related to specified cases.

Parameters

- **host_type** (*str*) – Host type.
- **config** (apyfal.configuration.Configuration, *path-like object or file-like object*) – If not set, will search it in current working directory, in current user "home" folder. If none found, will use default configuration values. Path-like object can be path, URL or cloud object URL.
- **host_name_prefix** (*str*) – Prefix to add to host name.
- **host_ip** (*str*) – IP or URL address of an already existing host to use. If not specified, create a new host.
- **stop_mode** (*str or int*) – Define the "stop" method behavior. Default to 'term' if new host, or 'keep' if already existing host. See "stop_mode" property for more information and possible values.

DOC_URL = ''
Link to host documentation or website

STOP_MODES = {0: 'term', 1: 'stop', 2: 'keep'}
Possible stop_mode int values

TIMEOUT = 420.0
Timeout for host status change in seconds

get_configuration_env(***config_env*)
Return environment to pass to "apyfal.accelerator.AcceleratorClient.start" "csp_env" argument.

Parameters **config_env** – Overwrites environment values.

Returns Configuration environment.

Return type dict

host_name
Name of the current host.

Returns Name

Return type str

host_type
Host type

Returns Host type

Return type str

info
Returns some host information.

Returns

Dictionary containing information on current host.

Return type dict

iter_hosts(*host_name_prefix=True*)
Iterates over accelerator hosts of current type.

Parameters **host_name_prefix** (*bool or str*) – If True, use "host_name_prefix" from configuration, if False don't filter by prefix, if str, uses this str as prefix

Returns dicts contains attributes values of the host.

Return type generator of dict

start(*accelerator=None*, *accel_parameters=None*, *stop_mode=None*)
Start host if not already started.

Needs "accel_client" or "accel_parameters".

Parameters

- **accelerator** (*str*) – Name of the accelerator.
- **accel_parameters** (*dict*) – Can override parameters from accelerator client.
- **stop_mode** (*str or int*) – See "stop_mode" property for more information.

stop(*stop_mode=None*)
Stop host accordingly with the current stop_mode. See "stop_mode" property for more information.

Parameters **stop_mode** (*str or int*) – If not None, override current "stop_mode" value.

stop_mode
Define the "stop" method behavior.

Possible values ares: 0: TERM, terminate and delete host. 1: STOP, stop and pause host. 2: KEEP, let host running.

"stop" can be called manually but is also called when:

- "with" exit if class is used as context manager.
- On object deletion by garbage collector.
- On OS signals if "exit_host_on_signal" was set to True on class instantiation.

Returns stop mode.

Return type str

url
URL of the current host.

Returns URL

Return type str

apyfal.host.alibaba

Alibaba Cloud ECS

Security: RAM Role and policy: By default, the ECS instance is configured to use a RAM role and policy generated by Apyfal on first instantiation.

This give access in read and write to all OSS Buckets. This behavior can be changed by overriding RAM Role and/or policy.

This can be done using by:

- Selecting an already existing role or policy on class instantiation (using `role` or `policy` argument).
- Updating policy and role documents in class attributes before instantiating the class for the first time(`POLICY_DOCUMENT` and `ASSUME_ROLE_POLICY_DOCUMENT` class attributes).
- Modifying policy or role parameters generated by Apyfal in RAM console.

Notes that FPGA related rules in policy are mandatory to any Accelerator use and needs to be present in custom policy.

Apyfal create the default RAM rules and policy only if missing and will not overwrite existing.

apyfal.host.alibaba.**API_VERSION = {'ecs': '2014-05-26', 'ram': '2015-05-01', 'sts': '201**
Alibaba Cloud API version (Key is subdomain name, value is API version)

class apyfal.host.alibaba.**AlibabaCSP**(*role=None*, *policy=None*, *acs_client_kwargs=None*, *acs_create_instance_kwargs=None*, ***kwargs*)

Alibaba Cloud CSP Class

Parameters

- **host_type** (*str*) – Cloud service provider name. (Default to "Alibaba").
- **config** (apyfal.configuration.Configuration, *path-like object or file-like object*) – If not set, will search it in current working directory, in current user "home" folder. If none found, will use default configuration values. Path-like object can be path, URL or cloud object URL.
- **client_id** (*str*) – Alibaba Access Key ID.
- **secret_id** (*str*) – Alibaba Secret Access Key.
- **region** (*str*) – Alibaba region. Needs a region supporting instances with FPGA devices.
- **instance_type** (*str*) – Alibaba instance type. Default defined by accelerator.
- **key_pair** (*str*) – Alibaba Key pair. Default to 'AccelizeAlibabaKeyPair'.
- **security_group** – Alibaba Security group. Default to'AccelizeSecurityGroup'.
- **instance_id** (*str*) – Instance ID of an already existing Alibaba ECS instance to use. If not specified, create a new instance.
- **host_name_prefix** (*str*) – Prefix to add to instance name. Also used as value of the "Apyfal" tag.
- **host_ip** (*str*) – IP or URL address of an already existing Alibaba ECS instance to use. If not specified, create a new instance.
- **use_private_ip** (*bool*) – If True, on new instances, uses private IP instead of public IP as default host IP.

- **role** (*str*) – Alibaba RAM role. Generated to allow instance to load FPGA bitstream and access to OSS. Default to 'AccelizeRole'.
- **policy** (*str*) – Alibaba RAM policy. Generated to allow instance to load FPGA bitstream and access to OSS. Default to 'AccelizePolicy'.
- **stop_mode** (*str or int*) – Define the "stop" method behavior. Default to 'term' if new instance, or 'keep' if already existing instance. See "stop_mode" property for more information and possible values.
- **init_config** (*bool or apyfal.configuration.Configuration, path-like object or file-like object*) – Configuration file to pass to instance on initialization. This configuration file will be used as default for host side accelerator. If value is True, use 'config' configuration. If value is a configuration use this configuration. If value is None or False, don't passe any configuration file (This is default behavior).
- **init_script** (*path-like object or file-like object*) – A bash script to execute on instance startup.
- **ssl_cert_crt** (*path-like object or file-like object or bool*) – Public ".crt" key file of the SSL ssl_cert_key used to provides HTTPS. If not specified, uses already generated certificate if found. If False, disable HTTPS.
- **ssl_cert_key** (*path-like object or file-like object*) – Private ".key" key file of the SSL ssl_cert_key used to provides HTTPS. If not specified, uses already generated key if found.
- **ssl_cert_generate** (*bool*) – Generate a self signed ssl_cert_key. The ssl_cert_key and private key will be stored in files specified by "ssl_cert_crt" and "ssl_cert_key" (Or temporary certificates if not specified). Note that this ssl_cert_key is only safe if other client verify it by providing "ssl_cert_crt". No Certificate Authority are available to trust this ssl_cert_key.
- **acs_client_kwargs** (*dict*) – Extra keyword arguments for aliyunsdkcore.client.AcsClient.
- **acs_create_instance_kwargs** (*dict*) – Extra keyword arguments for "CreateInstance" ECS function.

ASSUME_ROLE_POLICY_DOCUMENT = {'Statement': [{'Effect': 'Allow', 'Principal': {'Ser
RAM Assume Role Policy Document – Grant rights defined by policy to role.

DOC_URL = 'https://www.alibabacloud.com'
Alibaba Website

NAME = 'Alibaba'
Provider name

POLICY_DOCUMENT = {'Statement': [{'Effect': 'Allow', 'Resource': ['acs:ecs:*:*:*'],
RAM Policy document – Grant FPGA Access and OSS Buckets access

get_configuration_env(***config_env*)
Return environment to pass to "apyfal.accelerator.AcceleratorClient.start" "csp_env" argument.

