Pykube

Release 20.7.2

Jul 24, 2020

Contents

Pykube (pykube-ng) is a lightweight Python 3+ client library for Kubernetes.

CHAPTER 1

Installation

To install pykube, use pip:

```
pip install pykube-ng
```

CHAPTER 2

Usage

Query for all ready pods in a custom namespace:

```
import operator
import pykube

api = pykube.HTTPClient(pykube.KubeConfig.from_file())
pods = pykube.Pod.objects(api).filter(namespace="gondor-system")
ready_pods = filter(operator.attrgetter("ready"), pods)
```

CHAPTER 3

Documentation

3.1 How Tos

3.1.1 How to access a Service

To access the service defined with your `Service` object, you can use the HTTP proxy provided by the API server. There are convenience methods to do `GET`, `POST`, `PUT` and `DELETE` and the generic `proxy_http_request` version, where you can pass any HTTP verb.

```
service = pykube.Service.objects(api).filter(namespace="default").get(name="test")
res = service.proxy_http_get("my/path")
assert res is not None
assert res.content == b"it works"
assert res.status_code == 200
```

3.1.2 How to find Pods by label

This section explains how to find Kubernetes pods by a set of known labels.

Calling *`objects`* on the *`Pod`* class returns a *`Query`* object which provides the *`filter`* method. The `selector` parameter can take a dictionary of label names and values to filter by:

```
for pod in Pod.objects(api).filter(namespace=pykube.all, selector={'app': 'myapp'}):
    print(pod.namespace, pod.name)
```

Note that the special value of `pykube.all` needs to be passed, otherwise it would only return pods in the current namespace (i.e. usually "default").

3.1.3 How to print container logs

To print the most recent logs for multiple pods and containers, you can use the `timestamps` parameter and sort log lines afterwards. Note that this will not work correctly if your container logs contain newlines!

```
tail_lines = 100

logs = []

for pod in pods:
    for container in pod.obj["spec"]["containers"]:
        container_log = pod.logs(
            container=container["name"],
            timestamps=True,
            tail_lines=tail_lines,
        )
        for line in container_log.split("\n"):
            logs.append(line)

logs.sort()

for log in logs:
    print(log)
```

3.1.4 How to update a Deployment image

To update the Docker image for an existing deployment:

```
from pykube import Deployment, HTTPClient, KubeConfig

new_docker_image = "hjacobs/kube-web-view"

api = HTTPClient(KubeConfig.from_file())
deploy = Deployment.objects(api).get(name="mydeploy")
deploy.obj["spec"]["template"]["spec"]["containers"][0]["image"] = new_docker_image
deploy.update()
```

Note that the call to `deploy.update()` might fail if the resource was modified between loading and updating. In this case you need to retry.

3.1.5 How to use the Interactive Console

Pykube can be started as an interactive Python console:

```
python3 -m pykube
```

The interactive console automatically loads the Kubernetes configuration from the default location (`~/.kube/config`) and provides the objects `api` and `config`:

```
>>> api
<pykube.http.HTTPClient object at 0x7f2112263160>

>>> config
<pykube.config.KubeConfig object at 0x7f6631bbc2e8>
```

All standard classes from pykube-package are automatically imported, so you can use them, e.g.:

```
>>> for deploy in Deployment.objects(api):
...     print(f'{deploy.name}: {deploy.replicas}')
```

You can also pass a Python command via the `-c` option for non-interactive usage:

```
python3 -m pykube -c 'print(config.current_context, api.version)'
```

3.1.6 How to write an Operator

Pykube can be used to implement Kubernetes Operators. Here is how to write a very simple operator which adds a label `foo` with value `bar` to every deployment object which has the `pykube-test-operator` annotation:

```
# simplified example script, no error handling!
import pykube, time

while True:
    # loads in-cluster auth or local ~/.kube/config for testing
    config = pykube.KubeConfig.from_env()
    api = pykube.HTTPClient(config)
    for deploy in pykube.Deployment.objects(api, namespace=pykube.all):
        if 'pykube-test-operator' in deploy.annotations:
            print(f'Updating deployment {deploy.namespace}/{deploy.name}..')
            deploy.labels['foo'] = 'bar'
            deploy.update()
    time.sleep(15)
```

Save the above Python script as `main.py`.

