
Kubernetes-job Documentation

Release 0.3.3

R. Claasen

Mar 19, 2021

CONTENTS

1	Project details	3
1.1	Kubernetes-job: simple Kubernetes job creation	3
1.2	Kubernetes details	5
1.3	API reference	8
1.4	Examples	10
	Python Module Index	15
	Index	17

Kubernetes-job is a Python library for starting Kubernetes batch jobs, in the simplest way possible:

```
from kubernetes_job import JobManager

manager = JobManager(kubernetes_client, 'job.yaml')
job = manager.create_job(my_job_function, arg_1, arg_2)
```


PROJECT DETAILS

Project homepage: <https://gitlab.com/roemer/kubernetes-job>

Documentation: <https://kubernetes-job.readthedocs.io>

Pypi: <https://pypi.org/project/kubernetes-job>

Version: 0.3.3

1.1 Kubernetes-job: simple Kubernetes job creation

A library for starting a Kubernetes batch job as a normal Python function call.

For source code and tickets, see our [project page](#) on [Gitlab](#). The documentation is hosted on [ReadTheDocs](#). Kubernetes-job can be found on [Pypi](#) for easy installation with `pip`.

1.1.1 Installation

Installation with Pip:

```
pip install kubernetes-job
```

1.1.2 Quick start

```
from kubernetes_job import JobManager

def add(a, b):
    return a + b

manager = JobManager(k8s_client=k8s_client, k8s_job_spec='job.yaml', namespace=
↳ 'default')
job = manager.create_job(add, 1, 2)
```

The `JobManager` will now create a Kubernetes job using the basic job specification in the `job.yaml` file. The call to `add` is then passed on to the new job node, where the function is subsequently executed.

The `job.yaml` file should be adjusted to your needs. This is the place to put Kubernetes node selectors, Docker base images, etc. etc. Please refer to the [Kubernetes documentation](#) for details.

Please note: this is a very silly example, for two obvious reasons.

First, *add* will take a very short time to complete, and is therefore not a function you would want to spawn a Kubernetes job for. A job should be created for a task that is not easily performed on the calling machine. A good example would be training Machine Learning models on a heavy CUDA node, started from a web server node with modest resources.

Second, *Kubernetes jobs do not return values!* This means the result of this addition will be lost. In a Kubernetes job, it is up to the job to save its work. In this case, the result of $(1 + 2)$ will be lost for humanity.

Please see the *examples* and the `test/` directory.

1.1.3 API usage

Initializing the JobManager

The JobManager must be supplied a `yaml` template file (see above) and the Kubernetes client.

```
from pathlib import Path
from kubernetes_job import JobManager

# Path to worker configuration
yaml_spec = Path(__file__).parent / 'job.yaml'

# initialize the job manager
manager = JobManager(k8s_client=k8s_client, k8s_job_spec=yaml_spec, namespace='default
↪')
```

The `k8s_job_spec` may be a path to a file, or a dict instance. The latter is handy for generating configuration on the fly!

JobManager also needs a Kubernetes client. More information about *how to connect to Kubernetes* can be found [here](#).

Creating a new job

A job can be started by invoking `create_job` on the JobManager instance:

```
# function to pass to the job
def add(a, b):
    result = a + b
    print(result)
    return result

# create a new job
job = manager.create_job(add, 123, 456)
```

`create_job` takes a *function pointer*. This function pointer and all arguments (`*args` and `**kwargs`) are then “pickled”, and merged in the *job template*.

Our job is now running on the Kubernetes cluster!

Listing jobs

```
# list all jobs
for job in manager.list_jobs():
    print(f"Found: {job.metadata.name}")
```

Retrieving job status

```
from kubernetes_job import is_active, is_succeeded, is_failed, is_completed, job_
    ↪status

# get the status of a job
job = manager.read_job(name)

print(f"Status: {job_status(job)}")
print(f"Running: {is_active(job)} Completed: {is_completed(job)}")
print(f"Succeeded: {is_succeeded(job)} Failed: {is_failed(job)}")
```

Cleaning up finished jobs

```
# cleaning up finished jobs
manager.cleanup_jobs()
```

Deleting jobs

```
# delete a job
manager.delete_job(name)
```

1.2 Kubernetes details

1.2.1 Connecting to Kubernetes

There is more than one way to connect to a Kubernetes cluster.