Parameters **config_env** – Overwrites environment values.

Returns Configuration environment.

Return type dict

host_ip
Host IP of the current instance. This may return public or private IP based on configuration.

Returns IP address

Return type str

Raises *apyfal.exceptions.HostRuntimeException* – No instance from which get IP.

host_name
Name of the current host.

Returns Name

Return type str

host_type
Host type

Returns Host type

Return type str

info
Returns some host information.

Returns

Dictionary containing information on current host.

Return type dict

instance_id
ID of the current instance.

Returns ID

Return type str

iter_hosts(*host_name_prefix=True*)
Iterates over accelerator hosts of current type.

Parameters **host_name_prefix** (*bool or str*) – If True, use "host_name_prefix" from configuration, if False don't filter by prefix, if str, uses this str as prefix

Returns dicts contains attributes values of the host.

Return type generator of dict

key_pair
SSH Key pair linked to this instance.

Returns Name of key pair.

Return type str

private_ip
Private IP of the current instance.

Returns IP address

Return type str

Raises *apyfal.exceptions.HostRuntimeException* – No instance from which get IP.

public_ip
Public IP of the current instance.

Returns IP address

Return type str

Raises *apyfal.exceptions.HostRuntimeException* – No instance from which get IP.

ssl_cert_crt
SSL ssl_cert_key used.

Returns Path to SSL ssl_cert_key.

Return type str

start(*accelerator=None*, *accel_parameters=None*, *stop_mode=None*)
Start instance if not already started. Create instance if necessary.

Needs "accel_client" or "accel_parameters".

Parameters

- **accelerator** (*str*) – Name of the accelerator.
- **accel_parameters** (*dict*) – Can override parameters from accelerator client.
- **stop_mode** (*str or int*) – See "stop_mode" property for more information.

stop(*stop_mode=None*)
Stop instance accordingly with the current stop_mode. See "stop_mode" property for more information.

Parameters **stop_mode** (*str or int*) – If not None, override current "stop_mode" value.

stop_mode
Define the "stop" method behavior.

Possible values ares: 0: TERM, terminate and delete host. 1: STOP, stop and pause host. 2: KEEP, let host running.

"stop" can be called manually but is also called when:

- "with" exit if class is used as context manager.
- On object deletion by garbage collector.
- On OS signals if "exit_host_on_signal" was set to True on class instantiation.

Returns stop mode.

Return type str

url
URL of the current host.

Returns URL

Return type str

apyfal.host.alibaba.**MAIN_DOMAIN = 'aliyuncs.com'**
Alibaba Cloud API main domain

apyfal.host.aws

Amazon Web Services EC2

Security: IAM Role and policy: By default, the EC2 instance is configured to use a IAM role and policy generated by Apyfal on first instantiation.

This give access in read and write to all S3 Buckets. This behavior can be changed by overriding IAM Role and/or policy.

This can be done using by:

- Selecting an already existing role or policy on class instantiation (using `role` or `policy` argument).
- Updating policy and role documents in class attributes before instantiating the class for the first time (`POLICY_DOCUMENT` and `ASSUME_ROLE_POLICY_DOCUMENT` class attributes).
- Modifying policy or role parameters generated by Apyfal in AIM console.

Notes that FPGA related rules in policy are mandatory to any Accelerator use and needs to be present in custom policy.

Apyfal create the default AIM rules and policy only if missing and will not overwrite existing.

class apyfal.host.aws.**AWSHost**(*role=None*, *policy=None*, *boto3_session_kwargs=None*, *boto3_client_kwargs=None*, *boto3_create_instances_kwargs=None*, ***kwargs*)

AWS EC2 CSP

Parameters

- **host_type** (*str*) – Cloud service provider name. Default to "AWS".
- **config** (apyfal.configuration.Configuration, *path-like object or file-like object*) – If not set, will search it in current working directory, in current user "home" folder. If none found, will use default configuration values. Path-like object can be path, URL or cloud object URL.
- **client_id** (*str*) – AWS Access Key ID.
- **secret_id** (*str*) – AWS Secret Access Key.
- **region** (*str*) – AWS region. Needs a EC2 region supporting instances with FPGA devices.
- **instance_type** (*str*) – AWS EC2 Instance type. Default defined by accelerator.
- **key_pair** (*str*) – AWS Key pair. Default to 'AccelizeAWSKeyPair'.
- **security_group** – AWS Security group. Default to 'AccelizeSecurityGroup'.
- **instance_id** (*str*) – Instance ID of an already existing AWS EC2 instance to use. If not specified, create a new instance.
- **host_name_prefix** (*str*) – Prefix to add to instance name. Also used as value of the "Apyfal" tag.
- **host_ip** (*str*) – IP or URL address of an already existing AWS EC2 instance to use. If not specified, create a new instance.
- **use_private_ip** (*bool*) – If True, on new instances, uses private IP instead of public IP as default host IP.
- **role** (*str*) – AWS IAM role. Generated to allow instance to load AGFI (FPGA bitstream) and access to S3. Default to 'AccelizeRole'.
- **policy** (*str*) – AWS IAM policy. Generated to allow instance to load AGFI (FPGA bitstream) and access to S3. Default to 'AccelizePolicy'.

- **stop_mode** (*str or int*) – Define the "stop" method behavior. Default to 'term' if new instance, or 'keep' if already existing instance. See "stop_mode" property for more information and possible values.
- **init_config** (*bool or apyfal.configuration.Configuration, path-like object or file-like object*) – Configuration file to pass to instance on initialization. This configuration file will be used as default for host side accelerator. If value is True, use 'config' configuration. If value is a configuration use this configuration. If value is None or False, don't passe any configuration file (This is default behavior).
- **init_script** (*path-like object or file-like object*) – A bash script to execute on instance startup.
- **ssl_cert_crt** (*path-like object or file-like object or bool*) – Public ".crt" key file of the SSL ssl_cert_key used to provides HTTPS. If not specified, uses already generated certificate if found. If False, disable HTTPS.
- **ssl_cert_key** (*path-like object or file-like object*) – Private ".key" key file of the SSL ssl_cert_key used to provides HTTPS. If not specified, uses already generated key if found.
- **ssl_cert_generate** (*bool*) – Generate a self signed ssl_cert_key. The ssl_cert_key and private key will be stored in files specified by "ssl_cert_crt" and "ssl_cert_key" (Or temporary certificates if not specified). Note that this ssl_cert_key is only safe if other client verify it by providing "ssl_cert_crt". No Certificate Authority are available to trust this ssl_cert_key.
- **boto3_session_kwargs** (*dict*) – Extra keyword arguments for boto3.session.Session
- **boto3_client_kwargs** (*dict*) – Extra keyword arguments for boto3.session.Session.client.
- **boto3_create_instances_kwargs** (*dict*) – Extra Keyword arguments for boto3.session.Session.resource('ec2').create_instances.

ASSUME_ROLE_POLICY_DOCUMENT = {'Statement': {'Effect': 'Allow', 'Principal': {'Serv
IAM Assume Role Policy Document – Grant rights defined by policy to role.

DOC_URL = 'https://aws.amazon.com'
AWS Website

NAME = 'AWS'
Provider name to use

POLICY_DOCUMENT = {'Statement': [{'Sid': 'AllowFpgaCommands', 'Effect': 'Allow', 'A
IAM Policy document – Grant FPGA Access and S3 Buckets access

get_configuration_env(***config_env*)
Return environment to pass to "apyfal.accelerator.AcceleratorClient.start" "csp_env" argument.

Parameters **config_env** – Overwrites environment values.

Returns Configuration environment.

