



cloudnative-pg / cloudnative-pg



<> Code



Issues

184



Pull requests

106



Discussions



Actions



Projects

Fixes Issue 7793: Stuck reconciliation fixes #7794

Edit

<> Code



Try the new experience

Open

jmealo wants to merge 1 commit into cloudnative-pg:main from jmealo:feat/fix-issue-7793

Conversation

14

Commits

1

Checks

1

Files changed

6

Changes from all commits

File filter

Conversations

Jump to



241 internal/controller/cluster_controller.go

```
723     return nil
724 }
725
726 + // checkAndClearStuckScalingPhase checks if the cluster is stuck in a
727 + // but already has the correct number of instances, and clears the phase
728 + // if needed
729 + func (r *ClusterReconciler) checkAndClearStuckScalingPhase(
730 +     ctx context.Context,
731 +     cluster *apiv1.Cluster,
732 +     resources *managedResources,
733 + ) error {
734 +     contextLogger := log.FromContext(ctx)
735 +
736 +     // Check if we're in a scaling-related phase
737 +     scalingPhases := []string{
```



jmealo 10 minutes ago · edited

Contributor

Author

Do these need to come from a const/be-extracted to a helper function `isInScalingPhase` ?



Reply...

Resolve conversation

```
737 +     "Creating a new replica",
738 +     "Scaling up",
739 +     "Scaling down",
740 + }
741 +
742 + isInScalingPhase := false
743 + for _, phase := range scalingPhases {
744 +     if cluster.Status.Phase == phase {
745 +         isInScalingPhase = true
746 +         break
747 +     }
748 + }
749 +
750 + if !isInScalingPhase {
751 +     return nil // Not in a scaling phase, nothing to check
752 + }
753 +
754 + // Check if we already have the correct number of instances
755 + currentInstances := len(resources.instances.Items)
756 + desiredInstances := cluster.Spec.Instances
757 +
758 + contextLogger.Debug("Checking scaling phase consistency",
759 +     "phase", cluster.Status.Phase,
760 +     "currentInstances", currentInstances,
761 +     "desiredInstances", desiredInstances,
762 +     "statusInstances", cluster.Status.Instances)
763 +
764 + // If we have the right number of instances and no running jobs,
765 + // clear the scaling phase
766 + if currentInstances == desiredInstances &&
767 +     len(resources.runningJobNames()) == 0 {
768 +     contextLogger.Info("Clearing stuck scaling phase - cluster has
769 + correct number of instances",
770 +         "phase", cluster.Status.Phase,
771 +         "instances", currentInstances)
772 +
773 +     r.Recorder.Eventf(cluster, "Normal", "ScalingPhaseCleared",
774 +         "Cleared stuck scaling phase '%s' - cluster has correct
775 + number of instances (%d)",
776 +         cluster.Status.Phase, currentInstances)
777 +
778 +     // Clear the phase to allow normal operation
779 +     if err := r.RegisterPhase(ctx, cluster, apiv1.PhaseHealthy, "");
780 + err != nil {
781 +         return fmt.Errorf("failed to clear stuck scaling phase: %w",
782 + err)
```

```
777 +     }
778 + }
779 +
780 +     return nil
781 + }
782 +
783 + // clearStuckScalingPhaseAfterJobDeletion clears scaling phases after
784 + // deleting failed jobs
785 + func (r *ClusterReconciler) clearStuckScalingPhaseAfterJobDeletion(
786 +     ctx context.Context,
787 +     cluster *apiv1.Cluster,
788 + ) error {
789 +     contextLogger := log.FromContext(ctx)
790 +     // Check if we're in a scaling-related phase that might be stuck due
791 +     // to failed jobs
```

jmealo 7 minutes ago

Contributor

Author

TODO: Extract this to `isInScalingPhase` method.



Reply...

Resolve conversation

```
791 +     scalingPhases := []string{
792 +         "Creating a new replica",
793 +         "Scaling up",
794 +         "Scaling down",
795 +     }
796 +
797 +     isInScalingPhase := false
798 +     for _, phase := range scalingPhases {
799 +         if cluster.Status.Phase == phase {
800 +             isInScalingPhase = true
801 +             break
802 +         }
803 +     }
804 +
805 +     if !isInScalingPhase {
806 +         return nil // Not in a scaling phase
807 +     }
808 +
809 +     contextLogger.Info("Clearing scaling phase after job deletion to
810 +     allow retry",
811 +         "phase", cluster.Status.Phase)
```

```

811 +
812 +     r.Recorder.Eventf(cluster, "Normal", "ScalingPhaseReset",
813 +         "Reset scaling phase '%s' after deleting failed job to allow
retry",
814 +         cluster.Status.Phase)
815 +
816 +     // Clear the phase reason to allow the scaling operation to be
retried
817 +     // We don't set it to healthy here because the scaling operation
should be retried
818 +     cluster.Status.PhaseReason = ""
819 +
820 +     return r.Status().Update(ctx, cluster)
821 + }
822 +
726 823 // reconcileResources updates all the objects managed by the controller
727 824 func (r *ClusterReconciler) reconcileResources(
728 825     ctx context.Context, cluster *apiv1.Cluster,
731 828     contextLogger := log.FromContext(ctx)
732 829     runningJobs := resources.runningJobNames()
733 830
831 +     // Check if we're stuck in a scaling phase but already have the
correct number of instances
832 +     if err := r.checkAndClearStuckScalingPhase(ctx, cluster, resources);
err != nil {
833 +         contextLogger.Error(err, "Error while checking stuck scaling
phase")
834 +         return ctrl.Result{}, err
835 +     }
836 +
734 837 // Act on Pods and PVCs only if there is nothing that is currently
being created or deleted
735 838
736 839 if len(runningJobs) > 0 {
840 +     // Let's check for failed or stuck jobs and handle them
841 +     stuckJobTimeout := 10 * time.Minute // Jobs stuck for more than
10 minutes are considered failed

```

jmealo 7 minutes ago

Contributor

Author

This feels arbitrary but I'm not sure what the cut-off should be?



Reply...

