Zeblok Python SDK

Python SDK will help in integrating your existing MLOps pipeline with Zeblok platform.

Components of the Python SDK

Auth

- API Authentication
- Datalake Authentication

Resources

- Plan
- Namespace

Managed Service

- Microservice
- Orchestration
- Ai-API
- Ai-Pipeline

Interface

Authentication

- API Authentication: `APIAuth(app_url, api_access_key,api_access_secret)`
- Datalake Authentication: `DataSet(api_auth, access_key, secret_key, bucket_name)`

Plans

- Get all plans: `get_all()`
- Get a plan by id: `get_by_id(plan_id)`
- Validate a plan by id: `validate_id(plan_id)`
- Get filtered details of a plan: `get_filtered_details(plan_id, fields_req)`

Namespaces

- Get all namespaces: `get_all()`
- Get a namespace by id: `get_by_id(namespace_id)`
- Validate a namespace by id: `validate_id(namespace_id)`

MicroServices

- Get all microservices: `get_all()`
- Get a microservice by id: `get_by_id(plan_id)`
- Validate a microservice by id: `validate_id(microservice_id)`
- Spawn a microservices by id: `spawn(display_name, microservice_id, plan_id, microservice_name, namespace_id, envs, ports, args, command)`

Orchestrations

- Get all orchestrations: `get_all()`
- Get an orchestration by id: `get_by_id(orchestration_id)`
- Validate an orchestration by id: `validate_id(orchestration_id)`
- Spawn an orchestrations by id: `spawn(orchestration_id, plan_id, namespace_id, orchestration_name, min_workers, max_workers)`

AI-APIs

- Get all AI-APIs with `deployed` state: `get_all(state=deployed)`
- Get all AI-APIs with `ready` state: `get_all(state=ready)`
- Validate an AI-API: `validate(image_name, state)`
- Create and Spawn an AI-API: `create_and_spawn(ai_api_name, model_folder_path, plan_id, namespace_id, ai_api_type)`
- Create an AI-API: `create(ai_api_name, model_folder_path, ai_api_type)`
- Spawn an AI-API: `spawn(image_name, namespace_id, plan_id)`

AI-Pipelines

- Get all AI-Pipelines with `created` state: `get_all(state=created)`
- Get all AI-Pipelines with `ready` state: `get_all(state=ready)`
- Validate an AI-Pipeline: `validate(image_name, state)`
- Create and Spawn an AI-API: `create_and_spawn(ai_pipeline_name, ai_pipeline_folder_path, caas_plan_id, ai_pipeline_plan_id, namespace_id)`
- Create an AI-Pipeline: `create(ai_pipeline_name, ai_pipeline_folder_path, caas_plan_id, ai_pipeline_plan_id, namespace_id)`
- Spawn an AI-Pipeline: `spawn(ai_pipeline_plan_id, namespace_id, ai_pipeline_image_name)`

DataSet

- Check if a bucket exists: `bucket_exists()`
- Check if an object exists: `object_exists(object_name)`
- Upload an object: `upload_object(local_file_pathname, object_name)`
- Download an object: `download_object(object_name, local_dir, filename)`
- Get object PreSigned URL: `get_presigned_url(object_name)`

Usage

Table of Content

Authentication

We have two types of Authentications: `API Authentication` and `Datalake Authentication`

API Authentication

- If `api_access_key` is
    - `empty` or `None` then raises `ValueError`.
    - Not of type `str` then raises `TypeError`.
- If `api_access_secret` is
    - `empty` or `None` then raises `ValueError`.
    - Not of type `str` then raises `TypeError`.
- Raises `ServerError` for any other errors.

```
from zeblok.auth import APIAuth

api_auth = APIAuth(
    app_url=f'<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)
```

Datalake Authentication

- If `api_auth` is
    - `None` then raises `ValueError`.
    - Not of type `APIAuth` then raises `TypeError`.
- If `access_key` is
    - `empty` or `None` then raises `ValueError`.
    - Not of type `str` then raises `TypeError`.
- If `secret_key` is
    - `empty` or `None` then raises `ValueError`.
    - Not of type `str` then raises `TypeError`.
- If `bucket_name` is
    - `empty` or `None` then raises `ValueError`.
    - Not of type `str` then raises `TypeError`.
    - Doesn't exist then raises `BucketDoesNotExistsError`.

```
from zeblok.dataset import DataSet
from zeblok.auth import APIAuth

api_auth = APIAuth(
    app_url=f'<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

# While using Azure object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<account-name>",
  secret_key="<account-key>",
  bucket_name="<your-bucket-name>"
)

# While using AWS object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<aws-access-key-id>",
  secret_key="<aws-secret-access-key>",
  bucket_name="<your-bucket-name>"
)
```

Example

API Authentication

```
from zeblok.auth import APIAuth

api_auth = APIAuth(
    app_url=f'https://app.test.zeblok.com',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)
```

Datalake Authentication

```
from zeblok.dataset import DataSet
from zeblok.auth import APIAuth

api_auth = APIAuth(
    app_url=f'https://app.test.zeblok.com',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

# While using Azure object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<azure-account-name-from-your-UI>",
  secret_key="<azure-account-key-from-your-UI>",
  bucket_name="<your-bucket-name>"
)

# While using AWS object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<aws-access-key-id-from-your-UI>",
  secret_key="<aws-secret-access-key-from-your-UI>",
  bucket_name="<your-bucket-name>"
)
```

Plans

Using the SDK, we can

- Get all plans: `get_all()`
- Get a plan by id: `get_by_id(plan_id)`
- Validate a plan by id: `validate_id(plan_id)`
- Get filtered details of a plan: `get_filtered_details(plan_id, fields_req)`

Get all plans

- Returns all plans as `List[Dict]`.
- Raises
    - `AuthenticationError` if user not authenticated.
    - `AuthorizationError` if user is not permitted to view the list of plans.
    - `NoResourcesError` if no `plans` are available in the environment.

Get all plans info

```
from zeblok.plan import Plan
from zeblok.auth import APIAuth

api_auth = APIAuth(
    app_url=f'<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

plan_obj = Plan(api_auth=api_auth)

plan_obj.get_all(print_stdout=False)
# returns a list of dict wherein each dict contains a plan's info. Raises an exception if `no-resources-found` or there's an error.
```

Get all plans info and print on console

```
from zeblok.plan import Plan
from zeblok.auth import APIAuth

api_auth = APIAuth(
    app_url=f'<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

plan_obj = Plan(api_auth=api_auth)

plan_obj.get_all(print_stdout=True)
# returns a list of dict wherein each dict contains a plan's info. Prints the formatted list of plans on stdout. Raises an exception
```

Get plan information by id

- Returns a `Dict`.
- Raises
  - `AuthenticationError` if user not authenticated.
  - `AuthorizationError` if user is not permitted to view the list of plans.
- If `plan-id` is
  - `empty` or `None` then raises `ValueError`.
  - Not of type `str` then raises `TypeError`.
  - Not found in the environment then raises `ResourceNotFoundError`.
- Raises `ServerError` for any other errors.

Get a plan's info by id

```
from zeblok.plan import Plan
from zeblok.auth import APIAuth

api_auth = APIAuth(
    app_url=f'<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

plan_obj = Plan(api_auth=api_auth)

plan_obj.get_by_id(plan_id = <your-plan-id>, print_stdout=False)
# returns a dict containing `plan-id` info. Raises an exception if `plan-id` not found or there's an error.
```

Get a plan's info by id and print on console

```
from zeblok.plan import Plan
from zeblok.auth import APIAuth

api_auth = APIAuth(
    app_url=f'<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

plan_obj = Plan(api_auth=api_auth)

plan_obj.get_by_id(plan_id = <your-plan-id>, print_stdout=True)
# returns a dict containing `plan-id` info. Prints the formatted info on stdout. Raises an exception if `plan-id` not found or there
```

Validate a plan by id

- Returns `True` if `plan-id` exists in the environment else `False`.
- Raises
  - `AuthenticationError` if user not authenticated.

