Kube Documentation

Release 0.10.0

Floris Bruynooghe

May 26, 2017

Contents

Kube is an opinionated, Python wrapper around the Kubernetes API that enables you to interact with and manage your Kubernetes cluster. Kube's primary design goal is to enable easy access to all features offered by the Kubernetes API using the Python language, while hiding Kubernetes API peculiarities. The result is a consistent and easy to use pythonic API.

Currently, Kube has the following capabilities:

- Major resources wrapped: Nodes, Namespaces, Pods, ReplicaSets, ReplicationControllers, Daemonsets, Deployments, Services, Secrets.
- Good labelling support, you can read and modify resource labels.
- Blocking and non-blocking support for the WATCH API.
- Low-level access to the Kubernetes API.

At the moment creating, deleting and modifying resources in general must be done via the low level access Kube provides to the actual Kubernetes API however these features are in the process of being added.

Contents:

CHAPTER 1

Quickstart

Before you start you need to make the Kubernetes API available via a proxy, this is the officially recommended method to connect and the only one supported by `kube`. To do this, simply run `kubectl proxy` on the localhost and `kube` will use that connection, for example:

```
$ kubectl proxy
Starting to serve on localhost:8001
```

When running your `kube` code in an actual Kubernetes cluster you can simply run the kubectl proxy in a container in the same pod that your `kube` code is running in. Finally, if you haven't already done so, refer to the *Installation* chapter and install `kube`.

Cluster

The main entry point provided by `kube` is the *`kube.Cluster`* class. Creating an instance of this class is central to gaining access to the objects inside your Kubernetes cluster. The *`kube.Cluster`* class assumes the default endpoint used by `kubectl proxy` (i.e. `http://localhost:8001/api/`) so you can simply create an instance as follows:

```
import kube

cluster = kube.Cluster()
```

However if you're running your proxy at a non-default endpoint then you should instance your *`kube.Cluster`* class using the url parameter as follows:

```
cluster = kube.Cluster(url='http://localhost:8080/api')
```

Pretty much all the ways in which you would want to interact with the Kubernetes API are supported by `kube`, however a cluster provides a *`kube.APIServerProxy`* instance via the *`kube.Cluster.proxy`* attribute. This provides low-level access to the Kubernetes cluster and can be useful to manage API objects, or access objects not yet wrapped by `kube`.

Views and Items

`kube` has two important concepts: *views* and *items*. All API objects in Kubernetes have a *kind*, and *views* provide access to Kubernetes resources whose *kind* ends in `List` e.g. `PodList` or `NodeList`. *Items*, on the other hand, provide access to the individual resource items themselves, e.g. a `Pod` or a `Node`.

The *`kube.Cluster`* instance has appropriately named attributes representing the views that provide access to the Kubernetes resources for that cluster. So for example, to fetch the *ReplicaSet* named `auth-v3` in the `default` namespace you can simply use code like this:

```
>>> rs = cluster.replicasets.fetch('auth-v3', namespace='default')
>>> assert rs.meta.name == 'auth-v3'
>>> assert rs.meta.namespace = 'default'
>>> assert rs.kind is kube.Kind.ReplicaSet
```

A view is also an iterator of all the resource items it provides access to. So for example, retrieving the names of all the namespaces in your cluster can be done using a simple list comprehension:

```
>>> ns_names = [ns.meta.name for ns in cluster.namespaces]
>>> assert 'default' in ns_names
```

Note: Kubernetes versions all of its API objects. Whenever anything changes, the version for the resource is updated. Resource items returned by `kube` views are snapshots of a resource item's state at a certain version.

On that note, consider this:

```
>>> cluster = kube.Cluster()
>>> print([node.meta.version for node in cluster.nodes])
['6434482', '6434483', '6434481']
>>> # A bit later
...
>>> print([node.meta.version for node in cluster.nodes])
['6434485', '6434486', '6434484']
```

Note: This is just to show you that the metadata attribute carries the resource version and that it is updated when the resource changes. However comparing versions is not very useful as they are opaque blobs. It is advised that you compare resource items directly.

As you have probably noticed, all resource items have a `meta` attribute and the version of a resource item is kept in `meta.version`. The `meta` attribute is an instance of the *`kube.ObjectMeta`* class and provides convenient access to a Kubernetes resource item's metadata. For example, it provides access to the labels defined for a resource item:

```
>>> rs = cluster.replicasets.fetch('auth-v3', namespace='default')
>>> if 'mylabel' not in rs.meta.labels:
...     curr_rs = rs.meta.labels.set('mylabel', 'value')
...     assert curr_rs.meta.labels['mylabel'] == 'value'
...     print(rs.meta.version)
...     print(curr_rs.meta.version)
...     assert rs != curr_rs
...
6530399
6530416
```

There are a few points to note from the above:

- The `labels` attribute, an instance of *kube.ResourceLabels*, behaves as a mapping. Both the keys and values are strings. Note however that the mapping is immutable.
- To modify a resource item, `kube` will always require that you call a method.
- Any method which modifies a resource item will always return an instance of the newest revision of that resource item (i.e. `curr_rs`).

Almost all resource items have a specification and status associated with them. The specification is a copy of the raw data representing the resource, which for example, could be used to re-create it. The specification is accessible in raw dict form using an item's *kube.ItemABC.spec()* method.

The resource item's status is exposed directly on appropriately named attributes. So for example:

```
>>> assert rs.spec()['replicas'] == rs.observed_replicas
```

That should be enough to get you going, but do read on. The remainder of this documentation describes `kube` concepts and terminology in more detail, provides detailed information on using `kube` in its entirety and gives a full API reference.

CHAPTER 2

Installation

Kube requires Python 3. The current release is published on PyPI and the easiest way to install it is to use `pip` as follows:

```
$ pip install kube
```

Dependencies

The following libraries will be automatically installed from PyPI:

- requests >=2.5.0,<3.0.0

CHAPTER 3

Concepts and Terminology

Kubernetes is a large and complex system and the API (and related APIs in the ecosystem) expose it in all of its glory. Consequently the Kubernetes API can sometimes seem a bit confusing. Kube's remit is to try and insulate the developer from most of this complexity and provide a pythonic, intuitive interface to work with, while adhering to the main Kubernetes API concepts. This chapter outlines these concepts, and in addition describes how and where `kube` fits in. Throughout the documentation we will endeavour to consistently use the terminology defined here.

If you are interested in getting a deeper understanding of the concepts employed by the Kubernetes API (and we strongly recommend that you do) then the Kubernetes API conventions document is a must read.

Kubernetes API concepts from 10,000 feet

Principally, the Kubernetes API defines the following terms:

- **Kind**: The name of a particular object schema (e.g. *Node* or *Pod* kinds that have different attributes and properties).
- **Resource**: A representation of a system entity, sent or retrieved as JSON via HTTP to the server. Resources are represented as:
- *Collections*: A list of resources of the same type.
- *Elements*: An individual resource.