Resolve conversation

```
842 +
843 +     for _, job := range resources.jobs.Items {
844 +         if !utils.IsJobFailedOrStuck(job, stuckJobTimeout) {
845 +             continue
846 +         }
847 +
848 +         // This job is failed or stuck. We need to record the event,
            delete the job
849 +         // and reconcile again
850 +         if utils.IsJobFailed(job) {
851 +             r.Recorder.Eventf(cluster, "Warning", "FailingJob",
852 +                 "Job %v is failing, deleting it to retry", job.Name)
853 +             contextLogger.Warning("Deleting failed job", "jobName",
            job.Name)
854 +         } else {
855 +             r.Recorder.Eventf(cluster, "Warning", "StuckJob",
856 +                 "Job %v is stuck in pending state, deleting it to
            retry", job.Name)
857 +             contextLogger.Warning("Deleting stuck job", "jobName",
            job.Name,
858 +                 "creationTime", job.CreationTimestamp.Time,
859 +                 "active", job.Status.Active,
860 +                 "succeeded", job.Status.Succeeded,
861 +                 "failed", job.Status.Failed)
862 +         }
863 +
864 +         if err := r.Delete(ctx, &job,
            client.PropagationPolicy(metav1.DeletePropagationBackground)); err != nil
            {
865 +             contextLogger.Error(err, "Error while deleting
            failed/stuck job", "jobName", job.Name)
866 +             return ctrl.Result{}, err
867 +         }
868 +
869 +         // Clear any stuck scaling phases since we're cleaning up
            failed jobs
870 +         if err := r.clearStuckScalingPhaseAfterJobDeletion(ctx,
            cluster); err != nil {
871 +             contextLogger.Error(err, "Error clearing stuck scaling
            phase after job deletion")
872 +         }
873 +
874 +         // Requeue the reconciliation to recreate the job
875 +         return ctrl.Result{Requeue: true}, nil
876 +     }
877 +
878 +     // Check for equilibrium state - if we have been waiting for jobs
            for too long
```

```

879 +         // and the cluster state hasn't changed, we might be in a stuck
      state
880 +         if err := r.checkForEquilibriumState(ctx, cluster, resources);
      err != nil {
881 +             contextLogger.Warning("Detected potential equilibrium state",
      "error", err)
882 +             // Continue with normal processing to attempt recovery
883 +         }
884 +

```

```

737 885         contextLogger.Debug("A job is currently running. Waiting",
      "runningJobs", runningJobs)
738 886         return ctrl.Result{RequeueAfter: 5 * time.Second}, nil
739 887     }
1561 1709
1562 1710     return nil
1563 1711 }

```

```

1712 +
1713 + // checkForEquilibriumState detects when the cluster is stuck in an
      equilibrium state
1714 + // where jobs are running but making no progress, potentially due to
      missing PVCs or other issues
1715 + func (r *ClusterReconciler) checkForEquilibriumState(
1716 +     ctx context.Context,
1717 +     cluster *apiv1.Cluster,
1718 +     resources *managedResources,
1719 + ) error {
1720 +     contextLogger := log.FromContext(ctx)
1721 +
1722 +     // Check if we have long-running jobs that might be stuck
1723 +     equilibriumTimeout := 15 * time.Minute // Consider equilibrium after
      15 minutes

```



jmealo 5 minutes ago

Contributor

Author

This is somewhat arbitrary, should this be configurable? What would be a sane value.



Reply...

Resolve conversation

```

1724 +
1725 +     for _, job := range resources.jobs.Items {
1726 +         // Skip completed or failed jobs
1727 +         if utils.IsJobComplete(job) || utils.IsJobFailed(job) {
1728 +             continue
1729 +         }

```

```
1730 +
1731 +     // Check if job has been running for too long without progress
1732 +     if
1733 +         job.CreationTimestamp.Add(equilibriumTimeout).Before(time.Now()) {
1734 +         // Check if job has no active pods (stuck in pending)
1735 +         if job.Status.Active == 0 && job.Status.Succeeded == 0 &&
1736 +             job.Status.Failed == 0 {
```

jmealo 4 minutes ago

Contributor

Author

If the job is failing to schedule for a long time, I'd be more inclined to call it stuck and consider it a failure than if it gets scheduled.



Reply...

Resolve conversation

```
1735 +         contextLogger.Warning("Detected job in equilibrium state
1736 +             - no pods created",
1737 +             "jobName", job.Name,
1738 +             "creationTime", job.CreationTimestamp.Time,
1739 +             "age", time.Since(job.CreationTimestamp.Time))
1740 +         // Check if there are missing PVCs that might be causing
1741 +         the issue
1742 +         if err := r.checkForMissingPVCs(ctx, cluster, &job); err
1743 +             != nil {
1744 +             return fmt.Errorf("equilibrium state detected: job %s
1745 +                 stuck due to missing PVCs: %w",
```

jmealo 4 minutes ago

Contributor

Author

It might be useful to surface the different error types in the cluster status somehow? 🤔



Reply...

Resolve conversation

```
1743 +             job.Name, err)
1744 +         }
1745 +
1746 +         return fmt.Errorf("equilibrium state detected: job %s has
1747 +             been pending for %v without creating pods",
```

```
1747 +             job.Name, time.Since(job.CreationTimestamp.Time))
1748 +         }
1749 +
1750 +         // Check if job has active pods but they're not making
           progress
1751 +         if job.Status.Active > 0 {
1752 +             contextLogger.Warning("Detected job with long-running
           active pods",
1753 +                 "jobName", job.Name,
1754 +                 "activePods", job.Status.Active,
1755 +                 "age", time.Since(job.CreationTimestamp.Time))
1756 +         }
1757 +     }
1758 + }
1759 +
1760 + return nil
1761 + }
1762 +
1763 + // checkForMissingPVCs checks if a job is stuck due to missing PVCs
1764 + func (r *ClusterReconciler) checkForMissingPVCs(
1765 +     ctx context.Context,
1766 +     cluster *apiv1.Cluster,
1767 +     job *batchv1.Job,
1768 + ) error {
1769 +     contextLogger := log.FromContext(ctx)
1770 +
1771 +     // Extract PVC names from job template
1772 +     var requiredPVCs []string
1773 +     for _, volume := range job.Spec.Template.Spec.Volumes {
1774 +         if volume.PersistentVolumeClaim != nil {
1775 +             requiredPVCs = append(requiredPVCs,
           volume.PersistentVolumeClaim.ClaimName)
1776 +         }
1777 +     }
1778 +
1779 +     if len(requiredPVCs) == 0 {
1780 +         return nil // No PVCs required
1781 +     }
1782 +
1783 +     // Check if required PVCs exist
1784 +     var missingPVCs []string
1785 +     for _, pvcName := range requiredPVCs {
1786 +         pvc := &corev1.PersistentVolumeClaim{}
1787 +         err := r.Get(ctx, types.NamespacedName{
1788 +             Name:      pvcName,
1789 +             Namespace: cluster.Namespace,
1790 +         }, pvc)
1791 +
1792 +         if apierrs.IsNotFound(err) {
```

```

1793 +         missingPVCs = append(missingPVCs, pvcName)
1794 +     } else if err != nil {
1795 +         contextLogger.Error(err, "Error checking PVC existence",
1796 +             "pvcName", pvcName)
1797 +     }
1798 + }
1799 + if len(missingPVCs) > 0 {
1800 +     return fmt.Errorf("missing PVCs: %v", missingPVCs)
1801 + }
1802 +
1803 + return nil
1804 + }