During development, you will likely be best off using an existing `kubectl` configuration. In a production setting, you might prefer using a service account and token-based authentication.

Using `kubectl` configuration

During development, when working from a local development workstation, the easiest way to connect to a cluster is *probably* to use existing `kubectl` configuration:

```
from kubernetes import client, config

# This will initialize the client from an existing Kubectl config file in $HOME/.kube/
    ↪config
config.load_kube_config()

k8s_client = client.ApiClient()
```

Using a service account and token-based authentication

In a production setting, when the `kubernetes_job.JobManager` is run on the Kubernetes cluster, it is *probably* best to use a **Kubernetes service account** and a **bearer token**. This can be done as follows:

```
from kubernetes import client

configuration = client.Configuration()
configuration.api_key["authorization"] = '<token>'
configuration.api_key_prefix['authorization'] = 'Bearer'
configuration.host = 'https://<endpoint_of_api_server>'
configuration.ssl_ca_cert = '<path_to_cluster_ca_certificate>'

k8s_client = client.ApiClient(configuration)
```

How the correct settings for token, `endpoint_of_api_server`, and the cluster CA certificates can be retrieved is explained in the *section below*.

Please refer to [Python Kubernetes documentation](#) for more details.

1.2.2 The Kubernetes job spec template (e.g. `job.yaml`)

When Kubernetes-job spawns a new job, the Kubernetes job spec template is used as the base configuration for the new job.

This is an example:

```
apiVersion: batch/v1
kind: Job
metadata:
  # job name; a unique id will be added when launching a new job based on this_
  ↪template
  name: kubernetes-job
spec:
  # Try 1 time to execute this job
  backoffLimit: 1

  # Active deadline (timeout), in a number of seconds.
  activeDeadlineSeconds: 3600

  # Clean up pods and logs after finishing the job
  ttlSecondsAfterFinished: 3600

  template:
    spec:
      containers:
        - name: kubernetes-job
          image: registry.gitlab.com/roemer/kubernetes-job:latest
          restartPolicy: Never
```

Please adjust this template to your needs by specifying the right container image, job deadlines, etc. The [Kubernetes documentation](#) contains more information.

When Kubernetes-job spawns a new job, three things are added to the template:

1. A unique name, generated by adding a timestamp;
2. The function call, serialized (using Pickle), added as an environment variable;

3. A `cmd` entry calling `JobManager.execute`.

A working example can be found in the *test/directory*.

Make sure the Docker image in the job template contains the same packaged Python software as the process creating the job! Otherwise the function cannot be executed in the new job pod.

1.2.3 Setting up token-based authentication

Create a service account

First, create a service account:

```
# Create a service account
kubectl create -f service_account.yml --k8s_namespace=default
```

An example of `service_account.yml` can be found *here*

Kubernetes generates a unique name for the new service account. We need to retrieve that unique name, and to do that, we need to ask Kubernetes for its secrets:

```
# retrieve secret
kubectl get secrets --k8s_namespace=default | grep kubernetes-job-service-account
```

This returns something like this:

```
kubernetes-job-service-account-token-XXXXX   kubernetes.io/service-account-token   3   ↵
↪      66s
```

kubernetes-job-service-account-token-XXXXX is the name generated by Kubernetes.

Retrieving the access token

Now we are able to retrieve the access token for this service account:

```
kubectl describe secret/kubernetes-job-service-account-token-XXXXX | grep token
```

This returns something like:

```
token:          <token>
```

This token is the one we're looking for.

Cluster endpoint and cluster CA certificates

To connect to the cluster we also need the **cluster endpoint** and the **CA certificates**. Both can easily be retrieved through the Kubernetes dashboard, through the “cluster details” page.