Resources typically deal in data of a particular *kind*. For example, the kind `Pod` (a kind of resource *element*) is exposed as a *pods* resource (a resource *collection* with a kind of `PodList`) that allows end users to create, update, and delete pods. Kubernetes maintains a convention that resource collection names are all lowercase and plural, whereas kinds (that are the *types* of resource elements) are CamelCase and singular. Here are some resource *kinds*:

- Collections: e.g. `PodList`, `ServiceList`, `NodeList`
- Elements: e.g. `Pod`, `Service`, `Node`

Additionally, by convention the Kubernetes API makes a distinction between the *specification* of an object, and the *status* of an object at the current time.

The *specification* is a complete description of an object's desired state, including configuration settings provided by the user, default values expanded by the system, and properties initialized or otherwise changed after creation by other ecosystem components (e.g., schedulers, auto-scalers), and is persisted in stable storage with the API object. If the specification is deleted, the object will be purged from the system.

The *status* summarises the current state of the object in the system, and is usually persisted with the object by an automated processes but may be generated on the fly.

When a new version of an object is POSTed or PUT, the specification is updated and available immediately. Over time the system will work to bring the *status* into line with the *specification*. The system will drive toward the most recent "spec" regardless of previous versions of that stanza.

In other words Kubernetes' behavior is level-based rather than edge-based which enables robust behavior in the presence of missed intermediate state changes.

How Kube maps these concepts

Views and Items

`kube` maintains two important concepts that support the principle Kubernetes API concepts described above: *views* and *items*. As discussed all API objects in Kubernetes have a *kind*, and *views* provide access to Kubernetes resources whose *kind* ends in `List` e.g. `PodList` or `NodeList`. Note however that a `kube` View is not exactly the same as a resource collection; resource collections on the K8s API carry metadata and are versioned, this detail is not exposed in `kube` views.

Items, on the other hand, provide access to the individual resource items (elements) themselves, e.g. a `Pod` or a `Node`.

All `kube` views and items implement the abstract base classes *`kube.ViewABC`* and *`kube.ItemABC`* respectively, relating to the *Collections* and *Elements* concepts prescribed by the Kubernetes API. Consequently, View and Item instances that represent them all have a `kind` property. Additionally:

- As a minimum, Views have a:
- `fetch` method to get an Item.
- `filter` method to get a subset of the items in a resource collection.
- `watch` method to get an iterator that will provide access to watch events that represent updates to items belonging to the view.
- As a minimum, Items have a:
- `fetch` method to get the latest version of an Item.
- `spec` method that represents the specification of the Item.
- `meta` property that provides access to an Item's metadata.
- `watch` method to get an iterator that will provide access to any watch events that represent updates to the item.
- `resource` property that provides name of the Kubernetes API resource.

So as one can see, an Item's *specification* is available via any Item instance's `spec` method. An Item's *status* however, is represented by a selection attributes particular to the *kind* of Item. For example, in the case of a Pod, attributes like; *`kube.PodItem.phase`*, *`kube.PodItem.start_time`*, *`kube.PodItem.message`*.

How Kube handles Kubernetes API versions

`kube` maintains a list of API base paths for present and past API versions for each resource type. When iterating over items in a `kube` view, `kube` uses the most recent API version base path that is found to be available.

Additional Terminology

Cluster

A Kubernetes cluster is a set of physical or virtual machines and other infrastructure which runs containerised applications. You would normally interact with the `apiserver` running on the Kubernetes master node. This is represented with `kube`'s entry-point class *`kube.Cluster`*, an instance of which is used to access, among other things, *Views* representing resource collections.

API object

The Kubernetes system is almost entirely controllable via the HTTP ReSTful API which provides the standard HTTP methods to control the API objects using various HTTP verbs. We use the general term *API object* to refer to *collections* (Views) or an *element* (Item), that is any object which can be retrieved using HTTP GET and has a `kind` in the returned JSON.

Object Metadata

The Kubernetes API exposes metadata for API Objects. This includes information like the namespace the object resides in, its name, any labels set and their values, version, uid and when the object was created. `kube` carries much of this information for *Items* but as discussed, however *Views* are not exactly the same as collections and only expose properties for particular metadata items, e.g. `namespace`.

Global Views

Views come in two flavours: global views and views bound to a namespace. Instances of global views are directly accessible from an attribute on a *`kube.Cluster`* instance. A global view will only contain all resource items of a certain kind that exist in the cluster, regardless of the namespace they reside in. When using a view bound to a namespace only the resource items residing in the given namespace are accessible.

Node

A `Node` is a worker machine in a Kubernetes Cluster. A `Node` may be a virtual or physical machine, depending on the cluster. `kube` exposes a cluster's `NodeList` resource via the *`kube.Cluster.nodes`* view. Each item in the view is a *`kube.NodeItem`* instance.

Namespace

Kubernetes supports multiple virtual clusters backed by the same physical cluster. These virtual clusters are called namespaces. `kube` exposes a cluster's `NamespaceList` resource via the *`kube.Cluster.namespaces`* view. Each item in the view is a *`kube.NamespaceItem`* instance. Furthermore `kube` exposes an API object's namespace

(if defined) on an instance property. For Views this is on the `namespace` property, for Items this in on the `meta.namespace` property.

Replication Controller

A `ReplicationController` ensures that a specified number of pod "replicas" are running at any one time. In other words, a `ReplicationController` makes sure that a pod or homogeneous set of pods are always up and available. If there are too many pods, it will kill some. If there are too few, the ReplicationController will start more. Unlike manually created pods, the pods maintained by a Replication-Controller are automatically replaced if they fail, get deleted, or are terminated. `kube` exposes a cluster's `ReplicationControllerList` resource via the `Cluster.replicationcontrollers` view. Each item in the view is a *`kube.ReplicationControllerItem`* instance.

ReplicaSet

A `ReplicaSet` is the next-generation Replication Controller. The only difference between a `ReplicaSet` and a Replication Controller is the selector support. `kube` only has views and items representing `ReplicaSetLists` and `ReplicaSet` respectively. `kube` exposes a cluster's `ReplicaSetList` resource via the `Cluster.replicasets` view. Each item in the view is a *`kube.ReplicaSetItem`* instance.

Daemonset

A DaemonSet ensures that all (or some) nodes run a copy of a pod. As nodes are added to the cluster, pods are added to them. As nodes are removed from the cluster, those pods are garbage collected. Deleting a DaemonSet will clean up the pods it created. `kube` exposes a cluster's `DaemonSetList` resource via the `Cluster.daemonsets` view. Each item in the view is a *`kube.DaemonSetItem`* instance.

Deployment

A Deployment provides declarative updates for Pods and Replica Sets (the next-generation Replication Controller). You only need to describe the desired state in a Deployment object, and the Deployment controller will change the actual state to the desired state at a controlled rate for you. You can define Deployments to create new resources, or replace existing ones by new ones. `kube` exposes a cluster's `DeploymentList` resource via the `Cluster.deployments` view. Each item in the view is a *`kube.DeploymentItem`* instance.

