Kubespawner Documentation

Release 0.10.0.dev0

Project Jupyter

Sep 20, 2018

Contents

CHAPTER 1

Overview

The *kubespawner* (also known as JupyterHub Kubernetes Spawner) enables JupyterHub to spawn single-user notebook servers on a Kubernetes cluster.

1.1 Features

Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications. If you want to run a JupyterHub setup that needs to scale across multiple nodes (anything with over ~50 simultaneous users), Kubernetes is a wonderful way to do it. Features include:

- Easily and elastically run anywhere between 2 and thousands of nodes with the same set of powerful abstractions. Scale up and down as required by simply adding or removing nodes.
- Run JupyterHub itself inside Kubernetes easily. This allows you to manage many JupyterHub deployments with only Kubernetes, without requiring an extra layer of Ansible / Puppet / Bash scripts. This also provides easy integrated monitoring and failover for the hub process itself.
- Spawn multiple hubs in the same kubernetes cluster, with support for namespaces. You can limit the amount of resources each namespace can use, effectively limiting the amount of resources a single JupyterHub (and its users) can use. This allows organizations to easily maintain multiple JupyterHubs with just one kubernetes cluster, allowing for easy maintenance & high resource utilization.
- Provide guarantees and limits on the amount of resources (CPU / RAM) that single-user notebooks can use. Kubernetes has comprehensive resource control that can be used from the spawner.
- Mount various types of persistent volumes onto the single-user notebook's container.
- Control various security parameters (such as userid/groupid, SELinux, etc) via flexible Pod Security Policies.
- Run easily in multiple clouds (or on your own machines). Helps avoid vendor lock-in. You can even spread out your cluster across multiple clouds at the same time.

In general, Kubernetes provides a ton of well thought out, useful features - and you can use all of them along with this spawner.

1.2 Requirements

1.2.1 Kubernetes

Everything should work from Kubernetes v1.6+.

The Kube DNS addon is not strictly required - the spawner uses environment variable based discovery instead. Your kubernetes cluster will need to be configured to support the types of volumes you want to use.

If you are just getting started and want a kubernetes cluster to play with, Google Container Engine is probably the nicest option. For AWS/Azure, kops is probably the way to go.

CHAPTER 2

Changes in KubeSpawner

2.1 KubeSpawner 0.9

KubeSpawner 0.9 is a big release of KubeSpawner.

Change highlights:

- Require Kubernetes >= 1.6
- Require JupyterHub >= 0.8
- Require Python >= 3.5
- Expose lots more Kubernetes options
- Support configuration profiles via `KubeSpawner.profile_list`
- Support Kubernetes events for the progress API in JupyterHub 0.9.
- Update Kubernetes Python client to 6.0 (supporting Kubernetes 1.10 APIs)
- Numerous bugfixes

CHAPTER 3

KubeSpawner

JupyterHub Spawner to spawn user notebooks on a Kubernetes cluster.

After installation, you can enable it by adding:

```
c.JupyterHub.spawner_class = 'kubespawner.KubeSpawner'
```

in your `jupyterhub_config.py` file.

class `kubespawner.`**KubeSpawner**(**args*, ***kwargs*)

Implement a JupyterHub spawner to spawn pods in a Kubernetes Cluster.

config c.KubeSpawner.args = List()

Extra arguments to be passed to the single-user server.

Some spawners allow shell-style expansion here, allowing you to use environment variables here. Most, including the default, do not. Consult the documentation for your spawner to verify!

config c.KubeSpawner.cmd = Command()

The command used for starting the single-user server.

Provide either a string or a list containing the path to the startup script command. Extra arguments, other than this path, should be provided via `args`.

This is usually set if you want to start the single-user server in a different python environment (with virtualenv/conda) than JupyterHub itself.

Some spawners allow shell-style expansion here, allowing you to use environment variables. Most, including the default, do not. Consult the documentation for your spawner to verify!

If set to `None`, Kubernetes will start the `CMD` that is specified in the Docker image being started.

config c.KubeSpawner.common_labels = Dict()

Kubernetes labels that both spawned singleuser server pods and created user PVCs will get.

Note that these are only set when the Pods and PVCs are created, not later when this setting is updated.

config c.KubeSpawner.consecutive_failure_limit = Int(0)

Maximum number of consecutive failures to allow before shutting down JupyterHub.

This helps JupyterHub recover from a certain class of problem preventing launch in contexts where the Hub is automatically restarted (e.g. systemd, docker, kubernetes).

A limit of 0 means no limit and consecutive failures will not be tracked.

config c.KubeSpawner.cpu_guarantee = Float(None)
Minimum number of cpu-cores a single-user notebook server is guaranteed to have available.

If this value is set to 0.5, allows use of 50% of one CPU. If this value is set to 2, allows use of up to 2 CPUs.

This is a configuration setting. Your spawner must implement support for the limit to work. The default spawner, `LocalProcessSpawner`, does **not** implement this support. A custom spawner **must** add support for this setting for it to be enforced.

config c.KubeSpawner.cpu_limit = Float(None)
Maximum number of cpu-cores a single-user notebook server is allowed to use.

If this value is set to 0.5, allows use of 50% of one CPU. If this value is set to 2, allows use of up to 2 CPUs.

The single-user notebook server will never be scheduled by the kernel to use more cpu-cores than this. There is no guarantee that it can access this many cpu-cores.

