# Alerting on Unschedulable or Failed Pods

In this lesson, we will see how to handle alerts in the case of Unschedulable or Failed Pods.

WE'LL COVER THE FOLLOWING

## Cause of unschedulable or failed pods

Knowing whether our applications are having trouble responding quickly to requests, whether they are being bombed with more requests than they could handle, whether they produce too many errors, and whether they are saturated, is of no use if they are not even running. Even if our alerts detect that something is wrong by notifying us that there are too many errors or that response times are slow due to an insufficient number of replicas, we should still be informed if, for example, one or even all the replicas failed to run. In the best-case scenario, such a notification would provide additional info about the cause of an issue. In a much worse situation, we might find out that one of the replicas of the DB is not running. That would not necessarily slow it down, nor would it produce any errors. However, it would put us in a situation where data could not be replicated (additional replicas are not running), and we might face a total loss of its state if the last standing replica fails as well.

There are many reasons why an application would fail to run. There might not be enough unreserved resources in the cluster. `Cluster Autoscaler` will deal with that problem if we have it. But, there are many other potential issues. Maybe, the image of the new release is not available in the registry. Or perhaps, the Pods are requesting PersistentVolumes that cannot be claimed perhaps, the Pods are requesting PersistentVolumes that cannot be claimed.

As you might have guessed, the list of the things that might cause our Pods to fail, be unschedulable or in an unknown state, is almost infinite.

## Generating an alert after a while

We cannot address all of the causes of problems with Pods individually. However, we can be notified if the phase of one or more Pods is `Failed`, `Unknown`, or `Pending`. Over time, we might extend our self-healing scripts to address some of the specific causes of those statuses. For now, our best first step is to be notified if a Pod is in one of those phases for a prolonged period of time (e.g., fifteen minutes). Alerting as soon as the status of a Pod indicates a problem would be silly because that would generate too many false positives. We should get an alert and choose how to act only after waiting for a while, thus giving Kubernetes time to fix an issue. We should perform some reactive actions only if Kubernetes fails to remedy the situation.

Over time, we'll notice some patterns in the alerts we're receiving. When we do, alerts should be converted into automated responses that will remedy selected issues without our involvement. We already explored some of the low hanging fruits through `HorizontalPodAutoscaler` and `Cluster Autoscaler`. For now, we'll focus on receiving alerts for all other cases, and failed and unschedulable Pods are a few of those. Later on, we might explore how to automate responses. But, that moment is not now, so we'll move forward with yet another alert that will result in a notification to Slack.