Pod

A `Pod` is the smallest deployable unit of computing that can be created and managed in Kubernetes. It is a group of one or more containers (such as *Docker containers*), the shared storage for those containers, and options about how to run them. Pods model an application-specific "logical host". `kube` exposes a cluster's PodList resource via the `Cluster.pods` view. Each item in the view is a *`kube.PodItem`* instance.

Container

Containers (for example *Docker Containers*) are run-times that execute on a Node under the shared context of a `Pod`. The Kubernetes API doesn't represent containers directly as API Objects but indirectly through a Pod's *specification* and *status*. `kube` wraps a Pod's container information up in the *`kube.PodItem.containers`* property which provides a list of *`kube.Container`* instances that themselves have properties that are *`kube.ContainerState`* instances for the current and last known container state.

Service

A Kubernetes `Service` is an abstraction which defines a logical set of fungible Pods and a policy by which to access them. The set of Pods targeted by a Service is usually determined by a Label Selector. `kube` exposes a cluster's ServiceList resource via the *`kube.Cluster.services`* view. Each item in the view is a *`kube.ServiceItem`* instance.

Secret

A `Secret` is an API object that contains a small amount of sensitive data such as a password, a token, or a key. `kube` exposes a cluster's SecretList resource via the *`kube.Cluster.secrets`* view. Each item in the view is a *`kube.SecretItem`* instance.

Watching for changes

`kube` supports the Kubernetes API *Watch* capability. All Views and Items provided by `kube` have a `watch` method that returns an iterator of `kube.WatchEvent` instances. Whenever one of the resources in a view changes, or a watched Item changes, a `kube.WatchEvent` instance is yielded.

CHAPTER 4

Clusters

CHAPTER 5

Resource Views and Resource Items

CHAPTER 6

Using Resource Labels

CHAPTER 7

Using Resource Filters

CHAPTER 8

Using Resource Watchers

CHAPTER 9

Testing Kube

CHAPTER 10

API Reference

The full API documentation.

Exceptions

There are a number of common exceptions used.

exception `kube.`**`KubeError`**
The base class for all custom exceptions used by the the kube library.

exception `kube.`**`APIError`**
This is an exception which gets raised whenever there is a problem communicating with the Kubernetes API server or if the server returns the wrong HTTP status code.

`message`
An optional custom message for the exception.

`response`
The `requests.Response` object of the failed API server communication.

`status_code`
The HTTP status code of the failed response from the API server. This is a shortcut to the `status_code` attribute of the response object itself.

exception `kube.`**`StatusError`**
All resource items, represented by concrete instances of *`ItemABC`*, have a number of attributes which represent the status of the resource item. Not all status items are always available depending on the state of the resource item. If a status attribute is not available then this exception is used.

exception `kube.`**`NamespaceError`**
This represents the use of an invalid namespace. Some resources do not support namespaces, while others require a namespace. If the namespace use was wrong this exception will be raised.

Cluster

The cluster class is the global entry point to a Kubernetes API server. It holds some resources for the cluster it connects to.

Mostly this provides access to the API objects via the ref:*views* present as attributes.

class `kube.`**`Cluster`**(*url='http://localhost:8001/'*)

A Kubernetes cluster.

The entry point to control a Kubernetes cluster. There is only one connection mechanism, which is via a local API server proxy. This is normally achieved by running `kubectl proxy`.

Parameters **`url`** (*`str`*) – The URL of the API server.

The default of the *url* parameter coincides with the defaults used by `kubectl proxy` so will usually be the correct value.

The cluster instance can also be used as a context manager. When used like this *`close()`* will be called automatically when the context manager exits.

`proxy`

A *`APIServerProxy`* instance for this cluster. This provides low-level access to the API server if you need it.

`nodes`

A global *`NodeView`* instance providing convenient access to cluster nodes.

`namespaces`

A global *`NamespaceView`* instance providing convenient access to the namespaces present in the cluster.

`replicasets`

A global *`ReplicaSetView`* instance providing convenient access to all *ReplicaSet* objects present in the cluster. This view is not bound to a particular namespace.

`replicationcontrollers`

A global *`ReplicationControllerView`* instance providing convenient access to all *ReplicationController* objects present in the cluster. This view is not bound to a particular namespace.

`daemonsets`

A global *`DaemonSetView`* instance providing convenient access to all *DaemonSet* objects present in the cluster. This view is not bound to a particular namespace.

`deployments`

A global *`DeploymentView`* instance providing convenient access to all *Deployment* objects present in the cluster. This view is not bound to a particular namespace.

`pods`

A global *`PodView`* instance providing convenient access to all *Pod* objects present in the cluster. This view is not bound to a particular namespace.

`services`

A global *`ServiceView`* instance providing convenient access to all *Service* objects present in the cluster. This view is not bound to a particular namespace.

`secrets`

A global *`SecretView`* instance providing convenient access to all *Secret* objects present in the cluster. This view is not bound to a particular namespace.

`close`()

Close and clean up underlying resources.

create(*data*, *namespace=None*)
Create a new resource item.

Parameters

- **data** (*dict*) – The specification to create the resource from, this must include the `apiVersion`, `kind`, `metadata` and `spec` fields. It is usually simply the de-serialised YAML but allows you to insert template processing if you require so.
- **namespace** (*str*) – Create the resource item in the given namespace. If the `spec` includes a namespace this namespace must match or an exception will be raised.

Returns The newly created item.

Return type A *kube.ViewABC* instance of the right type according to the kind of resource item created based on the data in the spec.

Raises

- ***kube.APIError*** – For errors from the k8s API server.
- ***kube.KubeError*** – If the spec is incomplete or the kind is unknown.

classmethod kindimpl(*kind*)
Return the class which implements the resource kind.

Parameters **kind** (kube.Kind) – The *kube.Kind* instance.

Returns A class implementing either *kube.ViewABC* or *kube.ItemABC* depending on the kind.

Raises **ValueError** – If the kind is not known.

APIServerProxy

This provides low-level access to the Kubernetes cluster. It can be useful to interact with API objects not yet wrapped by the library.

class kube.**APIServerProxy**(*base_url='http://localhost:8001/'*)
Helper class to directly communicate with the API server.

Since most classes need to communicate with the Kubernetes cluster's API server in a common way, this class helps take care of the common logic. It also keeps the requests session alive to enable connection pooling to the API server.

Parameters **base_url** (*str*) – The URL of the API, not including the API version.

Most methods take a variable-length *path* argument which is used to make up the URL queried. These parts are joined together and attached to the base URL configured on the class (e.g. `http://localhost:8001/`) using the *urljoin()* method. Thus, to query a namespace at `http://localhost:8001/api/v1/namespaces/default`, you would use `['api/v1', 'namespace', 'default']` as *path*. Likewise, `['api/v1', 'namespace', 'default', 'pods', 'foo']` as *path* would result in a query to `http://localhost:8001/api/v1/namespaces/default/pods/foo`.

