Node Failure Recover with ForceEviction

Authors: jinxu@

Background

Kubernetes relies on a node agent (aka kubelet) to report the health of a node. There are a variety of reasons for kubelet fail to update its node status, including kernel crash, panic, or update which cause system reboot, system hang due to heavy load, network partition issue, or kubelet fails to start. When this happens, Kubernetes will taint the node status as “NotReady” and will trigger eviction of pods on the failed node with taint-based eviction and start new pods on other healthy nodes.

However, users often reported issues during the failure handling including old pods cannot be deleted correctly, new pods cannot be started correctly due to volume failing to detach from old node and attach to new node issues, or it takes a long time for new pods to start. This document is trying to investigate the failure handling issues for pods and their volumes with a goal of improving system reliability and failure recovery capabilities.

Goals

When kubelet is not responsive, the current failure recovery with Pod eviction does not work well. Pod eviction and resource clean up normally require a kubelet acknowledge so that Pod will be deleted from API server and also volume can be released (detached from the node). This proposal is trying to address this problem with a new type of taint based eviction, ForceEviction. The overall idea is to force delete Pods with zero grace period. This means that a Pod will be deleted immediately from API server without waiting for kubelet reponse. In this way, new statefulSet Pod can be started again on other healthy node and volumes will be detached quickly.

This force eviction can be very useful for quick failure recovery in certain scenarios. For example, for bare-metal cluster, shutting down a node is often used as the last recovery step if node is no longer responsive and there is no other way (or no time) to diagnose the root cause and fix it. When node is shutdown, all processes stops and volumes are no longer mounted, so Pods can be safely deleted without waiting for kubelet to perform any clean up. The main challenge for this ForceEviction is to make sure no race condition between Pod eviction and Kubelet restarts Pods when it comes back.

Proposal

Currently, Kubernetes already supports four types of taint effects:

- **NoSchedule**: Pods that do not tolerate this taint are not scheduled on the node.
- **PreferNoSchedule**: Kubernetes avoids scheduling Pods that do not tolerate this taint onto the node.
- **NoExecute**: Pod is evicted from the node if it is already running on the node, and is not scheduled onto the node if it is not yet running on the node.