    - `AuthorizationError` if user is not permitted to view the list of plans.
- If `raise_exception` parameter is `True` and `plan-id` is
    - `empty` or `None` then raises `ValueError`.
    - Not of type `str` then raises `TypeError`.
    - Not found in the environment then raises `ResourceNotFoundError`.
    - Raises `ServerError` for any other errors.

Validate if plan exists in the environment

```
from zeblok.plan import Plan
from zeblok.auth import APIAuth

api_auth = APIAuth(
    app_url=f'<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

plan_obj = Plan(api_auth=api_auth)

plan_obj.validate_id(plan_id = <your-plan-id>, raise_exception=False)
# returns True if `plan-id` exists else False.
```

Validate if plan exists in the environment. Raise an exception if `plan-id` not found or there's an error.

```
from zeblok.plan import Plan
from zeblok.auth import APIAuth

api_auth = APIAuth(
    app_url=f'<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

plan_obj = Plan(api_auth=api_auth)

plan_obj.validate_id(plan_id = <your-plan-id>, raise_exception=True)
# returns True if `plan-id` exists else raises an Exception.
```

Get filtered details of a plan

- Returns a `Dict` containing the keys in the `fields_req` parameter and their respective value. Value is `Non` if the key doesn't exist.
- Raises
    - `AuthenticationError` if user not authenticated.
    - `AuthorizationError` if user is not permitted to view the list of plans.
- If `plan-id` is
    - `empty` or `None` then raises `ValueError`.
    - Not of type `str` then raises `TypeError`.
    - Not found in the environment then raises `ResourceNotFoundError`.
- Raises `ServerError` for any other errors.

Get `id` and `type` of the plan if the `plan-id` exists in the environment

```
from zeblok.plan import Plan
from zeblok.auth import APIAuth

api_auth = APIAuth(
    app_url=f'<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

plan_obj = Plan(api_auth=api_auth)

plan_obj.get_filtered_details(plan_id="<your-plan-id>", fields_req=['id', 'type'])
# Returns a Dict: {'id': <plan-id>, 'type': '<type-of-plan>'}
```

Validate if plan exists in the environment. Raise an exception if `plan-id` not found or there's an error.

```
from zeblok.plan import Plan
from zeblok.auth import APIAuth

api_auth = APIAuth(
    app_url=f'<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

plan_obj = Plan(api_auth=api_auth)

plan_obj.validate_id(plan_id = <your-plan-id>, raise_exception=True)
# returns True if `plan-id` exists else raises an Exception.
```

Namespaces

Using the SDK, we can

- Get all namespaces: `get_all()`
- Get a namespace by id: `get_by_id(namespace_id)`
- Validate a namespace by id: `validate_id(namespace_id)`

Get all namespaces

- Returns all namespaces as `List[Dict]`.
- Raises
    - `AuthenticationError` if user not authenticated.
    - `AuthorizationError` if user is not permitted to view the list of plans.
    - `NoResourcesError` if no `namespaces` are available in the environment.

Get all namespaces info

```
from zeblok.namspace import Namespace
from zeblok.auth import APIAuth

api_auth = APIAuth(
    app_url='<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

namespace_obj = Namespace(api_auth=api_auth)

namespace_obj.get_all(print_stdout=False)
# returns a list of dict wherein each dict contains a namespace's info. Raises an exception if `no-resources-found` or there's an er
```

Get all namespaces info and print on console

```
from zeblok.namspace import Namespace
from zeblok.auth import APIAuth

api_auth = APIAuth(
    app_url='<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

namespace_obj = Namespace(api_auth=api_auth)

namespace_obj.get_all(print_stdout=True)
# returns a list of dict wherein each dict contains a namespace's info. Prints the formatted list of namespaces on stdout. Raises an
```

Get namespace information by id

- Returns a `Dict`.
- Raises
    - `AuthenticationError` if user not authenticated.
    - `AuthorizationError` if user is not permitted to view the list of plans.
- If `namespace-id` is
    - `empty` or `None` then raises `ValueError`.
    - Not of type `str` then raises `TypeError`.
    - Not found in the environment then raises `ResourceNotFoundError`.
- Raises `ServerError` for any other errors.

Get a namespace's info by id

```
from zeblok.namspace import Namespace
from zeblok.auth import APIAuth

api_auth = APIAuth(
    app_url='<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

namespace_obj = Namespace(api_auth=api_auth)

namespace_obj.get_by_id(namespace_id = <your-namespace-id>, print_stdout=False)
# returns a dict containing `namespace-id` info. Raises an exception if `namespace-id` not found or there's an error.
```

Get a namespace's info by id and print on console

```
from zeblok.namspace import Namespace
from zeblok.auth import APIAuth

api_auth = APIAuth(
    app_url='<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

namespace_obj = Namespace(api_auth=api_auth)

namespace_obj.get_by_id(namespace_id = <your-namespace-id>, print_stdout=True)
# returns a dict containing `namespace-id` info. Prints the formatted info on stdout. Raises an exception if `namespace-id` not foun
```

Validate a namespace by id

- Returns `True` if `namespace-id` exists in the environment else `False`.
- Raises
    - `AuthenticationError` if user not authenticated.

    - `AuthorizationError` if user is not permitted to view the list of plans.
- If `raise_exception` parameter is `True` and `namespace-id` is
    - `empty` or `None` then raises `ValueError`.
    - Not of type `str` then raises `TypeError`.
    - Not found in the environment then raises `ResourceNotFoundError`.
    - Raises `ServerError` for any other errors.

Validate if namespace exists in the environment

```
from zeblok.namspace import Namespace
from zeblok.auth import APIAuth

api_auth = APIAuth(
    app_url='<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

namespace_obj = Namespace(api_auth=api_auth)

namespace_obj.validate_id(namspace_id = <your-namspace-id>, raise_exception=False)
# returns True if `namspace-id` exists else False.
```

Validate if namspace exists in the environment. Raise an exception if `namspace-id` not found or there's an error.

```
from zeblok.namspace import Namespace
from zeblok.auth import APIAuth

api_auth = APIAuth(
    app_url='<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

namespace_obj = Namespace(api_auth=api_auth)

namespace_obj.validate_id(namspace_id = <your-plan-id>, raise_exception=True)
# returns True if `namspace-id` exists else raises an Exception.
```

MicroServices

Using the SDK, we can

- Get all microservices: `get_all()`
- Get a microservice by id: `get_by_id(microservice_id)`
- Validate a microservice by id: `validate_id(microservice_id)`
- Spawn a microservices by id: `spawn(display_name, microservice_id, plan_id, microservice_name, namespace_id, envs, ports, args, command)`

Get all microservices

- Returns all microservices as `List[Dict]`.
- Raises
    - `AuthenticationError` if user not authenticated.
    - `AuthorizationError` if user is not permitted to view the list of microservices.
    - `NoResourcesError` if no `microservices` are available in the environment.

Get all microservices info

```
from zeblok.microservice import MicroService
from zeblok.auth import APIAuth

api_auth = APIAuth(
    app_url=f'<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

microservice_obj = MicroService(api_auth=api_auth)

microservice_obj.get_all(print_stdout=False)
# returns a list of dict wherein each dict contains a microservice's info. Raises an exception if `no-resources-found` or there's an
```

Get all microservices info and print on console

```
from zeblok.microservice import MicroService
from zeblok.auth import APIAuth

api_auth = APIAuth(
    app_url=f'<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

microservice_obj = MicroService(api_auth=api_auth)

microservice_obj.get_all(print_stdout=True)
# returns a list of dict wherein each dict contains a microservice's info. Prints the formatted list of microservices on stdout. Rai
```

Get microservice information by id

- Returns a `Dict`.
- Raises
  - `AuthenticationError` if user not authenticated.
  - `AuthorizationError` if user is not permitted to view the list of microservices.
- If `microservice-id` is
  - `empty` or `None` then raises `ValueError`.
  - Not of type `str` then raises `TypeError`.
  - Not found in the environment then raises `ResourceNotFoundError`.
- Raises `ServerError` for any other errors.