Return type dict

host_ip
Host IP of the current instance. This may return public or private IP based on configuration.

Returns IP address

Return type str

Raises *apyfal.exceptions.HostRuntimeException* – No instance from which get IP.

host_name
Name of the current host.

Returns Name

Return type str

host_type
Host type

Returns Host type

Return type str

info
Returns some host information.

Returns

Dictionary containing information on current host.

Return type dict

instance_id
ID of the current instance.

Returns ID

Return type str

iter_hosts(*host_name_prefix=True*)
Iterates over accelerator hosts of current type.

Parameters **host_name_prefix** (*bool or str*) – If True, use "host_name_prefix" from configuration, if False don't filter by prefix, if str, uses this str as prefix

Returns dicts contains attributes values of the host.

Return type generator of dict

key_pair
SSH Key pair linked to this instance.

Returns Name of key pair.

Return type str

private_ip
Private IP of the current instance.

Returns IP address

Return type str

Raises *apyfal.exceptions.HostRuntimeException* – No instance from which get IP.

public_ip
Public IP of the current instance.

Returns IP address

Return type str

Raises *apyfal.exceptions.HostRuntimeException* – No instance from which get IP.

ssl_cert_crt
SSL ssl_cert_key used.

Returns Path to SSL ssl_cert_key.

Return type str

start(*accelerator=None*, *accel_parameters=None*, *stop_mode=None*)
Start instance if not already started. Create instance if necessary.

Needs "accel_client" or "accel_parameters".

Parameters

- **accelerator** (*str*) – Name of the accelerator.
- **accel_parameters** (*dict*) – Can override parameters from accelerator client.
- **stop_mode** (*str or int*) – See "stop_mode" property for more information.

stop(*stop_mode=None*)
Stop instance accordingly with the current stop_mode. See "stop_mode" property for more information.

Parameters **stop_mode** (*str or int*) – If not None, override current "stop_mode" value.

stop_mode
Define the "stop" method behavior.

Possible values ares: 0: TERM, terminate and delete host. 1: STOP, stop and pause host. 2: KEEP, let host running.

"stop" can be called manually but is also called when:

- "with" exit if class is used as context manager.
- On object deletion by garbage collector.
- On OS signals if "exit_host_on_signal" was set to True on class instantiation.

Returns stop mode.

Return type str

url
URL of the current host.

Returns URL

Return type str

apyfal.host.openstack

OpenStack Nova

class apyfal.host.openstack.**OpenStackHost**(*project_id=None*, *auth_url=None*, *nova_client_kwargs=None*, *nova_client_create_server_kwargs=None*, *neutron_client_kwargs=None*, ***kwargs*)
OpenStack based CSP

Parameters

- **host_type** (*str*) – Cloud service provider name. Default to "OpenStack".
- **config** (apyfal.configuration.Configuration, *path-like object or file-like object*) – If not set, will search it in current working directory, in current user "home" folder. If none found, will use default configuration values. Path-like object can be path, URL or cloud object URL.
- **client_id** (*str*) – OpenStack Access Key ID.
- **secret_id** (*str*) – OpenStack Secret Access Key.
- **region** (*str*) – OpenStack region. Needs a region supporting instances with FPGA devices.
- **instance_type** (*str*) – OpenStack Flavor. Default defined by accelerator.
- **key_pair** (*str*) – OpenStack Key pair. Default to 'Accelize<HostName>KeyPair'.
- **security_group** – OpenStack Security group. Default to 'AccelizeSecurityGroup'.
- **instance_id** (*str*) – Instance ID of an already existing OpenStack nova instance to use. If not specified, create a new instance.
- **host_name_prefix** (*str*) – Prefix to add to instance name. Also used as value of the "Apyfal" metadata.
- **host_ip** (*str*) – IP or URL address of an already existing OpenStack nova instance to use. If not specified, create a new instance.
- **use_private_ip** (*bool*) – If True, on new instances, uses private IP instead of public IP as default host IP.
- **project_id** (*str*) – OpenStack Project
- **auth_url** (*str*) – OpenStack auth-URL
- **stop_mode** (*str or int*) – Define the "stop" method behavior. Default to 'term' if new instance, or 'keep' if already existing instance. See "stop_mode" property for more information and possible values.
- **init_config** (*bool or* apyfal.configuration.Configuration, *path-like object or file-like object*) – Configuration file to pass to instance on initialization. This configuration file will be used as default for host side accelerator. If value is True, use 'config' configuration. If value is a configuration use this configuration. If value is None or False, don't passe any configuration file (This is default behavior).
- **init_script** (*path-like object or file-like object*) – A bash script to execute on instance startup.
- **ssl_cert_crt** (*path-like object or file-like object or bool*) – Public ".crt" key file of the SSL ssl_cert_key used to provides HTTPS. If not specified, uses already generated certificate if found. If False, disable HTTPS.
- **ssl_cert_key** (*path-like object or file-like object*) – Private ".key" key file of the SSL ssl_cert_key used to provides HTTPS. If not specified, uses already generated key if found.
- **ssl_cert_generate** (*bool*) – Generate a self signed ssl_cert_key. The ssl_cert_key and private key will be stored in files specified by "ssl_cert_crt" and "ssl_cert_key" (Or temporary certificates if not specified). Note that this ssl_cert_key is only safe if other client verify it by providing "ssl_cert_crt". No Certificate Authority are available to trust this ssl_cert_key.

- **nova_client_kwargs** (*dict*) – Extra keyword arguments for novaclient.client.Client.
- **nova_client_create_server_kwargs** (*dict*) – Extra Keyword arguments for novaclient.servers.ServerManager.create.
- **neutron_client_kwargs** (*dict*) – Extra keyword arguments for neutronclient.client.Client. By default, neutron client inherits from nova client session.

NAME = 'OpenStack'
Provider name to use

STATUS_ERROR = 'ERROR'
Instance status when in error

STATUS_RUNNING = 'ACTIVE'
Instance status when running

STATUS_STOPPED = 'PAUSED'
Instance status when stopped

get_configuration_env(***config_env*)
Return environment to pass to "apyfal.accelerator.AcceleratorClient.start" "csp_env" argument.

Parameters **config_env** – Overwrites environment values.

Returns Configuration environment.

Return type dict

host_ip
Host IP of the current instance. This may return public or private IP based on configuration.

Returns IP address

Return type str

Raises *apyfal.exceptions.HostRuntimeException* – No instance from which get IP.

host_name
Name of the current host.

Returns Name

Return type str

host_type
Host type

Returns Host type

Return type str

info
Returns some host information.

Returns

Dictionary containing information on current host.

Return type dict

instance_id
ID of the current instance.

Returns ID

Return type str

iter_hosts(*host_name_prefix=True*)
Iterates over accelerator hosts of current type.

Parameters **host_name_prefix** (*bool or str*) – If True, use "host_name_prefix" from configuration, if False don't filter by prefix, if str, uses this str as prefix

Returns dicts contains attributes values of the host.

Return type generator of dict

key_pair
SSH Key pair linked to this instance.

Returns Name of key pair.

Return type str

private_ip
Private IP of the current instance.

Returns IP address

Return type str

Raises *apyfal.exceptions.HostRuntimeException* – No instance from which get IP.

public_ip
Public IP of the current instance.

Returns IP address

Return type str

Raises *apyfal.exceptions.HostRuntimeException* – No instance from which get IP.

ssl_cert_crt
SSL ssl_cert_key used.

Returns Path to SSL ssl_cert_key.

Return type str

start(*accelerator=None*, *accel_parameters=None*, *stop_mode=None*)
Start instance if not already started. Create instance if necessary.

Needs "accel_client" or "accel_parameters".