This is a configuration setting. Your spawner must implement support for the limit to work. The default spawner, `LocalProcessSpawner`, does **not** implement this support. A custom spawner **must** add support for this setting for it to be enforced.

config c.KubeSpawner.debug = Bool(False)
Enable debug-logging of the single-user server

config c.KubeSpawner.default_url = Unicode('')
The URL the single-user server should start in.

`{username}` will be expanded to the user's username

Example uses:

- You can set `notebook_dir` to `/` and `default_url` to `/tree/home/{username}` to allow people to navigate the whole filesystem from their notebook server, but still start in their home directory.
- Start with `/notebooks` instead of `/tree` if `default_url` points to a notebook instead of a directory.
- You can set this to `/lab` to have JupyterLab start by default, rather than Jupyter Notebook.

config c.KubeSpawner.delete_stopped_pods = Bool(True)
Whether to delete pods that have stopped themselves. Set to False to leave stopped pods in the completed state, allowing for easier debugging of why they may have stopped.

config c.KubeSpawner.disable_user_config = Bool(False)
Disable per-user configuration of single-user servers.

When starting the user's single-user server, any config file found in the user's $HOME directory will be ignored.

Note: a user could circumvent this if the user modifies their Python environment, such as when they have their own conda environments / virtualenvs / containers.

config c.KubeSpawner.env_keep = List()
Whitelist of environment variables for the single-user server to inherit from the JupyterHub process.

This whitelist is used to ensure that sensitive information in the JupyterHub process's environment (such as `CONFIGPROXY_AUTH_TOKEN`) is not passed to the single-user server's process.

config c.KubeSpawner.environment = Dict()
Extra environment variables to set for the single-user server's process.

Environment variables that end up in the single-user server's process come from 3 sources:

- This `environment` configurable
- The JupyterHub process' environment variables that are whitelisted in `env_keep`
- Variables to establish contact between the single-user notebook and the hub (such as JUPYTERHUB_API_TOKEN)

The `environment` configurable should be set by JupyterHub administrators to add installation specific environment variables. It is a dict where the key is the name of the environment variable, and the value can be a string or a callable. If it is a callable, it will be called with one parameter (the spawner instance), and should return a string fairly quickly (no blocking operations please!).

Note that the spawner class' interface is not guaranteed to be exactly same across upgrades, so if you are using the callable take care to verify it continues to work after upgrades!

config c.KubeSpawner.events_enabled = Bool(True)
Enable event-watching for progress-reports to the user spawn page.

Disable if these events are not desirable or to save some performance cost.

config c.KubeSpawner.extra_annotations = Dict()
Extra Kubernetes annotations to set on the spawned single-user pods.

The keys and values specified here are added as annotations on the spawned single-user kubernetes pods. The keys and values must both be strings.

See the Kubernetes documentation for more info on what annotations are and why you might want to use them!

`{username}` and `{userid}` are expanded to the escaped, dns-label safe username & integer user id respectively, wherever they are used.

config c.KubeSpawner.extra_container_config = Dict()
Extra configuration (e.g. `envFrom`) for notebook container which is not covered by other attributes.

This dict will be directly merge into `container` of notebook server, so you should use the same structure. Each item in the dict must a field of the V1Container specification.

One usage is set `envFrom` on notebook container with configuration below:

```
c.KubeSpawner.extra_container_config = {
    "envFrom": [{
        "configMapRef": {
            "name": "special-config"
        }
    }]
}
```

The key could be either a camelCase word (used by Kubernetes yaml, e.g. `envFrom`) or a snake_case word (used by Kubernetes Python client, e.g. `env_from`).

config c.KubeSpawner.extra_containers = List()
List of containers belonging to the pod which besides to the container generated for notebook server.

This list will be directly appended under `containers` in the kubernetes pod spec, so you should use the same structure. Each item in the list is container configuration which follows spec at https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.11/#container-v1-core.

One usage is setting crontab in a container to clean sensitive data with configuration below:

```
c.KubeSpawner.extra_containers = [{
    "name": "crontab",
    "image": "supercronic",
    "command": ["/usr/local/bin/supercronic", "/etc/crontab"]
}]
```

config c.KubeSpawner.extra_labels = Dict()
Extra kubernetes labels to set on the spawned single-user pods.

The keys and values specified here would be set as labels on the spawned single-user kubernetes pods. The keys and values must both be strings that match the kubernetes label key / value constraints.

See the Kubernetes documentation for more info on what labels are and why you might want to use them!

`{username}` and `{userid}` are expanded to the escaped, dns-label safe username & integer user id respectively, wherever they are used.

config c.KubeSpawner.extra_pod_config = Dict()
Extra configuration for the pod which is not covered by other attributes.

This dict will be directly merge into pod,so you should use the same structure. Each item in the dict is field of pod configuration which follows spec at https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.11/#podspec-v1-core.