It is also possible to use the full URL path instead as a single argument. This is useful when using the `selfLink` metadata from an API object. So using `['/api/v1/namespaces/default']` as *path* would also result in a URL of `http://localhost:8001/api/v1/namespaces/default`.

close()
Close underlying connections.

Once the proxy has been closed then the it can no longer be used to issue further requests.

delete(**path*, *json=None*, ***params*)
HTTP DELETE to the relative path on the API server.

Parameters

- **path** (*str*) – Individual relative path components, they will be joined using *urljoin()*.
- **json** (*collections.abc.Mapping*) – The body, which will be JSON-encoded before posting.
- **params** (*str*) – Extra query parameters for the URL of the DELETE request.

Returns The decoded JSON data.

Return type pyrsistent.PMap

Raises ***kube.APIError*** – If the response status is not 200 OK.

get(**path*, ***params*)
HTTP GET the path from the API server.

Parameters

- **path** (*str*) – Individual API path components, they will be joined using "/". None of the path components should include a "/" separator themselves, other than the first component, the API path, which may.
- **params** (*dict*) – Extra query parameters for the URL of the GET request as a dictionary of strings.

Returns The decoded JSON data.

Return type pyrsistent.PMap

Raises ***kube.APIError*** – If the response status is not 200 OK.

patch(**path*, *patch=None*)
HTTP PATCH as application/strategic-merge-patch+json.

This allows using the Strategic Merge Patch to patch a resource on the Kubernetes API server.

Parameters

- **path** (*str*) – Individual relative path components, they will be joined using "/". None of the path components should include a "/" separator themselves, other than the first component, the API path, which may - unless you only provide one component, which will be joined to the base URL using `urllib.parse.urljoin()`. This case can be useful to use the links provided by the API itself directly, e.g. from a resource's `metadata.selfLink` field.
- **patch** (*dict*) – The decoded JSON object with the patch data.

Returns The decoded JSON object of the resource after applying the patch.

Raises ***APIError*** – If the response status is not 200 OK.

post(**path*, *json=None*, ***params*)
HTTP POST to the relative path on the API server.

Parameters

- **path** (*str*) – Individual relative path components, they will be joined using *urljoin()*.

- **json** (*collections.abc.Mapping*) – The body to post, which will be JSON-encoded before posting.
- **params** (*str*) – Extra query parameters for the URL of the POST request.

Returns The decoded JSON data.

Return type pyrsistent.PMap

Raises ***kube.APIError*** – If the response status is not 201 Created.

urljoin(**path*)
Wrapper around urllib.parse.urljoin for the configured base URL.

Parameters **path** – Individual relative path components, they will be joined using "/". None of the path components should include a "/" separator themselves, other than the first component, the API path, which may.

watch(**path*, *version=None*, *fields=None*)
Watch a list resource for events.

This issues a request to the API with the `watch` query string parameter set to `true` which returns a chunked response. An iterator is returned which continuously reads from the response, yielding received lines as bytes.

Parameters

- **path** – The URL path to the resource to watch. See *urljoin()*.
- **version** (*str*) – The resource version to start watching from.
- **fields** (*dict*) – A dict of fields which must match their values. This is a limited form of the full fieldSelector format, it is limited because filtering is done at client side for consistency.

Returns An special iterator which allows non-blocking iterating using a `.next(timeout)` method. Using it as a normal iterator will result in blocking behaviour.

Return type `kube._watch.JSONWatcher`.

Raises ***APIError*** – If there is a problem with the API server.

Resources Interface

ViewABC

Views are central to how the kube API works and are how you get hold of resource items. They give you a view into the resource items part which are part of the resource. Views are implemented in concrete classes for each resource type which is wrapped by kube. The view API presented in this abstract base class which all the concrete views have to implement and provides a consistent API.

Currently views come in two flavours: global views and views bound to a namespace. Instances of global views are directly accessible from attributes on the *Cluster* instance. When using a global view it will contain all resource items of a certain kind which exist in the cluster, regardless of the namespace they reside in. When using a view bound to a namespace only the resource items residing in the given namespace are accessible.

class `kube.`**ViewABC**(*cluster*, *namespace=None*)
Represents a view to a collection of resources.

All top-level resources in Kubernetes have a collection, resources of a `*List` kind, with some common functionality. This ABC defines views to provide access to resources in collections in a uniform way. Note that a

view is not the same as the collection resource, e.g. collections resources have some metadata associated with them and exist at a particular point in time, they have a metadata.resourceVersion, which views do not have.

It is always possible to create an instance of this without needing to do any requests to the real Kubernetes cluster.

Parameters

- **cluster** (kube.Cluster) – The cluster this resource list is part of.
- **namespace** (*str*) – The optional namespace this resource list is part of. If the resource list is not part of a namespace this will be None which means it will be a view to all resources of a certain type, regardless of their namespace.

Raises ***kube.NamespaceError*** – When a namespace is provided but the resource does not support one.

api_paths

The list of possible Kubernetes API version base paths for resource.

This is a list of the API base path string for each of the existing API versions that could be used in the construction of the API endpoint for a resource, if available. For example, ['api/v1', '/apis/extensions/v1beta1']. They are listed in reverse chronological order, the most recent API version appearing first. kube uses the list to establish and use the most recent API version available.

cluster

The *kube.Cluster* instance this resource is bound to.

fetch(*name*, *namespace=None*)

Retrieve the current version of a single resource item by name.

If the view itself is associated with a namespace, self.namespace is not None, then this will default to fetching the resource item from this namespace. If the view is not associated with a namespace, self.namespace is None, and the resource requires a namespace then a *kube.NamespaceError* is raised. Note that the default namespace is not automatically used in this case.

Parameters

- **name** (*str*) – The name of the resource.
- **namespace** (*str*) – The namespace to fetch the resource from.

Returns A single instance representing the resource.

Raises

- **LookupError** – If the resource does not exist.
- ***kube.NamespaceError*** – For an invalid namespace, either because the namespace is required for this resource but not provided or the resource does not support namespaces and one was provided.
- ***kube.APIError*** – For errors from the k8s API server.

filter(*, *labels=None*, *fields=None*)

Return an iterable of a subset of the resources.

Parameters

- **labels** (*dict or str*) – A label selector expression. This can either be a dictionary with labels which must match exactly, or a string label expression as understood by k8s itself.
- **fields** (*dict or str*) – A field selector expression. This can either be a dictionary with fields which must match exactly, or a string field selector as understood by k8s itself.

Returns An iterator of *kube.ItemABC* instances of the correct type for the resrouce which match the given selector.

Raises

- **ValueError** – If an empty selector is used. An empty selector is almost certainly not what you want. Kubernetes treats an **empty** selector as *all* items and treats a **null** selector as *no* items.
- ***kube.APIError*** – For errors from the k8s API server.

kind
The kind of the underlying Kubernetes resource.

This is a *kube.Kind* enum.

This should be implemented as a static attribute since it needs to be available on the class as well as on the instance.

namespace
The optional namespace this view is bound to.

If the view is not bound to a namespace this will be `None`, including for resources which do not support namespaces.

resource
The name of the Kubernetes API resource.