Parameters

- **accelerator** (*str*) – Name of the accelerator.
- **accel_parameters** (*dict*) – Can override parameters from accelerator client.
- **stop_mode** (*str or int*) – See "stop_mode" property for more information.

stop(*stop_mode=None*)
Stop instance accordingly with the current stop_mode. See "stop_mode" property for more information.

Parameters **stop_mode** (*str or int*) – If not None, override current "stop_mode" value.

stop_mode
Define the "stop" method behavior.

Possible values ares: 0: TERM, terminate and delete host. 1: STOP, stop and pause host. 2: KEEP, let host running.

"stop" can be called manually but is also called when:

- "with" exit if class is used as context manager.
- On object deletion by garbage collector.
- On OS signals if "exit_host_on_signal" was set to True on class instantiation.

Returns stop mode.

Return type str

url

URL of the current host.

Returns URL

Return type str

apyfal.host.ovh

OVH

class apyfal.host.ovh.**OVHHost**(*project_id=None*, *auth_url=None*, *nova_client_kwargs=None*, *nova_client_create_server_kwargs=None*, *neutron_client_kwargs=None*, ***kwargs*)

OVH CSP

Parameters

- **host_type** (*str*) – Cloud service provider name. Default to "OVH".
- **config** (apyfal.configuration.Configuration, *path-like object or file-like object*) – If not set, will search it in current working directory, in current user "home" folder. If none found, will use default configuration values. Path-like object can be path, URL or cloud object URL.
- **client_id** (*str*) – OVH Access Key ID.
- **secret_id** (*str*) – OVH Secret Access Key.
- **region** (*str*) – OVH region. Needs a region supporting instances with FPGA devices.
- **instance_type** (*str*) – OVH Flavor. Default defined by accelerator.
- **key_pair** (*str*) – OVH Key pair. Default to 'AccelizeOVHKeyPair'.
- **security_group** – OVH Security group. Default to 'AccelizeSecurityGroup'.
- **instance_id** (*str*) – Instance ID of an already existing OVH nova instance to use. If not specified, create a new instance.
- **host_name_prefix** (*str*) – Prefix to add to instance name. Also used as value of the "Apyfal" metadata.
- **host_ip** (*str*) – IP or URL address of an already existing AWS EC2 instance to use. If not specified, create a new instance.
- **use_private_ip** (*bool*) – If True, on new instances, uses private IP instead of public IP as default host IP.

- **project_id** (*str*) – OVH Project
- **stop_mode** (*str or int*) – Define the "stop" method behavior. Default to 'term' if new instance, or 'keep' if already existing instance. See "stop_mode" property for more information and possible values.
- **init_config** (*bool or apyfal.configuration.Configuration, path-like object or file-like object*) – Configuration file to pass to instance on initialization. This configuration file will be used as default for host side accelerator. If value is True, use 'config' configuration. If value is a configuration use this configuration. If value is None or False, don't passe any configuration file (This is default behavior).
- **init_script** (*path-like object or file-like object*) – A bash script to execute on instance startup.
- **ssl_cert_crt** (*path-like object or file-like object or bool*) – Public ".crt" key file of the SSL ssl_cert_key used to provides HTTPS. If not specified, uses already generated certificate if found. If False, disable HTTPS.
- **ssl_cert_key** (*path-like object or file-like object*) – Private ".key" key file of the SSL ssl_cert_key used to provides HTTPS. If not specified, uses already generated key if found.
- **ssl_cert_generate** (*bool*) – Generate a self signed ssl_cert_key. The ssl_cert_key and private key will be stored in files specified by "ssl_cert_crt" and "ssl_cert_key" (Or temporary certificates if not specified). Note that this ssl_cert_key is only safe if other client verify it by providing "ssl_cert_crt". No Certificate Authority are available to trust this ssl_cert_key.
- **nova_client_kwargs** (*dict*) – Extra keyword arguments for novaclient.client.Client.
- **nova_client_create_server_kwargs** (*dict*) – Extra Keyword arguments for novaclient.servers.ServerManager.create.
- **neutron_client_kwargs** (*dict*) – Extra keyword arguments for neutronclient.client.Client. By default, neutron client inherits from nova client session.

DOC_URL = 'https://horizon.cloud.ovh.net'
OVH Website

NAME = 'OVH'
Provider name to use

OPENSTACK_AUTH_URL = 'https://auth.cloud.ovh.net/'
OVH OpenStack auth-URL

get_configuration_env(***config_env*)
Return environment to pass to "apyfal.accelerator.AcceleratorClient.start" "csp_env" argument.

Parameters **config_env** – Overwrites environment values.

Returns Configuration environment.

Return type dict

host_ip
Host IP of the current instance. This may return public or private IP based on configuration.

Returns IP address

Return type str

Raises *apyfal.exceptions.HostRuntimeException* – No instance from which get IP.

host_name
Name of the current host.

Returns Name

Return type str

host_type
Host type

Returns Host type

Return type str

info
Returns some host information.

Returns

Dictionary containing information on current host.

Return type dict

instance_id
ID of the current instance.

Returns ID

Return type str

iter_hosts(*host_name_prefix=True*)
Iterates over accelerator hosts of current type.

Parameters **host_name_prefix** (*bool or str*) – If True, use "host_name_prefix" from configuration, if False don't filter by prefix, if str, uses this str as prefix

Returns dicts contains attributes values of the host.

Return type generator of dict

key_pair
SSH Key pair linked to this instance.

Returns Name of key pair.

Return type str

private_ip
Private IP of the current instance.

Returns IP address

Return type str

Raises *apyfal.exceptions.HostRuntimeException* – No instance from which get IP.

public_ip
Public IP of the current instance.

Returns IP address

Return type str

Raises *apyfal.exceptions.HostRuntimeException* – No instance from which get IP.

ssl_cert_crt
SSL ssl_cert_key used.

Returns Path to SSL ssl_cert_key.

Return type str

start(*accelerator=None*, *accel_parameters=None*, *stop_mode=None*)
Start instance if not already started. Create instance if necessary.

Needs "accel_client" or "accel_parameters".

Parameters

- **accelerator** (*str*) – Name of the accelerator.
- **accel_parameters** (*dict*) – Can override parameters from accelerator client.
- **stop_mode** (*str or int*) – See "stop_mode" property for more information.

stop(*stop_mode=None*)
Stop instance accordingly with the current stop_mode. See "stop_mode" property for more information.

Parameters **stop_mode** (*str or int*) – If not None, override current "stop_mode" value.

stop_mode
Define the "stop" method behavior.

Possible values ares: 0: TERM, terminate and delete host. 1: STOP, stop and pause host. 2: KEEP, let host running.

"stop" can be called manually but is also called when:

- "with" exit if class is used as context manager.
- On object deletion by garbage collector.
- On OS signals if "exit_host_on_signal" was set to True on class instantiation.

Returns stop mode.

Return type str

url
URL of the current host.

Returns URL

Return type str

2.6.3 apyfal.storage

Data storage I/O

Support I/O over different kind of storage that must be first mounted.

Theses features are provided by the "pycosio". Path are automatically handled with pycosio in Apyfal to provides automatic support of cloud storage.

Mount storage: Storage defined in configuration are automatically mounted on first call. It is possible to manually mount new storage with "mount" function.

The Apyfal configuration file is used to automatically mount all known storage on start up.

Storage URL format: All operations works with URL with format "scheme://path"

Common schemes are (mounted by default):

- "file://" or no scheme: Client local file.
- "host://": Host local file (Available only on REST client).
- "http://" or "https://": File access over HTTP.

Some storage use advanced same, basic form is "StorageType://path" with StorageType the storage type defining this storage.