Get a microservice's info by id

```
from zeblok.microservice import MicroService
from zeblok.auth import APIAuth

api_auth = APIAuth(
    app_url=f'<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

microservice_obj = MicroService(api_auth=api_auth)

microservice_obj.get_by_id(microservice_id=<your-microservice-id>, print_stdout=False)
# returns a dict containing `microservice-id` info. Raises an exception if `microservice-id` not found or there's an error.
```

Get a microservice's info by id and print on console

```
from zeblok.microservice import MicroService
from zeblok.auth import APIAuth

api_auth = APIAuth(
    app_url=f'<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

microservice_obj = MicroService(api_auth=api_auth)

microservice_obj.get_by_id(microservice_id=<your-microservice-id>, print_stdout=True)
# returns a dict containing `microservice-id` info. Prints the formatted info on stdout. Raises an exception if `microservice-id` no
```

Validate a microservice by id

- Returns `True` if `microservice-id` exists in the environment else `False`.
- Raises
  - `AuthenticationError` if user not authenticated.
  - `AuthorizationError` if user is not permitted to view the list of microservices.
- If `raise_exception` parameter is `True` and `microservice-id` is
  - `empty` or `None` then raises `ValueError`.
  - Not of type `str` then raises `TypeError`.
  - Not found in the environment then raises `ResourceNotFoundError`.
  - Raises `ServerError` for any other errors.

Validate if microservice exists in the environment

```
from zeblok.microservice import Microservice
from zeblok.auth import APIAuth

api_auth = APIAuth(
    app_url=f'<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

microservice_obj = Microservice(api_auth=api_auth)

microservice_obj.validate_id(microservice_id = <your-microservice-id>, raise_exception=False)
# returns True if `microservice-id` exists else False.
```

Validate if microservice exists in the environment. Raise an exception if `microservice-id` not found or there's an error.

```
from zeblok.microservice import Microservice
from zeblok.auth import APIAuth

api_auth = APIAuth(
    app_url=f'<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

microservice_obj = MicroService(api_auth=api_auth)

microservice_obj.validate_id(microservice_id=<your-microservice-id>, raise_exception=True)
# returns True if `plan-id` exists else raises an Exception.
```

Spawn a microservice by id

- User provides the following parameters:
  - `microservice_id` : Id of the microservice you want to spawn
  - `display_name` : Unique `display_name` of the respective microservice. You can get the `display_name(s)` of the respective microservice from the

function `get_by_id(microservice_id)`.
    - `plan_id` : Id of the resource plan.
    - `microservice_name` : A meaningful name to identify the microservice spawned in the Ai-MicroCloud environment.
    - `namespace_id` : Id of the namespace in which you wish to spawn the microservice.
    - `ports` : The list of ports along with protocol that user wishes to open. Pass it as a List[Dict]. For ex. `[{"protocol": "HTTP", "number": 8501}]`.
    - `envs` : The list of environment variables you wish to pass to your microservice. Pass it as a List[Dict]. For ex. `[{"key": "enable_feature", "value": True}]`.
    - `args` : The list of additional arguments you wish to pass to your microservice. Pass it as a List[Dict]. For ex. `[{"key": "arg 1"}]`.
    - `command` : The list of commands you wish to pass to your microservice. Pass it as a string with commands separated with commas. For ex. `ls, ps`
- Raises
    - `AuthenticationError` if user not authenticated.
    - `AuthorizationError` if user is not permitted to view the list of microservices.
- If `microservice_id` is
    - `empty` or `None` then raises `ValueError`.
    - Not of type `str` then raises `TypeError`.
    - Not found in the environment then raises `ResourceNotFoundError`.
- If `display_name` is
    - `empty` or `None` or `Not Found` then raises `ValueError`.
    - Not of type `str` then raises `TypeError`.
- If `plan_id` is
    - `empty` or `None` or `Not-of-Type-MicroService` then raises `ValueError`.
    - Not of type `str` then raises `TypeError`.
    - Not found in the environment then raises `ResourceNotFoundError`.
- If `microservice_name` is
    - `empty` or `None` then raises `ValueError`.
    - Not of type `str` then raises `TypeError`.
- If `namespace_id` is
    - `empty` or `None` or `Not-of-Type-MicroService` then raises `ValueError`.
    - Not of type `str` then raises `TypeError`.
    - Not found in the environment then raises `ResourceNotFoundError`.
- If `ports` is
    - Not of type `List[Dict]` then raises `TypeError`.
    - not according to the format specified above then raises `ValueError`.
- If `envs` is
    - Not of type `List[Dict]` then raises `TypeError`.
    - not according to the format specified above then raises `ValueError`.
- If `args` is
    - Not of type `List[Dict]` then raises `TypeError`.
    - not according to the format specified above then raises `ValueError`.
- If `command` is
    - Not of type `str` then raises `TypeError`.
- Raises `ServerError` for any other errors.

Spawn a MicroService with `microservice-id`.

```
 from zeblok.microservice import Microservice
from zeblok.auth import APIAuth

api_auth = APIAuth(
    app_url=f'<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

microservice_obj = MicroService(api_auth=api_auth)

microservice_obj.spawn(
    display_name='<valid-display-name>',
    microservice_id='<valid-microservice-id>',
    plan_id='<valid-plan-id>',
    microservice_name='<valid-microservice-name>',
    namespace_id='<valid-namespace-id>',
    ports=[{"protocol": "HTTP", "number": 8501}],
    envs= [{"key": "<valid-key>", "value": "<valid-value>"}],
    args=[{"key": "valid_key"}],
    command="<comma-separate-valid-commands>"
) # returns True with a user-friendly message if the microservice is spawned correctly else raises an Exception.
```

Orchestrations

Using the SDK, we can

- Get all orchestrations: `get_all()`
- Get a orchestration by id: `get_by_id(orchestration_id)`
- Validate a orchestration by id: `validate_id(orchestration_id)`
- Spawn a orchestrations by id: `spawn(orchestration_id, plan_id, namespace_id, orchestration_name=, min_workers, max_workers)`

Get all orchestrations

- Returns all orchestrations as `List[Dict]`.
- Raises
    - `AuthenticationError` if user not authenticated.
    - `AuthorizationError` if user is not permitted to view the list of orchestrations.
    - `NoResourcesError` if no `orchestrations` are available in the environment.

Get all orchestrations info

```
from zeblok.orchestration import Orchestration
from zeblok.auth import APIAuth

api_auth = APIAuth(
    app_url=f'<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

orchestration_obj = Orchestration(api_auth=api_auth)

orchestration_obj.get_all(print_stdout=False)
# returns a list of dict wherein each dict contains a orchestration's info. Raises an exception if `no-resources-found` or there's a
```

Get all orchestrations info and print on console

```
from zeblok.orchestration import Orchestration
from zeblok.auth import APIAuth

api_auth = APIAuth(
    app_url=f'<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

orchestration_obj = Orchestration(api_auth=api_auth)

orchestration_obj.get_all(print_stdout=True)
# returns a list of dict wherein each dict contains a orchestration's info. Prints the formatted list of orchestrations on stdout. R
```

Get orchestration information by id

- Returns a `Dict`.
- Raises
    - `AuthenticationError` if user not authenticated.
    - `AuthorizationError` if user is not permitted to view the list of orchestrations.
- If `orchestration-id` is
    - `empty` or `None` then raises `ValueError`.
    - Not of type `str` then raises `TypeError`.
    - Not found in the environment then raises `ResourceNotFoundError`.
- Raises `ServerError` for any other errors.