```

435 internal/controller/cluster_controller_stuck_reconciliation_test.go

```

... @@ -0,0 +1,435 @@
1 + /*
2 + Copyright © contributors to CloudNativePG, established as
3 + CloudNativePG a Series of LF Projects, LLC.
4 +
5 + Licensed under the Apache License, Version 2.0 (the "License");
6 + you may not use this file except in compliance with the License.
7 + You may obtain a copy of the License at
8 +
9 + http://www.apache.org/licenses/LICENSE-2.0
10 +
11 + Unless required by applicable law or agreed to in writing, software
12 + distributed under the License is distributed on an "AS IS" BASIS,
13 + WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14 + See the License for the specific language governing permissions and
15 + limitations under the License.
16 +
17 + SPDX-License-Identifier: Apache-2.0
18 + */
19 +
20 + package controller
21 +
22 + import (
23 +     "context"
24 +     "time"
25 +
26 +     batchv1 "k8s.io/api/batch/v1"
27 +     corev1 "k8s.io/api/core/v1"
28 +     apierrs "k8s.io/apimachinery/pkg/api/errors"
29 +     metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
30 +     "k8s.io/apimachinery/pkg/types"
31 +     "k8s.io/client-go/tools/record"
32 +     "sigs.k8s.io/controller-runtime/pkg/client"

```

```

33 +     "sigs.k8s.io/controller-runtime/pkg/client/fake"
34 +
35 +     apiv1 "github.com/cloudnative-pg/cloudnative-pg/api/v1"
36 +     schemeBuilder "github.com/cloudnative-pg/cloudnative-
pg/internal/scheme"
37 +     "github.com/cloudnative-pg/cloudnative-pg/pkg/postgres"
38 +     "github.com/cloudnative-pg/cloudnative-pg/pkg/utils"
39 +
40 +     . "github.com/onsi/ginkgo/v2"
41 +     . "github.com/onsi/gomega"
42 +     fakediscovery "k8s.io/client-go/discovery/fake"
43 +     k8stesting "k8s.io/client-go/testing"
44 + )
45 +
46 + var _ = Describe("Stuck Reconciliation Recovery", func() {
47 +     var (
48 +         ctx      context.Context
49 +         reconciler *ClusterReconciler
50 +         cluster   *apiv1.Cluster
51 +         namespace string
52 +     )
53 +
54 +     BeforeEach(func() {
55 +         ctx = context.Background()
56 +         namespace = "test-namespace"
57 +
58 +         // Create a fake client with the scheme
59 +         scheme := schemeBuilder.BuildWithAllKnownScheme()
60 +
61 +         fakeClient := fake.NewClientBuilder().
62 +             WithScheme(scheme).
63 +             WithStatusSubresource(&apiv1.Cluster{}).
64 +             WithIndex(&batchv1.Job{}, jobOwnerKey, jobOwnerIndexFunc).
65 +             WithIndex(&corev1.Pod{}, podOwnerKey, func(rawObj
client.Object) []string {
66 +                 pod := rawObj.(*corev1.Pod)
67 +                 if ownerName, ok := IsOwnedByCluster(pod); ok {
68 +                     return []string{ownerName}
69 +                 }
70 +                 return nil
71 +             }).
72 +             WithIndex(&corev1.PersistentVolumeClaim{}, pvcOwnerKey,
func(rawObj client.Object) []string {
73 +                 persistentVolumeClaim := rawObj.
(*corev1.PersistentVolumeClaim)
74 +                 if ownerName, ok :=
IsOwnedByCluster(persistentVolumeClaim); ok {
75 +                     return []string{ownerName}
76 +                 }

```

```
77 +         return nil
78 +     }).
79 +     Build()
80 +
81 +     // Create fake discovery client
82 +     fakeDiscoveryClient := &fakeDiscovery.FakeDiscovery{
83 +         Fake: &k8stesting.Fake{
84 +             Resources: []*metav1.APIResourceList{},
85 +         },
86 +     }
87 +
88 +     // Create a fake event recorder
89 +     fakeRecorder := record.NewFakeRecorder(100)
90 +
91 +     // Create the reconciler
92 +     reconciler = &ClusterReconciler{
93 +         Client:      fakeClient,
94 +         Scheme:      scheme,
95 +         Recorder:     fakeRecorder,
96 +         DiscoveryClient: fakeDiscoveryClient,
97 +     }
98 +
99 +     // Create a test cluster
100 +    cluster = &apiv1.Cluster{
101 +        ObjectMeta: metav1.ObjectMeta{
102 +            Name:      "test-cluster",
103 +            Namespace: namespace,
104 +            UID:      "test-uid",
105 +        },
106 +        Spec: apiv1.ClusterSpec{
107 +            Instances: 3, // Start with 3 instances
108 +        },
109 +        Status: apiv1.ClusterStatus{
110 +            Instances:      3,
111 +            ReadyInstances: 3,
112 +            Phase:          apiv1.PhaseHealthy,
113 +        },
114 +    }
115 +
116 +    // Create the cluster in the fake client
117 +    Expect(reconciler.Create(ctx, cluster)).To(Succeed())
118 + })
119 +
120 + Describe("End-to-End Stuck Reconciliation Recovery", func() {
121 +     It("should handle scale up → fail → scale down scenario", func() {
122 +         By("Starting with a healthy 3-instance cluster")
123 +         Expect(cluster.Spec.Instances).To(Equal(3))
124 +         Expect(cluster.Status.Phase).To(Equal(apiv1.PhaseHealthy))
```

```

125 +
126 +         By("Scaling up to 4 instances")
127 +         cluster.Spec.Instances = 4
128 +         Expect(reconciler.Update(ctx, cluster)).To(Succeed())
129 +
130 +         By("Simulating the cluster entering 'Creating a new replica'
131 + phase")
131 +         cluster.Status.Phase = "Creating a new replica"
132 +         cluster.Status.PhaseReason = "Creating replica test-cluster-
133 + 4-snapshot-recovery"
133 +         Expect(reconciler.Status().Update(ctx,
134 + cluster)).To(Succeed())
134 +
135 +         By("Creating a stuck snapshot-recovery job")
136 +         stuckJob := &batchv1.Job{
137 +             ObjectMeta: metav1.ObjectMeta{
138 +                 Name:         "test-cluster-4-snapshot-recovery",
139 +                 Namespace: namespace,
140 +                 OwnerReferences: []metav1.OwnerReference{
141 +                     {
142 +                         APIVersion:
143 + apiv1.SchemeGroupVersion.String(),
143 +                         Kind:         apiv1.ClusterKind,
144 +                         Name:         cluster.Name,
145 +                         UID:         cluster.UID,
146 +                         Controller: &[]bool{true}[0],
147 +                     },
148 +                 },
149 +                 CreationTimestamp: metav1.Time{Time:
150 + time.Now().Add(-20 * time.Minute)}, // Old job
150 +             },
151 +             Spec: batchv1.JobSpec{
152 +                 Template: corev1.PodTemplateSpec{
153 +                     Spec: corev1.PodSpec{
154 +                         Volumes: []corev1.Volume{
155 +                             {
156 +                                 Name: "pgdata",
157 +                                 VolumeSource: corev1.VolumeSource{
158 +                                     PersistentVolumeClaim:
159 + &corev1.PersistentVolumeClaimVolumeSource{
159 +                                         ClaimName: "missing-pvc", //
160 +                                         This PVC doesn't exist
160 +                                     },
161 +                                 },
162 +                             },
163 +                         },
164 +                         Containers: []corev1.Container{
165 +                             {
166 +                                 Name: "postgres",