1.3 API reference

1.3.1 JobManager

```
class kubernetes_job.JobManager (k8s_client:          kubernetes.client.api_client.ApiClient,  
                                k8s_job_spec: [ <class 'dict'>, <class 'str'> ], namespace:  
                                str = 'default')
```

Kubernetes JobManager

Parameters

- **k8s_client** – Kubernetes OpenAPI client
- **k8s_job_spec** – *dict* or path to YAML file containing the spec for the job worker
- **namespace** – Kubernetes k8s_namespace (default: 'default')

```
clean_jobs (field_selector=None, label_selector=None)
```

Clean up completed jobs

Parameters

- **field_selector** – A selector to restrict the list of returned objects by their fields. Defaults to everything.
- **label_selector** – A selector to restrict the list of returned objects by their labels. Defaults to everything.

```
create_job (func, *func_args, **func_kwargs) → kubernetes.client.models.v1_job.V1Job
```

Create a job

Parameters

- **func** – Function pointer
- **func_args** – Args to submit to the function
- **func_kwargs** – Kwargs to submit to the function

Returns V1Job

```
delete_job (job: [ <class 'str'>, <class 'kubernetes.client.models.v1_job.V1Job'> ],  
            grace_period_seconds: int = 0, propagation_policy: str = 'Background') → ku-  
bernetes.client.models.v1_status.V1Status
```

Delete a Job

Parameters

- **job** – Name or V1Job instance
- **grace_period_seconds** – (default: 0)
- **propagation_policy** – (default: 'Background')

Returns V1Status

```
static execute_job (job_func_def: Optional[str] = None)
```

Execute the JobFuncDef specified in the func_spec

Parameters **job_func_def** – Serialized job definition

Returns Job function return value (if any)

```
list_jobs (field_selector=None, label_selector=None) → Itera-  
tor[kubernetes.client.models.v1_job.V1Job]
```

List job objects

Parameters

- **field_selector** – A selector to restrict the list of returned objects by their fields. Defaults to everything.
- **label_selector** – A selector to restrict the list of returned objects by their labels. Defaults to everything.

Returns Iterator of V1Job

read_job (*job*: [*<class 'str'>*, *<class 'kubernetes.client.models.v1_job.V1Job'>*]) → *kubernetes.client.models.v1_job.V1Job*
 Read the status of the specified Job

Parameters *job* – Name or V1Job instance**Returns** V1Job

1.3.2 Helpers

kubernetes_job.job_name (*job*: [*<class 'str'>*, *<class 'kubernetes.client.models.v1_job.V1Job'>*]) → *str*
 Return the name of a job

kubernetes_job.job_status (*job*: *kubernetes.client.models.v1_job.V1Job*) → *str*
 Return SUCCEEDED, FAILED, ACTIVE, or PENDING, depending on the status of the job

kubernetes_job.is_completed (*job*: *kubernetes.client.models.v1_job.V1Job*)
 Return True if the job has completed (either failed or succeeded)

kubernetes_job.is_succeeded (*job*: *kubernetes.client.models.v1_job.V1Job*)
 Return True if the job has succeeded

kubernetes_job.is_failed (*job*: *kubernetes.client.models.v1_job.V1Job*)
 Return True if the job is failed

kubernetes_job.is_active (*job*: *kubernetes.client.models.v1_job.V1Job*)
 Return True if the job is active (running)

kubernetes_job.current_job = *kubernetes_job.job_func_def.JobFuncDef*
 Current *JobFuncDef* when executing a Kubernetes-job (as runner), otherwise *None*.