Get a orchestration's info by id

```
from zeblok.orchestration import Orchestration
from zeblok.auth import APIAuth

api_auth = APIAuth(
    app_url=f'<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

orchestration_obj = Orchestration(api_auth=api_auth)

orchestration_obj.get_by_id(orchestration_id=<your-orchestration-id>, print_stdout=False)
# returns a dict containing `orchestration-id` info. Raises an exception if `orchestration-id` not found or there's an error.
```

Get a orchestration's info by id and print on console

```
from zeblok.orchestration import Orchestration
from zeblok.auth import APIAuth

api_auth = APIAuth(
    app_url=f'<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

orchestration_obj = Orchestration(api_auth=api_auth)

orchestration_obj.get_by_id(orchestration_id=<your-orchestration-id>, print_stdout=True)
# returns a dict containing `orchestration-id` info. Prints the formatted info on stdout. Raises an exception if `orchestration-id`
```

Validate a orchestration by id

- Returns `True` if `orchestration-id` exists in the environment else `False`.
- Raises
  - `AuthenticationError` if user not authenticated.
  - `AuthorizationError` if user is not permitted to view the list of orchestrations.
- If `raise_exception` parameter is `True` and `orchestration-id` is
  - `empty` or `None` then raises `ValueError`.
  - Not of type `str` then raises `TypeError`.
  - Not found in the environment then raises `ResourceNotFoundError`.
  - Raises `ServerError` for any other errors.

Validate if orchestration exists in the environment

```
from zeblok.orchestration import Orchestration
from zeblok.auth import APIAuth

api_auth = APIAuth(
    app_url=f'<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

orchestration_obj = Orchestration(api_auth=api_auth)

orchestration_obj.validate_id(orchestration_id = <your-orchestration-id>, raise_exception=False)
# returns True if `orchestration-id` exists else False.
```

Validate if orchestration exists in the environment. Raise an exception if `orchestration-id` not found or there's an error.

```
 from zeblok.orchestration import Orchestration
from zeblok.auth import APIAuth

api_auth = APIAuth(
    app_url=f'<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

orchestration_obj = Orchestration(api_auth=api_auth)

orchestration_obj.validate_id(orchestration_id=<your-orchestration-id>, raise_exception=True)
# returns True if `orchestration-id` exists else raises an Exception.
```

Spawn a orchestration by id

- User provides the following parameters:
  - `orchestration_id` : Id of the orchestration you want to spawn
  - `plan_id` : Id of the resource plan.
  - `namespace_id` : Id of the namespace in which you wish to spawn the orchestration.
  - `orchestration_name` : A meaningful name to identify the orchestration spawned in the Ai-MicroCloud environment.
  - `min_workers` : Minimum number of worker nodes you want to spawn in the add-on orchestration.
  - `max_workers` : Maximum number of worker nodes you want to spawn in the add-on orchestration.
- Raises
  - `AuthenticationError` if user not authenticated.
  - `AuthorizationError` if user is not permitted to view the list of orchestrations.
- If `orchestration_id` is
  - `empty` or `None` then raises `ValueError`.
  - Not of type `str` then raises `TypeError`.
  - Not found in the environment then raises `ResourceNotFoundError`.
- If `plan_id` is
  - `empty` or `None` or `Not-of-Type-Orchestration` then raises `ValueError`.
  - Not of type `str` then raises `TypeError`.
  - Not found in the environment then raises `ResourceNotFoundError`.
- If `namespace_id` is
  - `empty` or `None` or `Not-of-Type-Orchestration` then raises `ValueError`.
  - Not of type `str` then raises `TypeError`.
  - Not found in the environment then raises `ResourceNotFoundError`.
- If `orchestration_name` is
  - `empty` or `None` then raises `ValueError`.
  - Not of type `str` then raises `TypeError`.
- If `min_workers` is
  - Not of type `int` then raises `TypeError`.
  - Less than 1 then raises `ValueError`
- If `max_workers` is
  - Not of type `int` then raises `TypeError`.
  - Less than `min_workers` then raises `ValueError`
- Raises `ServerError` for any other errors.

Spawn an Orchestration with `orchestration-id`.

```
 from zeblok.orchestration import Orchestration
from zeblok.auth import APIAuth

api_auth = APIAuth(
    app_url=f'<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

orchestration_obj = Orchestration(api_auth=api_auth)

orchestration_obj.spawn(
    orchestration_id='<valid-orchestration-id>',
    plan_id='<valid-plan-id>',
    namespace_id='<valid-namespace-id>',
    orchestration_name="<valid-orchestration-name>",
    min_workers=2, max_workers=3
) # returns True with a user-friendly message if the orchestration is spawned correctly else raises an Exception.
```

AI-APIs

Using the SDK, we can

- Get all AI-APIs with `deployed` state: `get_all(state=deployed)`
- Get all AI-APIs with `ready` state: `get_all(state=ready)`
- Validate an AI-API: `validate(image_name, state)`
- Create and Spawn an AI-API: `create_and_spawn(ai_api_name, model_folder_path, plan_id, namespace_id, ai_api_type)`
- Create an AI-API: `create(ai_api_name, model_folder_path, ai_api_type)`
- Spawn an AI-API: `spawn(image_name, namespace_id, plan_id)`

Get all AI-APIs with `deployed` state

- Returns all `deployed` state AI-APIs as `List[Dict]`.
- Raises
    - `AuthenticationError` if user not authenticated.
    - `AuthorizationError` if user is not permitted to view the list of AI-APIs.
    - `NoResourcesError` if no `deployed` state `AI-APIs` are available in the environment.
    - `ValueError` if `state` parameter value is not `deployed` or `ready`.

Get all `deployed` state AI-APIs info

```
from zeblok.dataset import DataSet
from zeblok.auth import APIAuth
from zeblok.api import API

api_auth = APIAuth(
    app_url='<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

# While using Azure object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<azure-account-name-from-your-UI>",
  secret_key="<azure-account-key-from-your-UI>",
  bucket_name="<your-bucket-name>"
)

# While using AWS object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<aws-access-key-id-from-your-UI>",
  secret_key="<aws-secret-access-key-from-your-UI>",
  bucket_name="<your-bucket-name>"
)

ai_api_obj = API(api_auth=api_auth, dataset=dataset_obj)

ai_api_obj.get_all(state="deployed", print_stdout=False)
# returns a list of dict wherein each dict contains an AI-API's info. Raises an exception if `no-resources-found` or there's an erro
```

Get all `deployed` state AI-APIs info and print on console

```
from zeblok.dataset import DataSet
from zeblok.auth import APIAuth
from zeblok.api import API

api_auth = APIAuth(
    app_url='<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

# While using Azure object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<azure-account-name-from-your-UI>",
  secret_key="<azure-account-key-from-your-UI>",
  bucket_name="<your-bucket-name>"
)

# While using AWS object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<aws-access-key-id-from-your-UI>",
  secret_key="<aws-secret-access-key-from-your-UI>",
  bucket_name="<your-bucket-name>"
)

ai_api_obj = API(api_auth=api_auth, dataset=dataset_obj)

ai_api_obj.get_all(state="deployed", print_stdout=True)
# returns a list of dict wherein each dict contains a AI-API's info. Prints the formatted list of AI-APIs on stdout. Raises an excep
```

Get all AI-APIs with `ready` state

- Returns all `ready` state AI-APIs as `List[Dict]`.
- Raises
    - `AuthenticationError` if user not authenticated.
    - `AuthorizationError` if user is not permitted to view the list of AI-APIs.
    - `NoResourcesError` if no `ready` state `AI-APIs` are available in the environment.
    - `ValueError` if `state` parameter value is not `deployed` or `ready`.