One usage is set restartPolicy and dnsPolicy with configuration below:

```
c.KubeSpawner.extra_pod_config = {
    "restartPolicy": "OnFailure",
    "dns_policy": "ClusterFirstWithHostNet"
}
```

The `key` could be either a camelCase word (used by Kubernetes yaml, e.g. `restartPolicy`) or a snake_case word (used by Kubernetes Python client, e.g. `dns_policy`).

config c.KubeSpawner.extra_resource_guarantees = Dict()
The dictionary used to request arbitrary resources. Default is None and means no additional resources are requested. For example, to request 1 Nvidia GPUs:

```
c.KubeSpawner.extra_resource_guarantees = {"nvidia.com/gpu": "1"}
```

config c.KubeSpawner.extra_resource_limits = Dict()
The dictionary used to limit arbitrary resources. Default is None and means no additional resources are limited. For example, to add a limit of 3 Nvidia GPUs:

```
c.KubeSpawner.extra_resource_limits = {"nvidia.com/gpu": "3"}
```

config c.KubeSpawner.fs_gid = Union()
The GID of the group that should own any volumes that are created & mounted.

A special supplemental group that applies primarily to the volumes mounted in the single-user server. In volumes from supported providers, the following things happen:

1. The owning GID will be the this GID
2. The setgid bit is set (new files created in the volume will be owned by this GID)

3. The permission bits are OR'd with rw-rw

The single-user server will also be run with this gid as part of its supplemental groups.

Instead of an integer, this could also be a callable that takes as one parameter the current spawner instance and returns an integer. The callable will be called asynchronously if it returns a future, rather than an int. Note that the interface of the spawner class is not deemed stable across versions, so using this functionality might cause your JupyterHub or kubespawner upgrades to break.

You'll *have* to set this if you are using auto-provisioned volumes with most cloud providers. See fsGroup for more details.

config c.KubeSpawner.gid = Union()

The GID to run the single-user server containers as.

This GID should ideally map to a group that already exists in the container image being used. Running as root is discouraged.

Instead of an integer, this could also be a callable that takes as one parameter the current spawner instance and returns an integer. The callable will be called asynchronously if it returns a future. Note that the interface of the spawner class is not deemed stable across versions, so using this functionality might cause your JupyterHub or kubespawner upgrades to break.

If set to `None`, the group of the user specified with the `USER` directive in the container metadata is used.

config c.KubeSpawner.http_timeout = Int(30)

Timeout (in seconds) before giving up on a spawned HTTP server

Once a server has successfully been spawned, this is the amount of time we wait before assuming that the server is unable to accept connections.

config c.KubeSpawner.hub_connect_ip = Unicode('')

DEPRECATED. Use c.JupyterHub.hub_connect_ip

config c.KubeSpawner.hub_connect_port = Int(0)

DEPRECATED. Use c.JupyterHub.hub_connect_url

config c.KubeSpawner.image_pull_policy = Unicode('IfNotPresent')

The image pull policy of the docker container specified in `image_spec`.

Defaults to `IfNotPresent` which causes the Kubelet to NOT pull the image specified in image_spec if it already exists, except if the tag is `:latest`. For more information on image pull policy, refer to the Kubernetes documentation.

This configuration is primarily used in development if you are actively changing the `image_spec` and would like to pull the image whenever a user container is spawned.

config c.KubeSpawner.image_pull_secrets = Unicode(None)

The kubernetes secret to use for pulling images from private repository.

Set this to the name of a Kubernetes secret containing the docker configuration required to pull the image specified in `image_spec`.

See the Kubernetes documentation for more information on when and why this might need to be set, and what it should be set to.

config c.KubeSpawner.image_spec = Unicode('jupyterhub/singleuser:latest')

Docker image spec to use for spawning user's containers.

Defaults to `jupyterhub/singleuser:latest`

Name of the container + a tag, same as would be used with a `docker pull` command. If tag is set to `latest`, kubernetes will check the registry each time a new user is spawned to see if there is a newer image available. If available, new image will be pulled. Note that this could cause long delays when spawning, especially if the image is large. If you do not specify a tag, whatever version of the image is first pulled on the node will be used, thus possibly leading to inconsistent images on different nodes. For all these reasons, it is recommended to specify a specific immutable tag for the imagespec.

If your image is very large, you might need to increase the timeout for starting the single user container from the default. You can set this with:

```
c.KubeSpawner.start_timeout = 60 * 5  # Upto 5 minutes
```

config c.KubeSpawner.init_containers = List()
List of initialization containers belonging to the pod.

This list will be directly added under `initContainers` in the kubernetes pod spec, so you should use the same structure. Each item in the dict must a field of the V1Container specification.

One usage is disabling access to metadata service from single-user notebook server with configuration below:

```
c.KubeSpawner.init_containers = [{
    "name": "init-iptables",
    "image": "<image with iptables installed>",
    "command": ["iptables", "-A", "OUTPUT", "-p", "tcp", "--dport", "80", "-d
↪", "169.254.169.254", "-j", "DROP"],
    "securityContext": {
        "capabilities": {
            "add": ["NET_ADMIN"]
        }
    }
}]
```

See the Kubernetes documentation for more info on what init containers are and why you might want to use them!

To user this feature, Kubernetes version must greater than 1.6.

config c.KubeSpawner.ip = Unicode('0.0.0.0')
The IP address (or hostname) the single-user server should listen on.

We override this from the parent so we can set a more sane default for the Kubernetes setup.

config c.KubeSpawner.k8s_api_threadpool_workers = Int(20)
Number of threads in thread pool used to talk to the k8s API.

Increase this if you are dealing with a very large number of users.

Defaults to `5 * cpu_cores`, which is the default for `ThreadPoolExecutor`.

config c.KubeSpawner.lifecycle_hooks = Dict()
Kubernetes lifecycle hooks to set on the spawned single-user pods.

