thanks to the controller performing health checks as described in Chapter 4, *Health Probe* and healing the Pod in case of failures.

The main thing to keep an eye on with this approach is the replica count, which should not be increased accidentally, as there is no platform-level mechanism to prevent a change of the replica count.

It's not entirely true that only one instance is running at all times, especially when things go wrong. Kubernetes primitives such as ReplicaSet, favor availability over consistency—a deliberate decision for achieving highly available and scalable distributed systems. That means a ReplicaSet applies "at least" rather than "at most" semantics for its replicas. If we configure a ReplicaSet to be a singleton with `replicas: 1`, the controller makes sure at least one instance is always running, but occasionally it can be more instances.

The most popular corner case here occurs when a node with a controller-managed Pod becomes unhealthy and disconnects from the rest of the Kubernetes cluster. In this scenario, a ReplicaSet controller starts another Pod instance on a healthy node (assuming there is enough capacity), without ensuring the Pod on the disconnected node is shut down. Similarly, when changing the number of replicas or relocating Pods to different nodes, the number of Pods can temporarily go above the desired number. That temporary increase is done with the intention of ensuring high availability and avoiding disruption, as needed for stateless and scalable applications.

Singletons can be resilient and recover, but by definition, are not highly available. Singletons typically favor consistency over availability. The Kubernetes resource that also favors consistency over availability and provides the desired strict singleton guarantees is the StatefulSet. If ReplicaSets do not provide the desired guarantees for your application, and you have strict singleton requirements, StatefulSets might be the answer. StatefulSets are intended for stateful applications and offer many features, including stronger singleton guarantees, but they come with increased complexity as well. We discuss concerns around singletons and cover StatefulSets in more detail in Chapter 11, *Stateful Service*.

Typically, singleton applications running in Pods on Kubernetes open outgoing connections to message brokers, relational databases, file servers, or other systems running on other Pods or external systems. However, occasionally, your singleton Pod may need to accept incoming connections, and the way to enable that on Kubernetes is through the Service resource.

We cover Kubernetes Services in depth in Chapter 12, *Service Discovery*, but let's discuss briefly the part that applies to singletons here. A regular Service (with `type: ClusterIP`) creates a virtual IP and performs load balancing among all the Pod instances that its selector matches. But a singleton Pod managed through a StatefulSet has only one Pod and a stable network identity. In such a case, it is better to create a