Testing

You can now test the script locally with Pipenv and Minikube (run `minikube start` first):

```
pipenv install pykube-ng
pipenv run python3 main.py
```

See the operator in action by creating a deployment with the right annotation:

```
kubectl run nginx --image=nginx
kubectl annotate deploy nginx pykube-test-operator=true
```

The operator should should now assign the `foo` label to the `nginx` deployment.

Building the Docker image

Create a `Dockerfile` in the same directory as `main.py`:

```
FROM python:3.7-alpine3.10

WORKDIR /

RUN pip3 install pykube-ng
```

(continues on next page)

(continued from previous page)

```
COPY main.py /

ENTRYPOINT ["python3", "main.py"]
```

Now build it:

```
docker build -t pykube-test-operator .
```

You need to push the Docker image to some Docker registry before you can deploy it.

Deployment

Now deploy the Docker image to your Kubernetes cluster using a service account with the necessary permissions (in this case to list and update deployments). To create such an service account with the necessary RBAC rights create `rbac.yaml` with these contents:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: pykube-test-operator
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: pykube-test-operator
rules:
- apiGroups:
  - apps
  resources:
  - deployments
  verbs:
  - get
  - watch
  - list
  - update
  - patch
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: pykube-test-operator
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: pykube-test-operator
subjects:
- kind: ServiceAccount
  name: pykube-test-operator
  namespace: default
```

Apply the RBAC role via `kubectl apply -f rbac.yaml`.

Finally, the deployment of the operator would then look like (`deployment.yaml`):

```
apiVersion: apps/v1
kind: Deployment
```

(continues on next page)

(continued from previous page)

```
metadata:
  name: pykube-test-operator
spec:
  replicas: 1
  selector:
    matchLabels:
      app: pykube-test-operator
  template:
    metadata:
      labels:
        app: pykube-test-operator
    spec:
      serviceAccountName: pykube-test-operator
      containers:
      - name: operator
        # this image needs have been pushed to some Docker registry!
        image: pykube-test-operator
        resources:
          limits:
            memory: 50Mi
          requests:
            cpu: 5m
            memory: 50Mi
        securityContext:
          readOnlyRootFilesystem: true
          runAsNonRoot: true
          runAsUser: 1000
```

Create the deployment via `kubectl apply -f deployment.yaml`.

You should now have a working operator deployment in your cluster.

3.2 API Modules

Pykube exports its main classes on the package level, so you can do:

```
from pykube import KubeConfig, HTTPClient, Pod
```

- *KubeConfig* is the main configuration class to load `~/.kube/config` or from in-cluster service account
- *HTTPClient* represents the Kubernetes API client
- Kubernetes resource kinds (*Pod*, etc) are defined in *pykube.objects*

3.2.1 pykube package

Submodules

pykube.config module

Configuration code.

class pykube.config.**BytesOrFile**(*filename=None*, *data=None*, *kubeconfig_path=None*)

Bases: object

Implements the same interface for files and byte input.

bytes()
Returns the provided data as bytes.

filename()
Returns the provided data as a file location.

classmethod maybe_set(*d*, *key*, *kubeconfig_path*)

class pykube.config.**KubeConfig**(*doc*, *current_context=None*)
Bases: object

Main configuration class.

cluster
Returns the current selected cluster by exposing as a read-only property.

clusters
Returns known clusters by exposing as a read-only property.

contexts
Returns known contexts by exposing as a read-only property.

current_context

filepath = None

classmethod from_env()
Convenience function to create an instance of KubeConfig from the current environment.

First tries to use in-cluster ServiceAccount, then tries default ~/.kube/config (or KUBECONFIG)

classmethod from_file(*filename=None*, ***kwargs*)
Creates an instance of the KubeConfig class from a kubeconfig file.

Parameters **filename** – The full path to the configuration file. Defaults to ~/.kube/config

classmethod from_service_account(*path='/var/run/secrets/kubernetes.io/serviceaccount'*, ***kwargs*)
Construct KubeConfig from in-cluster service account.

classmethod from_url(*url*, ***kwargs*)
Creates an instance of the KubeConfig class from a single URL (useful for interacting with kubectl proxy).

kubeconfig_file
Returns the path to kubeconfig file as string, if it exists

kubeconfig_path
Returns the path to kubeconfig file, if it exists

namespace
Returns the current context namespace by exposing as a read-only property.

persist_doc()

reload()

set_current_context(*value*)
Sets the context to the provided value.

Parameters

- *value*: The value for the current context

user
Returns the current user set by current context

users
Returns known users by exposing as a read-only property.

pykube.console module

pykube.console.**main**(*argv=None*)
Run the interactive Pykube console (usually invoked via python3 -m pykube)

pykube.exceptions module

Exceptions.

exception pykube.exceptions.**HTTPError**(*code*, *message*)
Bases: *pykube.exceptions.PyKubeError*

exception pykube.exceptions.**KubernetesError**
Bases: Exception

Base exception for all Kubernetes errors.

exception pykube.exceptions.**ObjectDoesNotExist**
Bases: *pykube.exceptions.PyKubeError*

exception pykube.exceptions.**PyKubeError**
Bases: *pykube.exceptions.KubernetesError*

PyKube specific errors.

pykube.http module

HTTP request related code.

class pykube.http.**HTTPClient**(*config: pykube.config.KubeConfig*, *timeout: float = 10*, *dry_run: bool = False*, *verify: bool = True*, *http_adapter: Optional[requests.adapters.HTTPAdapter] = None*)
Bases: object

Client for interfacing with the Kubernetes API.

delete(**args*, ***kwargs*)
Executes an HTTP DELETE.

Parameters

- *args*: Non-keyword arguments
- *kwargs*: Keyword arguments

get(**args*, ***kwargs*)
Executes an HTTP GET.

Parameters

- *args*: Non-keyword arguments
- *kwargs*: Keyword arguments

get_kwargs(***kwargs*) → dict
Creates a full URL to request based on arguments.

Parametes

- *kwargs*: All keyword arguments to build a kubernetes API endpoint

head(**args*, ***kwargs*)
Executes an HTTP HEAD.

Parameters

- *args*: Non-keyword arguments
- *kwargs*: Keyword arguments

http_adapter_cls
alias of *KubernetesHTTPAdapter*

options(**args*, ***kwargs*)
Executes an HTTP OPTIONS.

Parameters

- *args*: Non-keyword arguments
- *kwargs*: Keyword arguments

patch(**args*, ***kwargs*)
Executes an HTTP PATCH.

Parameters

- *args*: Non-keyword arguments
- *kwargs*: Keyword arguments

post(**args*, ***kwargs*)
Executes an HTTP POST.

Parameters

- *args*: Non-keyword arguments
- *kwargs*: Keyword arguments

put(**args*, ***kwargs*)
Executes an HTTP PUT.

Parameters

- *args*: Non-keyword arguments
- *kwargs*: Keyword arguments

raise_for_status(*resp*)

request(**args*, ***kwargs*)
Makes an API request based on arguments.

Parameters

- *args*: Non-keyword arguments
- *kwargs*: Keyword arguments

resource_list(*api_version*)

url

version
Get Kubernetes API version

class pykube.http.**KubernetesHTTPAdapter**(*kube_config: pykube.config.KubeConfig*, ***kwargs*)

Bases: requests.adapters.HTTPAdapter

send(*request*, ***kwargs*)

Sends PreparedRequest object. Returns Response object.

Parameters

- **request** – The PreparedRequest being sent.
- **stream** – (optional) Whether to stream the request content.
- **timeout** (*float or tuple or urllib3 Timeout object*) – (optional) How long to wait for the server to send data before giving up, as a float, or a (connect timeout, read timeout) tuple.
- **verify** – (optional) Either a boolean, in which case it controls whether we verify the server's TLS certificate, or a string, in which case it must be a path to a CA bundle to use
- **cert** – (optional) Any user-provided SSL certificate to be trusted.