The resource name is used in the construction of the API endpoint, e.g. for the API endpoint `/namespaces/default/pods/` the resource name is `pods`. The resource name is identical for both the resource as well as the resource item, e.g. both objects with `PodList` and `Pod` as kind will have a resource name of `pods`.

This should be implemented as a static attribute since it needs to be available on the class as well as on the instance.

watch()
Watch for changes to any of the resources in the view.

Whenever one of the resources in the view changes a new *kube.WatchEvent* instance is yielded. You can currently not control from "when" resources are being watched, other then from "now". So be aware of any race conditions with watching.

Returns An iterator of *kube.WatchEvent* instances.

Raises

- ***kube.NamespaceError*** – Whe the namespace no longer exists.
- ***kube.APIError*** – For errors from the k8s API server.

ResourceWatcher

A *ResourceWatcher* is used to watch resources and resource items. It should never be created directly but is instead returned by the *ViewABC.watch()* and *ItemABC.watch()* methods.

This class is also a context manager as it holds open active socket connections to the API server. On exiting the context manager the connections are closed. Typical usage would be:

```
cluster = kube.Cluster()
with cluster.pods.watch() as watcher:
```

```
for event in watcher:
    print(event)
```

class kube.**ResourceWatcher**(*cluster*, *jsonwatcher*, *itemcls*)

Watcher for a resource.

This is an iterator yielding watch events in either a blocking or non-blocking way, for non-blocking use `.next(timeout=0)`. It uses a `JSONWatcher` instance for retrieving the actual events, which must be configured correctly to return events for the same resource as this watcher is for.

Parameters

- **cluster** (*kube.Cluster*) – The cluster instance.
- **jsonwatcher** (*JSONWatcher*) – A correctly configured watcher instance which yields the decoded JSON objects.
- **itemcls** (*A callable, usually a class.*) – A constructor for the resource item being watched.

close()

Close the iterator and release it's resources.

This releases the underlying socket.

next(*, *timeout=None*)

Return the next watch event.

Parameters **timeout** (*int or float*) – The maximum time to wait for a new event. Not specifying this will block forever until a new event arrives, otherwise a `TimeoutError` is raised if no new event was received in time.

Raises **TimeoutError** – When no new event is available after the specified timeout.

WatchEvent

The namedtuple yielded by a *ResourceWatcher*.

class kube.**WatchEvent**

Events returned by the *ResourceWatcher* iterator, this is a namedtuple with the following fields:

evtype [field 0]

The first field of the tuple, representing the type of event, a *kube.WatchEventType* enum instance.

item [field 1]

The second field of the tuple, representing the API object itself. This will be a concrete instance of *ItemABC*.

MODIFIED

Shortcut to *kube.WatchEventType.MODIFIED*.

ADDED

Shortcut to *kube.WatchEventType.ADDED*.

DELETED

Shortcut to *kube.WatchEventType.DELETED*.

ERROR

Shortcut to *kube.WatchEventType.ERROR*.

WatchEventType

The types of watch events as an enum.

class kube.**WatchEventType**

MODIFIED

ADDED

DELETED

ERROR

Resource Items Interface

ItemABC

Individual resource items are always represented by a class implementing the *ItemABC* interface. Like with views each resource item is implemented in it's own concrete class but the abstract base class gives a consistent API.

Instances of an item represent the state of that resource item as a **snaphot in time**. All API objects in a Kubernetes cluster have a resource version associated with them and this particular version is what is represented by an item instance and thus an item instance is **immutable**. Items can and do implement methods which change the state of the resource item, in these cases a new instance of the item is returned by the method, representing the resource item in the state after the mutations have happened.

class kube.**ItemABC**(*cluster*, *raw*)

Representation for a kubernetes resource.

This is the interface all resource items must implement.

Parameters

- **cluster** (kube.Cluster) – The cluster this resource is bound to.
- **raw** (*dict*) – The decoded JSON representing the resource.

api_paths

The list of possible Kubernetes API version base paths for resource.

This is a list of the API base path string for each of the existing API versions that could be used in the construction of the API endpoint for a resource, if available. For example, `['api/v1', '/apis/extensions/v1beta1']`. They are listed in reverse chronological order, the most recent API version appearing first. kube uses the list to establish and use the most recent API version available.

cluster

The *kube.Cluster* instance this resource is bound to.

delete()

Delete the resource item.

Return type None

Raises ***APIError*** – For errors from the k8s API server.

fetch()

Fetch the current version of the resource item.

This will return a new instance of the current resource item at it's latest version. This is useful to see any changes made to the object since it was last retrieved.

Returns An instance of the relevant *ItemABC* subclass.

Raises ***kube.APIError*** – For errors from the k8s API server.

kind
The Kubernetes resource kind of the resource.

This is a *kube.Kind* enum.

This should be implemented as a static attribute since it needs to be available on the class as well as on the instance.

meta
The resource's metadata as a *kube.ObjectMeta* instance.

raw
The raw decoded JSON representing the resource.

This behaves like a dict but is actually an immutable view of the dict.

resource
The name of the Kubernetes API resource.

The resource name is used in the construction of the API endpoint, e.g. for the API endpoint `/namespaces/default/pods/` the resource name is `pods`. The resource name is identical for both the resource as well as the resource item, e.g. both objects with `PodList` and `Pod` as kind will have a resource name of `pods`.

This should be implemented as a static attribute since it needs to be available on the class as well as on the instance.

spec()
The spec of this node's resource.

This returns a copy of the raw, decoded JSON data representing the spec of this resource which can be used to re-create the resource.

watch()
Watch the resource item for changes.

Only changes after the current version will be part of the iterator. However it can not be guaranteed that *every* change is returned, if the current version is rather old some changes might no longer be available.

Returns An iterator of `kube.WatchEvents` instances for the resource item.

Raises ***kube.APIError*** – For errors from the k8s API server.

Kind

All API Objects in Kubernetes have a kind. The kind is represented as an enum instance where both the associated name and value matches the string used to describe the kind on the Kubernetes JSON API.

class `kube.`**Kind**

DaemonSet

DaemonSetList

Deployment

DeploymentList

Node

NodeList

Namespace

NamespaceList

Pod

PodList

ReplicaSet

ReplicaSetList

ReplicationController

ReplicationControllerList

Service

ServiceList

Secret

SecretList

ObjectMeta

Each instance of a concrete *ItemABC* class represents the metadata of the resource item in a *kube.ItemABC.meta* attribute. This attribute is always an instance of this *ObjectMeta* class to provide convenient access to the metadata. You would not normally create an instance manually.

class kube.**ObjectMeta**(*resource*)

Common metadata for API objects.

Parameters **resource** (*kube._base.ItemABC*) – The object representing the Kubernetes resource which this metadata describes.

created

The created timestamp as a `datetime.datetime` instance.

labels

The labels as a *ResourceLabels* instance.

link

A link to the resource itself.

