Monitoring Kubernetes Cluster

eG Innovations Product Documentation

www.eginnovations.com

eG
Total Performance Visibility

Restricted Rights Legend

The information contained in this document is confidential and subject to change without notice. No part of this document may be reproduced or disclosed to others without the prior permission of eG Innovations Inc. eG Innovations Inc. makes no warranty of any kind with regard to the software and documentation, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose.

Trademarks

Microsoft Windows, Windows 2008, Windows 7, Windows 8, Windows 10, Windows 2012, Windows 2016, and Windows 2019 are either registered trademarks or trademarks of Microsoft Corporation in United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Copyright

Table of Contents

Table of Figures

Chapter 1: Introduction to Monitoring the Kubernetes Cluster

Kubernetes is an open-source system for managing - i.e., running and co-ordinating - containerized applications across a cluster of machines. It allows users to define how their applications should run and how they should interact with other applications or the outside world. Using Kubernetes, users can ensure high-availability of their containerized applications, scale their services up or down, perform graceful rolling updates, and switch traffic between different versions of applications to test features or rollback problematic deployments.

At its base, Kubernetes brings together multiple physical or virtual servers into a cluster using a shared network to communicate between them. Though the cluster can contain any host that runs containerized applications, the most common or popular deployment of Kubernetes has it managing a cluster of Docker hosts. This cluster is the physical platform where all Kubernetes components, capabilities, and workloads are configured.

The machines in the cluster are each given a role within the Kubernetes ecosystem. One server (or a small group in highly available deployments) functions as the master server. This server acts as a gateway and brain for the cluster. It is the primary point of contact with the cluster and is responsible for most of the centralized logic Kubernetes provides.

The other servers in the cluster are designated as worker (or slave) nodes: servers responsible for accepting and running workloads using local and external resources. Worker nodes run applications and services in containers, and therefore require a container runtime (like Docker). The node receives work instructions from the master server and creates or destroys containers accordingly.

Together, the Kubernetes master and worker nodes form the Kubernetes control plane. To ensure the high availability of the containerized applications and services, the control plane responds to changes in the cluster and works to restore the cluster to its desired state.

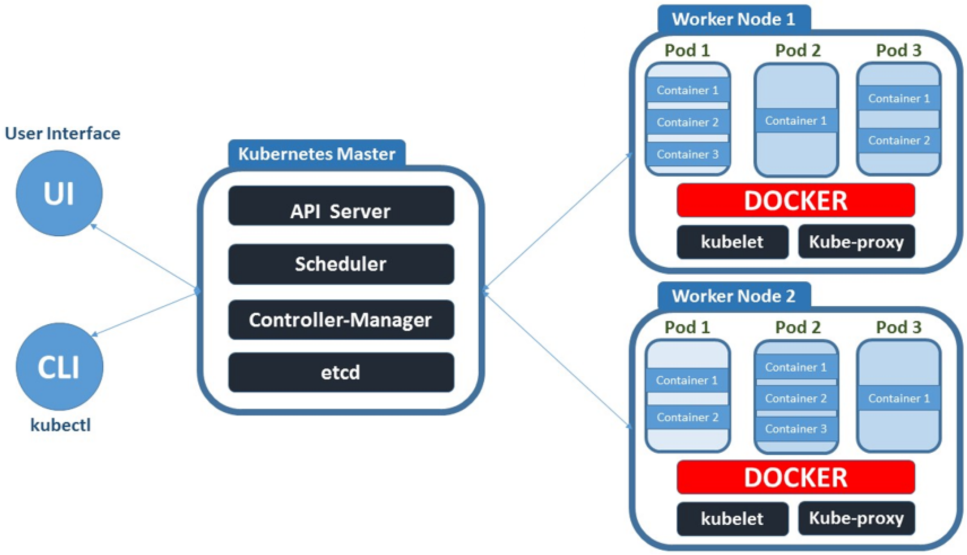


Figure 1.1: Basic architecture of a Kubernetes Cluster

The cluster's desired state is typically defined by the user (a developer/admin) who connects to the Kubernetes master server. To represent the state of a cluster, Kubernetes uses persistent entities called Objects. A Kubernetes object is a “record of intent”– once you create the object, the Kubernetes system will constantly work to ensure that object exists. By creating an object, you are effectively telling the Kubernetes system what you want your cluster’s workload to look like; this is your cluster’s desired state. Some of the most commonly used Kubernetes objects include:

- Pod: A Pod represents a unit of deployment: *a single instance of an application in Kubernetes*, which might consist of either a single Docker container or a small number of containers that are tightly coupled and that share resources. Other than container(s), a Pod encapsulates a unique network IP and options that govern how the container(s) should run.
- Service: A Service is an abstraction which defines a logical set of Pods and a policy by which to access them (sometimes this pattern is called a micro-service).
- Volume: At its core, a volume is just a directory, possibly with some data in it, which is accessible to the containers in a Pod.
- Namespace: Kubernetes supports multiple virtual clusters called Namespaces, which are backed by the same physical cluster. Namespaces are a way to divide cluster resources between multiple users (via resource quota).
- ReplicaSet: A ReplicaSet’s purpose is to maintain a stable set of replica Pods running at any given time. As such, it is often used to guarantee the availability of a specified number of identical Pods.