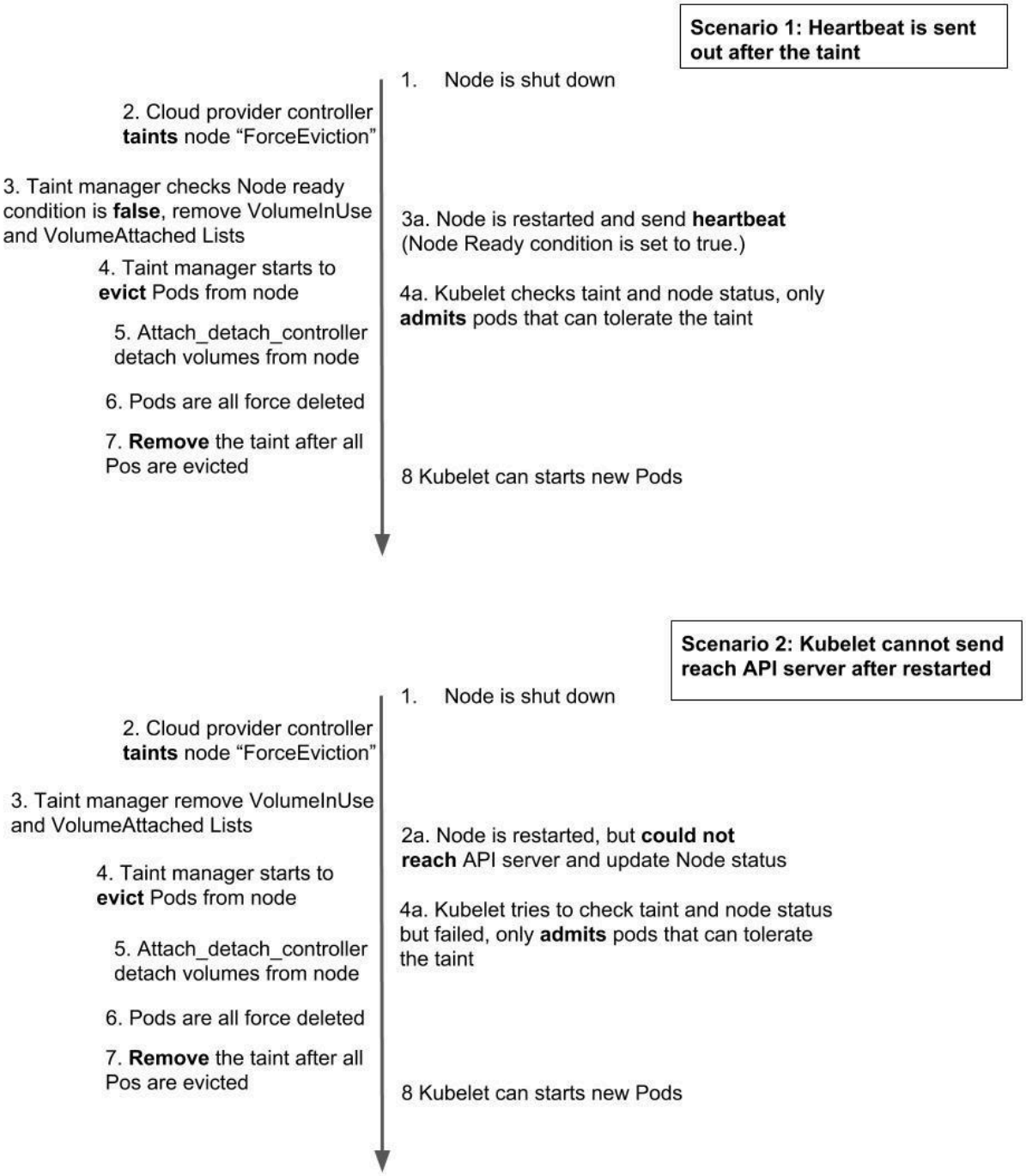
We propose to add a new effect, **TaintEffectForceEviction**. For a Pod that cannot tolerant this taint, it will be force deleted from the node with zero grace period. In node shut down case, controller could add the following taint to the node.

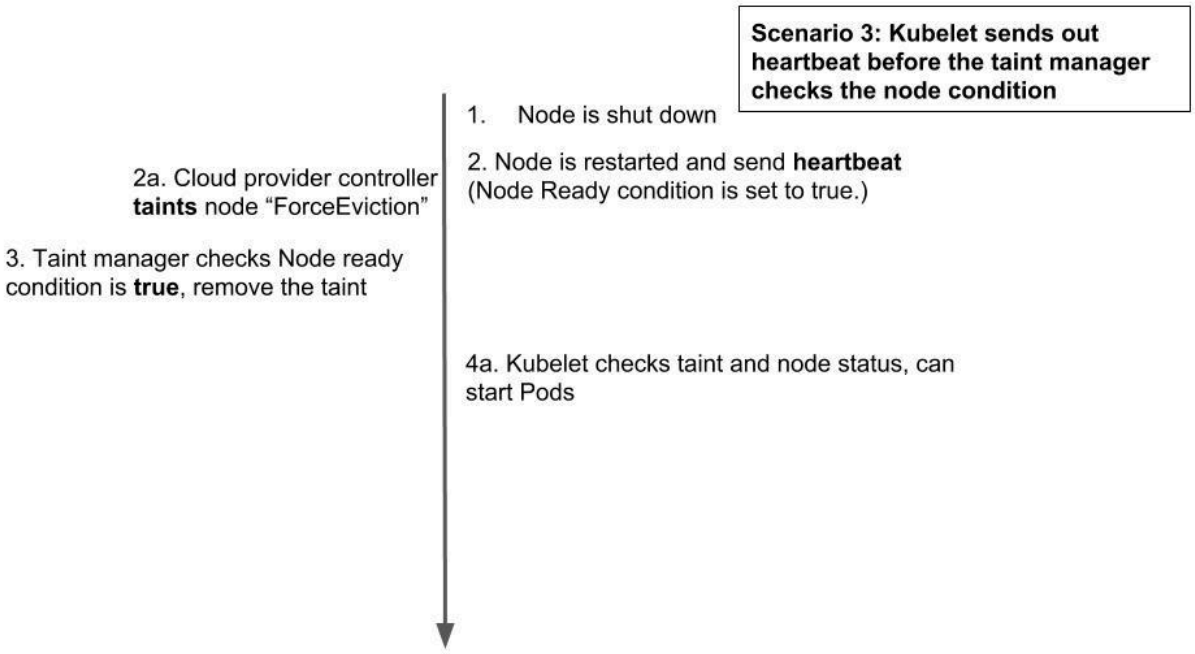
```
taint := &v1.Taint{
        Key:    nodeShutdown,
        Value:  "true",
        Effect: v1.TaintEffectForceEviction,
}
```