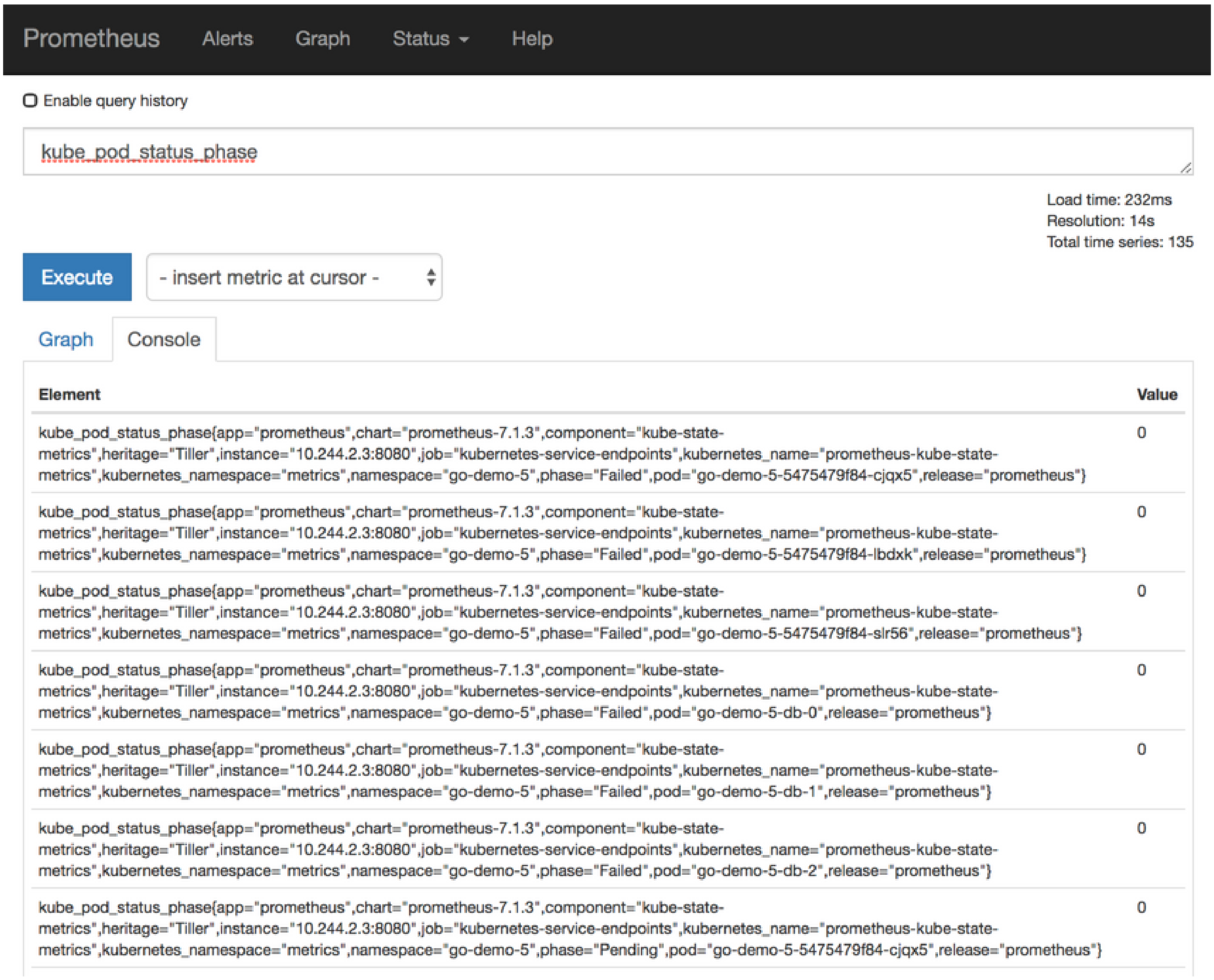
Let's open the `Prometheus`'s graph screen.

```
open "http://$PROM_ADDR/graph"
```

Please type the expression that follows and click the *Execute* button.

```
kube_pod_status_phase
```

The **output** shows us each of the Pods in the cluster. If you take a closer look, you'll notice that there are five results for each Pod, one for each of the five possible phases. If you focus on the `phase` field, you'll see that there is an entry for `Failed`, `Pending`, `Running`, `Succeeded`, and `Unknown`. So, each Pod has five results, but only one has the value `1`, while the values of the other four are all set to `0`.



```
kube_pod_status_phase
```

