kubetest Documentation

Release 0.9.5

Vapor IO

Mar 24, 2021

Contents

`kubetest` is a pytest plugin that makes it easier to write integration tests on Kubernetes. This allows you to automate tests for your Kubernetes infrastructure, networking, and disaster recovery.

kubetest was written out of a desire to test deployment behavior in a quasi-deterministic manner. Other solutions exist for on-cluster testing, such as chaos testing, but often those solutions lack the ability to test the specific state of a small piece of the deployment.

kubetest aims to make that easier, giving you control of your cluster from within your test cases, and providing a simple API for managing the cluster and the objects on it.

CHAPTER 1

Features

- Simple API for common cluster interactions.
- Uses the Kubernetes Python client as the backend, allowing more complex cluster control for actions not covered by our API.
- Load Kubernetes manifest YAMLs into their Kubernetes models.
- Each test is run in its own namespace and the namespace is created and deleted automatically.
- Detailed logging to help debug error cases.
- Wait functions for object readiness, deletion, and test conditions.
- Allows you to search container logs for expected log output.
- RBAC permissions can be set at a test-case granularity using pytest markers.

CHAPTER 2

Installation

kubetest can be installed with `pip`

```
$ pip install kubetest
```

Note: The kubetest package has entrypoint hooks defined in `setup.py` which allow it to be automatically made available to pytest. This means that it will run whenever pytest is run. Since kubetest expects a cluster to be set up and to be given configuration for that cluster, pytest will fail if those are not present. It is therefore recommended to only install kubetest in a virtual environment or other managed environment, such as a CI pipeline, where you can assure that cluster access and configuration are available.

CHAPTER 3

Feedback

Feedback for kubetest is greatly appreciated! If you experience any issues, find the documentation unclear, have feature requests, or just have questions about it, we'd love to know. Feel free to open an issue on GitHub for any feedback you may have. If you are reporting a bug, please provide as much context as you can.

CHAPTER 4

License

kubetest is free and open source software distributed under the terms of the GPLv3 license.

4.1 Command Line Usage

Once installed (see *Installation*), the following pytest command-line options become available:

```
pytest \
    [--kube-error-log-lines <COUNT>] \
    [--suppress-insecure-request] \
    [--kube-log-level <LEVEL>] \
    [--kube-context <CONTEXT>] \
    [--kube-config <PATH>] \
    [--kube-disable] \
    [--in-cluster]
```

kubernetes integration test support:

--kube-config=path the kubernetes config for kubetest; this is required for resources to be installed on the cluster

--kube-context=context the name of the kubernetes config context to use

--kube-disable [DEPRECATED] disable automatic configuration with the kubeconfig file

--in-cluster use the kubernetes in-cluster config

--kube-log-level=KUBE_LOG_LEVEL log level for the kubetest logger

--kube-error-log-lines=KUBE_ERROR_LOG_LINES set the number of lines to tail from container logs on error. to show all lines, set this to -1.

--suppress-insecure-request=SUPPRESS_INSECURE_REQUEST suppress the urllib3 InsecureRequestWarning. This is useful if testing against a cluster without HTTPS set up.

- `--kube-config <PATH>`

  Specifies the path to the config file to use for connecting to your cluster. Alternatively, you can set the KUBECONFIG env var, and then you will not need to specify. If this option is not specified, `kubetest` will not install resources onto the cluster, which may cause test failure.

- `--kube-context <CONTEXT>`

  Specifies the context to use in the kubeconfig. If not specified, it will use the current context, as set in the kubeconfig.

- `--kube-disable` **DEPRECATED**

  Note: *v0.2.0*: This flag no longer does anything. It will be removed in the next major release.

  Disable kubetest from running. This can be useful when running pytest when no backing cluster is needed (e.g. to view the registered markers via `pytest --markers`).

- `--kube-error-log-lines <COUNT>`

  Set the number of lines to tail from the container logs for a test namespace when a test fails. By default, this is set to 50. If you want to show all container logs, set this to -1. If you do not wish to display any container logs in the pytest results, set this to 0.

- `--kube-log-level <LEVEL>`

  Sets the logging level for kubetest. The default log level is *warining*. Setting the log level to *info* will provide logging for kubetest actions. Setting the log level to *debug* will log out the Kubernetes object state for various actions as well.

- `--in-cluster`

  Use the Kubernetes in cluster config. With this specified, you do not need to supply a kube-config via the `--kube-config` option.

- `--suppress-insecure-request`

  Suppress the urllib3 InsecureRequestWarning. This is useful if testing against a cluster without HTTPS set up.

4.2 Fixtures

This section defines the fixtures that `kubetest` makes available when installed.

Note: Use `pytest --fixtures` to get a complete list of available fixtures along with their descriptions.

4.2.1 kube

`kubetest.plugin.`**`kube`**(*kubeconfig*, *kubecontext*, *request*) → kubetest.client.TestClient

Return a client for managing a Kubernetes cluster for testing.

Summary

The `kube` fixture is the "entrypoint" for using kubetest. In addition to returning a client that can be used to manage your cluster from within the test case, it also provides automatic management of the test case on the cluster.

What this means is that it will create a Namespace specific to that test case (prefixed with `kubetest-`) and will create any resources specified by the kubetest *Markers*. Once the test has completed, the test client and all test-associated resources set up by this fixture are cleaned up.

For the full API for the test client provided by the `kube` fixture, see the *Client* documentation.

Example Usage

Below is a simple trivial example of how the `kube` fixture can be used. See the *Examples* page for more.

```
def test_simple_deployment(kube):
    """Test that a simple deployment runs as intended."""

    # Load and create a deployment
    deployment = kube.load_deployment('path/to/deployment.yaml')
    deployment.create()

    # Wait until the deployment is in the ready state and then
    # refresh its underlying object data
    deployment.wait_until_ready(timeout=10)
    deployment.refresh()

    # Get the pods from the deployment and check that we have
    # the right number of replicas
    pods = deployment.get_pods()
    assert len(pods) == 1

    # Get the pod, ensure that it is ready, then get the containers
    # for that pod.
    pod = pods[0]
    pod.wait_until_ready(timeout=10)

    containers = pod.get_containers()
    assert len(containers) == 1
```

4.2.2 kubeconfig

`kubetest.plugin.`**`kubeconfig`**(*request*) → Optional[str]

Return the name of the configured kube config file loaded for the tests.

Summary

The `kubeconfig` fixture provides the name of the kubeconfig file which was used to load cluster configuration for the test. This should be the same value which was passed in via the `--kube-config` command line parameter.

Example Usage

Below is a simple trivial example of how the `kubeconfig` fixture may be used.

```
def test_something(kubeconfig):
    """A test case that gets the config file name via fixture."""

    assert kubeconfig == '~/.kube/config'
```

4.2.3 clusterinfo

kubetest.plugin.**clusterinfo**(*kubeconfig*) → kubetest.plugin.ClusterInfo
Get a ClusterInfo instance which provides basic information about the cluster the tests are being run on.

Summary

The clusterinfo fixture provides some basic information about the cluster which the tests are being run on. This information is taken from the cluster configuration and current context, so it should match the corresponding entries in the kube config file.

Important: When using the clusterinfo fixture, you should *always* specify it **after** the kube fixture. This is because the clusterinfo fixture does not load the specified kubeconfig file, whereas the kube fixture does. Invoking the clusterinfo fixture before the kube fixture will lead to some default configuration values to be returned, which may not accurately reflect the actual configuration used when the tests are run.

Good

```
def test_example(kube, clusterinfo):
    ...
```

Bad

```
def test_example(clusterinfo, kube):
    ...
```

The clusterinfo fixture returns an instance of ClusterInfo:

class kubetest.plugin.**ClusterInfo**(*current*, *config*)
Information about the cluster the kubetest is being run on.

This info is gathered from the current context and the loaded configuration.

Variables

- **cluster** – The name of the cluster set for the current context.
- **user** – The name of the user set for the current context.
- ***context*** – The name of the current context.
- **host** – API server address.
- **verify_ssl** – SSL certificate verification when calling the API.

Example Usage

Below is a simple trivial example of how the kubeconfig fixture may be used.

```
def test_something(kube, clusterinfo):
    """A test case that gets cluster info via fixture."""

    assert clusterinfo.user == 'test-user'
    assert clusterinfo.context == 'text-context'
```

4.3 Markers

This section defines the markers that `kubetest` makes available when installed.

Note: Use `pytest --markers` to get a complete list of available markers along with their descriptions.

4.3.1 Apply Manifest

Summary

```
@pytest.mark.applymanifest(path)
```

`applymanifest` allows you to load a Kubernetes manifest file and create the resource(s) on the cluster.

Description

Load the YAML manifest with the specified `path` and create the corresponding resource(s) on the cluster.

Loading a manifest via this marker does not prohibit you from manually loading other manifests later in the test case. You can use the *kube* fixture to get object references to the created resources. Manifests loaded via this marker are registered with the internal test case `MetaInfo` and can be waited upon for creation via the *kube* fixture's `wait_until_created` method.

The path to the directory should either be an absolute path, or a path relative from the test file. This marker can be used multiple times on a test case. If you wish to load multiple manifests at once, consider using the *Apply Manifests* marker.

This marker is similar to the `kubectl apply -f <file>` command.

Examples

- Load a manifest YAML for a deployment

```
@pytest.mark.applymanifest('./deployment.yaml')
def test_something(kube):
    ...
```

- Load multiple manifests

```
@pytest.mark.applymanifest('manifests/test/service.yaml')
@pytest.mark.applymanifest('manifests/test/deployment.yaml')
def test_something(kube):
    ...
```

4.3.2 Apply Manifests

Summary

```
@pytest.mark.applymanifests(dir, files=None)
```

`applymanifests` allows you to load Kubernetes manifests from the specified directory and create the resources on the cluster.

Description

Load the YAML manifests from the specified `path` and create the corresponding resources on the cluster. By default all YAML files found in the specified `dir` will be loaded and created. A list of file names can be passed to the `files` parameter, which would limit manifest application to only those YAMLs matching the provided file names in the directory.

Loading manifests via this marker does not prohibit you from manually loading other manifests later in the test case. You can use the *kube* fixture to get object references to the created resources. Manifests loaded via this marker are registered with the internal test case `MetaInfo` and can be waited upon for creation via the *kube* fixture's `wait_until_created` method.

The path to the directory should either be an absolute path, or a path relative from the test file. This marker can be used multiple times on a test case.

When specifying specific files to use from within a directory, or when specifying multiple source directories, the order does not matter. The manifests are loaded, bucketed, and then applied to the cluster in the following order:

- Namespace
- RoleBinding
- ClusterRoleBinding
- Secret
- Service
- ConfigMap
- DeamonSet
- StatefulSet
- Deployment
- Pod

This marker is similar to the `kubectl apply -f <dir>` command.

Examples

- Load manifest YAMLs from a `manifests` directory

```
@pytest.mark.applymanifests('manifests')
def test_something(kube):
    ...
```

- Load specific manifest YAMLs from a `manifests` directory

```
@pytest.mark.applymanifests('manifests', files=[
    'deployment.yaml',
    'service.yml'
])
def test_something(kube):
    ...
```

- Load manifest YAMLs from a `manifests` directory and wait for the registered objects to be ready

```
@pytest.mark.applymanifests('manifests')
def test_something(kube):
    kube.wait_for_registered(timeout=60)
    ...
```

- Load manifests from multiple directories for a single test case

```
@pytest.mark.applymanifests('manifests')
@pytest.mark.applymanifests('common')
def test_something(kube):
    ...
```

4.3.3 Cluster Role Binding

Summary

```
@pytest.mark.clusterrolebinding(name, subject_kind=None, subject_name=None)
```

`clusterrolebinding` creates a `ClusterRoleBinding` resource for the cluster which will exist for the lifespan of the test case. The named cluster role must already exist on the cluster.

Description

Create and use a Kubernetes ClusterRoleBinding for the test case. The generated ClusterRoleBinding will be automatically created and removed for each marked test. The name of the cluster role must be specified and the ClusterRole must already exist. This marker can be used multiple times on a test case.

Optionally, the `subject_kind` (one of: *User*, *Group*, *ServiceAccount*) and `subject_name` can be specified to set a custom target subject for the generated ClusterRoleBinding. If a custom target subject is specified, both `subject_kind` and `subject_name` must be specified. If no custom subject is specified, the generated ClusterRoleBinding will default to all users and all service accounts.

The ClusterRoleBinding created via this marker will always use an `apiGroup` of "rbac.authorization.k8s.io" for both subjects and roleRefs. Generated ClusterRoleBindings will be created with the `kubetest:` prefix.

For more information, see: https://kubernetes.io/docs/reference/access-authn-authz/rbac/

- To see all existing ClusterRoleBindings, use `kubectl get clusterrolebindings`
- To see all existing ClusterRoles, use `kubectl get clusterroles`

Examples

- Use the "cluster-admin" role binding with the default subject

```
@pytest.mark.clusterrolebinding('cluster-admin')
def test_something(kube):
  ...
```

- Use the "cluster-admin" role on a custom target subject

```
@pytest.mark.clusterrolebinding('cluster-admin', subject_kind='ServiceAccount',␣
↪subject_name='custom-acct')
def test_something(kube):
    ...
```

- Set multiple ClusterRoleBindings for the test case

```
@pytest.mark.clusterrolebinding('system:node')
@pytest.mark.clusterrolebinding('system:discovery')
def test_something(kube):
    ...
```

4.3.4 Role Binding

Summary

```
@pytest.mark.rolebinding(kind, name, subject_kind=None, subject_name=None)
```

`rolebinding` creates a `RoleBinding` resource for the cluster which will exist for the lifespan of the test case. The named role must already exist on the cluster.

Description

Create and use a Kubernetes RoleBinding for the test case. The generated RoleBinding will use the generated test case namespace and will be automatically created for each marked test case and removed once each test completes. The role `kind` (one of: *Role*, *ClusterRole*) must be specified along with the `name` of the role. Only existing Roles or ClusterRoles can be used. This marker can be used multiple times on a test case.

Optionally, the `subject_kind` (one of: *User*, *Group*, *ServiceAccount*) and `subject_name` can be specified to set a custom target subject for the generated RoleBinding. If a custom target subject is specified, both `subject_kind` and `subject_name` must be specified. If no custom subject is specified, the generated RoleBinding will default to all users in the namespace and all service accounts.

The RoleBinding created via this marker will always use an `apiGroup` of "rbac.authorization.k8s.io" for both subjects and roleRefs.