The keys is name of hooks and there are only two hooks, postStart and preStop. The values are handler of hook which executes by Kubernetes management system when hook is called.

Below is an sample copied from the Kubernetes documentation:

```
c.KubeSpawner.lifecycle_hooks = {
    "postStart": {
        "exec": {
            "command": ["/bin/sh", "-c", "echo Hello from the postStart␣
↪handler > /usr/share/message"]
        }
```

(continues on next page)

(continued from previous page)

```
    },
    "preStop": {
        "exec": {
            "command": ["/usr/sbin/nginx", "-s", "quit"]
        }
    }
}
```

See the Kubernetes documentation for more info on what lifecycle hooks are and why you might want to use them!

config c.KubeSpawner.mem_guarantee = ByteSpecification(None)

Minimum number of bytes a single-user notebook server is guaranteed to have available.

Allows the following suffixes:

- K -> Kilobytes
- M -> Megabytes
- G -> Gigabytes
- T -> Terabytes

This is a configuration setting. Your spawner must implement support for the limit to work. The default spawner, `LocalProcessSpawner`, does **not** implement this support. A custom spawner **must** add support for this setting for it to be enforced.

config c.KubeSpawner.mem_limit = ByteSpecification(None)

Maximum number of bytes a single-user notebook server is allowed to use.

Allows the following suffixes:

- K -> Kilobytes
- M -> Megabytes
- G -> Gigabytes
- T -> Terabytes

If the single user server tries to allocate more memory than this, it will fail. There is no guarantee that the single-user notebook server will be able to allocate this much memory - only that it can not allocate more than this.

This is a configuration setting. Your spawner must implement support for the limit to work. The default spawner, `LocalProcessSpawner`, does **not** implement this support. A custom spawner **must** add support for this setting for it to be enforced.

config c.KubeSpawner.modify_pod_hook = Callable(None)

Callable to augment the Pod object before launching.

Expects a callable that takes two parameters:

1. The spawner object that is doing the spawning
2. The Pod object that is to be launched

You should modify the Pod object and return it.

This can be a coroutine if necessary. When set to none, no augmenting is done.

This is very useful if you want to modify the pod being launched dynamically. Note that the spawner object can change between versions of KubeSpawner and JupyterHub, so be careful relying on this!

config c.KubeSpawner.namespace = Unicode('')

Kubernetes namespace to spawn user pods in.

If running inside a kubernetes cluster with service accounts enabled, defaults to the current namespace. If not, defaults to `default`

config c.KubeSpawner.node_affinity_preferred = List()

Affinities describe where pods prefer or require to be scheduled, they may prefer or require a node to have a certain label or be in proximity / remoteness to another pod. To learn more visit https://kubernetes.io/docs/concepts/configuration/assign-pod-node/

Pass this field an array of "PreferredSchedulingTerm" objects.* * https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.11/#preferredschedulingterm-v1-core

config c.KubeSpawner.node_affinity_required = List()

Affinities describe where pods prefer or require to be scheduled, they may prefer or require a node to have a certain label or be in proximity / remoteness to another pod. To learn more visit https://kubernetes.io/docs/concepts/configuration/assign-pod-node/

Pass this field an array of "NodeSelectorTerm" objects.* * https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.11/#nodeselectorterm-v1-core

config c.KubeSpawner.node_selector = Dict()

The dictionary Selector labels used to match the Nodes where Pods will be launched.

Default is None and means it will be launched in any available Node.

For example to match the Nodes that have a label of `disktype: ssd` use:

```
c.KubeSpawner.node_selector = {'disktype': 'ssd'}
```

config c.KubeSpawner.notebook_dir = Unicode('')

Path to the notebook directory for the single-user server.

The user sees a file listing of this directory when the notebook interface is started. The current interface does not easily allow browsing beyond the subdirectories in this directory's tree.

~ will be expanded to the home directory of the user, and {username} will be replaced with the name of the user.

Note that this does *not* prevent users from accessing files outside of this path! They can do so with many other means.

config c.KubeSpawner.options_form = Union()

An HTML form for options a user can specify on launching their server.

The surrounding `<form>` element and the submit button are already provided.

For example:

```
Set your key:
<input name="key" val="default_key"></input>
<br>
Choose a letter:
<select name="letter" multiple="true">
  <option value="A">The letter A</option>
  <option value="B">The letter B</option>
</select>
```

The data from this form submission will be passed on to your spawner in `self.user_options`

Instead of a form snippet string, this could also be a callable that takes as one parameter the current spawner instance and returns a string. The callable will be called asynchronously if it returns a future, rather than a str. Note that the interface of the spawner class is not deemed stable across versions, so using this functionality might cause your JupyterHub upgrades to break.

config c.KubeSpawner.pod_affinity_preferred = List()

Affinities describe where pods prefer or require to be scheduled, they may prefer or require a node to have a certain label or be in proximity / remoteness to another pod. To learn more visit https://kubernetes.io/docs/concepts/configuration/assign-pod-node/

Pass this field an array of "WeightedPodAffinityTerm" objects.* * https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.11/#weightedpodaffinityterm-v1-core

config c.KubeSpawner.pod_affinity_required = List()