```

```
167 +             Image: "postgres:15",
168 +             },
169 +             },
170 +             },
171 +             },
172 +             },
173 +             Status: batchv1.JobStatus{
174 +                 Active:    0, // No active pods due to missing PVC
175 +                 Succeeded: 0,
176 +                 Failed:    0,
177 +             },
178 +         }
179 +         Expect(reconciler.Create(ctx, stuckJob)).To(Succeed())
180 +
181 +         By("Creating managed resources with the stuck job")
182 +         resources := &managedResources{
183 +             nodes: make(map[string]corev1.Node),
184 +             jobs: batchv1.JobList{
185 +                 Items: []batchv1.Job{*stuckJob},
186 +             },
187 +             instances: corev1.PodList{
188 +                 Items: []corev1.Pod{
189 +                     // Simulate 3 existing healthy instances
190 +                     createTestPod("test-cluster-1", namespace,
191 +                         cluster),
192 +                     createTestPod("test-cluster-2", namespace,
193 +                         cluster),
194 +                     createTestPod("test-cluster-3", namespace,
195 +                         cluster),
196 +                 },
197 +             },
198 +             pvcs: corev1.PersistentVolumeClaimList{
199 +                 Items: []corev1.PersistentVolumeClaim{
200 +                     // Only PVCs for existing instances, missing the
201 +                     // one for instance 4
202 +                     createTestPVC("test-cluster-1-pgdata", namespace,
203 +                         cluster),
204 +                     createTestPVC("test-cluster-2-pgdata", namespace,
205 +                         cluster),
206 +                     createTestPVC("test-cluster-3-pgdata", namespace,
207 +                         cluster),
208 +                 },
209 +             },
210 +         }
211 +
212 +         By("Testing stuck job handling through reconcileResources")
213 +         // The reconcileResources method checks for stuck jobs and
214 +         // deletes them
```

```
207 + // Pass an empty PostgresqlStatusList as it's not needed for
    stuck job detection
208 + var instancesStatus postgres.PostgresqlStatusList
209 + result, err := reconciler.reconcileResources(ctx, cluster,
    resources, instancesStatus)
210 + Expect(err).ToNot(HaveOccurred())
211 + Expect(result.Requeue).To(BeTrue(), "Should requeue after
    deleting stuck job")
212 +
213 + By("Verifying the stuck job was deleted")
214 + deletedJob := &batchv1.Job{}
215 + err = reconciler.Get(ctx, types.NamespacedName{
216 +     Name: stuckJob.Name,
217 +     Namespace: stuckJob.Namespace,
218 + }, deletedJob)
219 + Expect(apierrs.IsNotFound(err)).To(BeTrue(), "Stuck job
    should be deleted")
220 +
221 + By("Simulating user decision to scale down instead of
    retrying")
222 + // Refresh cluster state
223 + Expect(reconciler.Get(ctx, types.NamespacedName{
224 +     Name: cluster.Name,
225 +     Namespace: cluster.Namespace,
226 + }, cluster)).To(Succeed())
227 +
228 + // User scales down to 3 instances
229 + cluster.Spec.Instances = 3
230 + Expect(reconciler.Update(ctx, cluster)).To(Succeed())
231 +
232 + By("Running checkAndClearStuckScalingPhase with correct
    instance count")
233 + // Update resources to reflect no running jobs and correct
    instance count
234 + resources.jobs.Items = []batchv1.Job{} // No more jobs
235 +
236 + err = reconciler.checkAndClearStuckScalingPhase(ctx, cluster,
    resources)
237 + Expect(err).ToNot(HaveOccurred())
238 +
239 + By("Verifying the scaling phase was cleared")
240 + // Refresh cluster state
241 + Expect(reconciler.Get(ctx, types.NamespacedName{
242 +     Name: cluster.Name,
243 +     Namespace: cluster.Namespace,
244 + }, cluster)).To(Succeed())
245 +
246 + Expect(cluster.Status.Phase).To(Equal(apiv1.PhaseHealthy),
```

```

247 +         "Cluster phase should be cleared to healthy when instance
      count matches")
248 +     })
249 +
250 +     It("should detect and handle missing PVCs", func() {
251 +         By("Creating a job that requires a missing PVC")
252 +         jobWithMissingPVC := &batchv1.Job{
253 +             ObjectMeta: metav1.ObjectMeta{
254 +                 Name:         "test-job-missing-pvc",
255 +                 Namespace:      namespace,
256 +                 CreationTimestamp: metav1.Time{Time:
time.Now().Add(-20 * time.Minute)},
257 +             },
258 +             Spec: batchv1.JobSpec{
259 +                 Template: corev1.PodTemplateSpec{
260 +                     Spec: corev1.PodSpec{
261 +                         Volumes: []corev1.Volume{
262 +                             {
263 +                                 Name: "data",
264 +                                 VolumeSource: corev1.VolumeSource{
265 +                                     PersistentVolumeClaim:
&corev1.PersistentVolumeClaimVolumeSource{
266 +                                         ClaimName: "missing-pvc-
name",
267 +                                     },
268 +                                 },
269 +                             },
270 +                         },
271 +                         Containers: []corev1.Container{
272 +                             {Name: "test", Image: "test"},
273 +                         },
274 +                     },
275 +                 },
276 +             },
277 +             Status: batchv1.JobStatus{
278 +                 Active: 0,
279 +                 Succeeded: 0,
280 +                 Failed: 0,
281 +             },
282 +         }
283 +
284 +         By("Checking for missing PVCs")
285 +         err := reconciler.checkForMissingPVCs(ctx, cluster,
jobWithMissingPVC)
286 +         Expect(err).To(HaveOccurred())
287 +         Expect(err.Error()).To(ContainSubstring("missing PVCs:
[missing-pvc-name]"))
288 +     })
289 +