For more information, see: https://kubernetes.io/docs/reference/access-authn-authz/rbac/

- To see all existing RoleBindings, use `kubectl get rolebindings`
- To see all existing Roles, use `kubectl get roles`

Examples

- Use a RoleBinding with the default subject

```
@pytest.mark.rolebinding('Role', 'test-role')
def test_something(kube):
    ...
```

- Use a RoleBinding with a custom target subject

```
@pytest.mark.rolebinding('Role', 'test-role', subject_kind='Group', subject_name=
↪'example')
def test_something(kube):
    ...
```

- Set multiple RoleBindings for the test case

```
@pytest.mark.rolebinding('Role', 'test-role')
@pytest.mark.rolebinding('ClusterRole', 'custom-cluster-role')
def test_something(kube):
    ...
```

4.3.5 Namespace

Summary

```
@pytest.mark.namespace(create=True, name=None)
```

`namespace` helps define the way namespaces are handled for each test case, allowing tests to define custom namespaces or use existing ones.

Description

The `namespace` marker exposes options to control how namespaces are handled by kubetest.

By default, a new namespace is created for each test case where the namespace name is generated from the test name and a timestamp to ensure uniqueness. With this marker, this default behavior may be overridden: - Set `create` to *False* to disable namespace creation. - Set `name` to a string to give the namespace a specific name, or set to *None* to use the generated name.

Note: When `create` is *False*, the objects created inside the test and by the `applymanifest`/`applymanifests` markers are not automatically deleted.

Examples

- Do not create a namespace for a given test

```
@pytest.mark.namespace(create=False)
def test_something(kube):
    ...
```

- Do not create a namespace and create all objects in the `existing-ns` namespace

```
@pytest.mark.namespace(create=False, name='existing-ns')
def test_something(kube):
    ...
```

- Create a namespace with a specific name

```
@pytest.mark.namespace(create=True, name='specific-name')
def test_something(kube):
    ...
```

4.4 API Reference

This page contains the full API reference for kubetest.

4.4.1 Client

The test client for managing Kubernetes resources within test cases.

An instance of the `TestClient` defined in this module is automatically created for each test case that uses the `kube` fixture. The `kube` fixture provides the `TestClient` instance to the test case.

TestClient

New in version 0.0.1.

class kubetest.client.**TestClient**(*namespace: str*)

Test client for managing Kubernetes resources for a test case.

The `namespace` for the TestClient will be automatically generated and provided to the TestClient during the test setup process.

Parameters **namespace** – The namespace associated with the test client. Each test case will have its own namespace assigned.

create(*obj: kubetest.objects.api_object.ApiObject*) → None

Create the provided ApiObject on the Kubernetes cluster.

If the object does not already have a namespace assigned to it, the client's generated test case namespace will be used.

Parameters **obj** – A kubetest API Object wrapper.

delete(*obj: kubetest.objects.api_object.ApiObject*, *options: kubernetes.client.models.v1_delete_options.V1DeleteOptions = None*) → None

Delete the provided ApiObject from the Kubernetes cluster.

If the object does not already have a namespace assigned to it, the client's generated test case namespace will be used.

Parameters

- **obj** – A kubetest API Object wrapper.
- **options** – Additional options for deleting the resource from the cluster.

static refresh(*obj: kubetest.objects.api_object.ApiObject*) → None

Refresh the underlying Kubernetes resource status and state.

Parameters **obj** – A kubetest API Object wrapper.

static load_clusterrolebinding(*path: str*, *name: Optional[str] = None*) → kubetest.objects.clusterrolebinding.ClusterRoleBinding

Load a manifest YAML into a ClusterRoleBinding object.

Parameters

- **path** – The path to the ClusterRoleBinding manifest.
- **name** – The name of the resource to load. If the manifest file contains a single object definition for the type being loaded, it is not necessary to specify the name. If the manifest has multiple definitions containing the same type, a name is required to differentiate between them. If no name is specified in such case, an error is raised.

Returns The ClusterRoleBinding for the specified manifest.

load_configmap(*path: str*, *set_namespace: bool = True*, *name: Optional[str] = None*) → kubetest.objects.configmap.ConfigMap

Load a manifest YAML into a ConfigMap object.

By default, this will augment the ConfigMap object with the generated test case namespace. This behavior can be disabled with the `set_namespace` flag.

Parameters

- **path** – The path to the ConfigMap manifest.
- **set_namespace** – Enable/disable the automatic augmentation of the ConfigMap namespace.

- **name** – The name of the resource to load. If the manifest file contains a single object definition for the type being loaded, it is not necessary to specify the name. If the manifest has multiple definitions containing the same type, a name is required to differentiate between them. If no name is specified in such case, an error is raised.

Returns The ConfigMap for the specified manifest.

load_daemonset(*path: str*, *set_namespace: bool = True*, *name: Optional[str] = None*) → kubetest.objects.daemonset.DaemonSet

Load a manifest YAML into a DaemonSet object.

By default, this will augment the DaemonSet object with the generated test case namespace. This behavior can be disabled with the `set_namespace` flag.

Parameters

- **path** – The path to the DaemonSet manifest.
- **set_namespace** – Enable/disable the automatic augmentation of the DaemonSet namespace.
- **name** – The name of the resource to load. If the manifest file contains a single object definition for the type being loaded, it is not necessary to specify the name. If the manifest has multiple definitions containing the same type, a name is required to differentiate between them. If no name is specified in such case, an error is raised.

Returns The DaemonSet for the specified manifest.

load_deployment(*path: str*, *set_namespace: bool = True*, *name: Optional[str] = None*) → kubetest.objects.deployment.Deployment

Load a manifest YAML into a Deployment object.

By default, this will augment the Deployment object with the generated test case namespace. This behavior can be disabled with the `set_namespace` flag.

Parameters

- **path** – The path to the Deployment manifest.
- **set_namespace** – Enable/disable the automatic augmentation of the Deployment namespace.
- **name** – The name of the resource to load. If the manifest file contains a single object definition for the type being loaded, it is not necessary to specify the name. If the manifest has multiple definitions containing the same type, a name is required to differentiate between them. If no name is specified in such case, an error is raised.

Returns The Deployment for the specified manifest.

load_pod(*path: str*, *set_namespace: bool = True*, *name: Optional[str] = None*) → kubetest.objects.pod.Pod

Load a manifest YAML into a Pod object.

By default, this will augment the Pod object with the generated test case namespace. This behavior can be disabled with the `set_namespace` flag.

Parameters

- **path** – The path to the Pod manifest.
- **set_namespace** – Enable/disable the automatic augmentation of the Pod namespace.
- **name** – The name of the resource to load. If the manifest file contains a single object definition for the type being loaded, it is not necessary to specify the name. If the manifest has

multiple definitions containing the same type, a name is required to differentiate between them. If no name is specified in such case, an error is raised.

Returns The Pod for the specified manifest.

load_rolebinding(*path: str*, *set_namespace: bool = True*, *name: Optional[str] = None*) → kubetest.objects.rolebinding.RoleBinding

Load a manifest YAML into a RoleBinding object.

By default, this will augment the RoleBinding object with the generated test case namespace. This behavior can be disabled with the `set_namespace` flag.

Parameters

- **path** – The path to the RoleBinding manifest.
- **set_namespace** – Enable/disable the automatic augmentation of the RoleBinding namespace.
- **name** – The name of the resource to load. If the manifest file contains a single object definition for the type being loaded, it is not necessary to specify the name. If the manifest has multiple definitions containing the same type, a name is required to differentiate between them. If no name is specified in such case, an error is raised.

Returns The RoleBinding for the specified manifest.

load_secret(*path: str*, *set_namespace: bool = True*, *name: Optional[str] = None*) → kubetest.objects.secret.Secret

Load a manifest YAML into a Secret object.

By default, this will augment the Secret object with the generated test case namespace. This behavior can be disabled with the `set_namespace` flag.

Parameters

- **path** – The path to the Secret manifest.
- **set_namespace** – Enable/disable the automatic augmentation of the Secret namespace.
- **name** – The name of the resource to load. If the manifest file contains a single object definition for the type being loaded, it is not necessary to specify the name. If the manifest has multiple definitions containing the same type, a name is required to differentiate between them. If no name is specified in such case, an error is raised.

Returns The Secret for the specified manifest.

load_service(*path: str*, *set_namespace: bool = True*, *name: Optional[str] = None*) → kubetest.objects.service.Service

Load a manifest YAML into a Service object.

By default, this will augment the Service object with the generated test case namespace. This behavior can be disabled with the `set_namespace` flag.

Parameters

- **path** – The path to the Service manifest.
- **set_namespace** – Enable/disable the automatic augmentation of the Service namespace.
- **name** – The name of the resource to load. If the manifest file contains a single object definition for the type being loaded, it is not necessary to specify the name. If the manifest has multiple definitions containing the same type, a name is required to differentiate between them. If no name is specified in such case, an error is raised.

Returns The Service for the specified manifest.

load_persistentvolumeclaim(*path: str*, *set_namespace: bool = True*, *name: Optional[str] = None*) → kubetest.objects.persistentvolumeclaim.PersistentVolumeClaim

Load a manifest YAML into a PersistentVolumeClaim object.

By default, this will augment the PersistentVolumeClaim object with the generated test case namespace. This behavior can be disabled with the `set_namespace` flag.

Parameters

- **path** (*str*) – The path to the PersistentVolumeClaim manifest.
- **set_namespace** (*bool*) – Enable/disable the automatic augmentation of the PersistentVolumeClaim namespace.
- **name** – The name of the resource to load. If the manifest file contains a single object definition for the type being loaded, it is not necessary to specify the name. If the manifest has multiple definitions containing the same type, a name is required to differentiate between them. If no name is specified in such case, an error is raised.

Returns The PersistentVolumeClaim for the specified manifest.

Return type objects.PersistentVolumeClaim

load_ingress(*path: str*, *set_namespace: bool = True*, *name: Optional[str] = None*) → kubetest.objects.ingress.Ingress

Load a manifest YAML into a Ingress object.

By default, this will augment the Ingress object with the generated test case namespace. This behavior can be disabled with the `set_namespace` flag.

Parameters

- **path** (*str*) – The path to the Ingress manifest.
- **set_namespace** (*bool*) – Enable/disable the automatic augmentation of the Ingress namespace.
- **name** – The name of the resource to load. If the manifest file contains a single object definition for the type being loaded, it is not necessary to specify the name. If the manifest has multiple definitions containing the same type, a name is required to differentiate between them. If no name is specified in such case, an error is raised.

Returns The ingress for the specified manifest.

Return type objects.Ingress

load_replicaset(*path: str*, *set_namespace: bool = True*, *name: Optional[str] = None*) → kubetest.objects.replicaset.ReplicaSet

Load a manifest YAML into a ReplicaSet object.

By default, this will augment the ReplicaSet object with the generated test case namespace. This behavior can be disabled with the `set_namespace` flag.

Parameters

- **path** – The path to the ReplicaSet manifest.
- **set_namespace** – Enable/disable the automatic augmentation of the ReplicaSet namespace.
- **name** – The name of the resource to load. If the manifest file contains a single object definition for the type being loaded, it is not necessary to specify the name. If the manifest has

multiple definitions containing the same type, a name is required to differentiate between them. If no name is specified in such case, an error is raised.

Returns The ReplicaSet for the specified manifest.

load_statefulset(*path: str*, *set_namespace: bool = True*, *name: Optional[str] = None*) → kubetest.objects.statefulset.StatefulSet

Load a manifest YAML into a StatefulSet object.

By default, this will augment the StatefulSet object with the generated test case namespace. This behavior can be disabled with the `set_namespace` flag.

Parameters

- **path** – The path to the StatefulSet manifest.
- **set_namespace** – Enable/disable the automatic augmentation of the StatefulSet namespace.
- **name** – The name of the resource to load. If the manifest file contains a single object definition for the type being loaded, it is not necessary to specify the name. If the manifest has multiple definitions containing the same type, a name is required to differentiate between them. If no name is specified in such case, an error is raised.

Returns The StatefulSet for the specified manifest.

load_serviceaccount(*path: str*, *set_namespace: bool = True*, *name: Optional[str] = None*) → kubetest.objects.serviceaccount.ServiceAccount

Load a manifest YAML into a ServiceAccount object.

By default, this will augment the ServiceAccount object with the generated test case namespace. This behavior can be disabled with the `set_namespace` flag.

Parameters

- **path** – The path to the ServiceAccount manifest.
- **set_namespace** – Enable/disable the automatic augmentation of the ServiceAccount namespace.
- **name** – The name of the resource to load. If the manifest file contains a single object definition for the type being loaded, it is not necessary to specify the name. If the manifest has multiple definitions containing the same type, a name is required to differentiate between them. If no name is specified in such case, an error is raised.

Returns The ServiceAccount for the specified manifest.

get_configmaps(*namespace: str = None*, *fields: Dict[str, str] = None*, *labels: Dict[str, str] = None*) → Dict[str, kubetest.objects.configmap.ConfigMap]

Get ConfigMaps from the cluster.

Parameters

- **namespace** – The namespace to get the ConfigMaps from. If not specified, it will use the auto-generated test case namespace by default.
- **fields** – A dictionary of fields used to restrict the returned collection of ConfigMaps to only those which match these field selectors. By default, no restricting is done.
- **labels** – A dictionary of labels used to restrict the returned collection of ConfigMaps to only those which match these label selectors. By default, no restricting is done.

Returns A dictionary where the key is the ConfigMap name and the value is the ConfigMap itself.

get_daemonsets(*namespace: str = None*, *fields: Dict[str, str] = None*, *labels: Dict[str, str] = None*) → Dict[str, kubetest.objects.daemonset.DaemonSet]

Get DaemonSets from the cluster.

Parameters

- **namespace** – The namespace to get the DaemonSets from. If not specified, it will use the auto-generated test case namespace by default.
- **fields** – A dictionary of fields used to restrict the returned collection of DaemonSets to only those which match these field selectors. By default, no restricting is done.
- **labels** – A dictionary of labels used to restrict the returned collection of DaemonSets to only those which match these label selectors. By default, no restricting is done.

Returns A dictionary where the key is the DaemonSet name and the value is the DaemonSet itself.

get_deployments(*namespace: str = None*, *fields: Dict[str, str] = None*, *labels: Dict[str, str] = None*) → Dict[str, kubetest.objects.deployment.Deployment]

Get Deployments from the cluster.

Parameters

- **namespace** – The namespace to get the Deployments from. If not specified, it will use the auto-generated test case namespace by default.
- **fields** – A dictionary of fields used to restrict the returned collection of Deployments to only those which match these field selectors. By default, no restricting is done.
- **labels** – A dictionary of labels used to restrict the returned collection of Deployments to only those which match these label selectors. By default, no restricting is done.

Returns A dictionary where the key is the Deployment name and the value is the Deployment itself.

get_endpoints(*namespace: str = None*, *fields: Dict[str, str] = None*, *labels: Dict[str, str] = None*) → Dict[str, kubetest.objects.endpoints.Endpoints]

Get Endpoints from the cluster.