- Deployment: A Deployment provides declarative updates for Pods and ReplicaSets. You describe a desired state in a Deployment. You can define Deployments to create new ReplicaSets, or to remove existing Deployments and adopt all their resources with new Deployments.
- DaemonSet: A DaemonSet ensures that all (or some) Nodes run a copy of a Pod - eg., running a cluster storage daemon, such as *glusterd, ceph*, on each node.

Every Kubernetes object includes two nested object fields that govern the object's configuration: the *object spec* and the *object status*. When a user connects to the master server, he/she must provide a spec describing the desired state for the object–the characteristics that the user wants the object to have. For instance, a Kubernetes Deployment is an object that can represent an application running on the cluster. When the user creates the Deployment, he/she might set the Deployment spec to specify that they want three replicas of the application to be running.

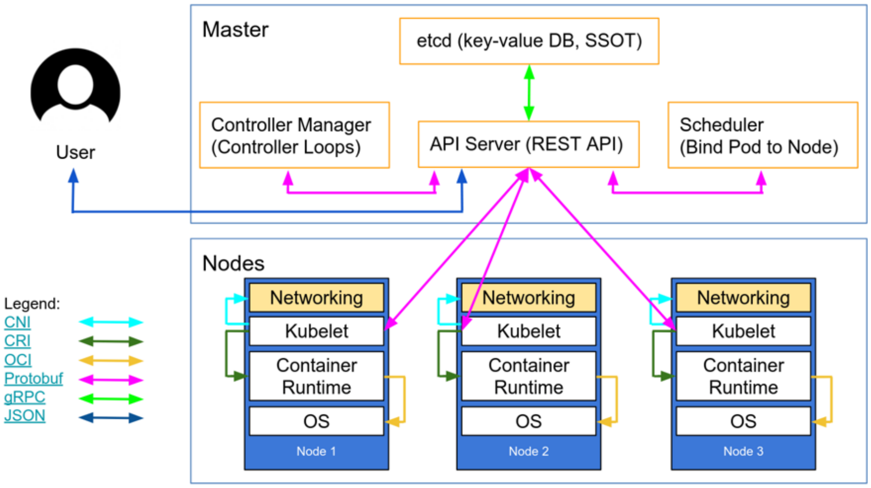


Figure 1.2: How the Kubernetes Cluster works

The master server exposes a Kubernetes API (the kube-apiserver process), which receives the object spec from the user. The API then runs the spec by the scheduler (the kube-scheduler process). The scheduler selects the worker (or slave) node to which the new objects should be assigned. Factors taken into account for scheduling decisions include individual and collective resource requirements, hardware/software/policy constraints, affinity and anti-affinity specifications, data locality, inter-workload interference and deadlines. Alongside, the master sever also stores the configuration and status data of objects created, in a consistent, highly-available key-value store called etcd.

Once the scheduler assigns a worker (or slave) node, the controller manager (the kube-controller-manager process) on the master node then sends the object spec to that node (via the Kubernetes API), so it can create the desired object.

Upon receipt of the object spec, the kubelet on that node ensures objects are created accordingly. The kubelet is the node-agent that resides on each worker node. The kubelet is also responsible for registering a node with a Kubernetes cluster, and sending events, pod status, and resource utilization reports to the master server.

At frequent intervals, the kubelet, via the API, updates the etcd on the master with the *Object status* of objects. This is the *actual state* of the objects. The watch functionality of etcd monitors changes to the *Object spec* (i.e., desired state) and the *Object status* (i.e., actual state). If the *Object spec* and *Object status* do not match, then the control loops run by the controller manager respond to these discrepancies and work to make the actual state of all the objects in the system match the desired state that the user provided. For example, if the kubelet reports that a Pod in a ReplicaSet is down, then the etcd's watch functionality figures out that the *object spec* is not in sync with the *object status*. To manage the state of objects, the controller manager, through control loops, sends instructions (via API) to the kubelet to create another Pod or restart the Pod that is down, and thus restores the ReplicaSet object to its desired state.