```

```
290 +         It("should detect equilibrium state", func() {
291 +             By("Creating a long-running job with no progress")
292 +             oldJob := &batchv1.Job{
293 +                 ObjectMeta: metav1.ObjectMeta{
294 +                     Name: "old-stuck-job",
295 +                     Namespace: namespace,
296 +                     CreationTimestamp: metav1.Time{Time:
297 +                         time.Now().Add(-20 * time.Minute)},
298 +                 },
299 +                 Status: batchv1.JobStatus{
300 +                     Active: 0,
301 +                     Succeeded: 0,
302 +                     Failed: 0,
303 +                 },
304 +             }
305 +             resources := &managedResources{
306 +                 jobs: batchv1.JobList{
307 +                     Items: []batchv1.Job{*oldJob},
308 +                 },
309 +             }
310 +
311 +             By("Checking for equilibrium state")
312 +             err := reconciler.checkForEquilibriumState(ctx, cluster,
313 +                 resources)
314 +             Expect(err).To(HaveOccurred())
315 +             Expect(err.Error()).To(ContainSubstring("equilibrium state
316 +                 detected"))
317 +             })
318 +
319 +             It("should clear scaling phase after job deletion", func() {
320 +                 By("Setting cluster to a scaling phase")
321 +                 cluster.Status.Phase = "Creating a new replica"
322 +                 cluster.Status.PhaseReason = "Creating replica test-cluster-
323 +                 4"
324 +                 Expect(reconciler.Status().Update(ctx,
325 +                     cluster)).To(Succeed())
326 +
327 +                 By("Clearing scaling phase after job deletion")
328 +                 err := reconciler.clearStuckScalingPhaseAfterJobDeletion(ctx,
329 +                     cluster)
330 +                 Expect(err).ToNot(HaveOccurred())
331 +
332 +                 By("Verifying phase reason was cleared")
333 +                 // Refresh cluster state
334 +                 Expect(reconciler.Get(ctx, types.NamespacedName{
335 +                     Name: cluster.Name,
336 +                     Namespace: cluster.Namespace,
337 +                 }, cluster)).To(Succeed())
```

```

333 +
334 +         Expect(cluster.Status.PhaseReason).To(BeEmpty(),
335 +             "Phase reason should be cleared to allow retry")
336 +     })
337 + })
338 +
339 + Describe("Job Utility Functions Integration", func() {
340 +     It("should correctly identify stuck jobs", func() {
341 +         By("Creating a stuck job")
342 +         stuckJob := batchv1.Job{
343 +             ObjectMeta: metav1.ObjectMeta{
344 +                 CreationTimestamp: metav1.Time{Time:
time.Now().Add(-15 * time.Minute)},
345 +             },
346 +             Status: batchv1.JobStatus{
347 +                 Active:    0,
348 +                 Succeeded: 0,
349 +                 Failed:    0,
350 +             },
351 +         }
352 +
353 +         By("Verifying job is detected as stuck")
354 +         isStuck := utils.IsJobStuck(stuckJob, 10*time.Minute)
355 +         Expect(isStuck).To(BeTrue())
356 +
357 +         isFailedOrStuck := utils.IsJobFailedOrStuck(stuckJob,
10*time.Minute)
358 +         Expect(isFailedOrStuck).To(BeTrue())
359 +     })
360 +
361 +     It("should correctly identify failed jobs", func() {
362 +         By("Creating a failed job")
363 +         failedJob := batchv1.Job{
364 +             Status: batchv1.JobStatus{
365 +                 Conditions: []batchv1.JobCondition{
366 +                     {
367 +                         Type:    batchv1.JobFailed,
368 +                         Status: corev1.ConditionTrue,
369 +                     },
370 +                 },
371 +             },
372 +         }
373 +
374 +         By("Verifying job is detected as failed")
375 +         isFailed := utils.IsJobFailed(failedJob)
376 +         Expect(isFailed).To(BeTrue())
377 +
378 +         isFailedOrStuck := utils.IsJobFailedOrStuck(failedJob,
10*time.Minute)

```

```
379 +         Expect(isFailedOrStuck).To(BeTrue())
380 +     })
381 + })
382 + })
383 +
384 + // Helper functions for creating test objects
385 +
386 + func createTestPod(name, namespace string, cluster *apiv1.Cluster)
    corev1.Pod {
387 +     return corev1.Pod{
388 +         ObjectMeta: metav1.ObjectMeta{
389 +             Name:      name,
390 +             Namespace: namespace,
391 +             OwnerReferences: []metav1.OwnerReference{
392 +                 {
393 +                     APIVersion: apiv1.SchemeGroupVersion.String(),
394 +                     Kind:       apiv1.ClusterKind,
395 +                     Name:       cluster.Name,
396 +                     UID:        cluster.UID,
397 +                     Controller: &[]bool{true}[0],
398 +                 },
399 +             },
400 +         },
401 +         Status: corev1.PodStatus{
402 +             Phase: corev1.PodRunning,
403 +             Conditions: []corev1.PodCondition{
404 +                 {
405 +                     Type:    corev1.PodReady,
406 +                     Status: corev1.ConditionTrue,
407 +                 },
408 +             },
409 +         },
410 +     }
411 + }
412 +
413 + func createTestPVC(name, namespace string, cluster *apiv1.Cluster)
    corev1.PersistentVolumeClaim {
414 +     return corev1.PersistentVolumeClaim{
415 +         ObjectMeta: metav1.ObjectMeta{
416 +             Name:      name,
417 +             Namespace: namespace,
418 +             OwnerReferences: []metav1.OwnerReference{
419 +                 {
420 +                     APIVersion: apiv1.SchemeGroupVersion.String(),
421 +                     Kind:       apiv1.ClusterKind,
422 +                     Name:       cluster.Name,
423 +                     UID:        cluster.UID,
424 +                     Controller: &[]bool{true}[0],
425 +                 },
```

```

426 +         },
427 +         Annotations: map[string]string{
428 +             utils.PVCStatusAnnotationName: "ready",
429 +         },
430 +     },
431 +     Status: corev1.PersistentVolumeClaimStatus{
432 +         Phase: corev1.ClaimBound,
433 +     },
434 + }
435 + }