Affinities describe where pods prefer or require to be scheduled, they may prefer or require a node to have a certain label or be in proximity / remoteness to another pod. To learn more visit https://kubernetes.io/docs/concepts/configuration/assign-pod-node/

Pass this field an array of "PodAffinityTerm" objects.* * https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.11/#podaffinityterm-v1-core

config c.KubeSpawner.pod_anti_affinity_preferred = List()

Affinities describe where pods prefer or require to be scheduled, they may prefer or require a node to have a certain label or be in proximity / remoteness to another pod. To learn more visit https://kubernetes.io/docs/concepts/configuration/assign-pod-node/

Pass this field an array of "WeightedPodAffinityTerm" objects.* * https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.11/#weightedpodaffinityterm-v1-core

config c.KubeSpawner.pod_anti_affinity_required = List()

Affinities describe where pods prefer or require to be scheduled, they may prefer or require a node to have a certain label or be in proximity / remoteness to another pod. To learn more visit https://kubernetes.io/docs/concepts/configuration/assign-pod-node/

Pass this field an array of "PodAffinityTerm" objects.* * https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.11/#podaffinityterm-v1-core

config c.KubeSpawner.pod_name_template = Unicode('jupyter-{username}{servername}')

Template to use to form the name of user's pods.

`{username}` and `{userid}` are expanded to the escaped, dns-label safe username & integer user id respectively.

This must be unique within the namespace the pods are being spawned in, so if you are running multiple jupyterhubs spawning in the same namespace, consider setting this to be something more unique.

config c.KubeSpawner.poll_interval = Int(30)

Interval (in seconds) on which to poll the spawner for single-user server's status.

At every poll interval, each spawner's `.poll` method is called, which checks if the single-user server is still running. If it isn't running, then JupyterHub modifies its own state accordingly and removes appropriate routes from the configurable proxy.

config c.KubeSpawner.port = Int(0)

The port for single-user servers to listen on.

Defaults to `0`, which uses a randomly allocated port number each time.

If set to a non-zero value, all Spawners will use the same port, which only makes sense if each server is on a different address, e.g. in containers.

New in version 0.7.

config c.KubeSpawner.post_stop_hook = Any(None)

An optional hook function that you can implement to do work after the spawner stops.

This can be set independent of any concrete spawner implementation.

config c.KubeSpawner.pre_spawn_hook = Any(None)

An optional hook function that you can implement to do some bootstrapping work before the spawner starts. For example, create a directory for your user or load initial content.

This can be set independent of any concrete spawner implementation.

Example:

```
from subprocess import check_call
def my_hook(spawner):
    username = spawner.user.name
    check_call(['./examples/bootstrap-script/bootstrap.sh', username])

c.Spawner.pre_spawn_hook = my_hook
```

config c.KubeSpawner.priority_class_name = Unicode('')

The priority class that the pods will use.

See https://kubernetes.io/docs/concepts/configuration/pod-priority-preemption for more information on how pod priority works.

config c.KubeSpawner.privileged = Bool(False)

Whether to run the pod with a privileged security context.

config c.KubeSpawner.profile_form_template = Unicode('n   <script>n   // Jup

Jinja2 template for constructing profile list shown to user.

Used when `profile_list` is set.

The contents of `profile_list` are passed in to the template. This should be used to construct the contents of a HTML form. When posted, this form is expected to have an item with name `profile` and the value the index of the profile in `profile_list`.

config c.KubeSpawner.profile_list = Union()

List of profiles to offer for selection by the user.

Signature is: `List(Dict())`, where each item is a dictionary that has two keys:

- `display_name`: the human readable display name (should be HTML safe)
- `description`: Optional description of this profile displayed to the user.
- `kubespawner_override`: a dictionary with overrides to apply to the KubeSpawner settings. Each value can be either the final value to change or a callable that take the `KubeSpawner` instance as parameter and return the final value.
- `default`: (optional Bool) True if this is the default selected option

Example:

```
c.KubeSpawner.profile_list = [
    {
        'display_name': 'Training Env - Python',
        'default': True,
        'kubespawner_override': {
            'image_spec': 'training/python:label',
            'cpu_limit': 1,
            'mem_limit': '512M',
        }
    }, {
```

(continues on next page)

(continued from previous page)

```
        'display_name': 'Training Env - Datascience',
        'kubespawner_override': {
            'image_spec': 'training/datascience:label',
            'cpu_limit': 4,
            'mem_limit': '8G',
        }
    }, {
        'display_name': 'DataScience - Small instance',
        'kubespawner_override': {
            'image_spec': 'datascience/small:label',
            'cpu_limit': 10,
            'mem_limit': '16G',
        }
    }, {
        'display_name': 'DataScience - Medium instance',
        'kubespawner_override': {
            'image_spec': 'datascience/medium:label',
            'cpu_limit': 48,
            'mem_limit': '96G',
        }
    }, {
        'display_name': 'DataScience - Medium instance (GPUx2)',
        'kubespawner_override': {
            'image_spec': 'datascience/medium:label',
            'cpu_limit': 48,
            'mem_limit': '96G',
            'extra_resource_guarantees': {"nvidia.com/gpu": "2"},
        }
    }
]
```

Instead of a list of dictionaries, this could also be a callable that takes as one parameter the current spawner instance and returns a list of dictionaries. The callable will be called asynchronously if it returns a future, rather than a list. Note that the interface of the spawner class is not deemed stable across versions, so using this functionality might cause your JupyterHub or kubespawner upgrades to break.

config c.KubeSpawner.pvc_name_template = Unicode('claim-{username}{servername}')
Template to use to form the name of user's pvc.