Load time: 232ms
Resolution: 14s
Total time series: 135

| Element | Value |
|---|---|
| kube_pod_status_phase{app="prometheus",chart="prometheus-7.1.3",component="kube-state-metrics",heritage="Tiller",instance="10.244.2.3:8080",job="kubernetes-service-endpoints",kubernetes_name="prometheus-kube-state-metrics",kubernetes_namespace="metrics",namespace="go-demo-5",phase="Failed",pod="go-demo-5-5475479f84-cjqx5",release="prometheus"} | 0 |
| kube_pod_status_phase{app="prometheus",chart="prometheus-7.1.3",component="kube-state-metrics",heritage="Tiller",instance="10.244.2.3:8080",job="kubernetes-service-endpoints",kubernetes_name="prometheus-kube-state-metrics",kubernetes_namespace="metrics",namespace="go-demo-5",phase="Failed",pod="go-demo-5-5475479f84-lbdxk",release="prometheus"} | 0 |
| kube_pod_status_phase{app="prometheus",chart="prometheus-7.1.3",component="kube-state-metrics",heritage="Tiller",instance="10.244.2.3:8080",job="kubernetes-service-endpoints",kubernetes_name="prometheus-kube-state-metrics",kubernetes_namespace="metrics",namespace="go-demo-5",phase="Failed",pod="go-demo-5-5475479f84-slr56",release="prometheus"} | 0 |
| kube_pod_status_phase{app="prometheus",chart="prometheus-7.1.3",component="kube-state-metrics",heritage="Tiller",instance="10.244.2.3:8080",job="kubernetes-service-endpoints",kubernetes_name="prometheus-kube-state-metrics",kubernetes_namespace="metrics",namespace="go-demo-5",phase="Failed",pod="go-demo-5-db-0",release="prometheus"} | 0 |
| kube_pod_status_phase{app="prometheus",chart="prometheus-7.1.3",component="kube-state-metrics",heritage="Tiller",instance="10.244.2.3:8080",job="kubernetes-service-endpoints",kubernetes_name="prometheus-kube-state-metrics",kubernetes_namespace="metrics",namespace="go-demo-5",phase="Failed",pod="go-demo-5-db-1",release="prometheus"} | 0 |
| kube_pod_status_phase{app="prometheus",chart="prometheus-7.1.3",component="kube-state-metrics",heritage="Tiller",instance="10.244.2.3:8080",job="kubernetes-service-endpoints",kubernetes_name="prometheus-kube-state-metrics",kubernetes_namespace="metrics",namespace="go-demo-5",phase="Failed",pod="go-demo-5-db-2",release="prometheus"} | 0 |
| kube_pod_status_phase{app="prometheus",chart="prometheus-7.1.3",component="kube-state-metrics",heritage="Tiller",instance="10.244.2.3:8080",job="kubernetes-service-endpoints",kubernetes_name="prometheus-kube-state-metrics",kubernetes_namespace="metrics",namespace="go-demo-5",phase="Pending",pod="go-demo-5-5475479f84-cjqx5",release="prometheus"} | 0 |

Prometheus' console view with the phases of the Pods

## Retrieve the number of Pods in each of the phases #

For now, our interest lies primarily with alerts, and they should, in most cases, be generic and not related to a specific node, application, replica, or some other type of resource. Only when we are alerted that there is an issue, should we start digging deeper and look for more granular data. With that in mind, we'll rewrite our expression to retrieve the number of Pods in each of the phases.

Please type the expression that follows and click the *Execute* button.

```
sum(
  kube_pod_status_phase
)
by (phase)
```

The **output** should show that all the Pods are in the `Running` phase. In my case, there are twenty-seven running Pods and none in any of the other

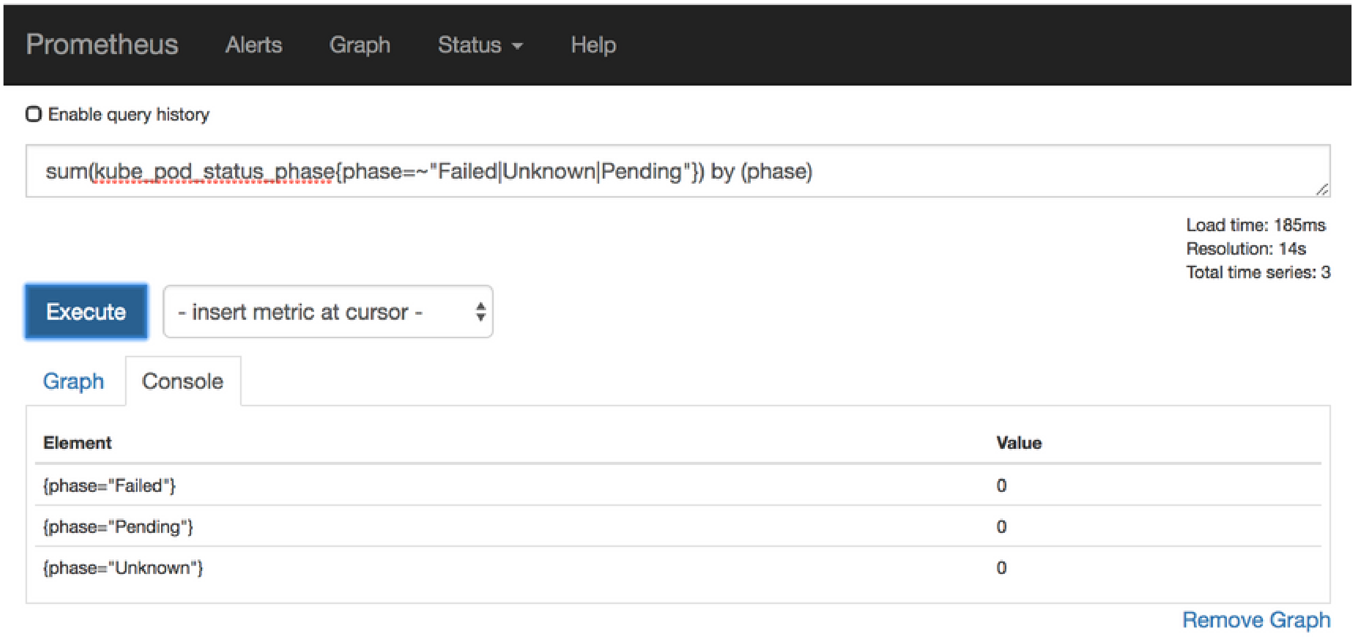
phases.