Parameters

- **namespace** – The namespace to get the Endpoints from. If not specified, it will use the auto-generated test case namespace by default.
- **fields** – A dictionary of fields used to restrict the returned collection of Endpoints to only those which match these field selectors. By default, no restricting is done.
- **labels** – A dictionary of labels used to restrict the returned collection of Endpoints to only those which match these label selectors. By default, no restricting is done.

Returns A dictionary where the key is the Endpoint name and the value is the Endpoint itself.

get_events(*fields: Dict[str, str] = None*, *labels: Dict[str, str] = None*, *all_namespaces: bool = False*) → Dict[str, kubetest.objects.event.Event]

Get the latest Events that occurred in the cluster.

Parameters

- **fields** – A dictionary of fields used to restrict the returned collection of Events to only those which match these field selectors. By default, no restricting is done.
- **labels** – A dictionary of labels used to restrict the returned collection of Events to only those which match these label selectors. By default, no restricting is done.
- **all_namespaces** – If True, get the events across all namespaces.

Returns A dictionary where the key is the Event name and the value is the Event itself.

get_namespaces(*fields: Dict[str, str] = None*, *labels: Dict[str, str] = None*) → Dict[str, kubetest.objects.namespace.Namespace]

Get Namespaces from the cluster.

Parameters

- **fields** – A dictionary of fields used to restrict the returned collection of Namespaces to only those which match these field selectors. By default, no restricting is done.
- **labels** – A dictionary of labels used to restrict the returned collection of Namespaces to only those which match these label selectors. By default, no restricting is done.

Returns A dictionary where the key is the Namespace name and the value is the Namespace itself.

static get_nodes(*fields: Dict[str, str] = None*, *labels: Dict[str, str] = None*) → Dict[str, kubetest.objects.node.Node]

Get the Nodes that make up the cluster.

Parameters

- **fields** – A dictionary of fields used to restrict the returned collection of Nodes to only those which match these field selectors. By default, no restricting is done.
- **labels** – A dictionary of labels used to restrict the returned collection of Nodes to only those which match these label selectors. By default, no restricting is done.

Returns A dictionary where the key is the Node name and the value is the Node itself.

get_pods(*namespace: str = None*, *fields: Dict[str, str] = None*, *labels: Dict[str, str] = None*) → Dict[str, kubetest.objects.pod.Pod]

Get Pods from the cluster.

Parameters

- **namespace** – The namespace to get the Pods from. If not specified, it will use the auto-generated test case namespace by default.
- **fields** – A dictionary of fields used to restrict the returned collection of Pods to only those which match these field selectors. By default, no restricting is done.
- **labels** – A dictionary of labels used to restrict the returned collection of Pods to only those which match these label selectors. By default, no restricting is done.

Returns A dictionary where the key is the Pod name and the value is the Pod itself.

get_secrets(*namespace: str = None*, *fields: Dict[str, str] = None*, *labels: Dict[str, str] = None*) → Dict[str, kubetest.objects.secret.Secret]

Get Secrets from the cluster.

Parameters

- **namespace** – The namespace to get the Secrets from. If not specified, it will use the auto-generated test case namespace by default.
- **fields** – A dictionary of fields used to restrict the returned collection of Secrets to only those which match these field selectors. By default, no restricting is done.
- **labels** – A dictionary of labels used to restrict the returned collection of Secrets to only those which match these label selectors. By default, no restricting is done.

Returns A dictionary where the key is the Secret name and the value is the Secret itself.

get_services(*namespace: str = None*, *fields: Dict[str, str] = None*, *labels: Dict[str, str] = None*) → Dict[str, kubetest.objects.service.Service]

Get Services under the test case namespace.

Parameters

- **namespace** – The namespace to get the Services from. If not specified, it will use the auto-generated test case namespace by default.
- **fields** – A dictionary of fields used to restrict the returned collection of Services to only those which match these field selectors. By default, no restricting is done.
- **labels** – A dictionary of labels used to restrict the returned collection of Services to only those which match these label selectors. By default, no restricting is done.

Returns A dictionary where the key is the Service name and the value is the Service itself.

get_persistentvolumeclaims(*namespace: str = None*, *fields: Dict[str, str] = None*, *labels: Dict[str, str] = None*) → Dict[str, kubetest.objects.persistentvolumeclaim.PersistentVolumeClaim]

Get PersistentVolumeClaims from the cluster.

Parameters

- **namespace** – The namespace to get the PersistentVolumeClaim from. If not specified, it will use the auto-generated test case namespace by default.
- **fields** – A dictionary of fields used to restrict the returned collection of PersistentVolumeClaim to only those which match these field selectors. By default, no restricting is done.
- **labels** – A dictionary of labels used to restrict the returned collection of PersistentVolumeClaim to only those which match these label selectors. By default, no restricting is done.

Returns A dictionary where the key is the PersistentVolumeClaim name and the value is the PersistentVolumeClaim itself.

get_ingresses(*namespace: str = None*, *fields: Dict[str, str] = None*, *labels: Dict[str, str] = None*) → Dict[str, kubetest.objects.ingress.Ingress]

Get Ingresses from the cluster.

Parameters

- **namespace** – The namespace to get the Ingress from. If not specified, it will use the auto-generated test case namespace by default.
- **fields** – A dictionary of fields used to restrict the returned collection of Ingress to only those which match these field selectors. By default, no restricting is done.
- **labels** – A dictionary of labels used to restrict the returned collection of Ingress to only those which match these label selectors. By default, no restricting is done.

Returns A dictionary where the key is the Ingress name and the value is the Ingress itself.

get_replicasets(*namespace: str = None*, *fields: Dict[str, str] = None*, *labels: Dict[str, str] = None*) → Dict[str, kubetest.objects.replicaset.ReplicaSet]

Get ReplicaSets from the cluster.

Parameters

- **namespace** – The namespace to get the ReplicaSets from. If not specified, it will use the auto-generated test case namespace by default.

- **fields** – A dictionary of fields used to restrict the returned collection of ReplicaSets to only those which match these field selectors. By default, no restricting is done.
- **labels** – A dictionary of labels used to restrict the returned collection of ReplicaSets to only those which match these label selectors. By default, no restricting is done.

Returns A dictionary where the key is the ReplicaSet name and the value is the ReplicaSet itself.

get_statefulsets(*namespace: str = None*, *fields: Dict[str, str] = None*, *labels: Dict[str, str] = None*) → Dict[str, kubetest.objects.statefulset.StatefulSet]

Get StatefulSets from the cluster.

Parameters

- **namespace** – The namespace to get the StatefulSets from. If not specified, it will use the auto-generated test case namespace by default.
- **fields** – A dictionary of fields used to restrict the returned collection of StatefulSets to only those which match these field selectors. By default, no restricting is done.
- **labels** – A dictionary of labels used to restrict the returned collection of StatefulSets to only those which match these label selectors. By default, no restricting is done.

Returns A dictionary where the key is the StatefulSet name and the value is the StatefulSet itself.

get_serviceaccounts(*namespace: str = None*, *fields: Dict[str, str] = None*, *labels: Dict[str, str] = None*) → Dict[str, kubetest.objects.serviceaccount.ServiceAccount]

Get ServiceAccounts from the cluster.

Parameters

- **namespace** – The namespace to get the ServiceAccount from. If not specified, it will use the auto-generated test case namespace by default.
- **fields** – A dictionary of fields used to restrict the returned collection of ServiceAccount to only those which match these field selectors. By default, no restricting is done.
- **labels** – A dictionary of labels used to restrict the returned collection of ServiceAccount to only those which match these label selectors. By default, no restricting is done.

Returns A dictionary where the key is the ServiceAccount name and the value is the ServiceAccount itself.

static wait_for_conditions(**args*, *timeout: int = None*, *interval: Union[float, int] = 1*, *policy: kubetest.condition.Policy = <Policy.ONCE: 1>*, *fail_on_api_error: bool = True*) → None

Wait for all of the provided Conditions to be met.

All Conditions must be met for this to unblock. If no Conditions are provided, this method will do nothing.

Parameters

- ***args** – Conditions to check.
- **timeout** – The maximum time to wait, in seconds, for the provided Conditions to be met. If all of the Conditions are not met within the given timeout, this will raise a TimeoutError. By default, there is no timeout so this will wait indefinitely.
- **interval** – The time, in seconds, to sleep before re-evaluating the conditions. Default: 1s
- **policy** – The condition checking policy that defines the checking behavior. Default: ONCE

- **fail_on_api_error** – Fail the condition checks if a Kubernetes API error is incurred. An API error can be raised for a number of reasons, including a Pod being restarted and temporarily unavailable. Disabling this will cause those errors to be ignored, allowing the check to continue until timeout or resolution. (default: True).

Raises

- `TimeoutError` – The Conditions were not met within the specified timeout period.
- `ValueError` – Not all arguments are a Condition.

wait_for_ready_nodes(*count: int*, *timeout: int = None*, *interval: Union[int, float] = 1*) → None
Wait until there are at least `count` number of nodes available in the cluster.

Notes

This should only be used for clusters that auto-scale the nodes. This will not create/delete nodes on its own.

Parameters

- **count** – The number of nodes to wait for.
- **timeout** – The maximum time to wait, in seconds.
- **interval** – The time, in seconds, to sleep before re-checking the number of nodes.

wait_for_registered(*timeout: int = None*, *interval: Union[int, float] = 1*) → None
Wait for all of the pre-registered objects to be ready on the cluster.

An object is pre-registered with the test client if it is specified to the test via the `applymanifests` pytest marker. The marker will load the manifest and add the object to the cluster, and register it with the test client. This method waits until all such loaded manifest objects are in the ready state simultaneously.

Parameters

- **timeout** – The maximum time to wait, in seconds.
- **interval** – The time, in seconds, to sleep before re-checking the ready state for pre-registered objects.

static wait_until_created(*obj: kubetest.objects.api_object.ApiObject*, *timeout: int = None*, *interval: Union[int, float] = 1*) → None
Wait until the specified object has been created.

Here, creation is judged on whether or not refreshing the object (e.g. getting it) returns an object (created) or an error (not yet created).

Parameters

- **obj** – The ApiObject to wait on.
- **timeout** – The maximum time to wait, in seconds.
- **interval** – The time, in seconds, to sleep before re-checking the created state of the object.

4.4.2 Objects

Kubetest wrappers around Kubernetes API Objects.

ApiObject

New in version 0.0.1.

class kubetest.objects.**ApiObject**(*api_object*)

ApiObject is the base class for many of the kubetest objects which wrap Kubernetes API objects.

This base class provides common functionality and common object properties for all API wrappers. It also defines the following abstract methods which all subclasses must implement:

- `create`: create the resource on the cluster
- `delete`: remove the resource from the cluster
- `refresh`: refresh the underlying object model
- `is_ready`: check if the object is in the ready state

Parameters **api_object** – The underlying Kubernetes API object.

Variables **obj** – The underlying Kubernetes API object.

obj_type = None

The default Kubernetes API object type for the class. Each subclass should define its own `obj_type`.

api_clients = None

A mapping of all the supported api clients for the API object type. Various resources can have multiple versions, e.g. "apps/v1", "apps/v1beta1", etc. The preferred version for each resource type should be defined under the "preferred" key. The preferred API client will be used when the apiVersion is not specified for the resource.

version

The API version of the Kubernetes object (*obj.apiVersion*').

name

The name of the Kubernetes object (`obj.metadata.name`).

namespace

The namespace of the Kubernetes object (`obj.metadata.namespace`).

api_client

The API client for the Kubernetes object. This is determined by the `apiVersion` of the object configuration.

Raises `ValueError` – The API version is not supported.

classmethod preferred_client()

The preferred API client for the Kubernetes object. This is defined in the `api_clients` class member dict for each object.

Raises `ValueError` – No preferred client is defined for the object.

wait_until_ready(*timeout: int = None*, *interval: Union[int, float] = 1*, *fail_on_api_error: bool = False*) → None

Wait until the resource is in the ready state.

Parameters

- **timeout** – The maximum time to wait, in seconds, for the resource to reach the ready state. If unspecified, this will wait indefinitely. If specified and the timeout is met or exceeded, a TimeoutError will be raised.
- **interval** – The time, in seconds, to wait before re-checking if the object is ready.

- **fail_on_api_error** – Fail if an API error is raised. An API error can be raised for a number of reasons, such as 'resource not found', which could be the case when a resource is just being started or restarted. When waiting for readiness we generally do not want to fail on these conditions.

Raises `TimeoutError` – The specified timeout was exceeded.

wait_until_deleted(*timeout: int = None*, *interval: Union[int, float] = 1*) → None

Wait until the resource is deleted from the cluster.

Parameters

- **timeout** – The maximum time to wait, in seconds, for the resource to be deleted from the cluster. If unspecified, this will wait indefinitely. If specified and the timeout is met or exceeded, a TimeoutError will be raised.
- **interval** – The time, in seconds, to wait before re-checking if the object has been deleted.

Raises `TimeoutError` – The specified timeout was exceeded.

classmethod load(*path: str*, *name: Optional[str] = None*) → kubetest.objects.api_object.ApiObject

Load the Kubernetes resource from file.

Generally, this is used to load the Kubernetes manifest files and parse them into their appropriate API Object type.

Parameters

- **path** – The path to the YAML config file (manifest) containing the configuration for the resource.
- **name** – The name of the resource to load. If the manifest file contains a single object definition for the type being loaded, it is not necessary to specify the name. If the manifest has multiple definitions containing the same type, a name is required to differentiate between them. If no name is specified in such case, an error is raised.

Returns The API object wrapper corresponding to the configuration loaded from manifest YAML file.

Raises

- `ValueError` – Multiple objects of the desired type were found in
- the manifest file and no name was specified to differentiate between
- `them.`

create(*namespace: str = None*) → None

Create the underlying Kubernetes resource in the cluster under the given namespace.

Parameters **namespace** – The namespace to create the resource under. If no namespace is provided, it will use the instance's namespace member, which is set when the object is created via the kubetest client.

delete(*options: kubernetes.client.models.v1_delete_options.V1DeleteOptions*) → kubernetes.client.models.v1_status.V1Status

Delete the underlying Kubernetes resource from the cluster.

This method expects the resource to have been loaded or otherwise assigned a namespace already. If it has not, the namespace will need to be set manually.

Parameters **options** – Options for resource deletion.

refresh() → None
Refresh the local state (obj) of the underlying Kubernetes resource.

is_ready() → bool
Check if the resource is in the ready state.

It is up to the wrapper subclass to define what "ready" means for that particular resource.

Returns True if in the ready state; False otherwise.

ClusterRoleBinding

New in version 0.0.1.

class kubetest.objects.**ClusterRoleBinding**(*api_object*)
Kubetest wrapper around a Kubernetes ClusterRoleBinding API Object.