- **proxies** – (optional) The proxies dictionary to apply to the request.

Return type requests.Response

pykube.mixins module

class pykube.mixins.**ReplicatedMixin**

Bases: object

replicas

scalable_attr = 'replicas'

class pykube.mixins.**ScalableMixin**

Bases: object

scalable

scale(*replicas=None*)

pykube.objects module

class pykube.objects.**APIObject**(*api: pykube.http.HTTPClient*, *obj: dict*)

Bases: object

Baseclass for all Kubernetes API objects

annotations

Annotations of the Kubernetes resource (metadata.annotations)

Annotations is an unstructured key value map stored with a resource that may be set by external tools to store and retrieve arbitrary metadata. They are not queryable and should be preserved when modifying objects. More info: http://kubernetes.io/docs/user-guide/annotations

api_kwargs(***kwargs*)

base = None

create()

delete(*propagation_policy: str = None*)
Delete the Kubernetes resource by calling the API.

The parameter propagation_policy defines whether to cascade the delete. It can be "Foreground", "Background" or "Orphan". See https://kubernetes.io/docs/concepts/workloads/controllers/garbage-collection/#setting-the-cascading-deletion-policy

exists(*ensure=False*)

labels
Labels of the Kubernetes resource (metadata.labels)

Map of string keys and values that can be used to organize and categorize (scope and select) objects. May match selectors of replication controllers and services. More info: http://kubernetes.io/docs/user-guide/labels

metadata

name
Name of the Kubernetes resource (metadata.name)

Name must be unique within a namespace. Is required when creating resources, although some resources may allow a client to request the generation of an appropriate name automatically. Name is primarily intended for creation idempotence and configuration definition. Cannot be updated. More info: http://kubernetes.io/docs/user-guide/identifiers#names

namespace

objects

patch(*strategic_merge_patch*, *, *subresource=None*)
Patch the Kubernetes resource by calling the API with a "strategic merge" patch.

reload()

set_obj(*obj: dict*)

update(*is_strategic=True*, *, *subresource=None*)
Update the Kubernetes resource by calling the API (patch)

watch()

class pykube.objects.**ClusterRole**(*api: pykube.http.HTTPClient*, *obj: dict*)
Bases: *pykube.objects.APIObject*

endpoint = 'clusterroles'

kind = 'ClusterRole'

version = 'rbac.authorization.k8s.io/v1'

class pykube.objects.**ClusterRoleBinding**(*api: pykube.http.HTTPClient*, *obj: dict*)
Bases: *pykube.objects.APIObject*

endpoint = 'clusterrolebindings'

kind = 'ClusterRoleBinding'

version = 'rbac.authorization.k8s.io/v1'

class pykube.objects.**ConfigMap**(*api: pykube.http.HTTPClient*, *obj: dict*)
Bases: *pykube.objects.NamespacedAPIObject*

endpoint = 'configmaps'

kind = 'ConfigMap'

version = 'v1'

class pykube.objects.**CronJob**(*api: pykube.http.HTTPClient*, *obj: dict*)
Bases: *pykube.objects.NamespacedAPIObject*

endpoint = 'cronjobs'

kind = 'CronJob'

version = 'batch/v1beta1'

class pykube.objects.**CustomResourceDefinition**(*api: pykube.http.HTTPClient*, *obj: dict*)
Bases: *pykube.objects.APIObject*

endpoint = 'customresourcedefinitions'

kind = 'CustomResourceDefinition'

version = 'apiextensions.k8s.io/v1'

class pykube.objects.**DaemonSet**(*api: pykube.http.HTTPClient*, *obj: dict*)
Bases: *pykube.objects.NamespacedAPIObject*

endpoint = 'daemonsets'

kind = 'DaemonSet'

version = 'apps/v1'

class pykube.objects.**Deployment**(*api: pykube.http.HTTPClient*, *obj: dict*)
Bases: *pykube.objects.NamespacedAPIObject*, *pykube.mixins.ReplicatedMixin*, *pykube.mixins.ScalableMixin*

endpoint = 'deployments'

kind = 'Deployment'

ready

rollout_undo(*target_revision=None*)
Produces same action as kubectl rollout undo deployment command. Input variable is revision to rollback to (in kubectl, –to-revision)

version = 'apps/v1'

class pykube.objects.**Endpoint**(*api: pykube.http.HTTPClient*, *obj: dict*)
Bases: *pykube.objects.NamespacedAPIObject*

endpoint = 'endpoints'

kind = 'Endpoint'

version = 'v1'

class pykube.objects.**Event**(*api: pykube.http.HTTPClient*, *obj: dict*)
Bases: *pykube.objects.NamespacedAPIObject*

endpoint = 'events'

kind = 'Event'

version = 'v1'

class pykube.objects.**HorizontalPodAutoscaler**(*api: pykube.http.HTTPClient*, *obj: dict*)
Bases: *pykube.objects.NamespacedAPIObject*

endpoint = 'horizontalpodautoscalers'

kind = 'HorizontalPodAutoscaler'

version = 'autoscaling/v1'

class pykube.objects.**Ingress**(*api: pykube.http.HTTPClient*, *obj: dict*)
Bases: *pykube.objects.NamespacedAPIObject*

endpoint = 'ingresses'

kind = 'Ingress'

version = 'networking.k8s.io/v1beta1'

class pykube.objects.**Job**(*api: pykube.http.HTTPClient*, *obj: dict*)
Bases: *pykube.objects.NamespacedAPIObject*, *pykube.mixins.ScalableMixin*

endpoint = 'jobs'

kind = 'Job'

parallelism

scalable_attr = 'parallelism'

version = 'batch/v1'

class pykube.objects.**LimitRange**(*api: pykube.http.HTTPClient*, *obj: dict*)
Bases: *pykube.objects.NamespacedAPIObject*

endpoint = 'limitranges'

kind = 'LimitRange'

version = 'v1'

class pykube.objects.**Namespace**(*api: pykube.http.HTTPClient*, *obj: dict*)
Bases: *pykube.objects.APIObject*

endpoint = 'namespaces'

kind = 'Namespace'

version = 'v1'

class pykube.objects.**NamespacedAPIObject**(*api: pykube.http.HTTPClient*, *obj: dict*)