This is currently an absolute URL without the hostname, but you don't have to care about that. The *kube.APIServerProxy* will be just fine with it as it's `path` argument.

name

The name of the object.

namespace

Namespace the object resides in, or `None`.

uid

The Universal ID of the item.

This is unique for this resource kind.

version
The opaque resource version.

ResourceLabels

This class is a `collections.abc.Mapping` of the labels applied to a resource item. It is not created manually but instead accessed via the *`ObjectMeta`* class.

Manipulation of the labels is supported using explicit method calls.

class `kube.`**`ResourceLabels`**(*resource*)
The labels applied to an API resource item.

This allows introspecting the labels as a normal mapping and provides a few methods to directly manipulate the labels on the resource item.

`delete`(*key*)
Delete a label.

This will remove the label for a given key from the resource.

Returns A new instance of the resource.

Raises ***`kube.APIError`*** – If there is a problem with the API server.

`set`(*key*, *value*)
Set a (new) label.

This will set or update the label's value on the resource.

Returns A new instance of the resource.

Raises ***`kube.APIError`*** – If there is a problem with the API server.

Nodes

NodeView

class `kube.`**`NodeView`**(*cluster*, *namespace=None*)
View of all the Node resource items in the cluster.

Parameters

- **`cluster`** (`kube.Cluster`) – The cluster instance.
- **`namespace`** (*`NoneType`*) – Limit the view to resource items in this namespace. This is here for the *`kube.ViewABC`* compatibility but can not be used for the NodeList resource. A *`kube.NamespaceError`* is raised when this is not `None`.

Raises ***`kube.NamespaceError`*** – If instantiated using a namespace.

Variables

- ***`kind`*** – The kind of the underlying Kubernetes resource item.
- ***`resource`*** – The name of the Kubernetes API resource.
- ***`cluster`*** – The *`kube.Cluster`* instance.

fetch(*name*, *namespace=None*)
Retrieve an individual node by name.

This returns the current verison of the resource item.

Parameters

- **name** (*str*) – The name of the node to retrieve.
- **namespace** (*str*) – Must be *None* or a *kube.NamespaceError* is raised. Here only for compatibility with the ABC.

Returns A single *kube.NodeItem* instance.

Raises

- **LookupError** – If the node does not exist.
- ***kube.APIError*** – For errors from the k8s API server.
- ***kube.NamespaceError*** – If a namespace is used.

NodeItem

class kube.**NodeItem**(*cluster*, *raw*)
A node in the Kubernetes cluster.

See http://kubernetes.io/docs/admin/node/ for details.

Parameters

- **cluster** (kube.Cluster) – The cluster this node belongs to.
- **raw** (*pyrsistent.PMap*) – The raw data of the resource item.

Variables

- ***kind*** – The kind of the underlying Kubernetes resource item.
- ***resource*** – The name of the Kubernetes API resource.

addresses
An iterator of the addresses for this node.

Each address is a namedtuple with `(type, addr)` as fields. Known types are in the `kube.AddressType` enumeration.

An empty list is returned if there are not yet any addresses associated with the node.

According to the K8s API spec (and K8s code) the node address array may contain addresses of the types defined by `kube.AddressType`. The `Hostname` address type, while unlikely, may present itself for certain cloud providers and will contain a hostname string, not an IP address.

capacity
The capacity of the node.

CPU is expressed in cores and can use fractions of cores, while memory is expressed in bytes.

conditions
List of conditions.

Namespaces

NamespaceView

class kube.**NamespaceView**(*cluster*, *namespace=None*)

View of all the Namespace resource items in the cluster.

Parameters

- **cluster** (kube.Cluster) – The cluster instance.
- **namespace** (*NoneType*) – Limit the view to resource items in this namespace. This is here for the *kube.ViewABC* compatibility, namespaces can not be used for the NamespaceList resource. A *kube.NamespaceError* is raised when this is not None.

Raises ***kube.NamespaceError*** – If instantiated using a namespace.

Variables

- ***kind*** – The kind of the underlying Kubernetes resource item.
- ***resource*** – The name of the Kubernetes API resource.

fetch(*name*, *namespace=None*)

Retrieve an individual Namespace resource item by name.

This returns the current version of the resource item.

Parameters

- **name** (*str*) – The name of the namespace resource item to retrieve.
- **namespace** (*str*) – Must be *None* or a *kube.NamespaceError* is raised. Here only for compatibility with the ABC.

Returns A *kube.NamespaceItem* instance.

Raises

- **LookupError** – If the namespace does not exist.
- ***kube.APIError*** – For errors from the k8s API server.
- ***kube.NamespaceError*** – If a namespace is used.

NamespaceItem

class kube.**NamespaceItem**(*cluster*, *raw*)

A namespace in the Kubernetes cluster.

See http://kubernetes.io/docs/admin/namespaces/ for details.

Variables

- ***kind*** – The kind of the underlying Kubernetes resource item.
- ***resource*** – The name of the Kubernetes API resource.
- ***NamespacePhase*** – Convenience alias of *NamespacePhase*.

class NamespacePhase

Enumeration of all possible namespace phases.

This is aliased to NamespaceResource.NamespacePhase for convenience.

NamespaceItem.**delete**()
Delete the namespace resource item.

For Namespace deletion K8s may have some work to do and could return a 409 (Conflict) instead of a 404 (Not Found) when a subsequent delete call occurs while status is trying to catch up with spec. We hide this idiosyncrasy from the kube user.

Return type None

Raises ***APIError*** – For errors from the k8s API server.

NamespaceItem.**phase**
Phase of the namespace as a kube.NamespacePhase.

ReplicaSets

ReplicaSetView

class kube.**ReplicaSetView**(*cluster*, *namespace=None*)
View of the ReplicaSet resource items in the cluster.

Parameters

- **cluster** (kube.Cluster) – The cluster instance.
- **namespace** (*str*) – Limit the view to resource items in this namespace.

Variables

- ***kind*** – The kind of the underlying Kubernetes resource item.
- ***resource*** – The name of the Kubernetes API resource.

ReplicaSetItem

class kube.**ReplicaSetItem**(*cluster*, *raw*)
A ReplicaSet in the Kubernetes cluster.

A ReplicaSet, formerly known as a ReplicationController, is responsible for keeping a desired number of pods running.

Parameters

- **cluster** (kube.Cluster) – The cluster this ReplicaSet exists in.
- **raw** (*pyrsistent.PMap*) – The raw data of the resource item.

Variables

- ***kind*** – The kind of the underlying Kubernetes resource item.
- ***resource*** – The name of the Kubernetes API resource.

available_replicas
The number of available replicas (ready for at least minReadySeconds) for the ReplicaSet.

fully_labeled_replicas
Number of pods which have an exact matching set of labels.

This counts the pods which have the exact same set of labels as the labelselector of this replicaset.

observed_generation
The (integer) generation of the ReplicaSet.

observed_replicas
The current number of replicas observed.

ready_replicas
The number of ready replicas for the ReplicaSet.