The actual kubernetes.client.V1ClusterRoleBinding instance that this wraps can be accessed via the obj instance member.

This wrapper provides some convenient functionality around the API Object and provides some state management for the ClusterRoleBinding.

obj_type
alias of kubernetes.client.models.v1_cluster_role_binding.V1ClusterRoleBinding

create(*namespace: str = None*) → None
Create the ClusterRoleBinding under the given namespace.

Parameters **namespace** – This argument is ignored for ClusterRoleBindings.

delete(*options: kubernetes.client.models.v1_delete_options.V1DeleteOptions = None*) → kubernetes.client.models.v1_status.V1Status
Delete the ClusterRoleBinding.

This method expects the ClusterRoleBinding to have been loaded or otherwise assigned a namespace already. If it has not, the namespace will need to be set manually.

Parameters **options** – Options for ClusterRoleBinding deletion.

Returns The status of the delete operation.

refresh() → None
Refresh the underlying Kubernetes ClusterRoleBinding resource.

is_ready() → bool
Check if the ClusterRoleBinding is in the ready state.

ClusterRoleBindings do not have a "status" field to check, so we will measure their readiness status by whether or not they exist on the cluster.

Returns True if in the ready state; False otherwise.

ConfigMap

New in version 0.0.1.

class kubetest.objects.**ConfigMap**(*api_object*)
Kubetest wrapper around a Kubernetes ConfigMap API Object.

The actual `kubernetes.client.V1ConfigMap` instance that this wraps can be accessed via the `obj` instance member.

This wrapper provides some convenient functionality around the API Object and provides some state management for the ConfigMap.

obj_type
alias of `kubernetes.client.models.v1_config_map.V1ConfigMap`

create(*namespace: str = None*) → None
Create the ConfigMap under the given namespace.

Parameters **namespace** – The namespace to create the ConfigMap under. If the ConfigMap was loaded via the kubetest client, the namespace will already be set, so it is not needed here. Otherwise, the namespace will need to be provided.

delete(*options: kubernetes.client.models.v1_delete_options.V1DeleteOptions = None*) → kubernetes.client.models.v1_status.V1Status
Delete the ConfigMap.

This method expects the ConfigMap to have been loaded or otherwise assigned a namespace already. If it has not, the namespace will need to be set manually.

Parameters **options** – Options for ConfigMap deletion.

Returns The status of the delete operation.

refresh() → None
Refresh the underlying Kubernetes ConfigMap resource.

is_ready() → bool
Check if the ConfigMap is in the ready state.

ConfigMaps do not have a "status" field to check, so we will measure their readiness status by whether or not they exist on the cluster.

Returns True if in the ready state; False otherwise.

Container

New in version 0.0.1.

class kubetest.objects.**Container**(*api_object*, *pod*)
Kubetest wrapper around a Kubernetes Container API Object.

The actual `kubernetes.client.V1Container` instance that this wraps can be accessed via the `obj` instance member.

This wrapper provides some convenient functionality around the API Object and provides some state management for the Container.

This wrapper does **NOT** subclass the `objects.ApiObject` like other object wrappers because it is not intended to be created or managed from manifest file. It is merely meant to wrap the Container spec for a Pod to make Container-targeted actions easier.

get_restart_count() → int
Get the number of times the Container has been restarted.

Returns The number of times the Container has been restarted.

get_logs() → str
Get all the logs for the Container.

Returns The Container logs.

search_logs(**keyword*) → bool
Search for keywords/phrases in the Container's logs.

Parameters ***keyword** – Keywords to search for within the logs.

Returns True if found; False otherwise.

Deployment

New in version 0.0.1.

class kubetest.objects.**Deployment**(**args*, ***kwargs*)
Kubetest wrapper around a Kubernetes Deployment API Object.

The actual `kubernetes.client.V1Deployment` instance that this wraps can be accessed via the `obj` instance member.

This wrapper provides some convenient functionality around the API Object and provides some state management for the Deployment.

obj_type
alias of `kubernetes.client.models.v1_deployment.V1Deployment`

create(*namespace: str = None*) → None
Create the Deployment under the given namespace.

Parameters **namespace** – The namespace to create the Deployment under. If the Deployment was loaded via the kubetest client, the namespace will already be set, so it is not needed here. Otherwise, the namespace will need to be provided.

delete(*options: kubernetes.client.models.v1_delete_options.V1DeleteOptions = None*) → kubernetes.client.models.v1_status.V1Status
Delete the Deployment.

This method expects the Deployment to have been loaded or otherwise assigned a namespace already. If it has not, the namespace will need to be set manually.

Parameters **options** – Options for Deployment deletion.

Returns The status of the delete operation.

refresh() → None
Refresh the underlying Kubernetes Deployment resource.

is_ready() → bool
Check if the Deployment is in the ready state.

Returns True if in the ready state; False otherwise.

status() → kubernetes.client.models.v1_deployment_status.V1DeploymentStatus
Get the status of the Deployment.

Returns The status of the Deployment.

get_pods() → List[kubetest.objects.pod.Pod]
Get the pods for the Deployment.

Returns A list of pods that belong to the deployment.

Namespace

New in version 0.0.1.

class kubetest.objects.**Namespace**(*api_object*)
Kubetest wrapper around a Kubernetes Namespace API Object.

The actual `kubernetes.client.V1Namespace` instance that this wraps can be accessed via the `obj` instance member.

This wrapper provides some convenient functionality around the API Object and provides some state management for the Namespace.

obj_type
alias of `kubernetes.client.models.v1_namespace.V1Namespace`

classmethod new(*name: str*) → kubetest.objects.namespace.Namespace
Create a new Namespace with object backing.

Parameters **name** – The name of the new Namespace.

Returns A new Namespace instance.

create(*name: str = None*) → None
Create the Namespace under the given name.

Parameters **name** – The name to create the Namespace under. If the name is not provided, it will be assumed to already be in the underlying object spec. If it is not, namespace operations will fail.

delete(*options: kubernetes.client.models.v1_delete_options.V1DeleteOptions = None*) → kubernetes.client.models.v1_status.V1Status
Delete the Namespace.

Parameters **options** – Options for Namespace deletion.

Returns The status of the delete operation.

refresh() → None
Refresh the underlying Kubernetes Namespace resource.

is_ready() → bool
Check if the Namespace is in the ready state.

Returns True if in the ready state; False otherwise.

Node

New in version 0.0.1.

class kubetest.objects.**Node**(*api_object*)
Kubetest wrapper around a Kubernetes Node API Object.

The actual `kubernetes.client.V1Node` instance that this wraps can be accessed via the `obj` instance member.

This wrapper provides some convenient functionality around the API Object and provides some state management for the Node.

This wrapper does **NOT** subclass the `objects.ApiObject` like other object wrappers because it is not intended to be created or managed from manifest file. It is merely meant to wrap the Node spec to make Node-based interactions easier

refresh() → None
Refresh the underlying Kubernetes Node resource.

status() → kubernetes.client.models.v1_node_status.V1NodeStatus
Get the status of the Node.

Returns The status of the Node.

is_ready() → bool
Check whether the Node is in the ready state.

Returns True if in the ready state; False otherwise.

Pod

New in version 0.0.1.

class kubetest.objects.**Pod**(*api_object*)
Kubetest wrapper around a Kubernetes Pod API Object.

The actual `kubernetes.client.V1Pod` instance that this wraps can be accessed via the `obj` instance member.

This wrapper provides some convenient functionality around the API Object and provides some state management for the Pod.

obj_type
alias of `kubernetes.client.models.v1_pod.V1Pod`

create(*namespace: str = None*) → None
Create the Pod under the given namespace.

Parameters **namespace** – The namespace to create the Pod under. If the Pod was loaded via the kubetest client, the namespace will already be set, so it is not needed here. Otherwise, the namespace will need to be provided.

delete(*options: kubernetes.client.models.v1_delete_options.V1DeleteOptions = None*) → kubernetes.client.models.v1_status.V1Status
Delete the Pod.

This method expects the Pod to have been loaded or otherwise assigned a namespace already. If it has not, the namespace will need to be set manually.

Parameters **options** – Options for Pod deletion.

Returns The status of the delete operation.

refresh() → None
Refresh the underlying Kubernetes Pod resource.

is_ready() → bool
Check if the Pod is in the ready state.

Returns True if in the ready state; False otherwise.

status() → kubernetes.client.models.v1_pod_status.V1PodStatus
Get the status of the Pod.

Returns The status of the Pod.

get_containers() → List[kubetest.objects.container.Container]
Get the Pod's containers.

Returns A list of containers that belong to the Pod.

get_container(*name: str*) → Optional[kubetest.objects.container.Container]
Get a container in the Pod by name.

Parameters **name** (*str*) – The name of the Container.

Returns The Pod's Container with the matching name. If no container with the given name is found, `None` is returned.

Return type *Container*

get_restart_count() → int
Get the total number of Container restarts for the Pod.

Returns The total number of Container restarts.

http_proxy_get(*path: str*, *query_params: Dict[str, str] = None*) → kubetest.response.Response
Issue a GET request to a proxy for the Pod.

Notes

This function does not use the kubernetes `connect_get_namespaced_pod_proxy_with_path` function because there appears to be lack of support for custom query parameters (as of the `kubernetes==9.0.0` package version). To bypass this, parts of the core functionality from the aforementioned function are used here with the modification of allowing user-defined query parameters to be passed along.

Parameters

- **path** – The URI path for the request.
- **query_params** – Any query parameters for the request.

Returns The response data.

http_proxy_post(*path: str*, *query_params: Dict[str, str] = None*, *data=None*) → kubetest.response.Response
Issue a POST request to a proxy for the Pod.

Notes

This function does not use the kubernetes `connect_post_namespaced_pod_proxy_with_path` function because there appears to be lack of support for custom query parameters (as of the `kubernetes==9.0.0` package version). To bypass this, parts of the core functionality from the aforementioned function are used here with the modification of allowing user-defined query parameters to be passed along.

Parameters

- **path** – The URI path for the request.
- **query_params** – Any query parameters for the request.
- **data** – The data to POST.

Returns The response data.

containers_started() → bool
Check if the Pod's Containers have all started.

Returns True if all Containers have started; False otherwise.

wait_until_containers_start (*timeout: int = None*) → None
Wait until all containers in the Pod have started.

This will wait for the images to be pulled and for the containers to be created and started. This will unblock once all Pod containers have been started.

This is different than waiting until ready, since a container may not be ready immediately after it has been started.

Parameters **timeout** – The maximum time to wait, in seconds, for the Pod's containers to be started. If unspecified, this will wait indefinitely. If specified and the timeout is met or exceeded, a TimeoutError will be raised.

Raises TimeoutError – The specified timeout was exceeded.

RoleBinding

New in version 0.0.1.

class kubetest.objects.**RoleBinding**(*api_object*)
Kubetest wrapper around a Kubernetes RoleBinding API Object.

The actual kubernetes.client.V1RoleBinding instance that this wraps can be accessed via the obj instance member.

This wrapper provides some convenient functionality around the API Object and provides some state management for the RoleBinding.

obj_type
alias of kubernetes.client.models.v1_role_binding.V1RoleBinding

create(*namespace: str = None*) → None
Create the RoleBinding under the given namespace.

Parameters **namespace** – The namespace to create the RoleBinding under. If the RoleBinding was loaded via the kubetest client, the namespace will already be set, so it is not needed here. Otherwise, the namespace will need to be provided.

delete(*options: kubernetes.client.models.v1_delete_options.V1DeleteOptions = None*) → kubernetes.client.models.v1_status.V1Status
Delete the RoleBinding.

This method expects the RoleBinding to have been loaded or otherwise assigned a namespace already. If it has not, the namespace will need to be set manually.

Parameters **options** – Options for RoleBinding deletion.

Returns The status of the delete operation.

refresh() → None
Refresh the underlying Kubernetes RoleBinding resource.

is_ready() → bool
Check if the RoleBinding is in the ready state.

RoleBindings do not have a "status" field to check, so we will measure their readiness status by whether or not they exist on the cluster.

Returns True if in the ready state; False otherwise.

Secret

New in version 0.0.1.

class kubetest.objects.**Secret**(*api_object*)
Kubetest wrapper around a Kubernetes Secret API Object.

The actual `kubernetes.client.V1Secret` instance that this wraps can be accessed via the `obj` instance member.

This wrapper provides some convenient functionality around the API Object and provides some state management for the Secret.

obj_type
alias of `kubernetes.client.models.v1_secret.V1Secret`

create(*namespace: str = None*) → None
Create the Secret under the given namespace.

Parameters **namespace** – The namespace to create the Secret under. If the Secret was loaded via the kubetest client, the namespace will already be set, so it is not needed here. Otherwise, the namespace will need to be provided.

delete(*options: kubernetes.client.models.v1_delete_options.V1DeleteOptions = None*) → kubernetes.client.models.v1_status.V1Status
Delete the Secret.

This method expects the Secret to have been loaded or otherwise assigned a namespace already. If it has not, the namespace will need to be set manually.

Parameters **options** – Options for Secret deletion.

Returns The status of the delete operation.

refresh() → None
Refresh the underlying Kubernetes Secret resource.

is_ready() → bool
Check if the Secret is in the ready state.

Secrets do not have a "status" field to check, so we will measure their readiness status by whether or not they exist on the cluster.

Returns True if in the ready state; False otherwise.

Service

New in version 0.0.1.

class kubetest.objects.**Service**(*api_object*)
Kubetest wrapper around a Kubernetes Service API Object.

The actual `kubernetes.client.V1Service` instance that this wraps can be accessed via the `obj` instance member.

This wrapper provides some convenient functionality around the API Object and provides some state management for the Service.

obj_type
alias of `kubernetes.client.models.v1_service.V1Service`

create(*namespace: str = None*) → None
Create the Service under the given namespace.

Parameters **namespace** – The namespace to create the Service under. If the Service was loaded via the kubetest client, the namespace will already be set, so it is not needed here. Otherwise, the namespace will need to be provided.

delete(*options: kubernetes.client.models.v1_delete_options.V1DeleteOptions = None*) → kubernetes.client.models.v1_status.V1Status
Delete the Service.

This method expects the Service to have been loaded or otherwise assigned a namespace already. If it has not, the namespace will need to be set manually.

Parameters **options** – Options for Service deletion.

Returns The status of the delete operation.

refresh() → None
Refresh the underlying Kubernetes Service resource.

is_ready() → bool
Check if the Service is in the ready state.

The readiness state is not clearly available from the Service status, so to see whether or not the Service is ready this will check whether the endpoints of the Service are ready.

This comes with the caveat that in order for a Service to have endpoints, there needs to be some backend hooked up to it. If there is no backend, the Service will never have endpoints, so this will never resolve to True.