1.3.3 job_func_def

job_func_def contains helper classes for the serialization and execution of the function call.

class *kubernetes_job.job_func_def.JobFuncDef* (*func*, *args=None*, *kwargs=None*, *meta: Optional[kubernetes_job.job_func_def.JobMeta] = None*)

Helper class to hold the job function definition

Parameters

- **func** – Pointer to the job function
- **args** – Args for the job function
- **kwargs** – Kwargs for the job function
- **meta** – Metadata for the job

args = None

Args for the job function

dump() → str

Dump the job function definition to a base64 string

execute()

Execute the job function

func = None

Pointer to the job function

kwargs = None

Kwargs for the job function

static load(s: str) → *kubernetes_job.job_func_def.JobFuncDef*

Load the job function definition from a base64 string

meta: *kubernetes_job.job_func_def.JobMeta* = None

Metadata for the job

class *kubernetes_job.job_func_def.JobMeta*

Helper class to hold job meta information

dt_scheduled: <module 'datetime' from '/home/docs/.pyenv/versions/3.7.9/lib/python3.7

Job scheduled datetime

host: str = '[HOST]'

Host responsible for spawning the job

name: str = '[JOB-NAME]'

Unique job name

1.4 Examples

1.4.1 demo.py

A small demo application illustrating the use of the Kubernetes-job API.

```
1 import logging
2 import os
3 import tempfile
4 import time
5 import yaml
6 import sys
7 from pathlib import Path
8 from kubernetes import client, config
9
10 from kubernetes_job import JobManager, is_completed
11 from funcs import add, calc_pi
12
13 logger = logging.getLogger()
14 logging.basicConfig(level=logging.DEBUG, stream=sys.stdout)
15
16 # retrieve cluster details from environment variables
17 host_url = os.environ.get("HOST_URL")
18 cacert = os.environ.get("CACERT")
19 token = os.environ.get("TOKEN")
```

(continues on next page)

(continued from previous page)

```

20
21 configuration = None
22
23 if host_url:
24     # initialize configuration for token authentication
25     # this is the way to go if we're using a service account
26     configuration = client.Configuration()
27     configuration.api_key["authorization"] = token
28     configuration.api_key_prefix['authorization'] = 'Bearer'
29     configuration.host = host_url
30
31     # configuration.ssl_ca_cert expects a file containing the certificates,
32     # so we generate a temporary file to hold those
33     with tempfile.NamedTemporaryFile(mode='w', delete=False) as tf:
34         tf.write(cacert)
35         configuration.ssl_ca_cert = tf.name
36
37 else:
38     # try to initialize from $HOME/.kube/config (eg. kubectl config file)
39     config.load_kube_config()
40
41 # initialize the Kubernetes client
42 k8s_client = client.ApiClient(configuration=configuration)
43
44 # Path to worker configuration
45 yaml_path = Path(__file__).parent / 'job.yml'
46
47 # we're loading the yaml file here;
48 # we could also supply the path when initializing the JobManager
49 with Path(yaml_path).open() as f:
50     yaml_spec = yaml.safe_load(f)
51
52 # initialize the job manager
53 manager = JobManager(k8s_client=k8s_client, k8s_job_spec=yaml_spec)
54
55 # create a new job
56 new_job = manager.create_job(calc_pi, 100, 1)
57 logging.info(f"Created job {new_job.metadata.name}")
58
59 # list all jobs
60 for job in manager.list_jobs():
61     logging.info(f"Found: {job.metadata.name}")
62
63 # get the status of a job
64 job_status = manager.read_job(new_job)
65 while not is_completed(job_status):
66     logging.info(f"Status: {job_status.status}")
67     job_status = manager.read_job(new_job)
68     time.sleep(5)
69
70 # clean up jobs
71 manager.clean_jobs()
72
73 # delete a job
74 # manager.delete_job(new_job)
75