ReplicationControllers

ReplicationControllerView

class kube.**ReplicationControllerView**(*cluster*, *namespace=None*)
View of the Replication Controller resource items in the cluster.

Parameters

- **cluster** (kube.Cluster) – The cluster instance.
- **namespace** (*str*) – Limit the view to resource items in this namespace.

Variables

- ***kind*** – The kind of the underlying Kubernetes resource item.
- ***resource*** – The name of the Kubernetes API resource.

ReplicationControllerItem

class kube.**ReplicationControllerItem**(*cluster*, *raw*)
A Replication Controller in the Kubernetes cluster.

A ReplicationController, is responsible for keeping a desired number of pods running.

Parameters

- **cluster** (kube.Cluster) – The cluster this Replication Controller exists in.
- **raw** (*pyrsistent.PMap*) – The raw data of the resource item.

Variables

- ***kind*** – The kind of the underlying Kubernetes resource item.
- ***resource*** – The name of the Kubernetes API resource.

available_replicas
The number of available replicas (ready for at least minReadySeconds) for the ReplicaSet.

fully_labeled_replicas
Number of pods which have an exact matching set of labels.

This counts the pods which have the exact same set of labels as the labelselector of this replication controller.

observed_generation
The (integer) generation of the ReplicaSet.

observed_replicas
The current number of replicas observed.

ready_replicas
The number of ready replicas for the ReplicaSet.

Daemonsets

DaemonsetView

class kube.**DaemonSetView**(*cluster*, *namespace=None*)
View of the DaemonSet resource items in the cluster.

Parameters

- **cluster** (kube.Cluster) – The cluster instance.
- **namespace** (*str*) – Limit the view to resource items in this namespace.

Variables

- ***kind*** – The kind of the underlying Kubernetes resource item.
- ***resource*** – The name of the Kubernetes API resource.

DaemonsetItem

class kube.**DaemonSetItem**(*cluster*, *raw*)
A DaemonSet in the Kubernetes cluster.

A Daemon Set ensures that all (or some) nodes run a copy of a pod.

Parameters

- **cluster** (kube.Cluster) – The cluster this DaemonSet exists in.
- **raw** (*pyrsistent.PMap*) – The raw data of the resource item.

Variables

- ***kind*** – The kind of the underlying Kubernetes resource item.
- ***resource*** – The name of the Kubernetes API resource.

current_number_scheduled
The number of nodes that are running at least 1 daemon pod.

The count is of those nodes that are running at least one daemon pod and that are supposed to run the daemon pod.

desired_number_scheduled
The total number of nodes that should be running the daemon pod.

This includes nodes correctly running the daemon pod.

number_misscheduled
Number of nodes running the daemon pod, but not supposed to be.

number_ready
The number of nodes that have one or more of the daemon pod ready.

Nodes counted are those that should be running the daemon pod and have one or more of the daemon pod running and ready.

Deployments

DeploymentView

class kube.**DeploymentView**(*cluster*, *namespace=None*)
View of the Deployment resource items in the cluster.

Parameters

- **cluster** (kube.Cluster) – The cluster instance.
- **namespace** (*str*) – Limit the view to resource items in this namespace.

Variables

- ***kind*** – The kind of the underlying Kubernetes resource item.
- ***resource*** – The name of the Kubernetes API resource.

DeploymentItem

class kube.**DeploymentItem**(*cluster*, *raw*)
A Deployment in the Kubernetes cluster.

A Deployment provides declarative updates for Pods and Replica Sets.

Parameters

- **cluster** (kube.Cluster) – The cluster this Deployment exists in.
- **raw** (*pyrsistent.PMap*) – The raw data of the resource item.

Variables

- ***kind*** – The kind of the underlying Kubernetes resource item.
- ***resource*** – The name of the Kubernetes API resource.

available_replicas
Number of available pods ready for at least minReadySeconds.

observed_generation
The (integer) generation of the Deployment.

observed_replicas
Total number of non-terminated pods targeted by this deployment.

unavailable_replicas
Total number of unavailable pods targeted by this deployment.

updated_replicas
Nr. of non-terminated pods targeted with desired template spec.

Pods

PodView

class kube.**PodView**(*cluster*, *namespace=None*)
View of the Pod resource items in the cluster.

Parameters

- **cluster** (kube.Cluster) – The cluster instance.
- **namespace** (*str*) – Limit the view to resource items in this namespace.

Variables

- ***kind*** – The kind of the underlying Kubernetes resource item.
- ***resource*** – The name of the Kubernetes API resource.

PodItem

class kube.**PodItem**(*cluster*, *raw*)

A pod in the Kubernetes cluster.

Each pod contains a number of containers and volumes which are executed on a node within the cluster. A pod may exist in a namespace. Pods are typically managed by a controller such as a replication controller or job.

Parameters

- **cluster** (kube.Cluster) – The cluster this pod exists in.
- **raw** (*pyrsistent.PMap*) – The raw data of the resource item.

Variables

- ***kind*** – The kind of the underlying Kubernetes resource item.
- ***resource*** – The name of the Kubernetes API resource.
- ***PodPhase*** – Convenience alias of *PodPhase*.

class PodPhase

Enumeration of all possible pod phases.

This is aliased to Pod.PodPhase for convenience.

PodItem.**containers**

Iterate over all *Container* instances in the pod.

PodItem.**host_ip**

IP address of the pod's host within the cluster.

This may be as a ipaddress.IPv4Address or a ipaddress.IPv6Address.

Raises ***kube.StatusError*** – If this status item is not present.

PodItem.**ip**

IP address of the pod within the cluster.

This may be as a ipaddress.IPv4Address or a ipaddress.IPV6Address.

Raises ***kube.StatusError*** – If this status item is not present.

PodItem.**message**

Human readable message explaining the pod's state.

Raises ***kube.StatusError*** – If this status item is not present.

PodItem.**phase**

Phase of the pod as a kube.PodPhase.

Raises ***kube.StatusError*** – If this status item is not present.

PodItem.**reason**
PascalCase string explaining the pod's state.

Raises *kube.StatusError* – If this status item is not present.

PodItem.**start_time**
Start the pod was started as a datetime.datetime.

Raises *kube.StatusError* – If this status item is not present.

Container

class kube.**Container**(*pod*, *raw*)
A container inside a pod.

Containers live inside a pod and may be restarted inside this pod as controlled by the restart policy set on the pod.

Parameters

- **pod** (PodItem) – The pod the container is part off.
- **raw** (*pyrsistent.PMap*) – The JSON-decoded object describing the status of the container.

Variables

- **pod** – The *PodItem* instance the container is bound to.
- ***raw*** – The raw JSON-decoded object representing the container.

id
The ID of the running container.

For Docker this is in the docker://<hex_id> format.

image
The image the container is running.

For Docker this is normally the repository name with tag appended.

image_id
The ImageID of the container's image.

For Docker this is in the docker://<hex_id> format.

last_state
Previous state of the container, if known.

This is represented by a *ContainerState* instance.

Raises *kube.StatusError* – If this status item is not present.

name
The name of the container as a string.

ready
Boolean indicating if the container passed it's readyness probe.