Get all `ready` state AI-APIs info

```
from zeblok.dataset import DataSet
from zeblok.auth import APIAuth
from zeblok.api import API

api_auth = APIAuth(
    app_url='<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

# While using Azure object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<azure-account-name-from-your-UI>",
  secret_key="<azure-account-key-from-your-UI>",
  bucket_name="<your-bucket-name>"
)

# While using AWS object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<aws-access-key-id-from-your-UI>",
  secret_key="<aws-secret-access-key-from-your-UI>",
  bucket_name="<your-bucket-name>"
)

ai_api_obj = API(api_auth=api_auth, dataset=dataset_obj)

ai_api_obj.get_all(state="ready", print_stdout=False)
# returns a list of dict wherein each dict contains an AI-API's info. Raises an exception if `no-resources-found` or there's an erro
```

Get all `ready` state AI-APIs info and print on console

```
from zeblok.dataset import DataSet
from zeblok.auth import APIAuth
from zeblok.api import API

api_auth = APIAuth(
    app_url='<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

# While using Azure object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<azure-account-name-from-your-UI>",
  secret_key="<azure-account-key-from-your-UI>",
  bucket_name="<your-bucket-name>"
)

# While using AWS object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<aws-access-key-id-from-your-UI>",
  secret_key="<aws-secret-access-key-from-your-UI>",
  bucket_name="<your-bucket-name>"
)

ai_api_obj = API(api_auth=api_auth, dataset=dataset_obj)

ai_api_obj.get_all(state="ready", print_stdout=True)
# returns a list of dict wherein each dict contains a AI-API's info. Prints the formatted list of AI-APIs on stdout. Raises an excep
```

Get all AI-APIs with `invalid-state` state

- Returns all `invalid-state` state AI-APIs as `List[Dict]`.
- Raises
    - `AuthenticationError` if user not authenticated.
    - `AuthorizationError` if user is not permitted to view the list of AI-APIs.
    - `NoResourcesError` if no `ready` state `AI-APIs` are available in the environment.
    - `ValueError` if `state` parameter value is not `deployed` or `ready`.

```
from zeblok.dataset import DataSet
from zeblok.auth import APIAuth
from zeblok.api import API

api_auth = APIAuth(
    app_url='<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

# While using Azure object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<azure-account-name-from-your-UI>",
  secret_key="<azure-account-key-from-your-UI>",
  bucket_name="<your-bucket-name>"
)

# While using AWS object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<aws-access-key-id-from-your-UI>",
  secret_key="<aws-secret-access-key-from-your-UI>",
  bucket_name="<your-bucket-name>"
)

ai_api_obj = API(api_auth=api_auth, dataset=dataset_obj)

try:
    print(ai_api_obj.get_all(state="invalid-state", print_stdout=False)) # Raises ValueError
except ValueError as e:
    print(f"{e.__class__.__name__}: {e}")
```

Validate an AI-API by image_name and state

- Returns `True` if given `image_name` with `state` exists in the environment else `False`.
- Raises
    - `AuthenticationError` if user not authenticated.
    - `AuthorizationError` if user is not permitted to view the list of AI-APIs.
- If `image_name` is
    - `empty` or `None` then raises `ValueError`.
    - Not of type `str` then raises `TypeError`.
- If `state` is
    - `empty` or `None` or `not one of ['deployed', 'ready']` then raises `ValueError`.
    - Not of type `str` then raises `TypeError`.
- Raises `ServerError` for any other errors.

Validate if AI-API with `image-name` and `valid-state` exists in the environment

```
from zeblok.dataset import DataSet
from zeblok.auth import APIAuth
from zeblok.api import API

api_auth = APIAuth(
    app_url='<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

# While using Azure object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<azure-account-name-from-your-UI>",
  secret_key="<azure-account-key-from-your-UI>",
  bucket_name="<your-bucket-name>"
)

# While using AWS object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<aws-access-key-id-from-your-UI>",
  secret_key="<aws-secret-access-key-from-your-UI>",
  bucket_name="<your-bucket-name>"
)

ai_api_obj = API(api_auth=api_auth, dataset=dataset_obj)

ai_api_obj.validate(image_name = "<image-name>", state="<valid-state>")
# returns True if `AI-API` exists else False. Raises error as specified in the description.
```

Create and Spawn an AI-API

- Creation followed automatic spawning of the AI-API.
- User provides the following parameters:
    - `ai_api_name` : A meaningful name to identify the AI-API created and spawned in the Ai-MicroCloud environment.
    - `model_folder_path` : Local model folder path to be spawned as an AI-API.
    - `plan_id` : Id of the resource plan.
    - `namespace_id` : Id of the namespace in which you wish to spawn the AI-API.
    - `ai_api_type` : Choose one of the following: `'bentoml', 'openvino', 'mlflow', 'llm'`
- Raises
    - `AuthenticationError` if user not authenticated.
    - `AuthorizationError` if user is not permitted to view the list of AI-APIs.
- If `ai_api_name` is
    - `empty` or `None` then raises `ValueError`.
    - Not of type `str` then raises `TypeError`.
- If `model_folder_path`
    - Is `empty` then raises `ValueError`.
    - Is `Invalid` or `doesn't exist` then raises `DirectoryNotFoundError`.
    - Is `Not-a-directory` then raises `NotADirectoryError`.
    - Does not have `write-permission` then raises `PermissionError`
- If `plan_id` is
    - `empty` or `None` or `Not-of-Type-Orchestration` then raises `ValueError`.
    - Not of type `str` then raises `TypeError`.
    - Not found in the environment then raises `ResourceNotFoundError`.
- If `namespace_id` is
    - `empty` or `None` or `Not-of-Type-Orchestration` then raises `ValueError`.
    - Not of type `str` then raises `TypeError`.
    - Not found in the environment then raises `ResourceNotFoundError`.
- If `ai_api_type` is
    - `empty` or `None` then raises `ValueError`.
    - Not of type `str` then raises `TypeError`.
    - Not one of the following `'bentoml', 'openvino', 'mlflow', 'llm'` then raises `ValueError`.
- Raises `ServerError` for any other errors.

create and spawn an AI-API

```
 from zeblok.dataset import DataSet
from zeblok.auth import APIAuth
from zeblok.api import API

api_auth = APIAuth(
    app_url='<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

# While using Azure object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<azure-account-name-from-your-UI>",
  secret_key="<azure-account-key-from-your-UI>",
  bucket_name="<your-bucket-name>"
)

# While using AWS object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<aws-access-key-id-from-your-UI>",
  secret_key="<aws-secret-access-key-from-your-UI>",
  bucket_name="<your-bucket-name>"
)

ai_api_obj = API(api_auth=api_auth, dataset=dataset_obj)

ai_api_obj.create_and_spawn(
    ai_api_name='<valid-ai-api-name>',
    model_folder_path='<valid-model-folder-path>',
    plan_id='<valid-plan-id>',
    namespace_id='<valid-namespace-id>',
    ai_api_type='<valid-ai-api-type>'
) # returns True with a user-friendly message if the AI-API is submitted correctly else raises an Exception.
```

Create an AI-API

- Creates an AI-API but doesn't spawn the AI-API automatically.
- User provides the following parameters:
  - `ai_api_name` : A meaningful name to identify the AI-API created and spawned in the Ai-MicroCloud environment.
  - `model_folder_path` : Local model folder path to be spawned as an AI-API.
  - `ai_api_type` : Choose one of the following: `'bentoml', 'openvino', 'mlflow', 'llm'`
- Raises
  - `AuthenticationError` if user not authenticated.
  - `AuthorizationError` if user is not permitted to view the list of AI-APIs.
- If `ai_api_name` is
  - `empty` or `None` then raises `ValueError`.
  - Not of type `str` then raises `TypeError`.
- If `model_folder_path`
  - Is `empty` then raises `ValueError`.
  - Is `Invalid` or `doesn't exist` then raises `DirectoryNotFoundError`.
  - Is `Not-a-directory` then raises `NotADirectoryError`.
  - Does't have `write-permission` then raises `PermissionError`
- If `ai_api_type` is
  - `empty` or `None` then raises `ValueError`.
  - Not of type `str` then raises `TypeError`.
  - Not one of the following `'bentoml', 'openvino', 'mlflow', 'llm'` then raises `ValueError`.
- Raises `ServerError` for any other errors.

create an AI-API

```
 from zeblok.dataset import DataSet
from zeblok.auth import APIAuth
from zeblok.api import API

api_auth = APIAuth(
    app_url='<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

# While using Azure object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<azure-account-name-from-your-UI>",
  secret_key="<azure-account-key-from-your-UI>",
  bucket_name="<your-bucket-name>"
)

# While using AWS object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<aws-access-key-id-from-your-UI>",
  secret_key="<aws-secret-access-key-from-your-UI>",
  bucket_name="<your-bucket-name>"
)

ai_api_obj = API(api_auth=api_auth, dataset=dataset_obj)

ai_api_obj.create(
    ai_api_name='<valid-ai-api-name>',
    model_folder_path='<valid-model-folder-path>',
    ai_api_type='<valid-ai-api-type>'
) # returns image_name and prints image_name if the AI-API creation request is submitted correctly else raises an Exception.
```