`{username}` and `{userid}` are expanded to the escaped, dns-label safe username & integer user id respectively.

This must be unique within the namespace the pvc are being spawned in, so if you are running multiple jupyterhubs spawning in the same namespace, consider setting this to be something more unique.

config c.KubeSpawner.scheduler_name = Unicode(None)
Set the pod's scheduler explicitly by name. See the Kubernetes documentation for more information.

config c.KubeSpawner.service_account = Unicode(None)
The service account to be mounted in the spawned user pod.

When set to `None` (the default), no service account is mounted, and the default service account is explicitly disabled.

This `serviceaccount` must already exist in the namespace the user pod is being spawned in.

WARNING: Be careful with this configuration! Make sure the service account being mounted has the minimal permissions needed, and nothing more. When misconfigured, this can easily give arbitrary users root over your entire cluster.

config c.KubeSpawner.start_timeout = Int(60)

Timeout (in seconds) before giving up on starting of single-user server.

This is the timeout for start to return, not the timeout for the server to respond. Callers of spawner.start will assume that startup has failed if it takes longer than this. start should return when the server process is started and its location is known.

config c.KubeSpawner.storage_access_modes = List()

List of access modes the user has for the pvc.

The access modes are:

- `ReadWriteOnce` – the volume can be mounted as read-write by a single node
- `ReadOnlyMany` – the volume can be mounted read-only by many nodes
- `ReadWriteMany` – the volume can be mounted as read-write by many nodes

See the Kubernetes documentation for more information on how access modes work.

config c.KubeSpawner.storage_capacity = Unicode(None)

The ammount of storage space to request from the volume that the pvc will mount to. This ammount will be the ammount of storage space the user has to work with on their notebook. If left blank, the kubespawner will not create a pvc for the pod.

This will be added to the `resources: requests: storage:` in the k8s pod spec.

See the Kubernetes documentation for more information on how storage works.

Quantities can be represented externally as unadorned integers, or as fixed-point integers with one of these SI suffices (`E, P, T, G, M, K, m`) or their power-of-two equivalents (`Ei, Pi, Ti, Gi, Mi, Ki`). For example, the following represent roughly the same value: `128974848`, `129e6`, `129M`, `123Mi`.

config c.KubeSpawner.storage_class = Unicode(None)

The storage class that the pvc will use. If left blank, the kubespawner will not create a pvc for the pod.

This will be added to the `annotations: volume.beta.kubernetes.io/storage-class:` in the pvc metadata.

This will determine what type of volume the pvc will request to use. If one exists that matches the criteria of the StorageClass, the pvc will mount to that. Otherwise, b/c it has a storage class, k8s will dynamically spawn a pv for the pvc to bind to and a machine in the cluster for the pv to bind to.

See the Kubernetes documentation for more information on how StorageClasses work.

config c.KubeSpawner.storage_extra_labels = Dict()

Extra kubernetes labels to set on the user PVCs.

The keys and values specified here would be set as labels on the PVCs created by kubespawner for the user. Note that these are only set when the PVC is created, not later when this setting is updated.

See the Kubernetes documentation for more info on what labels are and why you might want to use them!

`{username}` and `{userid}` are expanded to the escaped, dns-label safe username & integer user id respectively, wherever they are used.

config c.KubeSpawner.storage_pvc_ensure = Bool(False)

Ensure that a PVC exists for each user before spawning.

Set to true to create a PVC named with `pvc_name_template` if it does not exist for the user when their pod is spawning.

config c.KubeSpawner.supplemental_gids = Union()

A list of GIDs that should be set as additional supplemental groups to the user that the container runs as.

Instead of a list of integers, this could also be a callable that takes as one parameter the current spawner instance and returns a list of integers. The callable will be called asynchronously if it returns a future, rather than a list. Note that the interface of the spawner class is not deemed stable across versions, so using this functionality might cause your JupyterHub or kubespawner upgrades to break.

You may have to set this if you are deploying to an environment with RBAC/SCC enforced and pods run with a 'restricted' SCC which results in the image being run as an assigned user ID. The supplemental group IDs would need to include the corresponding group ID of the user ID the image normally would run as. The image must setup all directories/files any application needs access to, as group writable.

config c.KubeSpawner.tolerations = List()

List of tolerations that are to be assigned to the pod in order to be able to schedule the pod on a node with the corresponding taints. See the official Kubernetes documentation for additional details https://kubernetes.io/docs/concepts/configuration/taint-and-toleration/

Pass this field an array of "Toleration" objects

Example:

```
[
    {
        'key': 'key',
        'operator': 'Equal',
        'value': 'value',
        'effect': 'NoSchedule'
    },
    {
        'key': 'key',
        'operator': 'Exists',
        'effect': 'NoSchedule'
    }
]
```

config c.KubeSpawner.uid = Union()

The UID to run the single-user server containers as.

This UID should ideally map to a user that already exists in the container image being used. Running as root is discouraged.

Instead of an integer, this could also be a callable that takes as one parameter the current spawner instance and returns an integer. The callable will be called asynchronously if it returns a future. Note that the interface of the spawner class is not deemed stable across versions, so using this functionality might cause your JupyterHub or kubespawner upgrades to break.