Now, if the kubelet on the worker node fails to create a desired object - say, a Pod - then the desired state of the cluster will not be restored. Likewise, if a Pod running a critical application/service suddenly goes down, and the kubelet fails to restart that Pod or create another one in its stead, then again the actual state will not be in sync with the desired state. Under such circumstances, containerized applications and services may be rendered unavailable to end- users. Since Kubernetes is widely used in mission-critical environments - eg.,microservices, DevOps, serverless computing, and multi- cloud environments - for processing business- critical workloads, the non-availability of applications can adversely impact productivity and business continuity. To avoid this, administrators must closely monitor the status of the objects managed and operations performed by Kubernetes, proactively capture abnormalities, and resolve them well before end-users notice. This is where eG Enterprise helps!

eG Enterprise provides a dedicated monitoring model for those Kubernetes clusters that manage Docker hosts and containers. This model continuously monitors the status of the cluster nodes, the Kubernetes control plane services running on the master node, and the workloads and application services on the worker nodes. In the process, the test promptly detects and alerts administrators to real/potential operational failures that may cause a mismatch between the actual state of objects and the desired cluster state. Rapid problem detection enables swift problem resolution, which in turn ensures the high availability of business-critical applications/services running within the containers in the cluster.

1.1 How Does eG Enterprise Monitor a Kubernetes Cluster?

eG Enterprise monitors Kubernetes in an agentless manner. A single remote agent deployed on a Windows host in the environment uses the Kubernetes API on the master node of the Kubernetes cluster to pull useful metrics on cluster performance.

To enable the eG agent to use the Kubernetes API, you need to:

1. Configure the eG agent to connect to the master node of the Kubernetes cluster

2. Configure the eG agent with an authentication bearer token

Each of these requirements have been discussed in detail below.

1.1.1 Configuring the eG Agent to Connect to the Master Node

To connect to the Kubernetes API, you first need to configure the eG agent with the IP address of the master node of the cluster. If the target cluster consists of more than one master node, then the eG agent should be configured to connect to the load balancer that is managing the cluster. In this case, the load balancer will route the eG agent's connection request to any available master node in the cluster, thus enabling the agent to connect with the API server on that node, run API commands on it, and pull metrics.

You can provide the IP address of the master node/load balancer when adding a Kubernetes cluster for monitoring using the eG administrative interface. Refer to Section **Chapter 2** to know how.

1.1.2 Configuring the eG Agent with an Authentication Bearer Token

To access the Kubernetes API, run API commands on it, and pull metrics, the eG agent has to be configured with a valid authentication bearer token. To generate this token, follow the steps below:

1. Go to the master node shell in the Kubernetes cluster.
2. Type the below command to create the "eginnovations" service account

   kubectl create -f https://raw.githubusercontent.com/eGInnovationsInc/kubernetes/master/eginnovations.yaml

3. Type the below command to get the secret name associated with "eginnovations" service account. Usually, the secret name is in the format "eginnovations-token-xxxxx".

   kubectl get serviceaccount eginnovations | grep -i tokens

4. Type the below command and replace the <xxxxx> with the secret name token from step 3 to get the authentication bearer token.

   kubectl describe secrets <xxxxx> | grep -I "token:"

5. Copy the token from step 4 and paste to the **AUTHENTICATION TOKEN** field in the monitoring information section of the **Kubernetes Cluster Preferences** page that appears when managing a Kubernetes cluster using the eG admin interface. To know how to manage a cluster using the eG admin interface, refer to Section **Chapter 2**

Chapter 2: How to Monitor the Kubernetes Cluster Using eG Enterprise?

To monitor the Kubernetes cluster using eG Enterprise, you need to first manage it .

eG Enterprise does not automatically discover the Kubernetes cluster. To manage the cluster therefore, you need to manually add it to the eG Enterprise system using the eG admin interface. For this, follow the steps below:

1. Login to the eG admin interface.
2. Figure 2.1 then appears prompting you to pick what you want to monitor. Select **Containers** from Figure 2.1.