After the node is tainted, node lifecycle controller will trigger to force delete all the pods from the node. To avoid race condition, kubelet also requires some additional handling when node is tainted which is explained in more details below.

Master Node lifecycle controller	Node kubelet
When node is marked as “NotReady” and tainted with “ForceEviction”. Taint manager will perform the following steps 1. Check node condition whether Ready becomes true again. If yes, remove the taint if ForceEviction pods on the node is not started yet or already finished. Otherwise, skip the rest. If Ready is still false, continue with next step 2. 2. Remove VolumeInUse list for this node and Remove VolumesAttached list 3. Evict pods from the node **with --force and 0 grace period**	During startup: send out heartbeat by updating node status. Before starting up Pod, Pod canAdmit checks taint and pod tolerations to decide whether to admit pod or not. ● If node condition can not be retrieved, do not start Pods that cannot tolerant ForceEviction taint ● If node has taint or node Ready condition is not true (heartbeat has not yet sent out), do not start Pods that cannot tolerant ForceEviction taint

The following diagrams list the possible scenarios during force eviction. The test cases should cover all these scenarios






```
---
kep-number: 36
title: Node Failure Recovery with ForceEviction Taint
authors:
 - "@jingxu97"
 - "@yastij"
owning-sig: sig-node
participating-sigs:
 - sig-storage
 - sig-node
 - sig-architecture
reviewers:
 - "@yujuhong"
 - "@yguo00905"
 - "@liggitt"
 - "@davidopp"
approvers:
 - "@liggitt"
```

```
 - "@yujuhong"
editor: TBD
creation-date: 2019-1-18
last-updated: 2019-1-18
status: implementable
see-also:
 - n/a
replaces:
 - n/a
superseded-by:
 - n/a
---
```

Title

Node Failure Recovery with ForceEviction Taint

Table of Contents

Summary

Current taint-based eviction does not work well to handle node failures.
When a node fails and kubelet cannot respond, Pod will be struck in

terminating and cannot be deleted, and Volume cannot be detach either.
To address the issue of failing to recover Pod from node failure, we are proposing a new taint-base eviction effect, ForceEviction.
With this new type of eviction, Pods can be deleted forcefully without waiting for
Kubelet's acknowledgement and volumes can be detached faster. The main challenge
of this proposal is to make sure no race condition between Pod eviction and Kubelet
restarts Pods when it comes back.

Motivation

Kubernetes relies on a node agent (aka kubelet) to report the health of a node.
There are a variety of reasons that kubelet fails to update its node status,
such as kernel crash, panic, or update which causes system reboot,
system hangs due to heavy load, network partition, and machine is powered off
due to failures. When this happens, Kubernetes currently uses a taint-based eviction
trying to recovery from the failed node. The recovery process will taint the node
status as “NotReady” and will trigger eviction of pods on the failed node if the node
is in "NotReady" state for a while (set to 5 minutes by default) so that new pods
could be started on other healthy nodes.