Now, we should not really care about healthy Pods. They are running, and there's nothing we should do about that. Instead, we should focus on those that are problematic. So, we might just as well rewrite the previous expression to retrieve the sum only of those that are in the `Failed`, `Unknown`, or `Pending` phase.

Please type the expression that follows and click the *Execute* button.

```
sum(
  kube_pod_status_phase{
    phase=~"Failed|Unknown|Pending"
  }
)
by (phase)
```

As expected, unless you messed up something, the values of the output are all set to `0`.



```
sum(kube_pod_status_phase{phase=~"Failed|Unknown|Pending"}) by (phase)
```

Load time: 185ms
Resolution: 14s
Total time series: 3

| Element | Value |
|---|---|
| {phase="Failed"} | 0 |
| {phase="Pending"} | 0 |
| {phase="Unknown"} | 0 |

Prometheus' console view with the sum of the Pods in Failed, Unknown, or Pending phases

## Intentionally failing a pod

So far, there are no Pods we should worry about. We'll change that by creating one that will intentionally fail by using an image that apparently does not exist.

```
kubectl run problem \
    --image i-do-not-exist \
    --restart=Never
```

As we can see from the output, the `pod/problem` was `created`. If we created it through a script (e.g., CI/CD pipeline), we'd think that everything is OK. Even if we follow it with `kubectl rollout status`, we would only ensure that it started working, not that it continues working.

But, since we did not create that Pod through a CI/CD pipeline, but manually, we can just as well list all the Pods in the `default` Namespace.

```
kubectl get pods
```

The **output** is as follows.

```
NAME    READY STATUS       RESTARTS AGE
problem 0/1   ErrImagePull 0        27s
```

We'll imagine that we have only short-term memory and that we already forgot that the `image` is set to `i-do-not-exist`. What could the issue be? Well, the first step would be to describe the Pod.

```
kubectl describe pod problem
```

The **output**, limited to the messages of the `Events` section, is as follows.

```
...
Events:
...  Message
...  -------
...  Successfully assigned default/problem to aks-nodepool1-29770171-2
...  Back-off pulling image "i-do-not-exist"
...  Error: ImagePullBackOff
...  pulling image "i-do-not-exist"
...  Failed to pull image "i-do-not-exist": rpc error: code = Unknown des
c = Error response from daemon: repository i-do-not-exist not found: doe
s not exist or no pull access
  Warning  Failed     8s (x3 over 46s)   kubelet, aks-nodepool1-29770171-2
  Error: ErrImagePull
  Error: ErrImagePull
```

The problem is clearly manifested through the `Back-off pulling image "i-do-not-exist"` message. Further down, we can see the message from the container server stating that it `failed to pull image "i-do-not-exist"`.

## Creating another alert to notify us when pods fail

Of course, we knew in advance that that would be the result, but something similar could happen without us noticing that there is an issue. The cause could be a failure to pull the image, or one of a myriad of others. Nevertheless, we are not supposed to sit in front of a terminal listing and describing Pods and other types of resources. Instead, we should receive an alert that Kubernetes failed to run a Pod, and only after that, we should start digging for the cause of the issue. So, let's create one more alert that will notify us when Pods fail and do not recuperate.