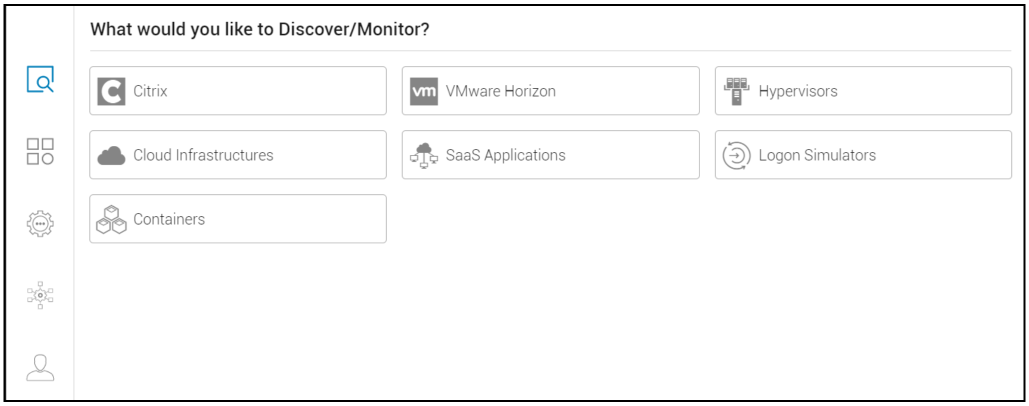
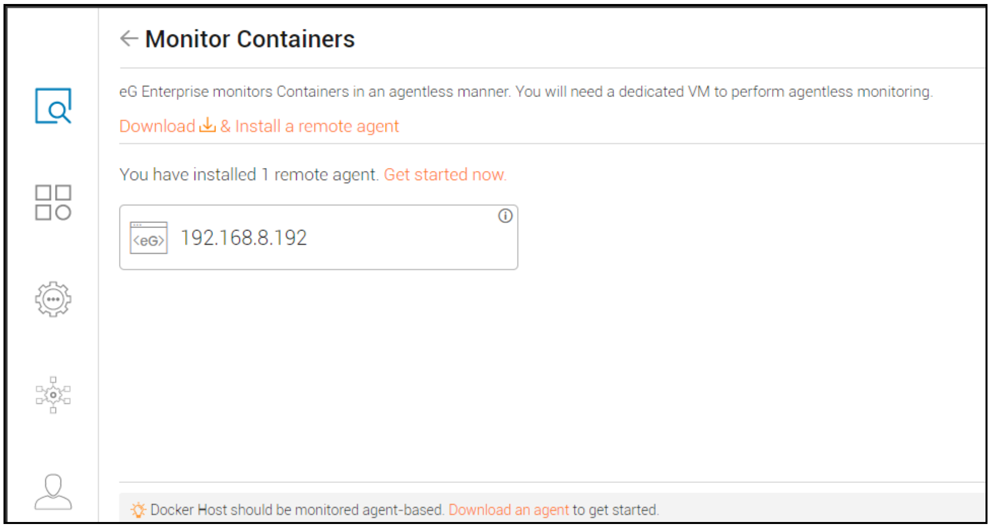


Figure 2.1: Choosing to monitor Containers

3. Since eG Enterprise monitors containers in an agentless manner, eG Enterprise automatically displays the remote agents that may pre-exist in the target environment (see Figure 2.2). From the list in Figure 2.2, select the remote agent you want to use for monitoring the Kubernetes cluster, by clicking on it.



← Monitor Containers

eG Enterprise monitors Containers in an agentless manner. You will need a dedicated VM to perform agentless monitoring.

Download & Install a remote agent

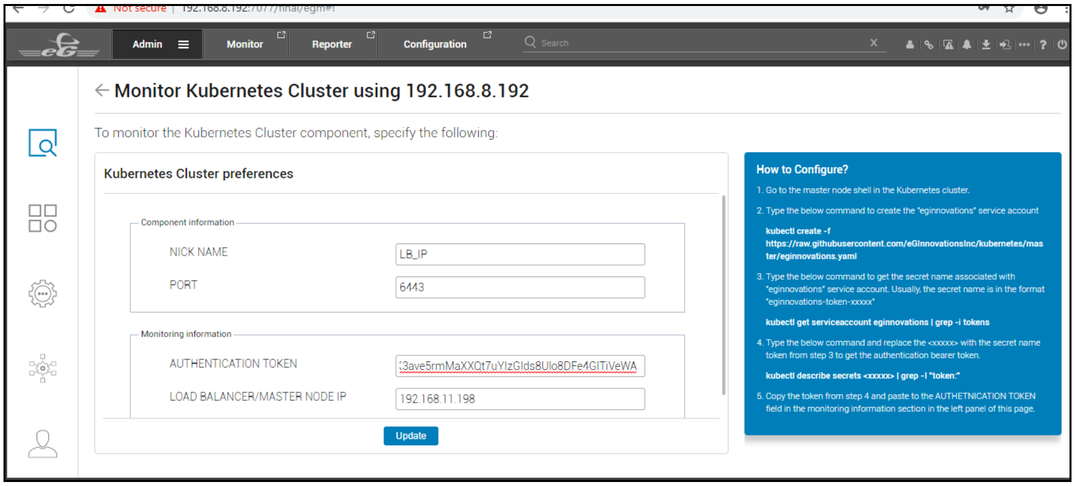
You have installed 1 remote agent. Get started now.

192.168.8.192

Docker Host should be monitored agent-based. Download an agent to get started.