```

86 internal/controller/cluster_controller_test.go

```

22 22 import (
23 23     "time"
24 24
25 - cnpgTypes "github.com/cloudnative-pg/machinery/pkg/types"
26 25 batchv1 "k8s.io/api/batch/v1"
27 26 corev1 "k8s.io/api/core/v1"
27 27 + metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
28 28     "k8s.io/apimachinery/pkg/types"
29 29 + "sigs.k8s.io/controller-runtime/pkg/client"
29 30     "sigs.k8s.io/controller-runtime/pkg/reconcile"
30 31
31 32     apiv1 "github.com/cloudnative-pg/cloudnative-pg/api/v1"
32 33     "github.com/cloudnative-pg/cloudnative-pg/internal/configuration"
33 34     "github.com/cloudnative-pg/cloudnative-pg/pkg/postgres"
34 35     "github.com/cloudnative-pg/cloudnative-pg/pkg/reconciler/persistentvolumeclaim"
35 36     "github.com/cloudnative-pg/cloudnative-pg/pkg/specs"
36 37 + "github.com/cloudnative-pg/cloudnative-pg/pkg/utils"
37 38 + cnpgTypes "github.com/cloudnative-pg/machinery/pkg/types"
38 39
39 40     . "github.com/onsi/ginkgo/v2"
40 41     . "github.com/onsi/gomega"
41 42 + apierrs "k8s.io/apimachinery/pkg/api/errors"
42 43 )
43 44
44 - var _ = Describe("Filtering cluster", func() {
45 -     metrics := make(map[string]string, 1)
46 -     metrics["a-secret"] = "test-version"
45 45 + var _ = Describe("reconcileResources", func() {
46 46 +     var env *testingEnvironment
47 47
48 -     cluster := apiv1.Cluster{
49 -         Spec: apiv1.ClusterSpec{
50 -             ImageName: "postgres:13.0",
51 -         },
52 -     },
49 -     Status: apiv1.ClusterStatus{

```

```

50         SecretsResourceVersion:
51         apiv1.SecretsResourceVersion{Metrics: metrics},
52     -     },
53     -     }
54
55     BeforeEach(func() {
56     +     env = buildTestEnvironment()
57     +     })
58
59     items := []apiv1.Cluster{cluster}
60     clusterList := apiv1.ClusterList{Items: items}
61
62     It("should delete a failed job and requeue", func(ctx SpecContext) {
63     +     namespace := newFakeNamespace(env.client)
64     +     cluster := newFakeCNPGBCluster(env.client, namespace)
65
66     55
67
68     It("using a secret", func() {
69     -     secret := corev1.Secret{}
70     -     secret.Name = "a-secret"
71     -     req := filterClustersUsingSecret(clusterList, &secret)
72     -     Expect(req).ToNot(BeNil())
73     -     })
74
75     // Create the CA secrets that the cluster expects
76     generateFakeCASecret(env.client, cluster.GetServerCASecretName(),
77     namespace, "cluster-test")
78     generateFakeCASecret(env.client, cluster.GetClientCASecretName(),
79     namespace, "cluster-test")
80
81     +
82     instanceName := cluster.Name + "-1"
83     failedJob := &batchv1.Job{
84     +     ObjectMeta: metav1.ObjectMeta{
85     +         Name:      instanceName + "-snapshot-recovery",
86     +         Namespace: namespace,
87     +         Labels: map[string]string{
88     +             utils.ClusterLabelName:      cluster.Name,
89     +             utils.InstanceNameLabelName: instanceName,
90     +             utils.JobRoleLabelName:      "snapshot-recovery",
91     +         },
92     +     },
93     +     Status: batchv1.JobStatus{
94     +         Conditions: []batchv1.JobCondition{
95     +             {
96     +                 Type:    batchv1.JobFailed,
97     +                 Status: corev1.ConditionTrue,
98     +             },
99     +         },
100    +     },
101    +     },
102    +     }
103    +

```

```

81 + // Create the failed job
82 + Expect(env.client.Create(ctx, failedJob)).To(Succeed())
83 +
84 + // Create minimal managed resources for the test
85 + managedResources := &managedResources{
86 +     nodes:      make(map[string]corev1.Node),
87 +     instances: corev1.PodList{Items: []corev1.Pod{}},
88 +     pvcs:        corev1.PersistentVolumeClaimList{Items:
89 + []corev1.PersistentVolumeClaim{}},
90 +     jobs:        batchv1.JobList{Items: []batchv1.Job{*failedJob}},
91 + }
92 +
93 + // Test the reconcileResources method directly to avoid
94 + // architecture validation
95 + var instancesStatus postgres.PostgresqlStatusList
96 + result, err := env.clusterReconciler.reconcileResources(ctx,
97 + cluster, managedResources, instancesStatus)
98 +
99 + // Check the result
100 + Expect(err).ToNot(HaveOccurred())
101 + Expect(result.Requeue).To(BeTrue())

```

64

99

65

66

67

68

69

```

- It("using a config map", func() {
-     configMap := corev1.ConfigMap{}
-     configMap.Name = "a-secret"
-     req := filterClustersUsingConfigMap(clusterList, &configMap)
-     Expect(req).ToNot(BeNil())

```

100

101

102

103

```

+ // Check if the job was deleted
+ err = env.client.Get(ctx, client.ObjectKeyFromObject(failedJob),
+ failedJob)
+ Expect(err).To(HaveOccurred())
+ Expect(apierrs.IsNotFound(err)).To(BeTrue())

```

70

104

71

105

72

106

```

    })
    })

```

14 internal/controller/suite_test.go

78

78

79

79

80

80

81

82

83

84

85

86

```

+ WithStatusSubresource(&apiv1.Cluster{}, &apiv1.Backup{},
+ &apiv1.Pooler{}, &corev1.Service{},
+ &corev1.ConfigMap{}, &corev1.Secret{}).
+ WithIndex(&batchv1.Job{}, jobOwnerKey, jobOwnerIndexFunc).
+ WithIndex(&corev1.Pod{}, podOwnerKey, func(rawObj client.Object)
+ []string {
+     pod := rawObj.(*corev1.Pod)
+     if ownerName, ok := IsOwnedByCluster(pod); ok {
+         return []string{ownerName}
+     }
+     return nil
+ }

```

```

87 +     }).
88 +     WithIndex(&corev1.PersistentVolumeClaim{}, pvcOwnerKey,
      func(rawObj client.Object) []string {
89 +         persistentVolumeClaim := rawObj.
          (*corev1.PersistentVolumeClaim)
90 +         if ownerName, ok := IsOwnedByCluster(persistentVolumeClaim);
          ok {
91 +             return []string{ownerName}
92 +         }
93 +         return nil
94 +     }).