However, users often reported issues during the failure handling including old pods
cannot be deleted correctly, new pods cannot be started correctly due to volume failing
to detach from old node and attach to new node issues, or it takes a long time for new

pods to start. The main reason for this problem is due to the fact that Pod deletion requires
kubelet's acknowlegement. When a Pod is evicted (deleted) from a node, the deletion timestamp
is first set on the Pod. However the Pod will not be actually deleted from API server until kubelet
finishes cleaning up all the resources of this Pod and kills the running containers. If the node
is in a failed state and kubelet cannot respond, Pod will be stuck in a teminating state forever.
In addtion, the containers status for the Pod are still running since kubelet cannot update it,
which blocks attach_detach controller to detach volumes. The current taint-based eviction actually
cannot give any benefit in the sense that Pod can only be succesfully evicted after Kubelet comes
back healthy, in which case there is no need of Pod eviction at all.

Goals

This proposal is trying to investigate the failure handling issues for pods
and their volumes with a goal of improving system reliability and failure recovery capabilities.
Among different kinds of failures, node shut down is the main user case this proposal tries
to address. In this case, we are sure that all containers are stopped, and all mounts are gone
so that it is safe to take some recovery actions more aggresively without waiting for kubelet correspondence.

to delete the Pods from API server without waiting for kubelet's acknoledgement.
Volumes can also be safely detached since mounts no longer exist.

Non-Goals

This proposal is not trying to address the recovery issue for all types of failures. For example for network
partition issue, because this is no source of truth about whether Pods are running or not, it is not possible
for controller decide what recovery actions should take based on the node status.

Proposal

Currently, Kubernetes already supports four types of taint effects:
- NoSchedule: Pods that do not tolerate this taint are not scheduled on the node.
- PreferNoSchedule: Kubernetes avoids scheduling Pods that do not tolerate this taint onto the node.
- NoExecute: Pod is evicted from the node if it is already running on the node, and is not scheduled
onto the node if it is not yet running on the node.

ForceEviction Taint Effect

We propose to add a new effect, TaintEffectForceEviction. For a Pod that cannot tolerant this taint,
it will be force deleted from the node with zero grace period which means that Pod can be deleted immediately
without waiting for Kubelet to responde. One important use case of this taint is node shut down case. If node
controller or some external controller detact that node is powered off, the controller could add the following
taint to the node.

```
taint := &v1.Taint{
  Key:    nodeShutdown,
  Value:  "true",
  Effect: v1.TaintEffectForceEviction,
```

}
```

After the node is tainted, node lifecycle controller will trigger to forcely delete all the Pods from the node.
Because the main goal of this effect is to recover from failures as soon as possible, the grace period of deleting
the pod will be set to 0. The controller or system admin who taint the node with ForceEviction should make sure that
no workload is running (pods are running) any more on the node to ensure the [pod
safety](https://github.com/kubernetes/community/blob/master/contributors/design-proposals/storage/pod-safety.md).

### Coordination between Node Controller and Kubelet

If Kubelet comes back healthy in the middle of force eviction process, there might be race condition between kubelet
starting Pods while they are being deleted since force deletion does not require kubelet coordination any more.
To avoid race condition between controller and kubelet, kubelet also requires some additional handling when node
is tainted which is explained in more details below.

For node lifecycle controller,  we plan to add the following eviction logic:

When node is marked as “NotReady” and tainted with “ForceEviction”. Taint manager will perform the following steps

1. Check node condition whether Ready becomes true again. If yes, remove the taint if ForceEviction pods on the node
is not started yet or already finished. Otherwise, skip the rest. If Ready is still false, continue with next step 2.
2. Remove VolumeInUse list for this node and Remove VolumesAttached list
3. Evict pods from the node with --force and 0 grace period
```

For node Kubelet, it will first send out heartbeat by updating node status during start up.