Figure 2.2: Selecting the remote agent that should monitor the Kubernetes cluster

4. This will invoke Figure 2.3, using which you can configure the details of the Kubernetes cluster you want to monitor.



← Monitor Kubernetes Cluster using 192.168.8.192

To monitor the Kubernetes Cluster component, specify the following:

Kubernetes Cluster preferences

Component information

NICK NAME	LB_IP
PORT	6443

Monitoring information

AUTHENTICATION TOKEN	3ave5rmMaXXQt7uYIzGIds8UIo8DFe4GITiVeWA
LOAD BALANCER/MASTER NODE IP	192.168.11.198

Update

How to Configure?

1. Go to the master node shell in the Kubernetes cluster.
2. Type the below command to create the "eginnovations" service account

   kubectl create -f https://raw.githubusercontent.com/eGInnovationsInc/kubernetes/master/eginnovations.yaml

3. Type the below command to get the secret name associated with "eginnovations" service account. Usually, the secret name is in the format "eginnovations-token-xxxxx"

   kubectl get serviceaccount eginnovations | grep -i tokens

4. Type the below command and replace the <xxxxx> with the secret name token from step 3 to get the authentication bearer token.

   kubectl describe secrets <xxxxx> | grep -I "token:"

5. Copy the token from step 4 and paste to the AUTHETNICATION TOKEN field in the monitoring information section in the left panel of this page.

Figure 2.3: Adding a Kubernetes cluster

5. In Figure 2.3, specify the following:

- Provide a unique **NICK NAME** for the Kubernetes cluster you want to monitor.
- Specify the **PORT** at which the cluster listens. The default port is 6443.
- The eG agent requires an authentication bearer token to access the Kubernetes API, run API commands on the cluster, and pull metrics of interest. The steps for generating this token have been detailed in Section **1.1** You will also find these steps displayed in the right panel of

Figure 2.3. Once you generate a token by following the displayed (or documented) steps, copy the token and paste it against **AUTHENTICATION TOKEN** in Figure 2.3.

- The Kubernetes API server exists on the master node of a Kubernetes cluster. This means that the eG agent should connect to the master node to use the API. To enable this connection, specify the IP address of the master node in the **LOAD BALANCER / MASTER NODE IP** text box in Figure 2.3. Some Kubernetes clusters may support multiple master nodes. When monitoring such a cluster, you will have to configure the eG agent with the IP address of the load balancer that is managing the cluster. In other words, you will have to specify the IP address of the load balancer in the **LOAD BALANCER / MASTER NODE IP** text box of Figure 2.3. In this case, the load balancer will route the eG agent's connection request to any available master node in the cluster, thus enabling the agent to connect with the API server on that node.
- Finally, click the **Update** button in Figure 2.3 to add the component to the eG Enterprise system.

Chapter 3:

Chapter 3: Monitoring the Kubernetes Cluster

Figure 1 depicts the layer model of the Kubernetes Cluster.

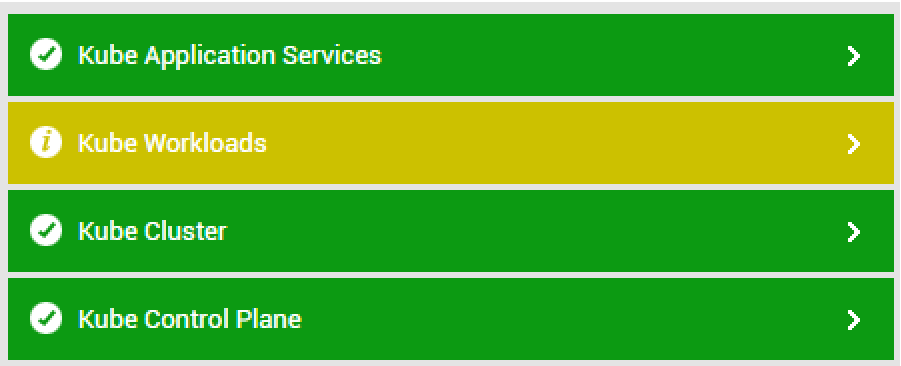


Figure 3.1: Layer model of the Kubernetes Cluster

Each layer of Figure 1 is mapped to tests that report a wide variety of status metrics - eg., node status, Pod status, Daemonset status, etc. - thus bringing abnormalities to the attention of administrators. Using these metrics, administrators can find quick and accurate answers to the following performance queries:

- Is the Kubernetes API server available?
- Are any nodes running Daemonsets they should not?
- Are any nodes not running the Daemonsets they should?
- Are all Deployments healthy? If not, whih are the Deployments that failed to create the desired number of Pod replicas?
- Is any Deployment unavailable?
- Has any Deployment failed to update all desired Pod replicas with changes to the Pod template?
- Are backoff conditions not allowing any Horizonal Pod autoscaler to perform scaling?
- Is any autoscaler unable to compute scales? If so, why?
- Has the scaling ability of any autoscaler been inhibited by the replica limits set?
- Has the target utilization level for scaling been set correctly for all autoscalers?