Bases: *pykube.objects.APIObject*

namespace
Namespace scope of the Kubernetes resource (metadata.namespace)

Namespace defines the space within each name must be unique. Cannot be updated. More info: http://kubernetes.io/docs/user-guide/namespaces

class pykube.objects.**Node**(*api: pykube.http.HTTPClient*, *obj: dict*)
Bases: *pykube.objects.APIObject*

cordon()

endpoint = 'nodes'

kind = 'Node'

uncordon()

unschedulable

version = 'v1'

class pykube.objects.**ObjectManager**
Bases: object

class pykube.objects.**PersistentVolume**(*api: pykube.http.HTTPClient*, *obj: dict*)
Bases: *pykube.objects.APIObject*

endpoint = 'persistentvolumes'

kind = 'PersistentVolume'

version = 'v1'

class pykube.objects.**PersistentVolumeClaim**(*api: pykube.http.HTTPClient*, *obj: dict*)
Bases: *pykube.objects.NamespacedAPIObject*

endpoint = 'persistentvolumeclaims'

kind = 'PersistentVolumeClaim'

version = 'v1'

class pykube.objects.**Pod**(*api: pykube.http.HTTPClient*, *obj: dict*)
Bases: *pykube.objects.NamespacedAPIObject*

endpoint = 'pods'

kind = 'Pod'

logs(*container=None*, *pretty=None*, *previous=False*, *since_seconds=None*, *since_time=None*, *timestamps=False*, *tail_lines=None*, *limit_bytes=None*)
Produces the same result as calling kubectl logs pod/<pod-name>. Check parameters meaning at http://kubernetes.io/docs/api-reference/v1/operations/, part 'read log of the specified Pod'. The result is plain text.

ready

version = 'v1'

class pykube.objects.**PodDisruptionBudget**(*api: pykube.http.HTTPClient*, *obj: dict*)
Bases: *pykube.objects.NamespacedAPIObject*

endpoint = 'poddisruptionbudgets'

kind = 'PodDisruptionBudget'

version = 'policy/v1beta1'

class pykube.objects.**PodSecurityPolicy**(*api: pykube.http.HTTPClient*, *obj: dict*)
Bases: *pykube.objects.APIObject*

endpoint = 'podsecuritypolicies'

kind = 'PodSecurityPolicy'

version = 'policy/v1beta1'

class pykube.objects.**ReplicaSet**(*api: pykube.http.HTTPClient*, *obj: dict*)
Bases: *pykube.objects.NamespacedAPIObject*, *pykube.mixins.ReplicatedMixin*, *pykube.mixins.ScalableMixin*

endpoint = 'replicasets'

kind = 'ReplicaSet'

version = 'apps/v1'

class pykube.objects.**ReplicationController**(*api: pykube.http.HTTPClient*, *obj: dict*)
Bases: *pykube.objects.NamespacedAPIObject*, *pykube.mixins.ReplicatedMixin*, *pykube.mixins.ScalableMixin*

endpoint = 'replicationcontrollers'

kind = 'ReplicationController'

ready

version = 'v1'

class pykube.objects.**ResourceQuota**(*api: pykube.http.HTTPClient*, *obj: dict*)
Bases: *pykube.objects.NamespacedAPIObject*

endpoint = 'resourcequotas'

kind = 'ResourceQuota'

version = 'v1'

class pykube.objects.**Role**(*api: pykube.http.HTTPClient*, *obj: dict*)
Bases: *pykube.objects.NamespacedAPIObject*

endpoint = 'roles'

kind = 'Role'

version = 'rbac.authorization.k8s.io/v1'

class pykube.objects.**RoleBinding**(*api: pykube.http.HTTPClient*, *obj: dict*)
Bases: *pykube.objects.NamespacedAPIObject*

endpoint = 'rolebindings'

kind = 'RoleBinding'

version = 'rbac.authorization.k8s.io/v1'

class pykube.objects.**Secret**(*api: pykube.http.HTTPClient*, *obj: dict*)
Bases: *pykube.objects.NamespacedAPIObject*

endpoint = 'secrets'

kind = 'Secret'

version = 'v1'

class pykube.objects.**Service**(*api: pykube.http.HTTPClient*, *obj: dict*)
Bases: *pykube.objects.NamespacedAPIObject*

endpoint = 'services'

kind = 'Service'

proxy_http_delete(*path: str*, *port: Optional[int] = None*, ***kwargs*) → requests.models.Response
Issue a HTTP DELETE request to proxy of a Service. Args: :rtype: `Response` :type path: `str`

param path The URI path for the request.

param port This value can be used to override the

default (first defined) port used to connect to the Service. :param kwargs: Keyword arguments for the proxy_http_get function. They are the same as for requests.models.Request object plus the additional 'port' kwarg, which can be used to override the default (first defined) port used to connect to the Service.

Returns: The requests.Response object.

proxy_http_get (*path: str*, *port: Optional[int] = None*, ***kwargs*) → requests.models.Response
Issue a HTTP GET request to proxy of a Service. Args: :rtype: `Response` :type path: `str`

param path The URI path for the request.

param port This value can be used to override the

default (first defined) port used to connect to the Service. :param kwargs: Keyword arguments for the proxy_http_get function. They are the same as for requests.models.Request object plus the additional 'port' kwarg, which can be used to override the default (first defined) port used to connect to the Service.

Returns: The requests.Response object.

proxy_http_post (*path: str*, *port: Optional[int] = None*, ***kwargs*) → requests.models.Response
Issue a HTTP POST request to proxy of a Service. Args: :rtype: `Response` :type path: `str`

param path The URI path for the request.

param port This value can be used to override the

default (first defined) port used to connect to the Service. :param kwargs: Keyword arguments for the proxy_http_get function. They are the same as for requests.models.Request object plus the additional 'port' kwarg, which can be used to override the default (first defined) port used to connect to the Service.

Returns: The requests.Response object.

