resources managed by the controller. In the former case, we already have all types available in the Kubernetes client library of our choice. For the CRD case, we don't have the type information out of the box, and we can either use a schemaless approach for managing CRD resources, or define the custom types on our own, possibly based on an OpenAPI schema contained in the CRD definition. Support for typed CRDs varies by client library and framework used.

Figure 23-1 shows our *Controller* and *Operator* categorization starting from simpler resource definition options to more advanced with the boundary between *Controller* and *Operator* being the use of custom resources.

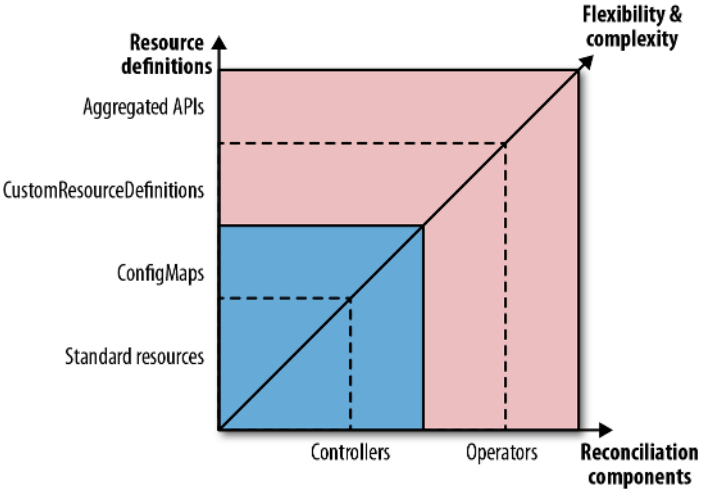

Figure 23-1. Spectrum of Controllers and Operators

For *Operators*, there is even a more advanced Kubernetes extension hook option. When Kubernetes-managed CRDs are not sufficient to represent a problem domain, you can extend the Kubernetes API with an own aggregation layer. We can add a custom implemented `APIService` resource as a new URL path to the Kubernetes API.

To connect a given Service with name `custom-api-server` and backed by a Pod with your service, you can use a resource like that shown in Example 23-4.

Example 23-4. API aggregation with a custom APIService

```
apiVersion: apiregistration.k8s.io/v1beta1
kind: APIService
metadata:
  name: v1alpha1.sample-api.k8spatterns.io
spec:
  group: sample-api.k8spattterns.io
  service:
```