Before starting up Pod, Pod canAdmit function checks taint and pod tolerations to decide whether to admit pod or not.
- If node condition can not be retrieved, do not start Pods that cannot tolerant ForceEviction taint. But we should allow
to start static Pods.
- If node has taint or node Ready condition is not true (heartbeat has not yet sent out), do not start Pods
that cannot tolerant ForceEviction taint.

The following iternate the possible events happened during the eviction process for node shut down use case.

Scenario 1: Heartbeat is sent out after the taint

1. Node is shut down
2. Cloud provider controller checks the node status and taints node with "ForceEviction" taint effect
3. Taint manager checks Node ready condition is false, remove VolumeInUse and VolumeAttached Lists
4. Taint manager confirms that no new heartbeat from kubelet and starts to evict Pods from node
5. Node might be restarted and kubelet will send heartbeat (Node Ready condition is set to true.)
Kubelet checks taint and only admits pods that can tolerate the taint.
6. Attach_detach_controller detach volumes from node
7. Remove the taint after all Pods are evicted
8. Kubelet can starts new Pods after taint is removed.

Scenario 2: Kubelet cannot reach API server after restarted

1. Node is shut down
2. Cloud provider controller taints node "ForceEviction"
3. Taint manager checks Node ready condition is false, remove VolumeInUse and VolumeAttached Lists

4. Node is restarted, but could not reach API server and update Node status
5. Kubelet tries to check taint and node status but failed, only admits pods that can tolerate the taint
6. Taint manager confirms that no new heartbeat from kubelet and starts to evict Pods from node
7. Attach_detach_controller detach volumes from node
8. Remove the taint after all Pods are evicted

Scenario 3: Kubelet sends out heartbeat before the taint manager checks the node condition

1. Node is shut down
2. Node is restarted and sends heartbeat (Node Ready condition is set to true.)
3. Taint manager checks Node ready condition is true, remove the taint
4. 4a. Kubelet checks taint and node status, can start Pods

User Stories

As a kubernetes administrator/IT automation tool, I want workload can be recovered quickly when some nodes fail.
Some controller could detect the node failures, and could shut it down so that the workload running on the node could
be recoved on other healthy nodes. The controller or system admin should make sure that no workload is running on
the node (e.g., shut down the node) when the taint is applied to the node.

Alternatives

Kubernetes administrator/IT automation tool could choose to delete the node API object from API server if it detect some
node failures. In this way, Pods will be garbage collected after a while. However, this way it will lose some meta data that
were applied to the node already which is not ideal. Also after fixing the node, administroator has to register this node back.

Risks and Mitigations

Graduation Criteria

We need to make sure this feature will not cause problem when kubelet comes back. Need comprehensive tests to cover all edge cases.

Implementation History

Check “shutdown” taint:

a. ~~If has taint, do nothing and check again later~~
b. Pod admit checks taint and pod tolerations to decide whether admit pod or not

after node shut down, pods might not be properly deleted from API server because kubelet is no longer responding. Furthermore, volumes cannot be detached because containers status is running (kubelet cannot update its status any more).

However, in reality is after node is shut down, all containers are stopped, and all mounts are gone. If system can check node is in shutdown state, it is safe to delete the Pods from API server so that new Pods can be started in other nodes. There are two efforts going on to address this.

- Taint the node “shutdown” state: cloud provider normally can check node status through their API. Therefore, cloud node controller could taint the node shutdown status by checking the node status through cloud provider. For bare-metal environment, there is a node fencing proposal for this purpose. The idea is to use external controller to taint the node.
- Controller force delete pods from nodes in shutdown state: this is the focus of this proposal. The idea is to monitor node shutdown state. Node controller can try to force delete the pods from shutdown node. However, If node restarts, make sure no new pods come up and mount when controller is detaching the volumes