See target storage class documentation for more information.

apyfal.storage.**copy**(*source*, *destination*)
Copy a file from source to destination.

Parameters

- **source** (*path-like object or file-like object*) – Source URL. Can be apyfal.storage URL, paths, file-like object.
- **destination** (*path-like object or file-like object*) – Destination URL. Can be apyfal.storage URL, paths, file-like object.

apyfal.storage.**mount**(*storage_type*, ***kwargs*)
Mount a new storage.

Parameters

- **storage_type** (*str*) – storage type
- **kwargs** – Storage keywords argument (see targeted storage class for more information)

apyfal.storage.**open**(*url*, *mode='rb'*, *encoding=None*, *errors=None*, *newline=None*)
Open file and return a corresponding file object.

Parameters

- **url** (*path-like object or file-like object*) – URL or file object to open.
- **mode** (*str*) – Mode in which the file is opened (Works like standard library open mode). Support at least 'r' (read), 'w' (write), 'b' (binary), 't' (text) modes, eventually more depending on source file.
- **encoding** (*str*) – with text mode, encoding used to decode or encode the file.
- **errors** (*str*) – with text mode, specifies how encoding and decoding errors are to be handled
- **newline** (*str*) – Controls how universal newlines mode works.

Returns Opened object handle

Return type file-like object

apyfal.storage.**parse_url**(*url*, *host=True*)
Return storage_type and path from URL.

If URL has no scheme, "file" scheme is inferred.

If URL is a file-like object, returns "stream" as storage_type.

Parameters

- **url** (*path-like object or file-like object*) – URL to parse
- **host** (*bool*) – If True, Scheme is returned from host point of view: "host" scheme is converted to "file" scheme.

Returns (storage_type, path)

Return type tuple of str

apyfal.storage.alibaba

Alibaba Cloud OSS

class apyfal.storage.alibaba.**OSSStorage**(*region=None*, *unsecure=None*, ***kwargs*)

Alibaba Cloud OSS Bucket

Storage URL: "oss://BucketName/ObjectKey"

Parameters

- **storage_type** (*str*) – Cloud service provider name. Default to "Alibaba".
- **config** (apyfal.configuration.Configuration, *path-like object or file-like object*) – If not set, will search it in current working directory, in current user "home" folder. If none found, will use default configuration values. Path-like object can be path, URL or cloud object URL.
- **unsecure** (*bool*) – if True (default) disables TLS/SSL/HTTPS for transfer. This can improve performance, but makes connection insecure.
- **client_id** (*str*) – Alibaba Access Key ID.
- **secret_id** (*str*) – Alibaba Secret Access Key.
- **region** (*str*) – Alibaba region.
- **storage_parameters** (*dict*) – Extra "storage_parameters". See "pycosio.mount".

DOC_URL = 'https://www.alibabacloud.com'

AWS Website

NAME = 'Alibaba'

Provider name

STORAGE_NAME = 'oss'

Storage name

STORAGE_PARAMETERS = {'access_key_id': 'self._client_id', 'access_key_secret': 'self

Storage parameters template

mount()

Mount storage.

apyfal.storage.aws

Amazon Web Services S3

class apyfal.storage.aws.**S3Storage**(*storage_type=None*, *config=None*, *client_id=None*, *secret_id=None*, *unsecure=None*, *storage_parameters=None*, ***_*)

AWS S3 Bucket

Storage URL: s3://BucketName/ObjectKey

Parameters

- **storage_type** (*str*) – Cloud service provider name. Default to "AWS".
- **config** (apyfal.configuration.Configuration, *path-like object or file-like object*) – If not set, will search it in current working directory, in current user "home" folder. If none found, will use default configuration values. Path-like object can be path, URL or cloud object URL.
- **unsecure** (*bool*) – if True (default) disables TLS/SSL/HTTPS for transfer. This can improve performance, but makes connection insecure.
- **client_id** (*str*) – AWS Access Key ID.
- **secret_id** (*str*) – AWS Secret Access Key.
- **storage_parameters** (*dict*) – Extra "storage_parameters". See "pycosio.mount".

DOC_URL = 'https://aws.amazon.com'
AWS Website

NAME = 'AWS'
Provider name

STORAGE_NAME = 's3'
Storage name

STORAGE_PARAMETERS = {'session': {'aws_access_key_id': 'self._client_id', 'aws_secre
Storage parameters template

mount()
Mount storage.

apyfal.storage.openstack

OpenStack Swift

class apyfal.storage.openstack.**SwiftStorage**(*region=None*, *project_id=None*, *auth_url=None*, ***kwargs*)

OpenStack Swift Object _Storage

Storage URL:

- "https://Domain/v1/ProjectID/Container/Object"
- "swift://Container/Object"

Parameters

- **storage_type** (*str*) – Cloud service provider name. Default to "OpenStack".
- **config** (apyfal.configuration.Configuration, *path-like object or file-like object*) – If not set, will search it in current working directory, in current user "home" folder. If none found, will use default configuration values. Path-like object can be path, URL or cloud object URL.
- **unsecure** (*bool*) – if True (default) disables TLS/SSL/HTTPS for transfer. This can improve performance, but makes connection insecure.
- **client_id** (*str*) – OpenStack Access Key ID.
- **secret_id** (*str*) – OpenStack Secret Access Key.
- **region** (*str*) – OpenStack region.

- **project_id** (*str*) – OpenStack Project
- **auth_url** (*str*) – OpenStack auth-URL
- **storage_parameters** (*dict*) – Extra "storage_parameters". See "pycosio.mount".

EXTRA_ROOT = 'swift://'
Extra root (For shorter URL)

NAME = 'OpenStack'
Provider name

STORAGE_NAME = 'swift'
Storage name

STORAGE_PARAMETERS = {'auth_version': '3', 'authurl': 'self._auth_url', 'key': 'sel
Storage parameters template

mount()
Mount storage.

apyfal.storage.ovh

OVH Object Store

class apyfal.storage.ovh.**OVHStorage**(*region=None*, *project_id=None*, *auth_url=None*, ***kwargs*)
OVH Object Store

Storage URL:

- "https://storage.Region.cloud.ovh.net/v1/AUTH_ProjectID/Container/Object"
- "ovh://Container/Object"

Parameters

- **storage_type** (*str*) – Cloud service provider name. Default to "OVH".
- **config** (apyfal.configuration.Configuration, *path-like object or file-like object*) – If not set, will search it in current working directory, in current user "home" folder. If none found, will use default configuration values. Path-like object can be path, URL or cloud object URL.
- **ssl** (*bool*) – If True (default) allow SSL for transfer, else tries to disable it. Disabling SSL can improve performance, but makes connection insecure.
- **client_id** (*str*) – OVH Access Key ID.
- **secret_id** (*str*) – OVH Secret Access Key.
- **region** (*str*) – OVH region.
- **project_id** (*str*) – OVH Project
- **storage_parameters** (*dict*) – Extra "storage_parameters". See "pycosio.mount".

DOC_URL = 'https://horizon.cloud.ovh.net'
OVH Website

EXTRA_ROOT = 'ovh://'
Extra root (For shorter URL)

NAME = 'OVH'
Provider name

OPENSTACK_AUTH_URL = 'https://auth.cloud.ovh.net/'
OVH OpenStack auth-URL

STORAGE_NAME = 'swift'
Storage name

mount()
Mount storage.

2.6.4 apyfal.configuration

See also *Configuration*

Manages Accelerator and CSP configuration

Notes

The "accelerator.conf" file provided with this package can be used as sample for creation of user configuration files.

Use of this file is optional. All parameters can also be passed to API class. Non specified ones will use default values.