Returns True if in the ready state; False otherwise.

status() → kubernetes.client.models.v1_service_status.V1ServiceStatus
Get the status of the Service.

Returns The status of the Service.

get_endpoints() → List[kubernetes.client.models.v1_endpoints.V1Endpoints]
Get the endpoints for the Service.

This can be useful for checking internal IP addresses used in containers, e.g. for container auto-discovery.

Returns A list of endpoints associated with the Service.

proxy_http_get(*path: str*, ***kwargs*) → tuple
Issue a GET request to proxy of a Service.

Parameters

- **path** – The URI path for the request.
- **kwargs** – Keyword arguments for the proxy_http_get function.

Returns The response data

proxy_http_post(*path: str*, ***kwargs*) → tuple
Issue a POST request to proxy of a Service.

Parameters

- **path** – The URI path for the request.
- **kwargs** – Keyword arguments for the proxy_http_post function.

Returns The response data

4.4.3 Conditions

Define test conditions for kubetest.

Policy

New in version 0.0.1.

class kubetest.condition.**Policy**
Condition checking policies.

A Policy defines the behavior of how Conditions are checked.

- **ONCE**: A condition only needs to be met once and the check will consider it met regardless of the state of any other conditions that may be checked alongside it. This is the default behavior.
- **SIMULTANEOUS**: A condition needs to be met simultaneous to all other conditions that are being checked alongside it for the check to be successful.

Condition

New in version 0.0.1.

class kubetest.condition.**Condition**(*name: str*, *fn: Callable*, **args*, ***kwargs*)
A Condition is a convenience wrapper around a function and its arguments which allows the function to be called at a later time.

The function is called in the `check` method, which resolves the result to a boolean value, thus the condition function should return a boolean or something that ultimately resolves to a Truthy or Falsey value.

Parameters

- **name** – The name of the condition to make it easier to identify.
- **fn** – The condition function that will be checked.
- ***args** – Any arguments for the condition function.
- ****kwargs** – Any keyword arguments for the condition function.

Variables

- ***name*** (*str*) – The name of the Condition.
- **fn** (*callable*) – The condition function that will be checked.
- **args** (*tuple*) – Arguments for the checking function.
- **kwargs** (*dict*) – Keyword arguments for the checking function.
- **last_check** (*bool*) – Holds the state of the last condition check.

Raises `ValueError` – The given `fn` is not callable.

check() → bool
Check that the condition was met.

Returns True if the condition was met; False otherwise.

Helpers

kubetest.condition.**check_all**(**args*) → bool
Check all the given Conditions.

Parameters ***args** – The Conditions to check.

Returns True if all checks pass; False otherwise.

kubetest.condition.**check_and_sort**(**args*) → Tuple[List[kubetest.condition.Condition], List[kubetest.condition.Condition]]
Check all the given Conditions and sort them into 'met' and 'unmet' buckets.

Parameters ***args** – The Conditions to check.

Returns The met and unmet condition buckets (in that order).

4.5 Writing Tests

kubetest is designed to interface with a Kubernetes cluster, so before you begin writing tests with kubetest, be sure that you have access to a cluster, whether on Google Container Engine, via minikube, or through your own custom cluster. Generally, where the cluster runs shouldn't be an issue, as long as you can access it from wherever the tests are being run.

4.5.1 Cluster Configuration

By default, kubetest will look for a config file at `~/.kube/config` and the current context – this is the same behavior that `kubectl` utilizes for the resolving cluster config. Generally, if you can reach your cluster via. `kubectl`, you should be able to use it with kubetest.

If you wish to specify a different config file and/or context, you can pass it in via the `--kube-config` and `--kube-context` flags. See *Command Line Usage* for more details.

You can also write a `kubeconfig` fixture which provides the path to the config file and/or a `kubecontext` fixture which provides the name of the context to be used. This may be useful in case your cluster is generated as part of the tests or you wish to use specific contexts in different parts of the suite.

```
import pytest
import subprocess
from typing import Optional


@pytest.fixture
def kubeconfig() -> str:
    # Here, Terraform creates a cluster and outputs a kubeconfig
    # at somepath
    subprocess.check_call(['terraform', 'apply'])
    return 'somepath/kubeconfig'


@pytest.fixture
def kubecontext() -> Optional[str]:
    # Return None to use the current context as set in the kubeconfig
    # Or return the name of a specific context in the kubeconfig
    return 'kubetest-cluster'
```

(continues on next page)

(continued from previous page)

```
def test_my_terraformed_cluster(kube):
    # Use your cluster!
    pass
```

4.5.2 Loading Manifests

It is recommended, though not required, to test against pre-defined manifest files. These files can be kept anywhere relative to your tests and can be organized however you like. Each test can have its own directory of manifests, or you can pick and choose individual manifest files for the test case.

While you can generate your own manifests within the tests themselves (e.g. by initializing a Kubernetes API object), this can become tedious and clutter up the tests. If you do choose to go this route, you can still use all of the kubetest functionality by wrapping supported objects with their equivalent kubetest wrapper. For example,

```
from kubernetes import client
from kubetest.objects import Deployment


# Create a Kubernetes API Object
raw_deployment = client.V1Deployment(
    metadata=client.V1ObjectMeta(
        name='test-deployment'
    ),
    spec=client.V1DeploymentSpec(
        replicas=2,
        template=client.V1PodTemplateSpec(
            ...
        )
    )
)

# Wrap it in the kubetest wrapper
wrapped_deployment = Deployment(raw_deployment)
```

If you use manifest files, you can load them directly into wrapped API objects easily via the kubetest *Client*, which is provided to a test case via the *kube* fixture.

```
def test_something(kube):

    f = os.path.join(
        os.path.dirname(os.path.realpath(__file__)),
        'manifests',
        'deployment.yaml'
    )

   deployment = kube.load_deployment(f)
```

Often, tests will multiple resources that need to be loaded from manifest YAMLs. It can be tedious to construct all of the paths, load them, and create them at the start of a test. kubetest provides the *Apply Manifests* marker that allows you to specify an entire directory to load, or specific files from a directory. The example below loads the same file as the previous example using the `applymanifests` marker.

```
@pytest.mark.applymanifests('manifests', files=[
    'deployment.yaml'
```

(continues on next page)

(continued from previous page)

```
])
def test_something(kube):
    ...
```

Once a manifest is loaded, you will have (or be able to get) a reference to the created API Objects which offer more functionality.

4.5.3 Creating Resources

If you use the *Apply Manifests*, as described in the previous section, the manifest will be loaded and created for you in the test case namespace of your cluster (test case namespaces are automatically managed via the *kube*).

You may want to load resources manually, or load and create some at a later time in the test. This can be done via the `kube` client

```
def test_something(kube):

    # ...
    # do something first
    # ...

    deployment = kube.load_deployment('path/to/deployment.yaml')
    kube.create(deployment)
```

It can also be done through the resource reference itself

```
def test_something(kube):

    # ...
    # do something first
    # ...

    deployment = kube.load_deployment('path/to/deployment.yaml')
    deployment.create()
```

4.5.4 Deleting Resources

It is not necessary to delete resources at the end of a test case. kubetest automatically manages the namespace for the test case. When the test completes, it will delete the namespace from the cluster which will also delete any remaining resources in that namespace.

It can still be useful to delete things while testing, e.g. to simulate a service failure and to test the subsequent disaster recovery process. Similar to resource creation, resource deletion can be done either through the object reference or through the `kube` client

```
def test_something(kube):

    # ...
    # created resource, did some testing, now need to remove
    # the resource
    # ...

    # Method #1 - delete via the kube client
    kube.delete(deployment)
```

(continues on next page)

(continued from previous page)

```
    # Method #2 - delete via the object reference
    deployment.delete()
```

4.5.5 Test Namespaces

By default, `kubetest` will automatically generate a new Namespace for each test case, using the test name and a timestamp for the namespace name to ensure uniqueness. This behavior may not be desired in all cases, such as when users may not have permissions to create a new namespace on the cluster, or the tests are written against an already-running deployment in an existing namespace. In such cases, the _namespace_marker may be used.

4.5.6 Waiting

The time it takes for a resource to start, stop, or become ready can vary across numerous factors. It is not always reliable to just `time.sleep(10)` and hope that the desired state is met (nor is it efficient). To help with this, there are a number of *wait* functions provided by kubetest. For a full accounting of all wait functions, see the *API Reference*.

Below are some simple examples of select wait function usage.

Ready Nodes

If you are running on a cluster that can scale automatically, you may need to wait for the correct number of nodes to be available and ready before the test can run.

```
@pytest.mark.applymanifests('manifests')
def test_something(kube):
    # wait for 3 nodes to be available and ready
    kube.wait_for_ready_nodes(3, timeout=5 * 60)
```

Created Object

Wait until an object has been created on the cluster.

```
def test_something(kube):
    deployment = kube.load_deployment('path/to/deployment.yaml')
    kube.create(deployment)
    kube.wait_until_created(deployment, timeout=30)
```

Pod Containers Start

Wait until a Pod's containers have all started.

```
@pytest.mark.applymanifests('manifests')
def test_something(kube):
    pods = kube.get_pods()
    for pod in pods.values():
        pod.wait_until_containers_start(timeout=60)
```

4.6 Examples

4.6.1 Deploy Nginx

In this example test, we define a simple Nginx deployment and test that when we deploy it, it has the expected number of replicas and each pod returns the default *"welcome to nginx"* text when we `HTTP GET "/"`.

The file structure for this example would look like:

```
configs/
    nginx.yaml
test_nginx.py
```

Where the files listed above have the following contents:

Listing 1: configs/nginx.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

Listing 2: test_nginx.py

```python
import pytest


@pytest.mark.applymanifests('configs', files=[
    'nginx.yaml'
])
def test_nginx(kube):
    """An example test against an Nginx deployment."""

    # wait for the manifests loaded by the 'applymanifests' marker
    # to be ready on the cluster
    kube.wait_for_registered(timeout=30)

    deployments = kube.get_deployments()
    nginx_deploy = deployments.get('nginx-deployment')
```

(continues on next page)

(continued from previous page)

```
    assert nginx_deploy is not None

    pods = nginx_deploy.get_pods()
    assert len(pods) == 3, 'nginx should deploy with three replicas'

    for pod in pods:
        containers = pod.get_containers()
        assert len(containers) == 1, 'nginx pod should have one container'

        resp = pod.http_proxy_get('/')
        assert '<h1>Welcome to nginx!</h1>' in resp.data
```

With `kubetest` installed and a cluster available and configurations at `~/.kube/config`, we can run the test

```
$ pytest -s .
================================== test session starts␣
↪===================================
platform darwin -- Python 3.6.5, pytest-3.8.0, py-1.6.0, pluggy-0.7.1
kubetest config file: default
rootdir: /Users/edaniszewski/dev/examples, inifile:
plugins: kubetest-0.0.1
collected 1 item

test_nginx.py .

================================= 1 passed in 5.35 seconds␣
↪==================================
_____________________________________________ summary _____________________________________
↪_____
  examples: commands succeeded
  congratulations :)
```

4.6.2 Test in error

Looking at the same setup as the previous example, we can modify the test to fail in order to examine what a failure response would look like. We'll change `test_nginx.py` to instead expect 1 replica, when it will actually have three.

Listing 3: test_nginx.py

```
import pytest


@pytest.mark.applymanifests('configs', files=[
    'nginx.yaml'
])
def test_nginx(kube):
    """An example test against an Nginx deployment."""

    # wait for the manifests loaded by the 'applymanifests' marker
    # to be ready on the cluster
    kube.wait_for_registered(timeout=30)

    deployments = kube.get_deployments()
    nginx_deploy = deployments.get('nginx-deployment')
```

(continues on next page)

(continued from previous page)

```
    assert nginx_deploy is not None

    pods = nginx_deploy.get_pods()
    assert len(pods) == 1, 'nginx should deploy with three replicas'

    for pod in pods:
        containers = pod.get_containers()
        assert len(containers) == 1, 'nginx pod should have one container'

        resp = pod.http_proxy_get('/')
        assert '<h1>Welcome to nginx!</h1>' in resp.data
```

Now, when we run the tests, we should expect to see an error.

```
$ pytest -s .
================================== test session starts␣
↪===================================
platform darwin -- Python 3.6.5, pytest-3.8.0, py-1.6.0, pluggy-0.7.1
kubetest config file: default
rootdir: /Users/edaniszewski/dev/examples, inifile:
plugins: kubetest-0.0.1
collected 1 item

test_nginx.py F

======================================== FAILURES␣
↪=========================================
_____________________________________ test_nginx ______________________________________
↪_____

kube = <kubetest.client.TestClient object at 0x105d7cdd8>

    @pytest.mark.applymanifests('configs', files=[
        'nginx.yaml'
    ])
    def test_nginx(kube):
        """An example test against an Nginx deployment."""

        # wait for the manifests loaded by the 'applymanifests' marker
        # to be ready on the cluster
        kube.wait_for_registered(timeout=30)

        deployments = kube.get_deployments()
        nginx_deploy = deployments.get('nginx-deployment')
        assert nginx_deploy is not None

        pods = nginx_deploy.get_pods()
>       assert len(pods) == 1, 'nginx should deploy with three replicas'
E       AssertionError: nginx should deploy with three replicas
E       assert 3 == 1
E        +  where 3 = len([{'api_version': None,\n 'kind': None,\n 'metadata': {
↪'annotations': None,\n
            'cluster_name': None,\n         ...ort',\n            'reason': None,\n
            'start_time': datetime.datetime(2018, 9, 28, 22, 9, 2, tzinfo=tzutc())}}])

examples/test_nginx.py:20: AssertionError
================================== 1 failed in 4.36 seconds␣
↪===================================
```

(continues on next page)

(continued from previous page)

```
ERROR: InvocationError: 'pytest -s .'
_____ summary _____
ERROR:   examples: commands failed
```