Like many times before, we'll take a look at the differences between the old and the new definition of `Prometheus` Chart values.

```
diff mon/prom-values-errors.yml \
    mon/prom-values-phase.yml
```

The **output** is as follows.

```
136a137,146
> - name: pods
>   rules:
>   - alert: ProblematicPods
>     expr: sum(kube_pod_status_phase{phase=~"Failed|Unknown|Pending"}) b
y (phase) > 0
>     for: 1m
>     labels:
>       severity: notify
>     annotations:
>       summary: At least one Pod could not run
>       description: At least one Pod is in a problematic phase
```

We defined a new group of alerts called `pod`. Inside it, we have an `alert` named `ProblematicPods` that will fire if there are one or more Pods with the

`Failed`, `Unknown`, or `Pending` phase for more than one minute (`1m`). I intentionally set it to a very low `for` duration so that we can test it easily.

Later on, we'll switch to a fifteen minutes interval that will be more than enough to allow Kubernetes to remedy the issue before we get a notification that will send us into panic mode.

Let's update the `Prometheus`'s Chart with the updated values.

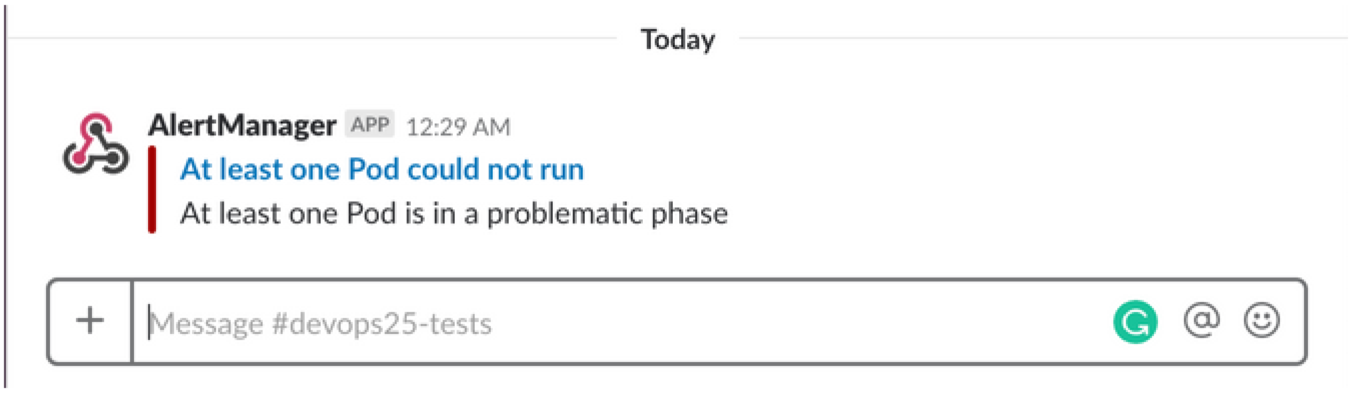
```
helm upgrade prometheus \
  stable/prometheus \
  --namespace metrics \
  --version 9.5.2 \
  --set server.ingress.hosts={$PROM_ADDR} \
  --set alertmanager.ingress.hosts={$AM_ADDR} \
  -f mon/prom-values-phase.yml
```

Since we did not yet fix the issue with the `problem` Pod, we should see a new notification in Slack soon. Let's confirm that.

```
open "https://devops20.slack.com/messages/CD8QJA8DS/"
```

If your notification did not yet arrive, please wait for a few moments.

We got the message stating that *at least one Pod could not run*.


Today

AlertManager APP 12:29 AM
At least one Pod could not run
At least one Pod is in a problematic phase

Slack with an alert message

Q

The `ProblematicPods` will fire if there are one or more Pods with the `Failed`, `Unknown`, or `Pending` phase for more than ten minute (`10m`).

COMPLETED 0%

1 of 1

Now that we received a notification that there is a problem with one of the Pods, we should go to `Prometheus`, dig through data until we find the cause of the issue, and fix it. But, since we already know what the problem is (we created it intentionally), we'll skip all that, and remove the faulty Pod, before we move onto the next subject.

```
kubectl delete pod problem
```

---

In the next lesson, we will see how to update the Old Pods.