API search automatically search for "accelerator.conf" in current working directory, or in current user home directory. A custom path to configuration file can also be passed to classes.

apyfal.configuration.**create_configuration**(*configuration_file*)
Create a configuration instance

Parameters **configuration_file** (apyfal.configuration.Configuration, *path-like object or file-like object*) – Configuration to use. Path-like object can be path, URL or cloud object URL.

class apyfal.configuration.**Configuration**(*configuration_file=None*)
Accelerator configuration.

Mapping of configuration sections. On section access, never raises KeyError but returns at least an empty section.

Sections are mapping of parameters. On parameter access, never raises KeyError but returns a default value of None.

If a configuration section name contain dot (ex: "host.csp"), it is a subsection and accessing parameter performs the follow: - Tries to get parameter from this section ("host.csp") - If parameter value is None, tries to get value from

parent section ("host")

Trying to setting an parameter to None does nothing and keep previous value. Use "del" to delete an parameter to set a reset an parameter value.

Parameters **configuration_file** (apyfal.configuration.Configuration, *path-like object or file-like object*) – If None, use default values. Path-like object can be path, URL or cloud object URL.

DEFAULT_CONFIG_FILE = 'accelerator.conf'
Default name for configuration file (Used for file detection)

access_token
Patched method

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

get_host_configurations(**args*, ***kwargs*)
Patched method

get_host_requirements(*host_type*, *accelerator*)
Gets accelerators requirements to use with host.

Parameters

- **host_type** (*str*) – Host type.
- **accelerator** (*str*) – Name of the accelerator

Returns AcceleratorClient requirements for host.

Return type dict

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

values() → an object providing a view on D's values

write(*fileobject*)
Write configuration file.

Parameters **fileobject** (*file-like object*) – file-like object open in text mode.

apyfal.configuration.**accelerator_executable_available**()
Returns True if accelerator executable available locally.

Returns Accelerator executable found.

Return type bool

apyfal.configuration.**ACCELERATOR_EXECUTABLE = '/opt/accelize/accelerator/accelerator'**
Accelerator Executable path

apyfal.configuration.**ACCELERATOR_TMP_ROOT = '/dev/shm'**
Accelerator temporary directory root

apyfal.configuration.**APYFAL_HOME = '/home/docs/.apyfal'**
Apyfal directory in user home

apyfal.configuration.**METERING_SERVER = 'https://master.metering.accelize.com'**
Metering server URL

apyfal.configuration.**METERING_TMP = '/tmp/meteringServer'**
Metering Server temporary directory

apyfal.configuration.**METERING_CLIENT_CONFIG = '/etc/sysconfig/meteringclient'**
Metering Client configuration

apyfal.configuration.**METERING_CREDENTIALS = '/etc/accelize/credentials.json'**
Metering Credentials JSON file

apyfal.configuration.**APYFAL_CERT_CRT = '/home/docs/.ssh/ApyfalCertificate.crt'**
Apyfal generated self signed wildcard certificate files

2.6.5 apyfal.exceptions

Apyfal Exceptions

exception apyfal.exceptions.**AcceleratorException**(*msg=None*, *gen_msg=None*, *exc=None*, **args*, ***kwargs*)

Base exception for apyfal exceptions

Parameters

- **msg** (*str*) – Exception message
- **gen_msg** (*tuple of str or str*) – Arguments for apyfal._utilities.gen_msg
- **exc** (*Exception or str*) – Exception or other details to add to description

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception apyfal.exceptions.**ClientAuthenticationException**(*msg=None*, *gen_msg=None*, *exc=None*, **args*, ***kwargs*)

Error while trying to authenticate user on Accelize server.

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception apyfal.exceptions.**ClientConfigurationException**(*msg=None*, *gen_msg=None*, *exc=None*, **args*, ***kwargs*)

Error with AcceleratorClient configuration.

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception apyfal.exceptions.**ClientException**(*msg=None*, *gen_msg=None*, *exc=None*, **args*, ***kwargs*)

Generic AcceleratorClient related exception.

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception apyfal.exceptions.**ClientRuntimeException**(*msg=None*, *gen_msg=None*, *exc=None*, **args*, ***kwargs*)

Error with AcceleratorClient running.

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception apyfal.exceptions.**ClientSecurityException**(*msg=None*, *gen_msg=None*, *exc=None*, **args*, ***kwargs*)

Generic security related exception.

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception apyfal.exceptions.**HostAuthenticationException**(*msg=None*, *gen_msg=None*, *exc=None*, **args*, ***kwargs*)

Error while trying to authenticate user on host.

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception apyfal.exceptions.**HostConfigurationException**(*msg=None*, *gen_msg=None*, *exc=None*, **args*, ***kwargs*)

Error with host configuration

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception apyfal.exceptions.**HostException**(*msg=None*, *gen_msg=None*, *exc=None*, **args*, ***kwargs*)

Generic host related exception

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception apyfal.exceptions.**HostRuntimeException**(*msg=None*, *gen_msg=None*, *exc=None*, **args*, ***kwargs*)

Error with host on runtime

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception apyfal.exceptions.**StorageAuthenticationException**(*msg=None*, *gen_msg=None*, *exc=None*, **args*, ***kwargs*)

Error while trying to authenticate user on storage.

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception apyfal.exceptions.**StorageConfigurationException**(*msg=None*, *gen_msg=None*, *exc=None*, **args*, ***kwargs*)

Error when configuring storage

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception apyfal.exceptions.**StorageException**(*msg=None*, *gen_msg=None*, *exc=None*, **args*, ***kwargs*)

Generic storage related exception

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception apyfal.exceptions.**StorageResourceNotExistsException**(*msg=None*, *gen_msg=None*, *exc=None*, **args*, ***kwargs*)

Exception when trying to access an non existing resource

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception apyfal.exceptions.**StorageRuntimeException**(*msg=None*, *gen_msg=None*, *exc=None*, **args*, ***kwargs*)

Error when running storage

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

2.7 Apyfal CLI

Apyfal CLI is a command line interface to use Apyfal directly in shell without the need to use Python.

2.7.1 Apyfal CLI commands

Apyfal CLI provides help using `-h` or `--help` arguments:

```
apyfal -h
```

Apyfal CLI needs to be run with a command, following command are available:

- `create`: Create accelerator and configure host. Equivalent to `apyfal.Accelerator` instantiation. Calling `create` is optional if called without arguments (By example when run locally on an already configured host).
- `start`: Start and configure Accelerator. Equivalent to `apyfal.Accelerator.start` method.
- `process`: Process with Accelerator. Equivalent to `apyfal.Accelerator.process` method.
- `stop`: Stop accelerator. Equivalent to `apyfal.Accelerator.stop` method.
- `copy`: Copy Apyfal Storage URL. Equivalent to `apyfal.storage.copy` function.
- `clear`: Clear cached accelerators. New command, specific to CLI mode (detailed below).

Each command accept the sames parameters as its `apyfal` library equivalent. Keywords arguments names needs to be prefixed with `--`.

```
# Values can be passed after argument separated with space or equal:
apyfal create --accelerator=my_accelerator --host_type=my_provider

# Is equivalent to
apyfal create --accelerator my_accelerator --host_type my_provider

# Some arguments proposes also short aliases prefixed by ``-``
apyfal create -a my_accelerator --host_type my_provider
```

Apyfal CLI provides command specific help. Uses it for more information.

```
apyfal create -h
```

Full commands documentation:

Apyfal CLI help

This section provides full CLI help (from `--help`) for each command.

apyfal

usage: apyfal [-h] {create,start,process,stop,copy,clear} ...