In this case, the error message isn't too bad, but if we wanted more context, we could run tests with kubetest at log level "info" (or, for lots of context at log level "debug". Debug output is omitted here for brevity).

```
$ pytest -s . --kube-log-level=info
============================= test session starts ==============================
platform darwin -- Python 3.6.5, pytest-3.8.0, py-1.6.0, pluggy-0.7.1
kubetest config file: default
rootdir: /Users/edaniszewski/dev/examples, inifile:
plugins: kubetest-0.0.1
collected 1 item

test_nginx.py F

================================== FAILURES ===================================
_____ test_nginx _____

kube = <kubetest.client.TestClient object at 0x103e012e8>

    @pytest.mark.applymanifests('configs', files=[
        'nginx.yaml'
    ])
    def test_nginx(kube):
        """An example test against an Nginx deployment."""

        # wait for the manifests loaded by the 'applymanifests' marker
        # to be ready on the cluster
        kube.wait_for_registered(timeout=30)

        deployments = kube.get_deployments()
        nginx_deploy = deployments.get('nginx-deployment')
        assert nginx_deploy is not None

        pods = nginx_deploy.get_pods()
>       assert len(pods) == 1, 'nginx should deploy with three replicas'
E       AssertionError: nginx should deploy with three replicas
E       assert 3 == 1
E        +  where 3 = len([{'api_version': None,\n 'kind': None,\n 'metadata': {
↪'annotations': None,\n              'cluster_name': None,\n         ...t',\n
            'reason': None,\n            'start_time': datetime.datetime(2018, 9, 28,
↪ 22, 10, 21, tzinfo=tzutc())}}])

examples/test_nginx.py:20: AssertionError
------------------------------ Captured log setup -------------------------------
manager.py                 308 INFO     creating test meta for examples/test_nginx.
↪py::test_nginx
namespace.py                61 INFO     creating namespace "kubetest-test-nginx-
↪1538172620"
```

(continues on next page)

(continued from previous page)

```
deployment.py                48 INFO     creating deployment "nginx-deployment" in␣
↪namespace "kubetest-test-nginx-1538172620"
utils.py                     90 INFO     waiting for condition: <Condition (name: wait␣
↪for <class 'kubetest.objects.deployment.Deployment'>:nginx-deployment to be created,
↪ met: False)>
utils.py                    121 INFO     wait completed (total=0.063870) <Condition␣
↪(name: wait for <class 'kubetest.objects.deployment.Deployment'>:nginx-deployment␣
↪to be created, met: True)>
------------------------------------------------------------------- Captured log call -
↪-------------------------------------------------------------------
utils.py                     90 INFO     waiting for condition: <Condition (name: wait␣
↪for pre-registered objects to be ready, met: False)>
utils.py                    121 INFO     wait completed (total=2.169333) <Condition␣
↪(name: wait for pre-registered objects to be ready, met: True)>
deployment.py               131 INFO     getting pods for deployment "nginx-deployment"
------------------------------------------------------------------- Captured log␣
↪teardown ------------------------------------------------------------------
namespace.py                 79 INFO     deleting namespace "kubetest-test-nginx-
↪1538172620"
=============================================================== 1 failed in 5.07␣
↪seconds ===================================================================
ERROR: InvocationError: 'pytest -s . --kube-log-level=info'
_____________________________________________________________________________ summary _______
↪_____________________________________________________________________________
ERROR:   examples: commands failed
```

4.6.3 Container logs on test error

In the above example, you got to see different log output that kubetest could provide. In addition to logging out the actions that kubetest performs (and at the "debug" level, the Kubernetes objects themselves), kubetest can also get logs out of the running contianers for the test.

The caveat here is that it will only get logs for containers that are running. In the example above, we don't see any of the container logs because the failure occurred before the containers were fully up. If we introduce an error later on, like changing the `<h1>` tags in the expected nginx response to `<h2>`, the test should fail while some containers are up, so the error output should contain some of the container logs. Below is a snippet of what that would look like.

```
---------------------------------------- Captured kubernetes container logs call -----
↪-----------------------------------
================================================================================================
=== examples/test_nginx.py::test_nginx -> nginx-deployment-75675f5897-9fp8n::nginx ===
================================================================================================
10.60.58.1 - - [28/Sep/2018:22:20:09 +0000] "GET /foobar HTTP/1.1" 404 168 "-"
↪"Swagger-Codegen/7.0.0/python" "68.162.240.6"
2018/09/28 22:20:09 [error] 6#0: *1 open() "/usr/share/nginx/html/foobar" failed (2:␣
↪No such file or directory), client: 10.60.58.1,
server: localhost, request: "GET /foobar HTTP/1.1", host: "35.232.2.153"
```

4.6.4 Using in-cluster config

If your test requirements limit you to only be able to run on a cluster where you may not have access to the kube config file, you can use in-cluster config instead by setting the `--in-cluster` flag.

As an extremely basic example, suppose you have a simple test case:

```
def test_configmap_count(kube):
    cms = kube.get_configmaps()
    assert len(cms) == 0
```

The test could be encapsulated in a Docker container so it could be run on the cluster. Note that this could be done in a number of ways, the example below is not meant to exemplify best-practices, it is only a basic functioning example.

Finally, a Job manifest can be created for the test:

```
apiVersion: batch/v1
kind: Job
metadata:
  name: kubetest-example
  namespace: default
  labels:
    app: kubetest-example
spec:
  backoffLimit: 0
  template:
    spec:
      restartPolicy: Never
      containers:
      - name: tests
        image: kubetest/example-test-image
        imagePullPolicy: Always
```

You can then apply the manifest and have it run on cluster

```
kubectl apply -f kubetest-job.yaml
```

4.7 Module Reference

Below is a complete module reference

4.7.1 kubetest

kubetest package

Submodules

kubetest.client module

The test client for managing Kubernetes resources within test cases.

An instance of the `TestClient` defined in this module is automatically created for each test case that uses the `kube` fixture. The `kube` fixture provides the `TestClient` instance to the test case.

class kubetest.client.**TestClient**(*namespace: str*)

Bases: `object`

Test client for managing Kubernetes resources for a test case.

The `namespace` for the TestClient will be automatically generated and provided to the TestClient during the test setup process.

Parameters **namespace** – The namespace associated with the test client. Each test case will have its own namespace assigned.

create(*obj: kubetest.objects.api_object.ApiObject*) → None
Create the provided ApiObject on the Kubernetes cluster.

If the object does not already have a namespace assigned to it, the client's generated test case namespace will be used.

Parameters **obj** – A kubetest API Object wrapper.

delete(*obj: kubetest.objects.api_object.ApiObject*, *options: kubernetes.client.models.v1_delete_options.V1DeleteOptions = None*) → None
Delete the provided ApiObject from the Kubernetes cluster.

If the object does not already have a namespace assigned to it, the client's generated test case namespace will be used.

Parameters

- **obj** – A kubetest API Object wrapper.
- **options** – Additional options for deleting the resource from the cluster.

static refresh(*obj: kubetest.objects.api_object.ApiObject*) → None
Refresh the underlying Kubernetes resource status and state.

Parameters **obj** – A kubetest API Object wrapper.

static load_clusterrolebinding(*path: str*, *name: Optional[str] = None*) → kubetest.objects.clusterrolebinding.ClusterRoleBinding
Load a manifest YAML into a ClusterRoleBinding object.

Parameters

- **path** – The path to the ClusterRoleBinding manifest.
- **name** – The name of the resource to load. If the manifest file contains a single object definition for the type being loaded, it is not necessary to specify the name. If the manifest has multiple definitions containing the same type, a name is required to differentiate between them. If no name is specified in such case, an error is raised.

Returns The ClusterRoleBinding for the specified manifest.

load_configmap(*path: str*, *set_namespace: bool = True*, *name: Optional[str] = None*) → kubetest.objects.configmap.ConfigMap
Load a manifest YAML into a ConfigMap object.

By default, this will augment the ConfigMap object with the generated test case namespace. This behavior can be disabled with the `set_namespace` flag.

Parameters

- **path** – The path to the ConfigMap manifest.
- **set_namespace** – Enable/disable the automatic augmentation of the ConfigMap namespace.
- **name** – The name of the resource to load. If the manifest file contains a single object definition for the type being loaded, it is not necessary to specify the name. If the manifest has multiple definitions containing the same type, a name is required to differentiate between them. If no name is specified in such case, an error is raised.

Returns The ConfigMap for the specified manifest.

load_daemonset(*path: str*, *set_namespace: bool = True*, *name: Optional[str] = None*) → kubetest.objects.daemonset.DaemonSet

Load a manifest YAML into a DaemonSet object.

By default, this will augment the DaemonSet object with the generated test case namespace. This behavior can be disabled with the `set_namespace` flag.

Parameters

- **path** – The path to the DaemonSet manifest.
- **set_namespace** – Enable/disable the automatic augmentation of the DaemonSet namespace.
- **name** – The name of the resource to load. If the manifest file contains a single object definition for the type being loaded, it is not necessary to specify the name. If the manifest has multiple definitions containing the same type, a name is required to differentiate between them. If no name is specified in such case, an error is raised.

Returns The DaemonSet for the specified manifest.

load_deployment(*path: str*, *set_namespace: bool = True*, *name: Optional[str] = None*) → kubetest.objects.deployment.Deployment

Load a manifest YAML into a Deployment object.

By default, this will augment the Deployment object with the generated test case namespace. This behavior can be disabled with the `set_namespace` flag.

Parameters

- **path** – The path to the Deployment manifest.
- **set_namespace** – Enable/disable the automatic augmentation of the Deployment namespace.
- **name** – The name of the resource to load. If the manifest file contains a single object definition for the type being loaded, it is not necessary to specify the name. If the manifest has multiple definitions containing the same type, a name is required to differentiate between them. If no name is specified in such case, an error is raised.

Returns The Deployment for the specified manifest.

load_pod(*path: str*, *set_namespace: bool = True*, *name: Optional[str] = None*) → kubetest.objects.pod.Pod

Load a manifest YAML into a Pod object.

By default, this will augment the Pod object with the generated test case namespace. This behavior can be disabled with the `set_namespace` flag.

Parameters

- **path** – The path to the Pod manifest.
- **set_namespace** – Enable/disable the automatic augmentation of the Pod namespace.
- **name** – The name of the resource to load. If the manifest file contains a single object definition for the type being loaded, it is not necessary to specify the name. If the manifest has multiple definitions containing the same type, a name is required to differentiate between them. If no name is specified in such case, an error is raised.

Returns The Pod for the specified manifest.

load_rolebinding(*path: str*, *set_namespace: bool = True*, *name: Optional[str] = None*) → kubetest.objects.rolebinding.RoleBinding

Load a manifest YAML into a RoleBinding object.

By default, this will augment the RoleBinding object with the generated test case namespace. This behavior can be disabled with the `set_namespace` flag.

Parameters

- **path** – The path to the RoleBinding manifest.
- **set_namespace** – Enable/disable the automatic augmentation of the RoleBinding namespace.
- **name** – The name of the resource to load. If the manifest file contains a single object definition for the type being loaded, it is not necessary to specify the name. If the manifest has multiple definitions containing the same type, a name is required to differentiate between them. If no name is specified in such case, an error is raised.

Returns The RoleBinding for the specified manifest.

load_secret(*path: str*, *set_namespace: bool = True*, *name: Optional[str] = None*) → kubetest.objects.secret.Secret

Load a manifest YAML into a Secret object.

By default, this will augment the Secret object with the generated test case namespace. This behavior can be disabled with the `set_namespace` flag.

Parameters

- **path** – The path to the Secret manifest.
- **set_namespace** – Enable/disable the automatic augmentation of the Secret namespace.
- **name** – The name of the resource to load. If the manifest file contains a single object definition for the type being loaded, it is not necessary to specify the name. If the manifest has multiple definitions containing the same type, a name is required to differentiate between them. If no name is specified in such case, an error is raised.

Returns The Secret for the specified manifest.

load_service(*path: str*, *set_namespace: bool = True*, *name: Optional[str] = None*) → kubetest.objects.service.Service

Load a manifest YAML into a Service object.

By default, this will augment the Service object with the generated test case namespace. This behavior can be disabled with the `set_namespace` flag.

Parameters

- **path** – The path to the Service manifest.
- **set_namespace** – Enable/disable the automatic augmentation of the Service namespace.
- **name** – The name of the resource to load. If the manifest file contains a single object definition for the type being loaded, it is not necessary to specify the name. If the manifest has multiple definitions containing the same type, a name is required to differentiate between them. If no name is specified in such case, an error is raised.

Returns The Service for the specified manifest.

load_persistentvolumeclaim(*path: str*, *set_namespace: bool = True*, *name: Optional[str] = None*) → kubetest.objects.persistentvolumeclaim.PersistentVolumeClaim

Load a manifest YAML into a PersistentVolumeClaim object.

By default, this will augment the PersistentVolumeClaim object with the generated test case namespace. This behavior can be disabled with the `set_namespace` flag.

Parameters

- **path** (*str*) – The path to the PersistentVolumeClaim manifest.
- **set_namespace** (*bool*) – Enable/disable the automatic augmentation of the PersistentVolumeClaim namespace.
- **name** – The name of the resource to load. If the manifest file contains a single object definition for the type being loaded, it is not necessary to specify the name. If the manifest has multiple definitions containing the same type, a name is required to differentiate between them. If no name is specified in such case, an error is raised.

Returns The PersistentVolumeClaim for the specified manifest.

Return type objects.PersistentVolumeClaim

load_ingress(*path: str*, *set_namespace: bool = True*, *name: Optional[str] = None*) → kubetest.objects.ingress.Ingress

Load a manifest YAML into a Ingress object.

By default, this will augment the Ingress object with the generated test case namespace. This behavior can be disabled with the `set_namespace` flag.

Parameters

- **path** (*str*) – The path to the Ingress manifest.
- **set_namespace** (*bool*) – Enable/disable the automatic augmentation of the Ingress namespace.
- **name** – The name of the resource to load. If the manifest file contains a single object definition for the type being loaded, it is not necessary to specify the name. If the manifest has multiple definitions containing the same type, a name is required to differentiate between them. If no name is specified in such case, an error is raised.

Returns The ingress for the specified manifest.

Return type objects.Ingress

load_replicaset(*path: str*, *set_namespace: bool = True*, *name: Optional[str] = None*) → kubetest.objects.replicaset.ReplicaSet

Load a manifest YAML into a ReplicaSet object.

By default, this will augment the ReplicaSet object with the generated test case namespace. This behavior can be disabled with the `set_namespace` flag.

Parameters

- **path** – The path to the ReplicaSet manifest.
- **set_namespace** – Enable/disable the automatic augmentation of the ReplicaSet namespace.
- **name** – The name of the resource to load. If the manifest file contains a single object definition for the type being loaded, it is not necessary to specify the name. If the manifest has multiple definitions containing the same type, a name is required to differentiate between them. If no name is specified in such case, an error is raised.

Returns The ReplicaSet for the specified manifest.

load_statefulset(*path: str*, *set_namespace: bool = True*, *name: Optional[str] = None*) → kubetest.objects.statefulset.StatefulSet

Load a manifest YAML into a StatefulSet object.

By default, this will augment the StatefulSet object with the generated test case namespace. This behavior can be disabled with the `set_namespace` flag.

Parameters

- **path** – The path to the StatefulSet manifest.
- **set_namespace** – Enable/disable the automatic augmentation of the StatefulSet namespace.
- **name** – The name of the resource to load. If the manifest file contains a single object definition for the type being loaded, it is not necessary to specify the name. If the manifest has multiple definitions containing the same type, a name is required to differentiate between them. If no name is specified in such case, an error is raised.