Spawn an AI-API

- Spawns an already created AI-API in the Ai-MicroCloud environment.
- User provides the following parameters:
    - `image_name` : A meaningful name to identify the AI-API created and spawned in the Ai-MicroCloud environment.
    - `plan_id` : Id of the resource plan.
    - `namespace_id` : Id of the namespace in which you wish to spawn the AI-API.
- Raises
    - `AuthenticationError` if user not authenticated.
    - `AuthorizationError` if user is not permitted to view the list of AI-APIs.
- If `ai_api_name` is
    - `empty` or `None` or `invalid` then raises `ValueError` .
    - Not of type `str` then raises `TypeError` .
- If `plan_id` is
    - `empty` or `None` or `Not-of-Type-Orchestration` then raises `ValueError` .
    - Not of type `str` then raises `TypeError` .
    - Not found in the environment then raises `ResourceNotFoundError` .
- If `namespace_id` is
    - `empty` or `None` or `Not-of-Type-Orchestration` then raises `ValueError` .
    - Not of type `str` then raises `TypeError` .
    - Not found in the environment then raises `ResourceNotFoundError` .
- Raises `ServerError` for any other errors.

spawn an AI-API

```
 from zeblok.dataset import DataSet
from zeblok.auth import APIAuth
from zeblok.api import API

api_auth = APIAuth(
    app_url='<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

# While using Azure object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<azure-account-name-from-your-UI>",
  secret_key="<azure-account-key-from-your-UI>",
  bucket_name="<your-bucket-name>"
)

# While using AWS object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<aws-access-key-id-from-your-UI>",
  secret_key="<aws-secret-access-key-from-your-UI>",
  bucket_name="<your-bucket-name>"
)

ai_api_obj = API(api_auth=api_auth, dataset=dataset_obj)

ai_api_obj.spawn(
    image_name='<valid-image-name>',
    plan_id='<valid-plan-id>',
    namespace_id='<valid-namespace-id>',
) # returns True with a user-friendly message if the AI-API spawning request is submitted correctly else raises an Exception.
```

AI-Pipelines

Using the SDK, we can

- Get all AI-Pipelines with `created` state: `get_all(state=created)`
- Get all AI-Pipelines with `ready` state: `get_all(state=ready)`
- Validate an AI-Pipeline: `validate(image_name, state)`
- Create and Spawn an AI-Pipeline: `create_and_spawn(ai_api_name, model_folder_path, plan_id, namespace_id, ai_api_type)`
- Create an AI-Pipeline: `create(ai_pipeline_name, ai_pipeline_folder_path, caas_plan_id, ai_pipeline_plan_id, namespace_id)`
- Spawn an AI-Pipeline: `spawn(ai_pipeline_plan_id, namespace_id, ai_pipeline_image_name)`

Get all AI-Pipelines with `created` state

- Returns all `created` state AI-APIs as `List[Dict]`.
- Raises
    - `AuthenticationError` if user not authenticated.
    - `AuthorizationError` if user is not permitted to view the list of AI-APIs.
    - `NoResourcesError` if no `created` state `AI-Pipelines` are available in the environment.
- If `state` is
    - `empty` or `None` then raises `ValueError`.
    - Not of type `str` then raises `TypeError`.
    - Not `created` or `ready` then raises `ValueError`.

Get all `created` state AI-Pipelines info

```
from zeblok.dataset import DataSet
from zeblok.auth import APIAuth
from zeblok.pipeline import Pipeline

api_auth = APIAuth(
    app_url='<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

# While using Azure object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<azure-account-name-from-your-UI>",
  secret_key="<azure-account-key-from-your-UI>",
  bucket_name="<your-bucket-name>"
)

# While using AWS object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<aws-access-key-id-from-your-UI>",
  secret_key="<aws-secret-access-key-from-your-UI>",
  bucket_name="<your-bucket-name>"
)

ai_pipeline_obj = Pipeline(api_auth=api_auth, dataset=dataset_obj)

ai_pipeline_obj.get_all(state="created", print_stdout=False)
# returns a list of dict wherein each dict contains an AI-Pipeline's info. Raises an exception if `no-resources-found` or there's an
```

Get all `created` state AI-Pipelines info and print on console

```
from zeblok.dataset import DataSet
from zeblok.auth import APIAuth
from zeblok.pipeline import Pipeline

api_auth = APIAuth(
    app_url='<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

# While using Azure object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<azure-account-name-from-your-UI>",
  secret_key="<azure-account-key-from-your-UI>",
  bucket_name="<your-bucket-name>"
)

# While using AWS object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<aws-access-key-id-from-your-UI>",
  secret_key="<aws-secret-access-key-from-your-UI>",
  bucket_name="<your-bucket-name>"
)

ai_pipeline_obj = Pipeline(api_auth=api_auth, dataset=dataset_obj)

ai_pipeline_obj.get_all(state="created", print_stdout=True)
# returns a list of dict wherein each dict contains an AI-Pipeline's info. Prints the formatted list of AI-APIs on stdout. Raises an
```

Get all AI-Pipelines with `ready` state

- Returns all `ready` state AI-APIs as `List[Dict]`.
- Raises
    - `AuthenticationError` if user not authenticated.
    - `AuthorizationError` if user is not permitted to view the list of AI-APIs.
    - `NoResourcesError` if no `ready` state `AI-Pipelines` are available in the environment.
- If `state` is
    - `empty` or `None` then raises `ValueError`.
    - Not of type `str` then raises `TypeError`.
    - Not `created` or `ready` then raises `ValueError`.

Get all `ready` state AI-Pipelines info

```
from zeblok.dataset import DataSet
from zeblok.auth import APIAuth
from zeblok.pipeline import Pipeline

api_auth = APIAuth(
    app_url='<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

# While using Azure object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<azure-account-name-from-your-UI>",
  secret_key="<azure-account-key-from-your-UI>",
  bucket_name="<your-bucket-name>"
)

# While using AWS object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<aws-access-key-id-from-your-UI>",
  secret_key="<aws-secret-access-key-from-your-UI>",
  bucket_name="<your-bucket-name>"
)

ai_pipeline_obj = Pipeline(api_auth=api_auth, dataset=dataset_obj)

ai_pipeline_obj.get_all(state="ready", print_stdout=False)
# returns a list of dict wherein each dict contains an AI-Pipeline's info. Raises an exception if `no-resources-found` or there's an
```

Get all `ready` state AI-Pipelines info and print on console

```
from zeblok.dataset import DataSet
from zeblok.auth import APIAuth
from zeblok.pipeline import Pipeline

api_auth = APIAuth(
    app_url='<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

# While using Azure object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<azure-account-name-from-your-UI>",
  secret_key="<azure-account-key-from-your-UI>",
  bucket_name="<your-bucket-name>"
)

# While using AWS object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<aws-access-key-id-from-your-UI>",
  secret_key="<aws-secret-access-key-from-your-UI>",
  bucket_name="<your-bucket-name>"
)

ai_pipeline_obj = Pipeline(api_auth=api_auth, dataset=dataset_obj)

ai_pipeline_obj.get_all(state="ready", print_stdout=True)
# returns a list of dict wherein each dict contains an AI-Pipeline's info. Prints the formatted list of AI-APIs on stdout. Raises an
```

Validate an AI-Pipeline by image_name and state

- Returns `True` if given `image_name` with `state` exists in the environment else `False`.
- Raises
  - `AuthenticationError` if user not authenticated.
  - `AuthorizationError` if user is not permitted to view the list of AI-APIs.
- If `image_name` is
  - `empty` or `None` then raises `ValueError`.
  - Not of type `str` then raises `TypeError`.
- If `state` is
  - `empty` or `None` or `not one of ['created', 'ready']` then raises `ValueError`.
  - Not of type `str` then raises `TypeError`.
- Raises `ServerError` for any other errors.