If set to `None`, the user specified with the `USER` directive in the container metadata is used.

config c.KubeSpawner.volume_mounts = List()

List of paths on which to mount volumes in the user notebook's pod.

This list will be added to the values of the `volumeMounts` key under the user's container in the kubernetes pod spec, so you should use the same structure as that. Each item in the list should be a dictionary with at least these two keys:

- `mountPath` The path on the container in which we want to mount the volume.
- `name` The name of the volume we want to mount, as specified in the `volumes` config.

See the Kubernetes documentation for more information on how the `volumeMount` item works.

`{username}` and `{userid}` are expanded to the escaped, dns-label safe username & integer user id respectively, wherever they are used.

config c.KubeSpawner.volumes = List()

List of Kubernetes Volume specifications that will be mounted in the user pod.

This list will be directly added under `volumes` in the kubernetes pod spec, so you should use the same structure. Each item in the list must have the following two keys:

- `name` Name that'll be later used in the `volume_mounts` config to mount this volume at a specific path.
- `<name-of-a-supported-volume-type>` (such as `hostPath`, `persistentVolumeClaim`, etc) The key name determines the type of volume to mount, and the value should be an object specifying the various options available for that kind of volume.

See the Kubernetes documentation for more information on the various kinds of volumes available and their options. Your kubernetes cluster must already be configured to support the volume types you want to use.

`{username}` and `{userid}` are expanded to the escaped, dns-label safe username & integer user id respectively, wherever they are used.

config c.KubeSpawner.working_dir = Unicode(None)

The working directory where the Notebook server will be started inside the container. Defaults to `None` so the working directory will be the one defined in the Dockerfile.

`{username}` and `{userid}` are expanded to the escaped, dns-label safe username & integer user id respectively.

CHAPTER 4

Objects

Helper methods for generating k8s API objects.

`kubespawner.objects.`**`make_pod`**`(`*name*, *cmd*, *port*, *image_spec*, *image_pull_policy*, *image_pull_secret=None*, *node_selector=None*, *run_as_uid=None*, *run_as_gid=None*, *fs_gid=None*, *supplemental_gids=None*, *run_privileged=False*, *env=None*, *working_dir=None*, *volumes=None*, *volume_mounts=None*, *labels=None*, *annotations=None*, *cpu_limit=None*, *cpu_guarantee=None*, *mem_limit=None*, *mem_guarantee=None*, *extra_resource_limits=None*, *extra_resource_guarantees=None*, *lifecycle_hooks=None*, *init_containers=None*, *service_account=None*, *extra_container_config=None*, *extra_pod_config=None*, *extra_containers=None*, *scheduler_name=None*, *tolerations=None*, *node_affinity_preferred=None*, *node_affinity_required=None*, *pod_affinity_preferred=None*, *pod_affinity_required=None*, *pod_anti_affinity_preferred=None*, *pod_anti_affinity_required=None*, *priority_class_name=None*, *logger=None*`)`

Make a k8s pod specification for running a user notebook.

Parameters

- **`name`** – Name of pod. Must be unique within the namespace the object is going to be created in. Must be a valid DNS label.
- **`image_spec`** – Image specification - usually a image name and tag in the form of image_name:tag. Same thing you would use with docker commandline arguments
- **`image_pull_policy`** – Image pull policy - one of ‘Always’, ‘IfNotPresent’ or ‘Never’. Decides when kubernetes will check for a newer version of image and pull it when running a pod.
- **`image_pull_secret`** – Image pull secret - Default is None – set to your secret name to pull from private docker registry.
- **`port`** – Port the notebook server is going to be listening on

- **cmd** – The command used to execute the singleuser server.
- **node_selector** – Dictionary Selector to match nodes where to launch the Pods
- **run_as_uid** – The UID used to run single-user pods. The default is to run as the user specified in the Dockerfile, if this is set to None.
- **run_as_gid** – The GID used to run single-user pods. The default is to run as the primary group of the user specified in the Dockerfile, if this is set to None.
- **fs_gid** – The gid that will own any fresh volumes mounted into this pod, if using volume types that support this (such as GCE). This should be a group that the uid the process is running as should be a member of, so that it can read / write to the volumes mounted.
- **supplemental_gids** – A list of GIDs that should be set as additional supplemental groups to the user that the container runs as. You may have to set this if you are deploying to an environment with RBAC/SCC enforced and pods run with a 'restricted' SCC which results in the image being run as an assigned user ID. The supplemental group IDs would need to include the corresponding group ID of the user ID the image normally would run as. The image must setup all directories/files any application needs access to, as group writable.
- **run_privileged** – Whether the container should be run in privileged mode.
- **env** – Dictionary of environment variables.
- **volumes** – List of dictionaries containing the volumes of various types this pod will be using. See k8s documentation about volumes on how to specify these
- **volume_mounts** – List of dictionaries mapping paths in the container and the volume(specified in volumes) that should be mounted on them. See the k8s documentaiton for more details
- **working_dir** – String specifying the working directory for the notebook container
- **labels** – Labels to add to the spawned pod.
- **annotations** – Annotations to add to the spawned pod.
- **cpu_limit** – Float specifying the max number of CPU cores the user's pod is allowed to use.
- **cpu_guarentee** – Float specifying the max number of CPU cores the user's pod is guaranteed to have access to, by the scheduler.
- **mem_limit** – String specifying the max amount of RAM the user's pod is allowed to use. String instead of float/int since common suffixes are allowed
- **mem_guarantee** – String specifying the max amount of RAM the user's pod is guaranteed to have access to. String ins loat/int since common suffixes are allowed
- **lifecycle_hooks** – Dictionary of lifecycle hooks
- **init_containers** – List of initialization containers belonging to the pod.
- **service_account** – Service account to mount on the pod. None disables mounting
- **extra_container_config** – Extra configuration (e.g. envFrom) for notebook container which is not covered by parameters above.
- **extra_pod_config** – Extra configuration (e.g. tolerations) for pod which is not covered by parameters above.
- **extra_containers** – Extra containers besides notebook container. Used for some housekeeping jobs (e.g. crontab).

