Create a CustomResourceDefinition

When you create a new CustomResourceDefinition (CRD), the Kubernetes API Server creates a new RESTful resource path for each version you specify. The CRD can be either namespaced or cluster-scoped, as specified in the CRD's scope field. As with existing built-in objects, deleting a namespace deletes all custom objects in that namespace. CustomResourceDefinitions themselves are non-namespaced and are available to all namespaces.

For example, if you save the following CustomResourceDefinition to resourcedefinition.yaml:

apiVersion: apiextensions.k8s.io/v1

kind: CustomResourceDefinition

metadata:

# name must match the spec fields below, and be in the form: <plural>.<group>

name: crontabs.stable.example.com

spec:

# group name to use for REST API: /apis/<group>/<version>

group: stable.example.com

# list of versions supported by this CustomResourceDefinition

versions:

- name: v1

# Each version can be enabled/disabled by Served flag.

served: true

# One and only one version must be marked as the storage version.

storage: true

schema:

openAPIV3Schema:

type: object

properties:

spec:

type: object

properties:

cronSpec:

type: string

image:

type: string

replicas:

type: integer

# either Namespaced or Cluster

scope: Namespaced

names:

# plural name to be used in the URL: /apis/<group>/<version>/<plural>

plural: crontabs

# singular name to be used as an alias on the CLI and for display

singular: crontab

# kind is normally the CamelCased singular type. Your resource manifests use this.

kind: CronTab

# shortNames allow shorter string to match your resource on the CLI

shortNames:

- ct

and create it:

kubectl apply -f resourcedefinition.yaml

Then a new namespaced RESTful API endpoint is created at:

/apis/stable.example.com/v1/namespaces/\*/crontabs/...

This endpoint URL can then be used to create and manage custom objects. The kind of these objects will be CronTab from the spec of the CustomResourceDefinition object you created above.

It might take a few seconds for the endpoint to be created. You can watch the Established condition of your CustomResourceDefinition to be true or watch the discovery information of the API server for your resource to show up.

Create custom objects

After the CustomResourceDefinition object has been created, you can create custom objects. Custom objects can contain custom fields. These fields can contain arbitrary JSON. In the following example, the cronSpec and image custom fields are set in a custom object of kind CronTab. The kind CronTab comes from the spec of the CustomResourceDefinition object you created above.

If you save the following YAML to my-crontab.yaml:

apiVersion: "stable.example.com/v1"

kind: CronTab

metadata:

name: my-new-cron-object

spec:

cronSpec: "\* \* \* \* \*/5"

image: my-awesome-cron-image

and create it:

kubectl apply -f my-crontab.yaml

You can then manage your CronTab objects using kubectl. For example:

kubectl get crontab

Should print a list like this:

NAME AGE

my-new-cron-object 6s

Resource names are not case-sensitive when using kubectl, and you can use either the singular or plural forms defined in the CRD, as well as any short names.

You can also view the raw YAML data:

kubectl get ct -o yaml

You should see that it contains the custom cronSpec and image fields from the YAML you used to create it:

apiVersion: v1

kind: List

items:

- apiVersion: stable.example.com/v1

kind: CronTab

metadata:

creationTimestamp: 2017-05-31T12:56:35Z

generation: 1

name: my-new-cron-object

namespace: default

resourceVersion: "285"

uid: 9423255b-4600-11e7-af6a-28d2447dc82b

spec:

cronSpec: '\* \* \* \* \*/5'

image: my-awesome-cron-image

metadata:

resourceVersion: ""

Delete a CustomResourceDefinition

When you delete a CustomResourceDefinition, the server will uninstall the RESTful API endpoint and delete all custom objects stored in it.

kubectl delete -f resourcedefinition.yaml

kubectl get crontabs

Error from server (NotFound): Unable to list {"stable.example.com" "v1" "crontabs"}: the server could not find the requested resource (get crontabs.stable.example.com)

If you later recreate the same CustomResourceDefinition, it will start out empty.

Specifying a structural schema

CustomResources store structured data in custom fields (alongside the built-in fields apiVersion, kind and metadata, which the API server validates implicitly). With OpenAPI v3.0 validation a schema can be specified, which is validated during creation and updates, compare below for details and limits of such a schema.

With apiextensions.k8s.io/v1 the definition of a structural schema is mandatory for CustomResourceDefinitions. In the beta version of CustomResourceDefinition, the structural schema was optional.

A structural schema is an OpenAPI v3.0 validation schema which:

specifies a non-empty type (via type in OpenAPI) for the root, for each specified field of an object node (via properties or additionalProperties in OpenAPI) and for each item in an array node (via items in OpenAPI), with the exception of:

a node with x-kubernetes-int-or-string: true

a node with x-kubernetes-preserve-unknown-fields: true

for each field in an object and each item in an array which is specified within any of allOf, anyOf, oneOf or not, the schema also specifies the field/item outside of those logical junctors (compare example 1 and 2).

does not set description, type, default, additionalProperties, nullable within an allOf, anyOf, oneOf or not, with the exception of the two pattern for x-kubernetes-int-or-string: true (see below).

if metadata is specified, then only restrictions on metadata.name and metadata.generateName are allowed.

Non-structural example 1:

allOf:

- properties:

foo:

...

conflicts with rule 2. The following would be correct:

properties:

foo:

...

allOf:

- properties:

foo:

...

Non-structural example 2:

allOf:

- items:

properties:

foo:

...

conflicts with rule 2. The following would be correct:

items:

properties:

foo:

...

allOf:

- items:

properties:

foo:

...

Non-structural example 3:

properties:

foo:

pattern: "abc"

metadata:

type: object

properties:

name:

type: string

pattern: "^a"

finalizers:

type: array

items:

type: string

pattern: "my-finalizer"

anyOf:

- properties:

bar:

type: integer

minimum: 42

required: ["bar"]

description: "foo bar object"

is not a structural schema because of the following violations:

the type at the root is missing (rule 1).

the type of foo is missing (rule 1).

bar inside of anyOf is not specified outside (rule 2).

bar's type is within anyOf (rule 3).

the description is set within anyOf (rule 3).

metadata.finalizers might not be restricted (rule 4).

In contrast, the following, corresponding schema is structural:

type: object

description: "foo bar object"

properties:

foo:

type: string

pattern: "abc"

bar:

type: integer

metadata:

type: object

properties:

name:

type: string

pattern: "^a"

anyOf:

- properties:

bar:

minimum: 42

required: ["bar"]

Violations of the structural schema rules are reported in the NonStructural condition in the CustomResourceDefinition.