Validate if AI-Pipeline with `image-name` and `valid-state` exists in the environment

```
 from zeblok.dataset import DataSet
from zeblok.auth import APIAuth
from zeblok.pipeline import Pipeline

api_auth = APIAuth(
    app_url='<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

# While using Azure object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<azure-account-name-from-your-UI>",
  secret_key="<azure-account-key-from-your-UI>",
  bucket_name="<your-bucket-name>"
)

# While using AWS object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<aws-access-key-id-from-your-UI>",
  secret_key="<aws-secret-access-key-from-your-UI>",
  bucket_name="<your-bucket-name>"
)

ai_pipeline_obj = Pipeline(api_auth=api_auth, dataset=dataset_obj)

ai_pipeline_obj.validate(image_name = "<image-name>", state="<valid-state>")
# returns True if `AI-Pipeline` exists else False. Raises error as specified in the description.
```

Create and Spawn an AI-Pipeline

- Creation followed automatic spawning of the AI-Pipeline.
- User provides the following parameters:
    - `ai_pipeline_name` : A meaningful name to identify the AI-Pipeline created and spawned in the Ai-MicroCloud environment.
    - `ai_pipeline_folder_path` : Local folder path to be spawned as an AI-Pipeline.
    - `caas_plan_id` : Id for the containerization-as-a-service plan.
    - `ai_pipeline_plan_id` : Id of the resource plan.
    - `namespace_id` : Id of the namespace in which you wish to spawn the AI-Pipeline.
- Raises
    - `AuthenticationError` if user not authenticated.
    - `AuthorizationError` if user is not permitted to view the list of AI-APIs.
- If `ai_pipeline_name` is
    - `empty` or `None` then raises `ValueError`.
    - Not of type `str` then raises `TypeError`.
- If `ai_pipeline_folder_path`
    - Is `empty` then raises `ValueError`.
    - Is `Invalid` or `doesn't exist` then raises `DirectoryNotFoundError`.
    - Is `Not-a-directory` then raises `NotADirectoryError`.
    - Does not have `write-permission` then raises `PermissionError`
- If `caas_plan_id` is
    - `empty` or `None` or `Not-of-Type-CAAS` then raises `ValueError`.
    - Not of type `str` then raises `TypeError`.
    - Not found in the environment then raises `ResourceNotFoundError`.
- If `ai_pipeline_plan_id` is
    - `empty` or `None` or `Not-of-Type-MicroService` then raises `ValueError`.
    - Not of type `str` then raises `TypeError`.
    - Not found in the environment then raises `ResourceNotFoundError`.
- If `namespace_id` is
    - `empty` or `None` then raises `ValueError`.
    - Not of type `str` then raises `TypeError`.
    - Not found in the environment then raises `ResourceNotFoundError`.
- Raises `ServerError` for any other errors.

create and spawn an AI-Pipeline

```
 from zeblok.dataset import DataSet
from zeblok.auth import APIAuth
from zeblok.pipeline import Pipeline

api_auth = APIAuth(
    app_url='<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

# While using Azure object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<azure-account-name-from-your-UI>",
  secret_key="<azure-account-key-from-your-UI>",
  bucket_name="<your-bucket-name>"
)

# While using AWS object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<aws-access-key-id-from-your-UI>",
  secret_key="<aws-secret-access-key-from-your-UI>",
  bucket_name="<your-bucket-name>"
)

ai_pipeline_obj = Pipeline(api_auth=api_auth, dataset=dataset_obj)

ai_pipeline_obj.create_and_spawn(
    ai_pipeline_name="<valid-ai-pipeline-name>",
    ai_pipeline_folder_path='<valid-pipeline-folder-path>',
    caas_plan_id='<valid-caas-plan-id>',
    ai_pipeline_plan_id='<valid-pipeline-plan-id>',
    namespace_id='<valid-namespace-id>'
) # returns True with a user-friendly message if the AI-Pipeline is submitted successfully else raises an Exception.
```

Create an AI-Pipeline

- Creates an AI-Pipeline but doesn't spawn the AI-Pipeline automatically.
- User provides the following parameters:
  - `ai_pipeline_name` : A meaningful name to identify the AI-Pipeline created and spawned in the Ai-MicroCloud environment.
  - `ai_pipeline_folder_path` : Local folder path to be spawned as an AI-Pipeline.
  - `caas_plan_id` : Id for the containerization-as-a-service plan.
  - `ai_pipeline_plan_id` : Id of the resource plan.
  - `namespace_id` : Id of the namespace in which you wish to spawn the AI-Pipeline.
- Raises
  - `AuthenticationError` if user not authenticated.
  - `AuthorizationError` if user is not permitted to view the list of AI-APIs.
- If `ai_pipeline_name` is
  - `empty` or `None` then raises `ValueError`.
  - Not of type `str` then raises `TypeError`.
- If `ai_pipeline_folder_path`
  - Is `empty` then raises `ValueError`.
  - Is `Invalid` or `doesn't exist` then raises `DirectoryNotFoundError`.
  - Is `Not-a-directory` then raises `NotADirectoryError`.
  - Does not have `write-permission` then raises `PermissionError`
- If `caas_plan_id` is
  - `empty` or `None` or `Not-of-Type-CAAS` then raises `ValueError`.
  - Not of type `str` then raises `TypeError`.
  - Not found in the environment then raises `ResourceNotFoundError`.
- If `ai_pipeline_plan_id` is
  - `empty` or `None` or `Not-of-Type-MicroService` then raises `ValueError`.
  - Not of type `str` then raises `TypeError`.
  - Not found in the environment then raises `ResourceNotFoundError`.
- If `namespace_id` is
  - `empty` or `None` then raises `ValueError`.
  - Not of type `str` then raises `TypeError`.
  - Not found in the environment then raises `ResourceNotFoundError`.
- Raises `ServerError` for any other errors.

create an AI-Pipeline

```
 from zeblok.dataset import DataSet
from zeblok.auth import APIAuth
from zeblok.pipeline import Pipeline

api_auth = APIAuth(
    app_url='<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

# While using Azure object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<azure-account-name-from-your-UI>",
  secret_key="<azure-account-key-from-your-UI>",
  bucket_name="<your-bucket-name>"
)

# While using AWS object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<aws-access-key-id-from-your-UI>",
  secret_key="<aws-secret-access-key-from-your-UI>",
  bucket_name="<your-bucket-name>"
)

ai_pipeline_obj = Pipeline(api_auth=api_auth, dataset=dataset_obj)

ai_pipeline_obj.create(
    ai_pipeline_name="<valid-ai-pipeline-name>",
    ai_pipeline_folder_path='<valid-pipeline-folder-path>',
    caas_plan_id='<valid-caas-plan-id>',
    ai_pipeline_plan_id='<valid-pipeline-plan-id>',
    namespace_id='<valid-namespace-id>'
) # returns image_name and prints image_name if the AI-Pipeline creation request is submitted successfully else raises an Exception.
```

Spawn an AI-Pipeline

- Spawns an already created AI-Pipeline in the Ai-MicroCloud environment. ai_pipeline_plan_id: str, namespace_id: str, ai_pipeline_image_name: str
- User provides the following parameters:
  - `ai_pipeline_image_name` : AI-Pipeline image name returned by the `create` function.
  - `ai_pipeline_plan_id` : Id of the resource plan.
  - `namespace_id` : Id of the namespace in which you wish to spawn the AI-Pipeline.
- Raises
  - `AuthenticationError` if user not authenticated.
  - `AuthorizationError` if user is not permitted to view the list of AI-APIs.
- If `ai_pipeline_image_name`
  - `empty` or `None` or `Not Found` then raises `ValueError` .
  - Not of type `str` then raises `TypeError` .
- If `ai_pipeline_plan_id` is
  - `empty` or `None` or `Not-of-Type-MicroService` then raises `ValueError` .
  - Not of type `str` then raises `TypeError` .
  - Not found in the environment then raises `ResourceNotFoundError` .
- If `namespace_id` is
  - `empty` or `None` then raises `ValueError` .
  - Not of type `str` then raises `TypeError` .
  - Not found in the environment then raises `ResourceNotFoundError` .
- Raises `ServerError` for any other errors.