Returns The StatefulSet for the specified manifest.

load_serviceaccount(*path: str*, *set_namespace: bool = True*, *name: Optional[str] = None*) → kubetest.objects.serviceaccount.ServiceAccount

Load a manifest YAML into a ServiceAccount object.

By default, this will augment the ServiceAccount object with the generated test case namespace. This behavior can be disabled with the `set_namespace` flag.

Parameters

- **path** – The path to the ServiceAccount manifest.
- **set_namespace** – Enable/disable the automatic augmentation of the ServiceAccount namespace.
- **name** – The name of the resource to load. If the manifest file contains a single object definition for the type being loaded, it is not necessary to specify the name. If the manifest has multiple definitions containing the same type, a name is required to differentiate between them. If no name is specified in such case, an error is raised.

Returns The ServiceAccount for the specified manifest.

get_configmaps(*namespace: str = None*, *fields: Dict[str, str] = None*, *labels: Dict[str, str] = None*) → Dict[str, kubetest.objects.configmap.ConfigMap]

Get ConfigMaps from the cluster.

Parameters

- **namespace** – The namespace to get the ConfigMaps from. If not specified, it will use the auto-generated test case namespace by default.
- **fields** – A dictionary of fields used to restrict the returned collection of ConfigMaps to only those which match these field selectors. By default, no restricting is done.
- **labels** – A dictionary of labels used to restrict the returned collection of ConfigMaps to only those which match these label selectors. By default, no restricting is done.

Returns A dictionary where the key is the ConfigMap name and the value is the ConfigMap itself.

get_daemonsets(*namespace: str = None*, *fields: Dict[str, str] = None*, *labels: Dict[str, str] = None*) → Dict[str, kubetest.objects.daemonset.DaemonSet]

Get DaemonSets from the cluster.

Parameters

- **namespace** – The namespace to get the DaemonSets from. If not specified, it will use the auto-generated test case namespace by default.

- **fields** – A dictionary of fields used to restrict the returned collection of DaemonSets to only those which match these field selectors. By default, no restricting is done.
- **labels** – A dictionary of labels used to restrict the returned collection of DaemonSets to only those which match these label selectors. By default, no restricting is done.

Returns A dictionary where the key is the DaemonSet name and the value is the DaemonSet itself.

get_deployments(*namespace: str = None*, *fields: Dict[str, str] = None*, *labels: Dict[str, str] = None*) → Dict[str, kubetest.objects.deployment.Deployment]

Get Deployments from the cluster.

Parameters

- **namespace** – The namespace to get the Deployments from. If not specified, it will use the auto-generated test case namespace by default.
- **fields** – A dictionary of fields used to restrict the returned collection of Deployments to only those which match these field selectors. By default, no restricting is done.
- **labels** – A dictionary of labels used to restrict the returned collection of Deployments to only those which match these label selectors. By default, no restricting is done.

Returns A dictionary where the key is the Deployment name and the value is the Deployment itself.

get_endpoints(*namespace: str = None*, *fields: Dict[str, str] = None*, *labels: Dict[str, str] = None*) → Dict[str, kubetest.objects.endpoints.Endpoints]

Get Endpoints from the cluster.

Parameters

- **namespace** – The namespace to get the Endpoints from. If not specified, it will use the auto-generated test case namespace by default.
- **fields** – A dictionary of fields used to restrict the returned collection of Endpoints to only those which match these field selectors. By default, no restricting is done.
- **labels** – A dictionary of labels used to restrict the returned collection of Endpoints to only those which match these label selectors. By default, no restricting is done.

Returns A dictionary where the key is the Endpoint name and the value is the Endpoint itself.

get_events(*fields: Dict[str, str] = None*, *labels: Dict[str, str] = None*, *all_namespaces: bool = False*) → Dict[str, kubetest.objects.event.Event]

Get the latest Events that occurred in the cluster.

Parameters

- **fields** – A dictionary of fields used to restrict the returned collection of Events to only those which match these field selectors. By default, no restricting is done.
- **labels** – A dictionary of labels used to restrict the returned collection of Events to only those which match these label selectors. By default, no restricting is done.
- **all_namespaces** – If True, get the events across all namespaces.

Returns A dictionary where the key is the Event name and the value is the Event itself.

get_namespaces(*fields: Dict[str, str] = None*, *labels: Dict[str, str] = None*) → Dict[str, kubetest.objects.namespace.Namespace]

Get Namespaces from the cluster.

Parameters

- **fields** – A dictionary of fields used to restrict the returned collection of Namespaces to only those which match these field selectors. By default, no restricting is done.
- **labels** – A dictionary of labels used to restrict the returned collection of Namespaces to only those which match these label selectors. By default, no restricting is done.

Returns A dictionary where the key is the Namespace name and the value is the Namespace itself.

static get_nodes(*fields: Dict[str, str] = None*, *labels: Dict[str, str] = None*) → Dict[str, kubetest.objects.node.Node]

Get the Nodes that make up the cluster.

Parameters

- **fields** – A dictionary of fields used to restrict the returned collection of Nodes to only those which match these field selectors. By default, no restricting is done.
- **labels** – A dictionary of labels used to restrict the returned collection of Nodes to only those which match these label selectors. By default, no restricting is done.

Returns A dictionary where the key is the Node name and the value is the Node itself.

get_pods(*namespace: str = None*, *fields: Dict[str, str] = None*, *labels: Dict[str, str] = None*) → Dict[str, kubetest.objects.pod.Pod]

Get Pods from the cluster.

Parameters

- **namespace** – The namespace to get the Pods from. If not specified, it will use the auto-generated test case namespace by default.
- **fields** – A dictionary of fields used to restrict the returned collection of Pods to only those which match these field selectors. By default, no restricting is done.
- **labels** – A dictionary of labels used to restrict the returned collection of Pods to only those which match these label selectors. By default, no restricting is done.

Returns A dictionary where the key is the Pod name and the value is the Pod itself.

get_secrets(*namespace: str = None*, *fields: Dict[str, str] = None*, *labels: Dict[str, str] = None*) → Dict[str, kubetest.objects.secret.Secret]

Get Secrets from the cluster.

Parameters

- **namespace** – The namespace to get the Secrets from. If not specified, it will use the auto-generated test case namespace by default.
- **fields** – A dictionary of fields used to restrict the returned collection of Secrets to only those which match these field selectors. By default, no restricting is done.
- **labels** – A dictionary of labels used to restrict the returned collection of Secrets to only those which match these label selectors. By default, no restricting is done.

Returns A dictionary where the key is the Secret name and the value is the Secret itself.

get_services(*namespace: str = None*, *fields: Dict[str, str] = None*, *labels: Dict[str, str] = None*) → Dict[str, kubetest.objects.service.Service]

Get Services under the test case namespace.

Parameters

- **namespace** – The namespace to get the Services from. If not specified, it will use the auto-generated test case namespace by default.

- **fields** – A dictionary of fields used to restrict the returned collection of Services to only those which match these field selectors. By default, no restricting is done.
- **labels** – A dictionary of labels used to restrict the returned collection of Services to only those which match these label selectors. By default, no restricting is done.

Returns A dictionary where the key is the Service name and the value is the Service itself.

get_persistentvolumeclaims(*namespace: str = None*, *fields: Dict[str, str] = None*, *labels: Dict[str, str] = None*) → Dict[str, kubetest.objects.persistentvolumeclaim.PersistentVolumeClaim]

Get PersistentVolumeClaims from the cluster.

Parameters

- **namespace** – The namespace to get the PersistentVolumeClaim from. If not specified, it will use the auto-generated test case namespace by default.
- **fields** – A dictionary of fields used to restrict the returned collection of PersistentVolumeClaim to only those which match these field selectors. By default, no restricting is done.
- **labels** – A dictionary of labels used to restrict the returned collection of PersistentVolumeClaim to only those which match these label selectors. By default, no restricting is done.

Returns A dictionary where the key is the PersistentVolumeClaim name and the value is the PersistentVolumeClaim itself.

get_ingresses(*namespace: str = None*, *fields: Dict[str, str] = None*, *labels: Dict[str, str] = None*) → Dict[str, kubetest.objects.ingress.Ingress]

Get Ingresses from the cluster.

Parameters

- **namespace** – The namespace to get the Ingress from. If not specified, it will use the auto-generated test case namespace by default.
- **fields** – A dictionary of fields used to restrict the returned collection of Ingress to only those which match these field selectors. By default, no restricting is done.
- **labels** – A dictionary of labels used to restrict the returned collection of Ingress to only those which match these label selectors. By default, no restricting is done.

Returns A dictionary where the key is the Ingress name and the value is the Ingress itself.

get_replicasets(*namespace: str = None*, *fields: Dict[str, str] = None*, *labels: Dict[str, str] = None*) → Dict[str, kubetest.objects.replicaset.ReplicaSet]

Get ReplicaSets from the cluster.

Parameters

- **namespace** – The namespace to get the ReplicaSets from. If not specified, it will use the auto-generated test case namespace by default.
- **fields** – A dictionary of fields used to restrict the returned collection of ReplicaSets to only those which match these field selectors. By default, no restricting is done.
- **labels** – A dictionary of labels used to restrict the returned collection of ReplicaSets to only those which match these label selectors. By default, no restricting is done.

Returns A dictionary where the key is the ReplicaSet name and the value is the ReplicaSet itself.

get_statefulsets(*namespace: str = None*, *fields: Dict[str, str] = None*, *labels: Dict[str, str] = None*) → Dict[str, kubetest.objects.statefulset.StatefulSet]

Get StatefulSets from the cluster.

Parameters

- **namespace** – The namespace to get the StatefulSets from. If not specified, it will use the auto-generated test case namespace by default.
- **fields** – A dictionary of fields used to restrict the returned collection of StatefulSets to only those which match these field selectors. By default, no restricting is done.
- **labels** – A dictionary of labels used to restrict the returned collection of StatefulSets to only those which match these label selectors. By default, no restricting is done.

Returns A dictionary where the key is the StatefulSet name and the value is the StatefulSet itself.

get_serviceaccounts(*namespace: str = None*, *fields: Dict[str, str] = None*, *labels: Dict[str, str] = None*) → Dict[str, kubetest.objects.serviceaccount.ServiceAccount]

Get ServiceAccounts from the cluster.

Parameters

- **namespace** – The namespace to get the ServiceAccount from. If not specified, it will use the auto-generated test case namespace by default.
- **fields** – A dictionary of fields used to restrict the returned collection of ServiceAccount to only those which match these field selectors. By default, no restricting is done.
- **labels** – A dictionary of labels used to restrict the returned collection of ServiceAccount to only those which match these label selectors. By default, no restricting is done.

Returns A dictionary where the key is the ServiceAccount name and the value is the ServiceAccount itself.

static wait_for_conditions(**args*, *timeout: int = None*, *interval: Union[float, int] = 1*, *policy: kubetest.condition.Policy = <Policy.ONCE: 1>*, *fail_on_api_error: bool = True*) → None

Wait for all of the provided Conditions to be met.

All Conditions must be met for this to unblock. If no Conditions are provided, this method will do nothing.

Parameters

- ***args** – Conditions to check.
- **timeout** – The maximum time to wait, in seconds, for the provided Conditions to be met. If all of the Conditions are not met within the given timeout, this will raise a TimeoutError. By default, there is no timeout so this will wait indefinitely.
- **interval** – The time, in seconds, to sleep before re-evaluating the conditions. Default: 1s
- **policy** – The condition checking policy that defines the checking behavior. Default: ONCE
- **fail_on_api_error** – Fail the condition checks if a Kubernetes API error is incurred. An API error can be raised for a number of reasons, including a Pod being restarted and temporarily unavailable. Disabling this will cause those errors to be ignored, allowing the check to continue until timeout or resolution. (default: True).

Raises

- `TimeoutError` – The Conditions were not met within the specified timeout period.

- `ValueError` – Not all arguments are a Condition.

wait_for_ready_nodes(*count: int*, *timeout: int = None*, *interval: Union[int, float] = 1*) → None
Wait until there are at least `count` number of nodes available in the cluster.

Notes

This should only be used for clusters that auto-scale the nodes. This will not create/delete nodes on its own.

Parameters

- **count** – The number of nodes to wait for.
- **timeout** – The maximum time to wait, in seconds.
- **interval** – The time, in seconds, to sleep before re-checking the number of nodes.

wait_for_registered(*timeout: int = None*, *interval: Union[int, float] = 1*) → None
Wait for all of the pre-registered objects to be ready on the cluster.

An object is pre-registered with the test client if it is specified to the test via the `applymanifests` pytest marker. The marker will load the manifest and add the object to the cluster, and register it with the test client. This method waits until all such loaded manifest objects are in the ready state simultaneously.

Parameters

- **timeout** – The maximum time to wait, in seconds.
- **interval** – The time, in seconds, to sleep before re-checking the ready state for pre-registered objects.

static wait_until_created(*obj: kubetest.objects.api_object.ApiObject*, *timeout: int = None*, *interval: Union[int, float] = 1*) → None
Wait until the specified object has been created.

Here, creation is judged on whether or not refreshing the object (e.g. getting it) returns an object (created) or an error (not yet created).

Parameters

- **obj** – The ApiObject to wait on.
- **timeout** – The maximum time to wait, in seconds.
- **interval** – The time, in seconds, to sleep before re-checking the created state of the object.

kubetest.manager module

Kubetest manager for test client instances and namespace management.

class kubetest.manager.**ObjectManager**
Bases: `object`

ObjectManager is a convenience class used to manage Kubernetes API objects that are registered with a test case.

The core usage of the ObjectManager is to sort each of the registered objects into different buckets by type. An "apply order" is also defined here so we can get the bucketed objects in the order that they should be applied onto the cluster.

This manager will only be used for API objects loaded from manifests that are specified by the *pytest.mark.applymanifests* marker on a test case by default.

ordered_buckets = ['namespace', 'rolebinding', 'clusterrolebinding', 'secret', 'networ

add(**args*) → None
Add API objects to the object manager.

This method will take in any number of ApiObjects and sort them into the correct buckets. It will not check for duplicates. If a non-ApiObject is passed in, an error will be raised.

Parameters **args** – Any subclass of the kubetest ApiObject wrapping a Kubernetes API object.

Raises ValueError – One or more arguments passed to the function are not ApiObject subclasses.

get_objects_in_apply_order() → Generator[kubetest.objects.api_object.ApiObject, None, None]
Get all of the managed objects in the order that they should be applied onto the cluster.

Within the buckets themselves, API objects are not sorted. This function only yields the buckets in the correct order.

Each of the buckets corresponds to an ApiObject wrapper that is supported by kubetest. As more ApiObject wrappers are added, the buckets here should be updated to reflect that.