proxy_http_put (*path: str*, *port: Optional[int] = None*, ***kwargs*) → requests.models.Response
Issue a HTTP PUT request to proxy of a Service. Args: :rtype: `Response` :type path: `str`

param path The URI path for the request.

param port This value can be used to override the

default (first defined) port used to connect to the Service. :param kwargs: Keyword arguments for the proxy_http_get function. They are the same as for requests.models.Request object plus the additional 'port' kwarg, which can be used to override the default (first defined) port used to connect to the Service.

Returns: The requests.Response object.

proxy_http_request (*method: str*, *path: str*, *port: Optional[int] = None*, ***kwargs*) → requests.models.Response
Issue a HTTP request with specific HTTP method to proxy of a Service. Args: :rtype: `Response` :type path: `str` :type method: `str`

param method The http request method e.g. 'GET', 'POST' etc.

param path The URI path for the request.

param port This value can be used to override the

default (first defined) port used to connect to the Service. :param kwargs: Keyword arguments for the proxy_http_get function. They are the same as for requests.models.Request object.

Returns: The requests.Response object.

version = 'v1'

class pykube.objects.**ServiceAccount**(*api: pykube.http.HTTPClient*, *obj: dict*)
Bases: *pykube.objects.NamespacedAPIObject*

endpoint = 'serviceaccounts'

kind = 'ServiceAccount'

version = 'v1'

class pykube.objects.**StatefulSet**(*api: pykube.http.HTTPClient*, *obj: dict*)
Bases: *pykube.objects.NamespacedAPIObject*, *pykube.mixins.ReplicatedMixin*, *pykube.mixins.ScalableMixin*

endpoint = 'statefulsets'

kind = 'StatefulSet'

version = 'apps/v1'

pykube.objects.**object_factory**(*api*, *api_version*, *kind*) → Type[pykube.objects.APIObject]
Dynamically builds a Python class for the given Kubernetes object in an API.

For example:

> api = pykube.HTTPClient(...) NetworkPolicy = pykube.object_factory(api, "networking.k8s.io/v1", "NetworkPolicy")

This enables construction of any Kubernetes object kind without explicit support from pykube.

Currently, the HTTPClient passed to this function will not be bound to the returned type. It is planned to fix this, but in the mean time pass it as you would normally.

pykube.query module

class pykube.query.**BaseQuery**(*api: pykube.http.HTTPClient*, *api_obj_class*, *namespace: str = None*)
Bases: object

all() → pykube.query.BaseQuery

filter(*namespace: str = None*, *selector: Union[str, dict] = None*, *field_selector: Union[str, dict] = None*) → pykube.query.BaseQuery
Filter objects by namespace, labels, or fields

Parameters

- **namespace** (str) – Namespace to filter by (pass pykube.all to get objects in all namespaces)
- **selector** – Label selector, can be a dictionary of label names/values

class pykube.query.**Query**(*api: pykube.http.HTTPClient*, *api_obj_class*, *namespace: str = None*)
Bases: *pykube.query.BaseQuery*

as_table() → pykube.query.Table
Execute query and return result as Table (similar to what kubectl does) See https://kubernetes.io/docs/reference/using-api/api-concepts/#receiving-resources-as-tables

execute(***kwargs*)

get(**args*, ***kwargs*)
Get a single object by name, namespace, label, ..

get_by_name(*name: str*)
Get object by name, raises ObjectDoesNotExist if not found

get_or_none(**args*, ***kwargs*)
Get object by name, return None if not found

iterator()
Execute the API request and return an iterator over the objects. This method does not use the query cache.

query_cache

response

watch(*since=None*, ***, *params=None*)

class pykube.query.**Table**(*api_obj_class*, *obj: dict*)
Bases: object

Tabular resource representation See https://kubernetes.io/docs/reference/using-api/api-concepts/#receiving-resources-as-tables

columns

rows

class pykube.query.**WatchQuery**(**args*, ***kwargs*)
Bases: *pykube.query.BaseQuery*

object_stream()

response

pykube.query.**as_selector**(*value: Union[str, dict]*) → str

pykube.utils module

pykube.utils.**join_url_path**(**components*, *join_empty: bool = False*) → str
Join given URL path components and return absolute path starting with '/'.

pykube.utils.**jsonpath_parse**(*template*, *obj*)

pykube.utils.**obj_check**(*obj_value*, *original_obj_value*, *is_strategic=True*)

pykube.utils.**obj_merge**(*obj*, *original_obj*, *is_strategic=True*)

Module contents

Python client for Kubernetes

3.3 pykube-ng

Pykube (pykube-ng) is a lightweight Python 3.6+ client library for Kubernetes.

This is a fork of kelproject/pykube which is no longer maintained (archived). Here the original text of the pykube README:

> Kel is an open source Platform as a Service (PaaS) from Eldarion, Inc. that makes it easy to manage web application deployment and hosting through the entire lifecycle from development through testing to production. It adds components and tools on top of Kubernetes that help developers manage their application infrastructure. Kel builds on Eldarion's 7+ years experience running one of the leading Python and Django PaaSes. For more information about Kel, see kelproject.com or follow us on Twitter @projectkel.

3.3.1 Features

- HTTP interface using requests using kubeconfig for authentication
- Python native querying of Kubernetes API objects

3.3.2 Installation

To install pykube, use pip:

```
pip install pykube-ng
```

3.3.3 Interactive Console

The `pykube` library module can be run as an interactive console locally for quick exploration. It will automatically load `~/.kube/config` to provide the `api` object, and it loads pykube classes (`Deployment`, `Pod`, ..) into local context:

```
python3 -m pykube
>>> [d.name for d in Deployment.objects(api)]
```

3.3.4 Usage

Query for all ready pods in a custom namespace:

```
import operator
import pykube

api = pykube.HTTPClient(pykube.KubeConfig.from_file())
pods = pykube.Pod.objects(api).filter(namespace="gondor-system")
ready_pods = filter(operator.attrgetter("ready"), pods)
```

Access any attribute of the Kubernetes object:

```
pod = pykube.Pod.objects(api).filter(namespace="gondor-system").get(name="my-pod")
pod.obj["spec"]["containers"][0]["image"]
```

Selector query:

```
pods = pykube.Pod.objects(api).filter(
    namespace="gondor-system",
    selector={"gondor.io/name__in": {"api-web", "api-worker"}},
)
pending_pods = pykube.objects.Pod.objects(api).filter(
    field_selector={"status.phase": "Pending"}
)
```

Watch query:

```
watch = pykube.Job.objects(api, namespace="gondor-system")
watch = watch.filter(field_selector={"metadata.name": "my-job"}).watch()