```

```

81 95         Build()
82 96         Expect(err).ToNot(HaveOccurred())
83 97

```

✓ ⓘ 71 ■■■■ pkg/utils/jobs.go

... @@ -0,0 +1,71 @@

```

1 + /*
2 + Copyright © contributors to CloudNativePG, established as
3 + CloudNativePG a Series of LF Projects, LLC.
4 +
5 + Licensed under the Apache License, Version 2.0 (the "License");
6 + you may not use this file except in compliance with the License.
7 + You may obtain a copy of the License at
8 +
9 +     http://www.apache.org/licenses/LICENSE-2.0
10 +
11 + Unless required by applicable law or agreed to in writing, software
12 + distributed under the License is distributed on an "AS IS" BASIS,
13 + WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14 + See the License for the specific language governing permissions and
15 + limitations under the License.
16 +
17 + SPDX-License-Identifier: Apache-2.0
18 + */
19 +
20 + package utils
21 +
22 + import (
23 +     "time"
24 +
25 +     batchv1 "k8s.io/api/batch/v1"
26 +     corev1 "k8s.io/api/core/v1"
27 + )
28 +
29 + // IsJobFailed check if a job has failed
30 + func IsJobFailed(job batchv1.Job) bool {
31 +     for _, condition := range job.Status.Conditions {

```

```
32 +     if condition.Type == batchv1.JobFailed && condition.Status ==
    corev1.ConditionTrue {
33 +         return true
34 +     }
35 + }
36 + return false
37 + }
38 +
39 + // IsJobStuck checks if a job is stuck in pending state for too long
40 + func IsJobStuck(job batchv1.Job, timeout time.Duration) bool {
41 +     // If the job is already marked as failed or complete, it's not stuck
42 +     if IsJobFailed(job) || IsJobComplete(job) {
43 +         return false
44 +     }
45 +
46 +     // Check if job has been pending for too long
47 +     if job.CreationTimestamp.Add(timeout).Before(time.Now()) {
48 +         // Check if any pods are unschedulable
49 +         if job.Status.Active == 0 && job.Status.Succeeded == 0 &&
            job.Status.Failed == 0 {
50 +             // No pods have been created or they're all unschedulable
51 +             return true
52 +         }
53 +     }
54 +
55 +     return false
56 + }
57 +
58 + // IsJobComplete checks if a job has completed successfully
59 + func IsJobComplete(job batchv1.Job) bool {
60 +     for _, condition := range job.Status.Conditions {
61 +         if condition.Type == batchv1.JobComplete && condition.Status ==
            corev1.ConditionTrue {
62 +             return true
63 +         }
64 +     }
65 +     return false
66 + }
67 +
68 + // IsJobFailedOrStuck checks if a job has failed or is stuck
69 + func IsJobFailedOrStuck(job batchv1.Job, stuckTimeout time.Duration) bool
    {
70 +     return IsJobFailed(job) || IsJobStuck(job, stuckTimeout)
71 + }
```

▼ ⓘ 279 pkg/utils/jobs_test.go

... @@ -0,0 +1,279 @@

1 + /*

```
2 + Copyright © contributors to CloudNativePG, established as
3 + CloudNativePG a Series of LF Projects, LLC.
4 +
5 + Licensed under the Apache License, Version 2.0 (the "License");
6 + you may not use this file except in compliance with the License.
7 + You may obtain a copy of the License at
8 +
9 +     http://www.apache.org/licenses/LICENSE-2.0
10 +
11 + Unless required by applicable law or agreed to in writing, software
12 + distributed under the License is distributed on an "AS IS" BASIS,
13 + WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14 + See the License for the specific language governing permissions and
15 + limitations under the License.
16 +
17 + SPDX-License-Identifier: Apache-2.0
18 + */
19 +
20 + package utils
21 +
22 + import (
23 +     "testing"
24 +     "time"
25 +
26 +     batchv1 "k8s.io/api/batch/v1"
27 +     corev1 "k8s.io/api/core/v1"
28 +     metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
29 + )
30 +
31 + func TestIsJobFailed(t *testing.T) {
32 +     tests := []struct {
33 +         name      string
34 +         job       batchv1.Job
35 +         expected bool
36 +     }{
37 +         {
38 +             name: "job with failed condition",
39 +             job: batchv1.Job{
40 +                 Status: batchv1.JobStatus{
41 +                     Conditions: []batchv1.JobCondition{
42 +                         {
43 +                             Type:    batchv1.JobFailed,
44 +                             Status: corev1.ConditionTrue,
45 +                         },
46 +                     },
47 +                 },
48 +                 expected: true,
49 +             },
50 +         },
```

```
51 +     {
52 +         name: "job without failed condition",
53 +         job: batchv1.Job{
54 +             Status: batchv1.JobStatus{
55 +                 Conditions: []batchv1.JobCondition{
56 +                     {
57 +                         Type:    batchv1.JobComplete,
58 +                         Status: corev1.ConditionTrue,
59 +                     },
60 +                 },
61 +             },
62 +         },
63 +         expected: false,
64 +     },
65 +     {
66 +         name: "job with no conditions",
67 +         job: batchv1.Job{
68 +             Status: batchv1.JobStatus{},
69 +         },
70 +         expected: false,
71 +     },
72 + }
73 +
74 + for _, tt := range tests {
75 +     t.Run(tt.name, func(t *testing.T) {
76 +         result := IsJobFailed(tt.job)
77 +         if result != tt.expected {
78 +             t.Errorf("IsJobFailed() = %v, expected %v", result,
79 +                 tt.expected)
80 +         }
81 +     })
82 + }
83 +
84 + func TestIsJobComplete(t *testing.T) {
85 +     tests := []struct {
86 +         name      string
87 +         job       batchv1.Job
88 +         expected  bool
89 +     }{
90 +         {
91 +             name: "job with complete condition",
92 +             job: batchv1.Job{
93 +                 Status: batchv1.JobStatus{
94 +                     Conditions: []batchv1.JobCondition{
95 +                         {
96 +                             Type:    batchv1.JobComplete,
97 +                             Status: corev1.ConditionTrue,
98 +                         },
```

```

99 +         },
100 +     },
101 + },
102 +     expected: true,
103 + },
104 + {
105 +     name: "job without complete condition",
106 +     job: batchv1.Job{
107 +         Status: batchv1.JobStatus{
108 +             Conditions: []batchv1.JobCondition{
109 +                 {
110 +                     Type: batchv1.JobFailed,
111 +                     Status: corev1.ConditionTrue,
112 +                 },
113 +             },
114 +         },
115 +     },
116 +     expected: false,
117 + },
118 + }
119 +
120 + for _, tt := range tests {
121 +     t.Run(tt.name, func(t *testing.T) {
122 +         result := IsJobComplete(tt.job)
123 +         if result != tt.expected {
124 +             t.Errorf("IsJobComplete() = %v, expected %v", result,
125 +                 tt.expected)
126 +         }
127 +     })
128 + }
129 +
130 + func TestIsJobStuck(t *testing.T) {
131 +     now := time.Now()
132 +     timeout := 10 * time.Minute
133 +
134 +     tests := []struct {
135 +         name      string
136 +         job        batchv1.Job
137 +         timeout    time.Duration
138 +         expected   bool
139 +     }{
140 +         {
141 +             name: "stuck job - old with no active pods",
142 +             job: batchv1.Job{
143 +                 ObjectMeta: metav1.ObjectMeta{
144 +                     CreationTimestamp: metav1.Time{Time: now.Add(-15 *
145 +                         time.Minute)},