Apyfal command line utility.

optional arguments:
    -h, –help show this help message and exit

Commands:
Apyfal must perform one of the following commands:

{create,start,process,stop,copy,clear}
Apyfal commands
create Create accelerator and configure host.
start Start and configure Accelerator.
process Process with Accelerator.
stop Stop accelerator.
copy Copy a file using Apyfal.storage.
clear Clear cached accelerators.

apyfal create

usage: apyfal create [-h] [–name NAME] [–accelerator ACCELERATOR]
[–config CONFIG]
[–accelize_client_id ACCELIZE_CLIENT_ID]
[–accelize_secret_id ACCELIZE_SECRET_ID]
[–host_type HOST_TYPE] [–host_ip HOST_IP]

Create accelerator and configure host.

optional arguments:
-h, –help show this help message and exit
–name NAME, -n NAME Save Accelerator with this name, it can be called
later with “start”, “process” and “stop” commands and
“–name” argument.
–accelerator ACCELERATOR, -a ACCELERATOR
–config CONFIG, -c CONFIG
–accelize_client_id ACCELIZE_CLIENT_ID
–accelize_secret_id ACCELIZE_SECRET_ID
–host_type HOST_TYPE
–host_ip HOST_IP

Extra parameters can be passed as “–parameter_name=parameter_value”. See Apyfal documentation for information on possible parameters for the targeted host (https://apyfal.readthedocs.io/). See accelerator documentation for information on its parameters

apyfal start

usage: apyfal start [-h] [–name NAME] [–src SRC] [–info_dict]
[–parameters PARAMETERS]

Start and configure Accelerator.

optional arguments:
-h, –help show this help message and exit
–name NAME, -n NAME Load Accelerator with this name that was created with
"create" commands
–src SRC, -i SRC
–info_dict
–parameters PARAMETERS, -j PARAMETERS

Extra parameters can be passed as "–parameter_name=parameter_value". See
accelerator documentation for information on specific configuration parameters

apyfal process

usage: apyfal process [-h] [–name NAME] [–src SRC] [–dst DST] [–info_dict]
[–parameters PARAMETERS]

Process with Accelerator.

optional arguments:
-h, –help show this help message and exit
–name NAME, -n NAME Load Accelerator with this name that was created with
"create" commands
–src SRC, -i SRC
–dst DST, -o DST
–info_dict
–parameters PARAMETERS, -j PARAMETERS

Extra parameters can be passed as "–parameter_name=parameter_value". See
accelerator documentation for information on specific process parameters

apyfal stop

usage: apyfal stop [-h] [–name NAME] [–info_dict] [–stop_mode STOP_MODE]

Stop accelerator.

optional arguments:
-h, –help show this help message and exit
–name NAME, -n NAME Load Accelerator with this name that was created with
"create" commands
–info_dict

–stop_mode STOP_MODE

apyfal copy

usage: apyfal copy [-h] source destination

Copy a file using Apyfal.storage.

positional arguments:
source
destination

optional arguments:
-h, –help show this help message and exit

apyfal clear

usage: apyfal clear [-h]

Clear cached accelerators.

optional arguments:
-h, –help show this help message and exit

2.7.2 Using accelerator with Apyfal CLI

The use of accelerator with CLI is the same as the library, see *Getting Started* for more information

This example show the basic use of an accelerator using Apyfal CLI :

```
# Accelerator instantiation
apyfal create --accelerator my_accelerator

# Accelerator start
apyfal start

# Accelerator process
apyfal process --src /path/myfile.dat --dst /path/result.dat

# Accelerator stop
# Do not forget this step, this is not automatically handled with CLI
apyfal stop
```

Specific start and process arguments are passed like others:

```
apyfal process --src /path/myfile.dat --specific1 1 --specific2 2
```

Using multiple accelerators

It is possible to use multiple accelerators at the same time by naming them with different names using the `--name` argument:

```
# Instantiate accelerator and name it
apyfal create --accelerator my_accelerator --name myaccel

# Run "start" on "myaccel"
apyfal start --name myaccel

# Run "process" on "myaccel"
apyfal process --name myaccel --src /path/myfile.dat --dst /path/result.dat

# Run "stop" on "myaccel"
# Do not forget this step, this is not automatically handled with CLI
apyfal stop --name myaccel
```

It is possible to clear all previously existing accelerator with the `clear` command. Warning: This don't stop accelerators.

```
apyfal clear
```

2.8 Apyfal REST API

It is possible to use accelerators using the REST API instead of Python But, note that in this case, host configuration is not supported.

The Apyfal REST API follow rhe OpenApi specification.

- Apyfal REST API Documentation
- `Apyfal OpenAPI specification`

2.8.1 Generating REST API client in any language using OpenApi

With OpenApi, a client can be generated for almost any language (Java, Javascript, ...)

- Download and install an OpenAPI client generator (like OpenAPI Generator or Swagger-Codegen).
- Download the Apyfal repository:

```
git clone https://github.com/Accelize/apyfal.git
```

- From the repository directory, run the client generator with the following command after replacing `$GENERATOR_CLI` with the path to the client generator `.jar` executable, and replacing `$LANGUAGE` by the language to generate (Please refer to client generator documentation for the list of possible languages)

```
java -jar $GENERATOR_CLI generate -i rest_api/input_spec.json -o rest_api/output -l
↪$LANGUAGE
```

- The generated API in the target language can be found in `rest_api/output` sub-directory inside the repository folder.

2.9 Accelerator CLI

On its host, accelerator uses a the `accelerator` CLI to communicate with FPGA.

Accelerator CLI do not provides as features as Apyfal CLI, but it can help to reduce latency in some cases.

A configured host is required to use this command. Apyfal can be used to configure an instance and access it with SSH. See *Getting Started* for more information.

2.9.1 The accelerator command

The accelerator command path is: `/opt/accelize/accelerator/accelerator`.

It needs to be run as `root` (or with `sudo`)

It support following arguments:

- `-m`: Accelerator mode. Possibles values are: `0` for configuration/start mode, `1` for process mode, `2` for stop mode. This is equivalent to `apyfal.Accelerator` `start`, `process` and `stop` methods.
- `-i`: Input local file path, used to pass `src` in configuration mode and `src` in process mode.
- `-o`: Output local file path, used to pass `dst` in process mode.
- `-j`: JSON parameter local file path, used to pass a JSON parameters files like described in *Advanced use*.
- `-p`: JSON output local file path, used to get some results in JSON format.
- `-v`: Verbosity level. Possible values: from `0` (Full verbosity) to `4` (Less verbosity).

```
# Configures accelerator with src and JSON parameters
sudo /opt/accelize/accelerator/accelerator -m 0 -i ${src} -j ${parameters}

# Processes src and save result to dst
sudo /opt/accelize/accelerator/accelerator -m 1 -i ${src} -o ${dst}
```

2.9.2 Metering services

For use accelerator command, metering services needs to be started. This should be the case by default.

Theses commands starts services:

```
sudo systemctl start meteringsession
sudo systemctl start meteringclient
```

Theses commands stops services:

```
sudo systemctl stop meteringclient
sudo systemctl stop meteringsession
```

In two cases, run order of commands is important.

2.10 Building Apyfal

2.10.1 Installing Apyfal build environment

Prepare Python environment

Ensure that `pip`, `setuptools` and `wheel` are installed and up to date.

```
pip install --upgrade setuptools pip wheel
```

Clone repository from Github

Apyfal development version is hosted on github, to get it use git clone:

```
git clone https://github.com/Accelize/apyfal.git
```

Installing all Apyfal requirements

To install all package that are required to run Apyfal, run the following line in repository directory:

```
pip install -e .[all]
```

2.10.2 Running tests

To run unittest, run the following line in repository directory:

```
./setup.py test
```

2.10.3 Generating documentation

To generate Sphinx documentation, run the following line in repository directory:

```
./setup.py build_sphinx
```

To see generated documentation open `/build/sphinx/html/index.html` with browser.