- **scheduler_name** – The pod's scheduler explicitly named.
- **tolerations** – Tolerations can allow a pod to schedule or execute on a tainted node. To learn more about pod tolerations, see https://kubernetes.io/docs/concepts/configuration/taint-and-toleration/.

  Pass this field an array of "Toleration" objects.* * https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.10/#nodeselectorterm-v1-core
- **node_affinity_preferred** – Affinities describe where pods prefer or require to be scheduled, they may prefer or require a node to have a certain label or be in proximity / remoteness to another pod. To learn more visit https://kubernetes.io/docs/concepts/configuration/assign-pod-node/

  Pass this field an array of "PreferredSchedulingTerm" objects.* * https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.10/#preferredschedulingterm-v1-core
- **node_affinity_required** – Affinities describe where pods prefer or require to be scheduled, they may prefer or require a node to have a certain label or be in proximity / remoteness to another pod. To learn more visit https://kubernetes.io/docs/concepts/configuration/assign-pod-node/

  Pass this field an array of "NodeSelectorTerm" objects.* * https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.10/#nodeselectorterm-v1-core
- **pod_affinity_preferred** – Affinities describe where pods prefer or require to be scheduled, they may prefer or require a node to have a certain label or be in proximity / remoteness to another pod. To learn more visit https://kubernetes.io/docs/concepts/configuration/assign-pod-node/

  Pass this field an array of "WeightedPodAffinityTerm" objects.* * https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.10/#weightedpodaffinityterm-v1-core
- **pod_affinity_required** – Affinities describe where pods prefer or require to be scheduled, they may prefer or require a node to have a certain label or be in proximity / remoteness to another pod. To learn more visit https://kubernetes.io/docs/concepts/configuration/assign-pod-node/

  Pass this field an array of "PodAffinityTerm" objects.* * https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.10/#podaffinityterm-v1-core
- **pod_anti_affinity_preferred** – Affinities describe where pods prefer or require to be scheduled, they may prefer or require a node to have a certain label or be in proximity / remoteness to another pod. To learn more visit https://kubernetes.io/docs/concepts/configuration/assign-pod-node/

  Pass this field an array of "WeightedPodAffinityTerm" objects.* * https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.10/#weightedpodaffinityterm-v1-core
- **pod_anti_affinity_required** – Affinities describe where pods prefer or require to be scheduled, they may prefer or require a node to have a certain label or be in proximity / remoteness to another pod. To learn more visit https://kubernetes.io/docs/concepts/configuration/assign-pod-node/

  Pass this field an array of "PodAffinityTerm" objects.* * https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.10/#podaffinityterm-v1-core
- **priority_class_name** – The name of the PriorityClass to be assigned the pod. This feature is Beta available in K8s 1.11.

`kubespawner.objects.`**`make_pvc`**(*name*, *storage_class*, *access_modes*, *storage*, *labels=None*, *annotations=None*)

Make a k8s pvc specification for running a user notebook.

Parameters

- **`name`** – Name of persistent volume claim. Must be unique within the namespace the object is going to be created in. Must be a valid DNS label.
- **`storage_class`** – String of the name of the k8s Storage Class to use.
- **`access_modes`** – A list of specifying what access mode the pod should have towards the pvc
- **`storage`** – The ammount of storage needed for the pvc

CHAPTER 5

Reflectors

class kubespawner.reflector.**NamespacedResourceReflector**(**args*, ***kwargs*)

Base class for keeping a local up-to-date copy of a set of kubernetes resources.

Must be subclassed once per kind of resource that needs watching.

CHAPTER 6

Custom Traitlets

Traitlets that are used in Kubespawner

class kubespawner.traitlets.**Callable**(*default_value=traitlets.Undefined*, *allow_none=False*, *read_only=None*, *help=None*, *config=None*, ***kwargs*)

A trait which is callable.

Classes are callable, as are instances with a __call__() method.

CHAPTER 7

Utilities

Misc. general utility functions, not tied to Kubespawner directly

`kubespawner.utils.`**`generate_hashed_slug`**(*slug*, *limit=63*, *hash_length=6*)

Generate a unique name that's within a certain length limit

Most k8s objects have a 63 char name limit. We wanna be able to compress larger names down to that if required, while still maintaining some amount of legibility about what the objects really are.

If the length of the slug is shorter than the limit - hash_length, we just return slug directly. If not, we truncate the slug to (limit - hash_length) characters, hash the slug and append hash_length characters from the hash to the end of the truncated slug. This ensures that these names are always unique no matter what.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index