spawn an AI-Pipeline

```
 from zeblok.dataset import DataSet
from zeblok.auth import APIAuth
from zeblok.pipeline import Pipeline

api_auth = APIAuth(
    app_url='<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

# While using Azure object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<azure-account-name-from-your-UI>",
  secret_key="<azure-account-key-from-your-UI>",
  bucket_name="<your-bucket-name>"
)

# While using AWS object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<aws-access-key-id-from-your-UI>",
  secret_key="<aws-secret-access-key-from-your-UI>",
  bucket_name="<your-bucket-name>"
)

ai_pipeline_obj = Pipeline(api_auth=api_auth, dataset=dataset_obj)

ai_pipeline_obj.spawn(
    ai_pipeline_image_name='<valid-image-name>',
    ai_pipeline_plan_id='<valid-plan-id>',
    namespace_id='<valid-namespace-id>',
) # returns True with a user-friendly message if the AI-Pipeline spawning request is submitted successfully else raises an Exception
```

DataSet

Using the SDK, we can

- Check if a bucket exists in the object storage: `bucket_exists()`
- Check if an object exists in the object storage: `object_exists(object_name)`
- Upload an object in the object storage: `upload_object(local_file_pathname, object_name)`
- Download an object from the object storage: `download_object(object_name, local_dir, filename)`
- Get a PreSigned URL for an object: `get_presigned_url(object_name)`

Check if a bucket exists

- Returns `True` if the given bucket exists in the object storage in the respective environment else `False`.

Validate if bucket exists in the object storage in the respective environment

```
from zeblok.dataset import DataSet
from zeblok.auth import APIAuth

api_auth = APIAuth(
    app_url=f'<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

# While using Azure object store
dataset_obj = DataSet(
    api_auth=api_auth,
    access_key="<account-name>",
    secret_key="<account-key>",
    bucket_name="<your-bucket-name>"
)


# While using AWS object store
dataset_obj = DataSet(
    api_auth=api_auth,
    access_key="<aws-access-key-id>",
    secret_key="<aws-secret-access-key>",
    bucket_name="<your-bucket-name>"
)

dataset_obj.bucket_exists() # returns True if `<your-bucket-name>` exists else False.
```

Check if an object exists

- Returns `True` if the given `object_name` exists in the bucket in the respective object storage else `False`.

Validate if the `object-name` exists in the bucket in the respective object storage

```
from zeblok.dataset import DataSet
from zeblok.auth import APIAuth

api_auth = APIAuth(
    app_url=f'<your-base-url>',
    api_access_key='<your-api-access-key-from-Web-UI>',
    api_access_secret='<your-api-access-secret-from-Web-UI>',
)

# While using Azure object store
dataset_obj = DataSet(
    api_auth=api_auth,
    access_key="<account-name>",
    secret_key="<account-key>",
    bucket_name="<your-bucket-name>"
)


# While using AWS object store
dataset_obj = DataSet(
    api_auth=api_auth,
    access_key="<aws-access-key-id>",
    secret_key="<aws-secret-access-key>",
    bucket_name="<your-bucket-name>"
)

dataset_obj.object_exists(object_name='<your-object-name>')
# returns True if `<your-object-name>` exists else False.
```

Upload an object to an object store

- Upload a local file in the respective bucket.
- User provides the following parameters:
  - `local_file_pathname` : Filepath to your local file.
  - `object_name` : Meaningful name for your object name. If not provided then it will be same as local file name.
- Raises
  - `BucketDoesNotExistsError` if the bucket does not exist.
  - `FileNotFoundError` if the local_file_pathname is not found.
  - `PermissionError` if local file does not have **read** permission.
  - `FileUploadError` if there is an error uploading the local file to the respective bucket.
- If `object_name` is
  - not None and `empty` then raises `ValueError` .
  - not None and not of type `str` then raises `TypeError` .

Upload a file (filepath: `<local-filepath-name>`) to the bucket with object name: `<object-name>`

```
from zeblok.dataset import DataSet
from zeblok.auth import APIAuth

api_auth = APIAuth(
  app_url=f'<your-base-url>',
  api_access_key='<your-api-access-key-from-Web-UI>',
  api_access_secret='<your-api-access-secret-from-Web-UI>',
)

# While using Azure object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<account-name>",
  secret_key="<account-key>",
  bucket_name="<your-bucket-name>"
)

# While using AWS object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<aws-access-key-id>",
  secret_key="<aws-secret-access-key>",
  bucket_name="<your-bucket-name>"
)

dataset_obj.upload_object(local_file_pathname="<local-filepath-name>", object_name="<object-name>")
# returns True if file uploaded successfully else False. Raises an exception if an error occurs while uploading the file.
```

Download an object from the object store

- Download an object from the respective bucket in your object store to your local system.
- User provides the following parameters:
  - `local_file_pathname` : Filepath to your local file.
  - `object_name` : The object name in the bucket to be downloaded.
  - `local_dir` : Directory path in your local system where you want to download the object.
  - `filename` : Meaningful filename for your object to be downloaded.
- Raises
  - `BucketDoesNotExistsError` if the bucket does not exist.
  - `NotADirectoryError` if local_dir is not a directory.
  - `PermissionError` if local_dir does not have permission to **write** a file.
- If `object_name` is
  - `empty` or `None` then raises `ValueError` .
  - Not of type `str` then raises `TypeError` .
  - Does not exist then `ObjectDoesNotExistsError`
- If `filename` is
  - `empty` or `None` then raises `ValueError` .
  - Not of type `str` then raises `TypeError` .

Download an object (object_name: `<object-name>`) from the bucket to the local system with local directory (local_dir: `<local-dir>`) and filename (filename: `filename`)

```
 from zeblok.dataset import DataSet
from zeblok.auth import APIAuth

api_auth = APIAuth(
  app_url=f'<your-base-url>',
  api_access_key='<your-api-access-key-from-Web-UI>',
  api_access_secret='<your-api-access-secret-from-Web-UI>',
)

# While using Azure object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<account-name>",
  secret_key="<account-key>",
  bucket_name="<your-bucket-name>"
)

# While using AWS object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<aws-access-key-id>",
  secret_key="<aws-secret-access-key>",
  bucket_name="<your-bucket-name>"
)

dataset_obj.download_object(object_name="<object-name>", local_dir="<local-dir>", filename="filename")
# returns True if file downloaded successfully else False. Raises an exception if an error occurs while downloading the file.
```

Generate a PreSigned url for the respective object

- Generate a pre-signed url with 1 hour expiry for an object in the respective bucket in your object store.
- User provides the following parameters:
    - `object_name` : The object name in the bucket for which you wish to create the pre-signed url.
- If `object_name` is
    - `empty` or `None` then raises `ValueError`.
    - Not of type `str` then raises `TypeError`.
    - Does not exist then `ObjectDoesNotExistsError`

Generate a pre-signed url of an object (object_name: `<object-name>`)

```
from zeblok.dataset import DataSet
from zeblok.auth import APIAuth

api_auth = APIAuth(
  app_url=f'<your-base-url>',
  api_access_key='<your-api-access-key-from-Web-UI>',
  api_access_secret='<your-api-access-secret-from-Web-UI>',
)

# While using Azure object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<account-name>",
  secret_key="<account-key>",
  bucket_name="<your-bucket-name>"
)

# While using AWS object store
dataset_obj = DataSet(
  api_auth=api_auth,
  access_key="<aws-access-key-id>",
  secret_key="<aws-secret-access-key>",
  bucket_name="<your-bucket-name>"
)

dataset_obj.get_presigned_url(object_name="<object-name>")
# returns the pre-signed url as a string. Raises an exception if an error occurs while generating the pre-signed url of the object.
```