- Did any autoscaler fail to scale the current replica count to the desired levels?
- Have any Jobs failed in a namespace? Which one is it?
- Are all Pods in a namespace, which were created by Jobs, running?
- Did any Job take too long to run? If so, which one is it?
- Are all nodes running? Which nodes are not running?
- Has any node been marked as 'unschedulable'? If so, which one?
- Is any node in a bad condition? If so, why? Is it because of a network misconfiguration? insufficient disk space? low memory? process pressure?
- Are all nodes ready to accept Pods? Which are the ones that are not ready?
- Is any node running to full Pod capacity?
- Are any node's resources been overcommitted? If so, which resource (CPU or memory) has been overcommitted, and which Pods on the nodes are over-subscribing to that resource?
- Is any node running out of CPU or memory resources?
- How many master and worker nodes does the cluster have?
- Are there any Pending Pods in the cluster? Which are they?
- Have any Pods in the cluster failed?
- Is the write-through cache of the etcd used optimally?
- Are Golang collectors spending too much time in garbage collection?
- Are all key master services up and running?
- Are any namespaces terminating?
- Has any namespace exhausted or is about to exhaust its quota of Pods and/or services?
- Is the (CPU and/or memory) resource quota of any namespace nearing exhaustion?
- Are there any free Persistent Volumes, or are all of them bound to a claim?
- Has any Persistent Volume failed automatic reclamation?
- How many Pods in a namespace are ready to serve requests? Which ones are they?
- Which Pod is in what phase of its lifecycle?
- Are there any Pods with containers that are not ready to service requests?
- Which Pods are not yet scheduled to nodes, and why?

- Does any Pod have containers that terminated abnormally? If so, which containers and which Pod terminated so, and why?
- Are any Services in a namespace in a Pending state currently? If so, why?
- Have any failure/problem events been detected recently in the Kubernetes cluster? What events are those - did Pod creation fail? did any containers get killed? did Pods get evicted? did any nodes run out of resources? did auto-scaling fail for any HPA? When did such events occur, why, and which nodes and Pods were impacted?

3.1 The Kube Control Plane Layer

Using the test mapped to this layer, you can:

- Track and capture failure events that occur in the Kubernetes cluster;
- Detect the unavailability of the Kube API server;
- Identify master services that are not running;
- Capture issues in garbage collection by Golang collectors

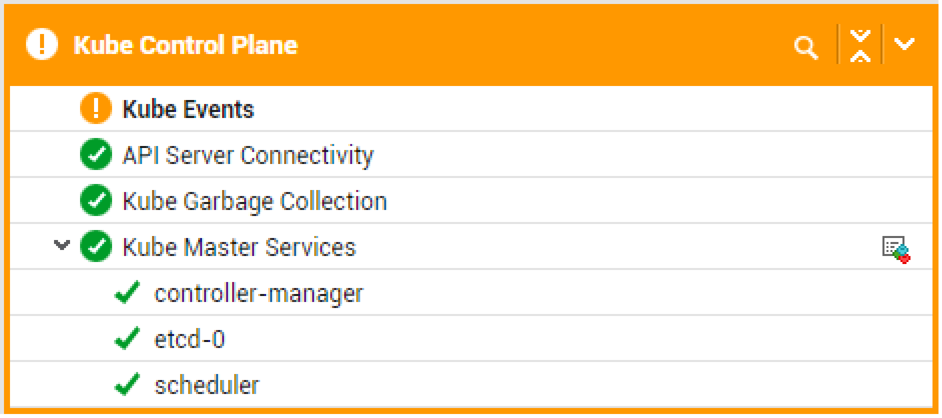


Figure 3.2: The tests mappedto the Kube Control Plane layer

3.1.1 Kube Events Test

Kubernetes events are a resource type in Kubernetes that are automatically created when other resources have state changes, errors, or other messages that should be broadcast to the system. These events are an invaluable resource when debugging issues in a Kubernetes cluster.

Hence, to be able to rapidly detect and troubleshoot issues impacting cluster performance, administrators should keep an eye out for Kubernetes events, and capture these events whenever they are created. The Kube Events test helps administrators achieve this!

This test intercepts Kubernetes events as and when they are created by the Kubernetes system, and brings every such event to the notice of administrators. Such events can point to normal cluster operations - eg., Pod creation, container creation etc. - and also abnormalities such as image pulling failures, scheduling failures etc. Whenever the test alerts administrators to an error or a failure event, administrators can use the detailed diagnostics provided by the test to determine why the error/failure occurred. This can greatly help in troubleshooting problem events.