The bucket order in which objects are yielded are:

- Namespace
- RoleBinding
- ClusterRoleBinding
- Secret
- NetworkPolicy
- Service
- ConfigMap
- PersistentVolumeClaim
- Ingress
- DaemonSet
- StatefulSet
- ReplicaSet
- Deployment
- Pod

Yields The kubetest ApiObject wrapper to be created on the cluster.

class kubetest.manager.**TestMeta**(*name: str*, *node_id: str*, *namespace_create: bool = True*, *namespace_name: str = None*)
Bases: object

TestMeta holds information associated with a single test node.

Parameters

- **name** – The name of the test.

- **node_id** – The id of the test node.
- **namespace_create** – Option to create a new namespace. This can be toggled off in the event that a user is using an existing namespace on their test cluster.
- **namespace_name** – The name of the namespace to use. If the namespace is to be auto-generated, this can be left as None.

client
Get the TestClient for the test case.

namespace
Get the Namespace API Object associated with the test case.

setup() → None
Setup the cluster state for the test case.

This performs all actions needed in order for the test client to be ready to use by a test case.

teardown() → None
Clean up the cluster state for the given test case.

This performs all actions needed in order to clean up the state that was previously set up for the test client in *setup*.

yield_container_logs(*tail_lines: int = None*) → Generator[str, None, None]
Yield the container logs for the test case.

These logs will be printed out if the test was in error to provide more context and make it easier to debug the issue.

Parameters **tail_lines** – The number of container log lines to print.

Yields *str* – Logs for the running containers on the cluster.

register_rolebindings(**rolebindings*) → None
Register a RoleBinding requirement with the test case.

Parameters **rolebindings** – The RoleBindings that are needed for the test case.

register_clusterrolebindings(**clusterrolebindings*) → None
Register a ClusterRoleBinding requirement with the test case.

Parameters **clusterrolebindings** – The ClusterRoleBindings that are needed for the test case.

register_objects(*api_objects: List[kubetest.objects.api_object.ApiObject]*)
Register the provided objects with the test case.

These objects will be registered to the test client and applied to the namespace on test setup.

Parameters **api_objects** – The wrapped Kubernetes API objects to create on the cluster.

class kubetest.manager.**KubetestManager**
Bases: object

The manager for kubetest state.

The KubetestManager is in charge of providing test clients for the tests that request them and mediating the namespace management corresponding to those test clients.

new_test(*node_id: str*, *test_name: str*, *namespace_create: bool = True*, *namespace_name: str = None*) → kubetest.manager.TestMeta
Create a new TestMeta for a test case.

This will be called by the test setup hook in order to create a new TestMeta for the manager to keep track of.

Parameters

- **node_id** – The id of the test node.
- **test_name** – The name of the test.
- **namespace_create** – Option to create a new namespace. This can be toggled off in the event that a user is using an existing namespace on their test cluster.
- **namespace_name** – The name of the namespace to use. If the namespace is to be autogenerated, this can be left as None.

Returns The newly created TestMeta for the test case.

get_test(*node_id: str*) → Optional[kubetest.manager.TestMeta]
Get the test metadata for the specified test node.

Parameters **node_id** – The id of the test node.

Returns The test metadata for the given node. If no test metadata is found for the given node, None is returned.

teardown(*node_id: str*) → None
Tear down the test case.

This is effectively a wrapper around the *teardown* method of the test client. It will also remove the test client from the manager.

Test client teardown will delete the test client's namespace from the cluster. Deleting a namespace will delete all the things in the namespace (e.g. API objects bound to the namespace).

Parameters **node_id** – The id of the test node.

kubetest.manifest module

Utility functions for loading Kubernetes manifest files and constructing the corresponding Kubernetes API models.

kubetest.manifest.**render**(*template: Union[str, TextIO], context: Dict[str, Any]*) → Union[str, TextIO]
Render a manifest template into a YAML document using the module render callable.

Rendering is performed by the module global *__render__* callable. The default implementation returns the input template unmodified. To implement a different default rendering behavior, set the *__render__* attribute to a *RenderCallable* object.

Parameters

- **template** – Then template to render.
- **context** – A dictionary of variables available to the template.

Returns The rendered content of the manifest template.

class kubetest.manifest.**ContextRenderer**(*renderer: Callable[[Union[str, TextIO], Dict[str, Any]], Union[str, TextIO]] = <function render>, context: Dict[str, Any] = {}*)
Bases: object

ContextRenderer objects manage context state and render manifest templates.

ContextRenderer objects maintain a persistent context state across an arbitrary number of rendering operations. They are useful in accumulating context state during test setup when manifest templates need access to state that was established earlier.

Parameters

- **renderer** – A callable that renders a templated manifest. Defaults to the module local *render* function which renders using the *__render__* callable.
- **context** – A dictionary of runtime values available to the template during rendering. Defaults to an empty dictionary.

context
The context variables set on the renderer.

__call__(*template: Union[str, TextIO]*, *context: Dict[str, Any]*) → Union[str, TextIO]
Render a manifest template file to a YAML docoment.

Parameters

- **template** – The template to render.
- **context** – A dictionary of template variables available for use during rendering.

Returns The rendered content of the manifest template.

kubetest.manifest.**load_file**(*path: str*, ***, *renderer: Callable[[Union[str, TextIO], Dict[str, Any]], Union[str, TextIO]]* = <*function render*>) → List[object]
Load an individual Kubernetes manifest YAML file.

This file may contain multiple YAML documents. It will attempt to auto-detect the type of each object to load.

Parameters

- **path** – The fully qualified path to the file.
- **renderer** – The callable responsible for rendering the contents of the manifest file to YAML.

Returns A list of the Kubernetes API objects for this manifest file.

kubetest.manifest.**load_path**(*path: str*, ***, *renderer: Callable[[Union[str, TextIO], Dict[str, Any]], Union[str, TextIO]]* = <*function render*>) → List[object]
Load all of the Kubernetes YAML manifest files found in the specified directory path.

Parameters

- **path** – The path to the directory of manifest files.
- **renderer** – The callable responsible for rendering the contents of the manifest files to YAML.

Returns A list of all the Kubernetes objects loaded from manifest file.

Raises ValueError – The provided path is not a directory.

kubetest.manifest.**get_type**(*manifest: Dict[str, Any]*) → Optional[object]
Get the Kubernetes object type from the manifest kind and version.

There is no easy way for determining the internal model that a manifest should use. What this tries to do is use the version info and the kind info to create a potential internal object name for the pair and look that up in the kubernetes package locals.

Parameters **manifest** – The manifest file, loaded into a dictionary.

Returns The Kubernetes API object for the manifest. If no Kubernetes API object type can be determined, None is returned.

Raises `ValueError` – The manifest dictionary does not have a *version* or *kind* specified.

`kubetest.manifest.`**`load_type`**(*obj_type, path: str, *, renderer: Callable[[Union[str, TextIO], Dict[str, Any]], Union[str, TextIO]] = <function render>*)

Load a Kubernetes YAML manifest file for the specified type.

While Kubernetes manifests can contain multiple object definitions in a single file (delimited with the YAML separator '—'), this does not currently support those files. This function expects a single object definition in the specified manifest file.

Parameters

- **`path`** – The path the manifest YAML to load.
- **`obj_type`** – The Kubernetes API object type that the YAML contents should be loaded into.
- **`renderer`** – The callable responsible for rendering the contents of the manifest file to YAML.

Returns A Kubernetes API object populated with the YAML contents.

Raises `FileNotFoundError` – The specified file was not found.

`kubetest.manifest.`**`new_object`**(*root_type*, *config*)

Create a new Kubernetes API object and recursively populate it with the provided manifest configuration.

The recursive population utilizes the swagger_types/openapi_types and attribute_map members of the Kubernetes API object class to determine which config fields correspond to which input parameter, and to cast them to their expected type.

This is all based on the premise that the Python Kubernetes client will continue to be based off of an auto-generated Swagger/Openapi spec and that these fields will be available for all API objects.

Parameters

- **`root_type`** – The Kubernetes API object type that will be populated with the manifest configuration. This is expected to be known ahead of time by the caller.
- **`config`** – The manifest configuration for the API object.

Returns A Kubernetes API object recursively populated with the YAML contents.

`kubetest.manifest.`**`cast_value`**(*value: Any*, *t: str*) → Any

Cast the given value to the specified type.

There are two general cases for possible casts:

- A cast to a builtin type (int, str, etc.)
- A cast to a Kubernetes object (V1ObjectMeta, etc)

In either case, check to see if the specified type exists in the correct type pool. If so, cast to that type, otherwise fail.

Parameters

- **`value`** – The value to cast.
- **`t`** – The type to cast the value to. This can be a builtin type or a Kubernetes API object type.

Returns The value, casted to the appropriate type.

Raises

- `ValueError` – Unable to cast the value to the specified type.
- `TypeError` – Unable to cast the given value to the specified type.

- `AttributeError` – The value is an invalid Kubernetes type.

kubetest.objects module

Kubetest wrappers around Kubernetes API Objects.

kubetest.plugin module

A pytest plugin which helps with integration testing for Kubernetes deployments.

This plugin helps with the managing of the Kubernetes cluster and provides useful test fixtures and functionality in order to interact with and test the state of the cluster.

`kubetest.plugin.`**`pytest_addoption`**(*parser*)
Add options to pytest to configure kubetest.

See also:

https://docs.pytest.org/en/latest/reference.html#_pytest.hookspec.pytest_addoption

`kubetest.plugin.`**`pytest_report_header`**(*config*)
Augment the pytest report header with kubetest info.

See also:

https://docs.pytest.org/en/latest/reference.html#_pytest.hookspec.pytest_report_header

`kubetest.plugin.`**`pytest_configure`**(*config*)
Configure pytest with kubetest additions.

This registers the kubetest markers with pytest.

See also:

https://docs.pytest.org/en/latest/reference.html#_pytest.hookspec.pytest_configure

`kubetest.plugin.`**`pytest_sessionstart`**(*session*)
Configure kubetest for the test session.

Kubetest setup happens at session start (pytest_sessionstart) rather than on configuration (pytest_configure) so that we only have the expectation of a cluster config and available cluster when there are actually tests available. For example, if we are simply calling any of

pytest –help pytest –markers pytest –fixtures

we do not want to configure kubetest and force the expectation of cluster availability.

See also:

https://docs.pytest.org/en/latest/reference.html#_pytest.hookspec.pytest_sessionstart

`kubetest.plugin.`**`pytest_runtest_setup`**(*item*)
Run setup actions to prepare the test case.

See also:

https://docs.pytest.org/en/latest/reference.html#_pytest.hookspec.pytest_runtest_setup

`kubetest.plugin.`**`pytest_runtest_teardown`**(*item*)
Run teardown actions to clean up the test client.

See also:

https://docs.pytest.org/en/latest/reference.html#_pytest.hookspec.pytest_runtest_teardown

kubetest.plugin.**pytest_runtest_makereport**(*item*, *call*)
Create a test report for the test case. If the test case was found to fail, this will log out the container logs to provide more debugging context.

See also:

https://docs.pytest.org/en/latest/reference.html#_pytest.hookspec.pytest_runtest_makereport

kubetest.plugin.**pytest_keyboard_interrupt**()
Clean up the cluster from kubetest artifacts if the tests are manually terminated via keyboard interrupt.

See also:

https://docs.pytest.org/en/latest/reference.html#_pytest.hookspec.pytest_keyboard_interrupt

class kubetest.plugin.**ClusterInfo**(*current*, *config*)
Bases: object

Information about the cluster the kubetest is being run on.

This info is gathered from the current context and the loaded configuration.

Variables

- **cluster** – The name of the cluster set for the current context.
- **user** – The name of the user set for the current context.
- ***context*** – The name of the current context.
- **host** – API server address.
- **verify_ssl** – SSL certificate verification when calling the API.

kubetest.plugin.**clusterinfo**(*kubeconfig*) → kubetest.plugin.ClusterInfo
Get a ClusterInfo instance which provides basic information about the cluster the tests are being run on.

kubetest.plugin.**kubeconfig**(*request*) → Optional[str]
Return the name of the configured kube config file loaded for the tests.

kubetest.plugin.**kubecontext**(*request*) → Optional[str]
Return the context in the kubeconfig to use for the tests.

When None, use the current context as set in the kubeconfig.

kubetest.plugin.**kube**(*kubeconfig*, *kubecontext*, *request*) → kubetest.client.TestClient
Return a client for managing a Kubernetes cluster for testing.

kubetest.utils module

Utility functions for kubetest.

kubetest.utils.**new_namespace**(*test_name: str*) → str
Create a new namespace for the given test name.

Kubernetes namespace names follow a DNS-1123 label that consists of lower case alphanumeric characters or '-' and must start with an alphanumeric.

The test name and current timestamp are formatted to comply to this spec and appended to the 'kubetest' prefix.

Parameters **test_name** – The name of the test case for the namespace.

Returns The namespace name.

kubetest.utils.**selector_string**(*selectors: Mapping[str, str]*) → str
Create a selector string from the given dictionary of selectors.

Parameters **selectors** – The selectors to stringify.

Returns The selector string for the given dictionary.

kubetest.utils.**selector_kwargs**(*fields: Mapping[str, str] = None*, *labels: Mapping[str, str] = None*) → Dict[str, str]
Create a dictionary of kwargs for Kubernetes object selectors.

Parameters

- **fields** – A mapping of fields used to restrict the returned collection of Objects to only those which match these field selectors. By default, no restricting is done.
- **labels** – A mapping of labels used to restrict the returned collection of Objects to only those which match these label selectors. By default, no restricting is done.

Returns A dictionary that can be used as kwargs for many Kubernetes API calls for label and field selectors.

kubetest.utils.**wait_for_condition**(*condition: kubetest.condition.Condition*, *timeout: int = None*, *interval: Union[int, float] = 1*, *fail_on_api_error: bool = True*) → None
Wait for a condition to be met.

Parameters

- **condition** – The Condition to wait for.
- **timeout** – The maximum time to wait, in seconds, for the condition to be met. If unspecified, this function will wait indefinitely. If specified and the timeout is met or exceeded, a TimeoutError will be raised.
- **interval** – The time, in seconds, to wait before re-checking the condition.
- **fail_on_api_error** – Fail the condition checks if a Kubernetes API error is incurred. An API error can be raised for a number of reasons, including a Pod being restarted and temporarily unavailable. Disabling this will cause those errors to be ignored, allowing the check to continue until timeout or resolution. (default: True).

Raises `TimeoutError` – The specified timeout was exceeded.

Module contents

kubetest – a Kubernetes integration test framework in Python.

Python Module Index

k

Index

Symbols

A

C

D

G

H

I

K

L

N

O

P

R

S

T

V

W

Y