2.10.4 Generating wheel package

To generate wheel package, run the following line in repository directory:

```
./setup.py bdist_wheel
```

Generated wheel can be found in `/dist`.

2.11 Changelog

2.11.1 1.2.3 (2018/10)

Backward incompatible changes:

- The `info_dict` argument behavior was changed. Previously, using it changed the returned result of methods. To avoid this, `info_dict` now waits a `dict` to populate with extra information. This argument is mainly intended to get debug or profiling information.

  Previous behavior:

  ```
  # This return only the processing result
  my_accel.process(dst='data.dat')

  # This return a tuple containing the processing result and the
  # information dict.
  my_accel.process(dst='data.dat', info_dict=True)
  ```

  New behavior:

  ```
  # This return only the processing result
  my_accel.process(dst='data.dat')

  # This still return only the processing result.
  # Information are now stored in the "info" dict.
  info = dict()
  my_accel.process(dst='data.dat', info_dict=info)
  ```

- The `info_dict` argument from `AcceleratorPoolExecutor.start`, `AcceleratorPoolExecutor.stop`, `AcceleratorPoolExecutor.process_map` and `Accelerator.process_map` methods is replaced by `info_list` and wait a `list` to populate instead of a `dict`.

2.11.2 1.2.2 (2018/10)

Fixes:

- Improve `stop` behavior depending if run by user or by garbage collector or `with` exit.

2.11.3 1.2.1 (2018/10)

Fixes:

- Fix broken input and output data on some accelerators when using cloud storage.
- `Accelerator` and `AcceleratorPoolExecutor` now waits completion of all asynchronous tasks (From `process_submit` or `process_map`) before exiting using `stop`. This avoid the accelerator or the host to be stopped before the end of tasks if `with` statement exited or Accelerator garbage collected when tasks are still running.
- Improve user public IP handling.

2.11.4 1.2.0 (2018/10)

New features

- Apyfal now fully support HTTPS between client and host.
- Apyfal can generate self signed certificates for generated hosts, theses certificates are verified by the client.
- Add of `process_map` and `process_submit` methods to the Accelerator class to performs `process` call asynchronously and improve performance on batch of processing tasks.
- Add the `AcceleratorPoolExecutor` that allow to perform processing tasks asynchronously over a pool of multiple accelerators hosts.

General improvements

- Apyfal CLI: `create` is now optional if can be called without any arguments, This is mainly intended to use local accelerator directly on host.
- It is now possible to use private IP instead of public IP as accelerator default URL. See `use_private_ip` parameter.
- Host instance have a new `Apyfal` tag/metadata with `host_name_prefix` value.
- Add `boto3` as default dependency. Actually AWS is the only provider ready for production and is the most commonly used. Other providers are available using extra setup options.
- Change logging levels to show only minimal information with INFO, implementation and step detail is still available using the DEBUG level. This allow to show more relevant information when using Apyfal with CLI or running Accelerators examples.
- Minimum packages versions are set in setup based on packages changelog or date.
- Hosts instantiation now support passing custom arguments to their libraries. See each specific host documentation for more information.

REST client improvements

- Uses `requests_toolbelt` instead of `PycURL` to upload big files. This simplify the Apyfal installation by using a far more easier to install library.
- Uses `requests` instead of Swagger codegen generated client. This simplify the REST client, removes some dependencies remove extra build step.
- Improves exceptions handling to add more detailed information from server and handle HTTP errors correctly.

Fixes:

- Fix bad text formatting in some exception messages.
- Server side logging was improved.
- Apyfal CLI: Fixed parsing of numeric parameters.
- Apyfal CLI: Fixed result dict handling.
- Fix accelerator application stopped if client `with` exited or garbage collected.
- Fix instance still running warning shown twice.
- Fix `stop_mode` overridden by default accelerator value.
- Fix case handling in configuration file.
- The host server now checks the Apyfal version used as client and raise a proper exception if not compatible.
- The host server was updated to be compatible with Apyfal starting from 1.0.0 instead of only 1.1.0.

- Apyfal now configures FPGA properly if run locally on a host no generated by Apyfal client (Ex: Host instance generated manually using accelerator image)
- Apyfal now runs the local accelerator if available even if a `host_type` is provided in configuration file.
- Fix Apyfal setup fail due to missing `ipgetter` package on PyPI (This package was removed by this author without notice).

Deprecations:

- The `optional` extra setup option is deprecated with the replacement of `PycURL`.

Pending deprecations:

- `file_in` and `file_out` argument in `process` method are replaced by `src` and `dst`. `datafile` argument in `start` method is replaced by `src`. This name change allow us to provides a better input and output data support in next version (No only files). The backward compatibility is kept for old arguments names but will be removed a future version.

2.11.5 1.1.0 (2018/08)

New features

- Add support for cloud storage and more in `apyfal.storage` package using `pycosio`.
- Add Apyfal CLI, this allow to use Apyfal from outside Python.
- Apyfal can now be used locally on host (as library or CLI).
- Add `apyfal.iter_accelerator` function to iterates over all existing accelerators for a configuration.
- Add Alibaba Cloud support.
- It is now possible to pass a SSL/TLS certificates to host instance to enable HTTPS.

General improvements

- Move OpenStack library from `openstacksdk` to `python-novaclient` and `python-neutronclient`. This adds more precision over the OpenStach Nova host control and reduces the overall number of required dependencies.
- Accelerator, Host and clients now have a proper string representation.
- Speed up cloud host configuration.
- Host now support the `init_script` argument to pass a custom bash script on instance startup, and the `init_config` argument to pass a configuration file.
- Apyfal now accepts path-like objects as path/URL arguments.

Configuration improvements:

- Add subsection support in configuration file (ex: `[host.host_type]`)
- Configuration file can be loaded from `apyfal.storage` URL.
- Configuration class is now a `Mapping` instead of `ConfigParser` subclass.
- Configuration file is now open with UTF-8 encoding.
- Add `host_name_prefix` in host section, This allow to add a custom prefix at the start of the created host name.

Fixes:

- Importing Apyfal from an unsupported Python version now raises `ImportError`.

- Host `stop_mode` is now correctly loaded from configuration file.
- Fix available regions list in exception message when trying to use a non existing region.
- Apyfal don't wait until end of timeout if instance is in `error` status during instance provisioning.
- Instance now terminates correctly if both `instance_id` and `host_ip` are provided.
- `stop` `stop_mode` with OpenStack now pauses instance instead of terminates it.
- Exception on AWS IAM policy first creation.
- Using `start` is not still mandatory when connecting to an already existing instance.

Deprecations:

- `exit_host_on_signal` host parameter was removed due to side effects. Use accelerator with the `with` statement to automatically terminate instance after run.

2.11.6 1.0.0 (2018/06)

Created the new *apyfal* library based on legacy *acceleratorAPI*.

Apyfal keeps all the features from acceleratorAPI but was largely improved. Apyfal is not backward compatible with acceleratorAPI (Read the documentation to see how update code). Future version of Apyfal will be compatible with this one.

Features of the 1.0.0 version:

- Accelerator start, process and stop in cloud environment.
- Accelerator configuration with arguments and/or configuration file.
- Support for *generic* OpenStack host.
- Support for AWS and OVH public host.
- Complete unittest for the core or the package.
- Full Sphinx documentation.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index

O

P

R

S

T

U

V

W