Target of the test : A Kubernetes Cluster

Agent deploying the test : A remote agent

Outputs of the test : One set of results for the Kubernetes cluster being monitored

Configurable parameters for the test

Parameter	Description
Test Period	How often should the test be executed.
Host	The IP address of the host for which this test is to be configured.
Port	Specify the port at which the specified Host listens. By default, this is *6443*.
Load Balancer / Master Node IP	To run this test and report metrics, the eG agent needs to connect to the Kubernetes API on the master node and run API commands. To enable this connection, the eG agent has to be configured with either of the following: • If only a single master node exists in the cluster, then configure the eG agent with the IP address of the master node. • If the target cluster consists of more than one master node, then you need to configure the eG agent with the IP address of the load balancer that is managing the cluster. In this case, the load balancer will route the eG agent's connection request to any available master node in the cluster, thus enabling the agent to connect with the API server on that node, run API commands on it, and pull metrics. By default, this parameter will display the **Load Balancer / Master Node IP** that you configured when manually adding the Kubernetes cluster for monitoring, using the **Kubernetes Cluster Preferences** page in the eG admin interface (see Figure 2.3). The steps for managing the cluster using the eG admin interface are discussed elaborately in How to Monitor the Kubernetes Cluster Using eG Enterprise? Whenever the eG agent runs this test, it uses the IP address that is displayed (by default) against this parameter to connect to the Kubernetes API. If there is any change

Parameter	Description
	in this IP address at a later point in time, then make sure that you update this parameter with it, by overriding its default setting.
SSL	By default, the Kubernetes cluster is SSL-enabled. This is why, the eG agent, by default, connects to the Kubernetes API via an HTTPS connection. Accordingly, this flag is set to **Yes** by default. If the cluster is not SSL-enabled in your environment, then set this flag to **No**.
Authentication Token	The eG agent requires an authentication bearer token to access the Kubernetes API, run API commands on the cluster, and pull metrics of interest. The steps for generating this token have been detailed in Section **1.1** Typically, once you generate the token, you can associate that token with the target Kubernetes cluster, when manually adding that cluster for monitoring using the eG admin interface. The steps for managing the cluster using the eG admin interface are discussed elaborately in Section **Chapter 2** By default, this parameter will display the **Authentication Token** that you provided in the **Kubernetes Cluster Preferences page** of the eG admin interface, when manually adding the cluster for monitoring (see Figure 2.3). Whenever the eG agent runs this test, it uses the token that is displayed (by default) against this parameter for accessing the API and pulling metrics. If for any reason, you generate a new authentication token for the target cluster at a later point in time, then make sure you update this parameter with the change. For that, copy the new token and paste it against this parameter.
Proxy Host	If the eG agent connects to the Kubernetes API on the master node via a proxy server, then provide the IP address of the proxy server here. If no proxy is used, then the default setting -*none* - of this parameter, need not be changed,
Proxy Port	If the eG agent connects to the Kubernetes API on the master node via a proxy server, then provide the port number at which that proxy server listens here. If no proxy is used, then the default setting -*none* - of this parameter, need not be changed,
Proxy Username, Proxy Password, Confirm Password	**These parameters are applicable only if the eG agent uses a proxy server to connect to the Kubernetes cluster, and that proxy server requires authentication.** In this case, provide a valid user name and password against the **Proxy Username** and **Proxy Password** parameters, respectively. Then, confirm the password by retyping it in the **Confirm Password** text box. If no proxy server is used, or if the proxy server used does not require authentication, then the default setting - *none* - of these parameters, need not be changed.

Parameter	Description
DD Frequency	Refers to the frequency with which detailed diagnosis measures are to be generated for this test. The default is *1:1*. This indicates that, by default, detailed measures will be generated every time this test runs, and also every time the test detects a problem. You can modify this frequency, if you so desire. Also, if you intend to disable the detailed diagnosis capability for this test, you can do so by specifying *none* against DD frequency.
Detailed Diagnosis	To make diagnosis more efficient and accurate, the eG Enterprise suite embeds an optional detailed diagnostic capability. With this capability, the eG agents can be configured to run detailed, more elaborate tests as and when specific problems are detected. To enable the detailed diagnosis capability of this test for a particular server, choose the **On** option. To disable the capability, click on the **Off** option. The option to selectively enable/disable the detailed diagnosis capability will be available only if the following conditions are fulfilled: • The eG manager license should allow the detailed diagnosis capability • Both the normal and abnormal frequencies configured for the detailed diagnosis measures should not be 0.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Terminated all pods	Indicates the number of times all the Pods in the cluster were terminated during the last measurement period.	Number	Use the detailed diagnosis of this measure to know which Pods on which nodes were terminated in which namespace, and why.
Nodes registered	Indicates the number of nodes that were registered during the last measurement period.	Number	
Removing nodes	Indicates the number of nodes that were gracefully removed/drained during the last measurement period.	Number	Draining a node does the following: • It cordons the node: Cordoning a node means that it will be marked unschedulable, so new pods can no longer be scheduled to the