Raises *kube.StatusError* – If this status item is not present.

restart_count
The number of times the container was restarted as an integer.

Note that this is currently not always accurate, it counts the number of dead containers which have not yet been removed. This means the gargage collection of containers caps this number at 5.

state
Current state of the container.

This is represented by a *ContainerState* instance.

Raises ***kube.StatusError*** – If this status item is not present.

ContainerState

class kube.**ContainerState**(*raw*)
The state of a container within a pod.

A container can be in one of three states: *running*, *waiting* or *terminated*. This class provides a uniform interface to all states and their associated details. Not all fields are always valid for each state so they can all raise an *kube.StatusError* when they are not available or not applicable.

The overall state of the container is available both as a string in the `state` attribute as well as booleans in the *waiting*, *running* and *terminated* attributes.

Parameters **raw** (*pyrsistent.PMap*) – The raw JSON-decoded `v1.ContainerState` API object as exposed by `v1.ContainerStatus` objects.

Variables ***raw*** – The raw JSON-decoded object representing the container state.

container_id
The container ID of the terminated container. Available for the *terminated* state.

Raises ***kube.StatusError*** – When this is not provided.

exit_code
Exit code of the container (int).

Available for the *terminated* state.

Raises ***kube.StatusError*** – When this is not provided.

finished_at
The time the container was terminated (datetime.datetime).

Available for the *terminated* state.

Raises ***kube.StatusError*** – When this is not provided.

message
Message regarding the container's state (str).

Available for *waiting* and *terminated* states.

Raises ***kube.StatusError*** – When this is not provided.

reason
Brief reason explaining the container's state (str).

This is normally a CamelCased message ID.

Available for *waiting* and *terminated* states.

Raises ***kube.StatusError*** – When this is not provided.

running
Boolean indicating if the container is running.

signal
Last signal sent to the container, if known (int).

Not all terminated containers can be expected to have this.

> **Warning:** The signal is identified numerically, however these signal numbers are not portable therefore it's ill-advised to attempt to compare this value with the constants provided by the built-in `singal` module.

Available for the *terminated* state.

Raises ***kube.StatusError*** – When this is not provided.

started_at
The time the container was started or restarted (datetime.datetime).

Available for the *running* state.

Raises ***kube.StatusError*** – When this is not provided.

terminated
Boolean indicating if the container has been terminated.

waiting
Boolean indicating if the container is waiting.

Services

ServiceView

class `kube.`**ServiceView**(*cluster*, *namespace=None*)
View of the Service resource items in the cluster.

Parameters

- **cluster** (`kube.Cluster`) – The cluster instance.
- **namespace** (*str*) – Limit the view to resource items in this namespace.

Variables

- ***kind*** – The kind of the underlying Kubernetes resource item.
- ***resource*** – The name of the Kubernetes API resource.

ServiceItem

class `kube.`**ServiceItem**(*cluster*, *raw*)
A Service in the Kubernetes cluster.

Parameters

- **cluster** (`kube.Cluster`) – The cluster this Service exists in.
- **raw** (*pyrsistent.PMap*) – The raw data of the resource item.

Variables

- ***kind*** – The kind of the underlying Kubernetes resource item.

- ***resource*** – The name of the Kubernetes API resource.

loadbalancer_ingress
The load balancer ingress endpoints.

This is a set of ingress endpoints in use by the load balancer. Depending on the infrastructure the cluster runs on the endpoint can be either an `ipaddress.IPv4Address`, `ipaddress.IPv6Address` or a hostname as a string.

Secrets

SecretView

class `kube.`**SecretView**(*cluster*, *namespace=None*)
View of the Secret resource items in the cluster.

Parameters

- **cluster** (`kube.Cluster`) – The cluster instance.
- **namespace** (*str*) – Limit the view to resource items in this namespace.

Variables

- ***kind*** – The kind of the underlying Kubernetes resource item.
- ***resource*** – The name of the Kubernetes API resource.

SecretItem

class `kube.`**SecretItem**(*cluster*, *raw*)
A Secret in the Kubernetes cluster.

Parameters

- **cluster** (`kube.Cluster`) – The cluster this Service exists in.
- **raw** (*pyrsistent.PMap*) – The raw data of the resource item.

Variables

- ***kind*** – The kind of the underlying Kubernetes resource item.
- ***resource*** – The name of the Kubernetes API resource.
- ***SecretType*** – Shortcut to `kube.SecretType`.

class SecretType
Enumeration of secret types.

`SecretItem.`**data**
A mapping of the secret data.

A copy of the secret data as a dict. The keys are the names of the secrets as a (unicode) string, while the values are the secrets as bytestrings.

Secret values are stored in a base64 encoding on the k8s master, but this is an implementation detail that this property takes care off for you.

SecretItem.**spec**()
An empty dictionary.

This is supposed to be the secret resource item's spec. But secrets do not have a spec, so to still follow the *kube.ItemABC* we return an empty dict.

SecretItem.**type**
The type of secret.

There currently is only the "Opaque" type.

CHAPTER 11

Glossary

ReplicaSet Formerly a replication controller, kube hides this transition from you and exposes this only under the `kube.ReplicaSetView` and `kube.ReplicaSetItem` names.

A replica set ensures a specified number of identical *Pod* instances are running by starting and stopping pods as required while watching for failed pods. See http://kubernetes.io/docs/user-guide/replication-controller/ for full details.

ReplicationController The old name for a *ReplicaSet*. The main difference is that the *labelSelector* for a ReplicationController can only select equality-based label sets. See https://kubernetes.io/docs/user-guide/replication-controller/ for details.

DaemonSet A DaemonSet ensures that all (or some) nodes run a copy of a pod. As nodes are added to the cluster, pods are added to them. As nodes are removed from the cluster, those pods are garbage collected. See https://kubernetes.io/docs/admin/daemons/ for full details.

Deployment A Deployment provides declarative updates for Pods and Replica Sets. You only need to describe the desired state in a Deployment object, and the Deployment controller will change the actual state to the desired state at a controlled rate for you. See https://kubernetes.io/docs/user-guide/deployments/ for full details.

Pod A pod is the smallest unit to run containers in the cluster. It is a co-located group of containers and volumes. See http://kubernetes.io/docs/user-guide/pods/ for the full details on pods.

Service A service groups a set of pods and makes them accessible via a single IP address and DNS name. See http://kubernetes.io/docs/user-guide/services/ for the full details on services.

Secret A secret stores sensitive data like authentication tokens which containers can then use. See http://kubernetes.io/docs/user-guide/secrets/ for the full details on secrets.

labelSelector Many objects in the Kubernetes clusters have labels associated with them. These can often be used to select a number of target objects and there is a fairly rich selector language to target objects using labels. See http://kubernetes.io/docs/user-guide/labels/#label-selectors for the full details on selectors.

CHAPTER 12

Indices and tables

- genindex
- modindex
- search

Python Module Index

k

Index

S

T

U

V

W