```

1.4.2 job.yaml

An example of a job spec template.

```

1  apiVersion: batch/v1
2  kind: Job
3  metadata:
4    # job name; a unique id will be added when launching a new job based on this
   ↪template
5    name: kubernetes-job
6  spec:
7    # The service account can be important if the security is restricted.
8    # It may not be necessary if the JobManager is initialized with elevated
   ↪credentials.
9    serviceAccountName: kubernetes-job-service-account
10
11   # Try 1 time to execute this job
12   backoffLimit: 1
13
14   # Active deadline (timeout), in a number of seconds.
15   activeDeadlineSeconds: 3600
16
17   # Clean up pods and logs after finishing the job
18   # N.B. This Kuberetes feature is still in alpha, GKE does not support it yet!
19   ttlSecondsAfterFinished: 3600
20
21   template:
22     spec:
23       containers:
24         - name: kubernetes-job
25
26           # Change this image to an image containing your codebase
27           image: registry.gitlab.com/roemer/kubernetes-job:latest
28
29           # A startup command is automatically added by Kubernetes-job, so no need to
   ↪set it here.
30           # We may set one, though, to override startup behaviour.
31           # Be sure to initialize the job runner then by spawning a `kubernetes-job`
   ↪process.
32           # cmd: kubernetes-job
33
34       restartPolicy: Never

```

1.4.3 service_account.yaml

An example of the Kubernetes configuration needed to create a service account for job management.

```

1  ---
2  apiVersion: v1
3  kind: ServiceAccount
4  metadata:
5    name: kubernetes-job-service-account
6  ---
7  kind: Role
8  apiVersion: rbac.authorization.k8s.io/v1
9  metadata:

```

(continues on next page)

(continued from previous page)

```
10  name: kubernetes-job-service-role
11  rules:
12  - apiGroups:
13    - ""
14    - "batch"
15    resources:
16    - jobs
17    verbs:
18    - get
19    - list
20    - watch
21    - create
22    - delete
23  ---
24  kind: RoleBinding
25  apiVersion: rbac.authorization.k8s.io/v1
26  metadata:
27    name: kubernetes-job-service-account
28    namespace: default
29  subjects:
30  - kind: ServiceAccount
31    name: kubernetes-job-service-account
32    namespace: default
33  roleRef:
34    apiGroup: rbac.authorization.k8s.io
35    kind: Role
36    name: kubernetes-job-service-role
```

To execute, run the following command:

```
kubectl apply -f service_account.yml
```

- [genindex](#)

PYTHON MODULE INDEX

k

kubernetes_job.job_func_def, 9

A

args (*kubernetes_job.job_func_def.JobFuncDef* attribute), 9

C

clean_jobs() (*kubernetes_job.JobManager* method), 8

create_job() (*kubernetes_job.JobManager* method), 8

current_job (*in module kubernetes_job*), 9

D

delete_job() (*kubernetes_job.JobManager* method), 8

dt_scheduled (*kubernetes_job.job_func_def.JobMeta* attribute), 10

dump() (*kubernetes_job.job_func_def.JobFuncDef* method), 10

E

execute() (*kubernetes_job.job_func_def.JobFuncDef* method), 10

execute_job() (*kubernetes_job.JobManager* static method), 8

F

func (*kubernetes_job.job_func_def.JobFuncDef* attribute), 10

H

host (*kubernetes_job.job_func_def.JobMeta* attribute), 10

I

is_active() (*in module kubernetes_job*), 9

is_completed() (*in module kubernetes_job*), 9

is_failed() (*in module kubernetes_job*), 9

is_succeeded() (*in module kubernetes_job*), 9

J

job_name() (*in module kubernetes_job*), 9

job_status() (*in module kubernetes_job*), 9

JobFuncDef (*class in kubernetes_job.job_func_def*), 9

JobManager (*class in kubernetes_job*), 8

JobMeta (*class in kubernetes_job.job_func_def*), 10

K

kubernetes_job.job_func_def
module, 9

kwargs (*kubernetes_job.job_func_def.JobFuncDef* attribute), 10

L

list_jobs() (*kubernetes_job.JobManager* method), 8

load() (*kubernetes_job.job_func_def.JobFuncDef* static method), 10

M

meta (*kubernetes_job.job_func_def.JobFuncDef* attribute), 10

module

kubernetes_job.job_func_def, 9

N

name (*kubernetes_job.job_func_def.JobMeta* attribute), 10

R

read_job() (*kubernetes_job.JobManager* method), 9