# watch is a generator:
for watch_event in watch:
    print(watch_event.type) # 'ADDED', 'DELETED', 'MODIFIED'
    print(watch_event.object) # pykube.Job object
```

Create a Deployment:

```
obj = {
    "apiVersion": "apps/v1",
    "kind": "Deployment",
    "metadata": {
        "name": "my-deploy",
        "namespace": "gondor-system"
    },
    "spec": {
        "replicas": 3,
        "selector": {
            "matchLabels": {
                "app": "nginx"
            }
        },
        "template": {
            "metadata": {
                "labels": {
                    "app": "nginx"
                }
            },
            "spec": {
                "containers": [
                    {
                        "name": "nginx",
                        "image": "nginx",
                        "ports": [
                            {"containerPort": 80}
                        ]
                    }
                ]
            }
        }
    }
}
pykube.Deployment(api, obj).create()
```

Delete a Deployment:

```
obj = {
    "apiVersion": "apps/v1",
    "kind": "Deployment",
    "metadata": {
        "name": "my-deploy",
        "namespace": "gondor-system"
    }
}
pykube.Deployment(api, obj).delete()
```

Check server version:

```
api = pykube.HTTPClient(pykube.KubeConfig.from_file())
api.version
```

3.3.5 Requirements

- Python 3.6+
- requests (included in `install_requires`)
- PyYAML (included in `install_requires`)

3.3.6 Local Development

You can run pykube against your current kubeconfig context, e.g. local Minikube:

```
poetry install
poetry run python3
>>> import pykube
>>> config = pykube.KubeConfig.from_file()
>>> api = pykube.HTTPClient(config)
>>> list(pykube.Deployment.objects(api))
```

To run PEP8 (flake8) checks and unit tests including coverage report:

```
make test
```

3.3.7 License

The code in this project is licensed under the Apache License, version 2.0 (included in this repository under LICENSE).

3.3.8 Contributing

Easiest way to contribute is to provide feedback! We would love to hear what you like and what you think is missing. Create an issue or ping try_except_ on Twitter.

PRs are welcome. Please also have a look at issues labeled with "help wanted".

3.3.9 Code of Conduct

In order to foster a kind, inclusive, and harassment-free community, this project follows the Contributor Covenant Code of Conduct.

3.4 Changelog

3.4.1 19.10.0

- use both `token` and `client-certificate` from Kubeconfig if defined (to support Azure Kubernetes)

3.4.2 19.9.2

- add `dry_run` option to `HTTPClient` to only make dry run requests

3.4.3 19.9.1

- support "oidc" auth type in KubeConfig (basic support with existing "id-token")

3.4.4 19.9.0

- changed to Calendar Versioning
- add convenience function `KubeConfig.from_env()` to load KubeConfig from in-cluster ServiceAccount or local KUBECONFIG

3.4.5 0.30

- allow passing a custom `Authorization header` with `HTTPClient.get(..., headers=..)` without getting overwritten

3.4.6 0.28

- support tabular representation like what kubectl uses (`Query.as_table()` method)

3.4.7 0.27

- allow passing arbitrary parameters to the `watch` query

3.4.8 0.26

- remember streaming response
- exclude "tests" from packaging

3.4.9 0.25

- add CustomResourceDefinition to top-level imports (allowing from pykube import CustomResourceDefinition)

3.4.10 0.24

- add `CustomResourceDefinition` class
- add `patch` method to APIObject class
- improve user-friendliness of `object_factory`

3.4.11 0.23

- remove debug print statement

3.4.12 0.22

- fix GCP support

3.4.13 0.21

- add optional `propagation_policy` parameter to APIObject.delete(), see https://pykube.readthedocs.io/en/latest/api/pykube.html#pykube.objects.APIObject.delete

3.4.14 0.20

- Fix handling of annotations and labels if the object had none set before

3.4.15 0.19

- Added interactive console (invoke with `python3 -m pykube`)

3.4.16 0.18

- Added `PodDisruptionBudget`
- Added HTTP timeout (default: 10 seconds)

3.4.17 0.17

- New release as `pykube-ng`
- Removed Python 2.7 compatibility
- Removed HTTPie plugin
- Added some tests

3.5 Users

The following projects use Pykube:

- kube-downscaler
- kube-janitor
- kube-ops-view
- kube-resource-report
- kube-web-view
- pytest-kind

Do you know of a project using Pykube and it's not listed here? Please open a PR to add it to the list.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

O

P

Q

R

S

T

U

V

W