Camel has a Kubernetes connector that also provides leader election and singleton capabilities. This connector goes a step further, and rather than accessing the Etcd API directly, it uses Kubernetes APIs to leverage ConfigMaps as a distributed lock. It relies on Kubernetes optimistic locking guarantees for editing resources such as ConfigMaps where only one Pod can update a ConfigMap at a time.

The Camel implementation uses this guarantee to ensure only one Camel route instance is active, and any other instance has to wait and acquire the lock before activating. It is a custom implementation of a lock, but achieves the same goal: when there are multiple Pods with the same Camel application, only one of them becomes the active singleton, and the others wait in passive mode.

An implementation with ZooKeeper, Etcd, or any other distributed lock implementation would be similar to the one described: only one instance of the application becomes the leader and activates itself, and other instances are passive and wait for the lock. This ensures that even if multiple Pod replicas are started and all are healthy, up, and running, only one service is active and performs the business functionality as a singleton, and other instances are waiting to acquire the lock in case the master fails or shuts down.

Pod Disruption Budget

While *Singleton Service* and leader election try to limit the maximum number of instances a service is running at a time, the PodDisruptionBudget functionality of Kubernetes provides a complementary and somewhat opposite functionality—limiting the number of instances that are simultaneously down for maintenance.

At its core, PodDisruptionBudget ensures a certain number or percentage of Pods will not voluntarily be evicted from a node at any one point in time. *Voluntary* here means an eviction that can be delayed for a particular time—for example, when it is triggered by draining a node for maintenance or upgrade (`kubectl drain`), or a cluster scaling down, rather than a node becoming unhealthy, which cannot be predicted or controlled.

The PodDisruptionBudget in Example 10-1 applies to Pods that match its selector and ensures two Pods must be available all the time.

Example 10-1. PodDisruptionBudget

```
apiVersion: policy/v1beta1
kind: PodDisruptionBudget
metadata:
  name: random-generator-pdb
spec:
  selector:
    matchLabels:                    ❶
```