```

```
146 +             Status: batchv1.JobStatus{
147 +                 Active:    0,
148 +                 Succeeded: 0,
149 +                 Failed:    0,
150 +             },
151 +         },
152 +         timeout: timeout,
153 +         expected: true,
154 +     },
155 +     {
156 +         name: "not stuck - recent job",
157 +         job: batchv1.Job{
158 +             ObjectMeta: metav1.ObjectMeta{
159 +                 CreationTimestamp: metav1.Time{Time: now.Add(-5 *
160 + time.Minute)},
161 +             },
162 +             Status: batchv1.JobStatus{
163 +                 Active:    0,
164 +                 Succeeded: 0,
165 +                 Failed:    0,
166 +             },
167 +             timeout: timeout,
168 +             expected: false,
169 +         },
170 +         {
171 +             name: "not stuck - has active pods",
172 +             job: batchv1.Job{
173 +                 ObjectMeta: metav1.ObjectMeta{
174 +                     CreationTimestamp: metav1.Time{Time: now.Add(-15 *
175 + time.Minute)},
176 +                 },
177 +                 Status: batchv1.JobStatus{
178 +                     Active:    1,
179 +                     Succeeded: 0,
180 +                     Failed:    0,
181 +                 },
182 +                 timeout: timeout,
183 +                 expected: false,
184 +             },
185 +             {
186 +                 name: "not stuck - already completed",
187 +                 job: batchv1.Job{
188 +                     ObjectMeta: metav1.ObjectMeta{
189 +                         CreationTimestamp: metav1.Time{Time: now.Add(-15 *
190 + time.Minute)},
191 +                     },
192 +                     Status: batchv1.JobStatus{
```

```
192 +             Active:    0,
193 +             Succeeded: 1,
194 +             Failed:    0,
195 +             Conditions: []batchv1.JobCondition{
196 +                 {
197 +                     Type:    batchv1.JobComplete,
198 +                     Status: corev1.ConditionTrue,
199 +                 },
200 +             },
201 +         },
202 +     },
203 +     timeout: timeout,
204 +     expected: false,
205 + },
206 + }
207 +
208 + for _, tt := range tests {
209 +     t.Run(tt.name, func(t *testing.T) {
210 +         result := IsJobStuck(tt.job, tt.timeout)
211 +         if result != tt.expected {
212 +             t.Errorf("IsJobStuck() = %v, expected %v", result,
213 +                 tt.expected)
214 +         }
215 +     })
216 + }
217 +
218 + func TestIsJobFailedOrStuck(t *testing.T) {
219 +     now := time.Now()
220 +     timeout := 10 * time.Minute
221 +
222 +     tests := []struct {
223 +         name      string
224 +         job        batchv1.Job
225 +         expected   bool
226 +     }{
227 +         {
228 +             name: "failed job",
229 +             job: batchv1.Job{
230 +                 Status: batchv1.JobStatus{
231 +                     Conditions: []batchv1.JobCondition{
232 +                         {
233 +                             Type:    batchv1.JobFailed,
234 +                             Status: corev1.ConditionTrue,
235 +                         },
236 +                     },
237 +                 },
238 +             },
239 +             expected: true,
```

```
240 +     },
241 +     {
242 +         name: "stuck job",
243 +         job: batchv1.Job{
244 +             ObjectMeta: metav1.ObjectMeta{
245 +                 CreationTimestamp: metav1.Time{Time: now.Add(-15 *
time.Minute)},
246 +             },
247 +             Status: batchv1.JobStatus{
248 +                 Active:    0,
249 +                 Succeeded: 0,
250 +                 Failed:    0,
251 +             },
252 +         },
253 +         expected: true,
254 +     },
255 +     {
256 +         name: "healthy job",
257 +         job: batchv1.Job{
258 +             ObjectMeta: metav1.ObjectMeta{
259 +                 CreationTimestamp: metav1.Time{Time: now.Add(-5 *
time.Minute)},
260 +             },
261 +             Status: batchv1.JobStatus{
262 +                 Active:    1,
263 +                 Succeeded: 0,
264 +                 Failed:    0,
265 +             },
266 +         },
267 +         expected: false,
268 +     },
269 + }
270 +
271 + for _, tt := range tests {
272 +     t.Run(tt.name, func(t *testing.T) {
273 +         result := IsJobFailedOrStuck(tt.job, timeout)
274 +         if result != tt.expected {
275 +             t.Errorf("IsJobFailedOrStuck() = %v, expected %v",
result, tt.expected)
276 +         }
277 +     })
278 + }
279 + }
```