Learning Objectives

By the end of this chapter, you should be able to:

- Grow available Kubernetes objects.
- Deploy a new Custom Resource Definition.
- Deploy a new resource and API endpoint.
- Discuss aggregated APIs.

CUSTOM RESOURCE DEFINITIONS

Custom Resources

We have been working with built-in resources, or API endpoints. The flexibility of Kubernetes allows for the dynamic addition of new resources as well. Once these Custom Resources have been added, the objects can be created and accessed using standard calls and commands, like **kubectl**. The creation of a new object stores new structured data in the etcd database and allows access via kube-apiserver.

To make a new custom resource part of a declarative API, there needs to be a controller to retrieve the structured data continually and act to meet and maintain the declared state. This controller, or operator, is an agent that creates and manages one or more instances of a specific stateful application. We have worked with built-in controllers such as Deployments, DaemonSets and other resources.

The functions encoded into a custom operator should be all the tasks a human would need to perform if deploying the application outside of Kubernetes. The details of building a custom controller are outside the scope of this course, and thus, not included.

There are two ways to add custom resources to your Kubernetes cluster. The easiest way, but less flexible, is by adding a Custom Resource Definition (CRD) to the cluster. The second way, which is more flexible, is the use of Aggregated APIs (AA), which requires a new API server to be written and added to the cluster.

Either way of adding a new object to the cluster, as distinct from a built-in resource, is called a Custom Resource.

If you are using RBAC for authorization, you probably will need to grant access to the new CRD resource and controller. If using an Aggregated API, you can use the same or a different authentication process.

Custom Resource Definitions

As we have already learnt, the decoupled nature of Kubernetes depends on a collection of watcher loops, or controllers, interrogating the kube-apiserver to determine if a particular configuration is true. If the current state does not match the declared state, the controller makes API calls to modify the state until they do match. If you add a new API object and controller, you can use the existing kube-apiserver to monitor and control the object. The addition of a Custom Resource Definition will be added to the cluster API path, currently under **apiextensions.k8s.io/v1**.

While this is the easiest way to add a new object to the cluster, it may not be flexible enough for your needs. Only the existing API functionality can be used. Objects must respond to REST requests and have their configuration state validated and stored in the same manner as built-in objects. They would also need to exist with the protection rules of built-in objects.

A CRD allows the resource to be deployed in a namespace or be available in the entire cluster. The YAML file sets this with the **scope:** parameter, which can be set to **Namespaced** or **Cluster**.

Prior to v1.8, there was a resource type called ThirdPartyResource (TPR). This has been deprecated and is no longer available. All resources will need to be rebuilt as CRD. After upgrading, existing TPRs will need to be removed and replaced by CRDs such that the API URL points to functional objects.

Configuration Example

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: backups.stable.linux.com
spec:
  group: stable.linux.com
  version: v1
  scope: Namespaced
  names:
    plural: backups
    singular: backup
    shortNames:
    - bks
    kind: BackUp
```

Explanation of Configuration Fiels

- apiVersion

This should match the current level of stability, which is **apiextensions.k8s.io/v1**.

- kind: CustomResourceDefinition

The object type being inserted by the kube-apiserver.

- name: backups.stable.linux.com

The name must match the **spec** field declared later. The syntax must be <plural name>.<group>

- group: stable.linux.com

The group name will become part of the REST API under **/apis/<group>/<version>** or **/apis/stable/v1** in this case with the version set to v1.

- scope

Determines if the object exists in a single namespace or is cluster-wide.

- plural

Defines the last part of the API URL, such as **apis/stable/v1/backups**.

- singular and shortNames

They represent the name displayed and make CLI usage easier.

- kind

A CamelCased singular type used in resource manifests.

New Object Configuration

```
apiVersion: "stable.linux.com/v1"
kind: BackUp
metadata:
  name: a-backup-object
spec:
  timeSpec: "* * * * */5"
  image: linux-backup-image
replicas: 5
```

Note that the **apiVersion** and **kind** match the CRD we created in a previous step. The **spec** parameters depend on the controller.

The object will be evaluated by the controller. If the syntax, such as **timeSpec**, does not match the expected value, you will receive and error, should validation be configured. Without validation, only the existence of the variable is checked, not its details.

Optional Hooks

Just as with built-in objects, you can use an asynchronous pre-delete hook known as a **Finalizer**. If an API **delete** request is received, the object metadata field **metadata.deletionTimestamp** is updated. The controller then triggers whichever finalizer has been configured. When the finalizer completes, it is removed from the list. The controller continues to complete and remove finalizers until the string is empty. Then, the object itself is deleted.

Finalizer:

```
metadata:
  finalizers:
  - finalizer.stable.linux.com
```

Validation:

```
validation:
    openAPIV3Schema:
      properties:
        spec:
          properties:
            timeSpec:
              type: string
              pattern: '^(\d+|\*)(/\d+)?(\s+(\d+|\*)(/\d+)?){4}$'
            replicas:
              type: integer
              minimum: 1
              maximum: 10
```

A feature in beta starting with v1.9 allows for validation of custom objects via the OpenAPI v3 schema. This will check various properties of the object configuration being passed by the API server. In the example above, the **timeSpec** must be a string matching a particular pattern and the number of allowed replicas is between 1 and 10. If the validation does not match, the error returned is the failed line of validation.

Understanding Aggregated APIs

The use of Aggregated APIs allows adding additional Kubernetes-type API servers to the cluster. The added server acts as a subordinate to kube-apiserver, which, as of v1.7, runs the aggregation layer in-process. When an extension resource is registered, the aggregation layer watches a passed URL path and proxies any requests to the newly registered API service.

The aggregation layer is easy to enable. Edit the flags passed during startup of the kube-apiserver to include **--enable-aggregator-routing=true**. Some vendors enable this feature by default.

The creation of the exterior can be done via YAML configuration files or APIs. Configuring TLS authorization between components and RBAC rules for various new objects is also required. A sample API server is available on GitHub. A project currently in the incubation stage is an API server builder which should handle much of the security and connection configuration.