Measurement	Description	Measurement Unit	Interpretation
			node. - It evicts or deletes the Pods on that node: After the node is made unschedulable, the drain command will try to evict the pods that are already running on that node. If eviction is supported on the cluster (from Kubernetes version 1.7) the drain command will use the Eviction API that takes disruption budgets into account, if it’s not supported it will simply delete the pods on the node. Use the detailed diagnosis of this measure to know which which nodes wereremoved/drained from which namespace, and why.
Deleting nodes	Indicates the number of nodes that were deleted during the last measurement period.	Number	Deleting the node object from Kubernetes causes all the Pod objects running on the node to be deleted from the apiserver, and frees up their names. Use the detailed diagnosis of this measure to know which nodes were deleted from which namespace.
Deleting all pods	Indicates the number of times all Pods on a node were deleted since the last measurement period.	Number	Use the detailed diagnosis of this measure to know which Pods were deleted from which nodes in which namespace, and why.
Terminating evicted pods	Indicates the number of times since the last measurement period, Pods were evicted.	Number	One of the most useful events to monitor is when a node begins

Measurement	Description	Measurement Unit	Interpretation
			evicting pods. This event happens when a node determines that pods need to be evicted to free up some resource such as CPU, memory, or disk. An eviction can have devastating consequences if the kubelet is unable to determine the best resources to evict. For instance, the kubelet detecting disk pressure may sometimes evict Pods that have no effect on disk usage. The evicted Pods may also get scheduled on other nodes, overloading their other resources and also causing evictions. Knowing when evictions happened, and being able to correlate it with other events in that time frame, can help avoid the issue. You can use the detailed diagnosis of this measure to know which Pods were evicted and when eviction occurred.
Ready nodes	Indicates the number of times the NodeReady event occurred since the last measurement period.	Number	
Nodes not ready	Indicates the number of times the NodeNotReady event occurred during the last measurement period.	Number	Use the detailed diagnosis of this measure to know which nodes were not ready , and when the event occurred.
Nodes are schedulable	Indicates the number of times the NodeSchedulable event occurred during the last measurement period.	Number	Use the detailed diagnosis of this measure to know when this event started, when it ended, and which nodes were found schedulable in the process, and which Pods were scheduled to those nodes.
CIDR not	Indicates the number of times the	Number	Kubernetes assigns each node a

Measurement	Description	Measurement Unit	Interpretation
available	CIDRNotAvailable event occurred during the last measurement period.		range of IP addresses, a CIDR block, so that each Pod can have a unique IP address. Use the detailed diagnosis of this measure to know when this event occurred, why, on which nodes, and which Pods were impacted.
CIDR assignments failed	Indicates the number of times the CIDRAssignmentFailed event occurred during the last measurement period.	Number	Use the detailed diagnosis of this measure to know when this event occurred, why, on which nodes, and which Pods were impacted.
Starting kubelets	Indicates the number of times the Starting event occurred during the last measurement period.	Number	
Kubelet setup failed	Indicates the number of times the KubeletSetupFailed event occurred during the last measurement period.	Number	Use the detailed diagnosis of this measure to know when this event occurred, why, on which nodes, and which Pods were impacted.
Volume mounts failed	Indicates the number of times the FailedMount event occurred during the last measurement period.	Number	The FailedMount and FailedAttachVolume events can help you debug issues with storage. These events will prevent Pods from starting correctly. You may think that your Pods are just slow to start, but if there are permissions or networking issues when creating network volumes, you will need to rectify them to get your Pods working again. Use the detailed diagnosis of this measure to know when this event occurred, why, on which nodes, and which Pods were impacted.
Nodes selector mismatch	Indicates the number of times in the last measurement period, the NodeSelectorMismatching event occurred.	Number	You can constrain a Pod to only be able to run on particular Node(s), or to prefer to run on particular nodes. nodeSelector is the simplest

Measurement	Description	Measurement Unit	Interpretation
			recommended form of node selection constraint. nodeSelector is a field of PodSpec. It specifies a map of key-value pairs. For the pod to be eligible to run on a node, the node must have each of the indicated key-value pairs as labels (it can have additional labels as well). The most common usage is one key-value pair. If, when attempting to schedule a Pod, scheduler finds that that Pod's nodeSelector does not have any matching node, the NodeSelectorMismatching event is triggered. In this case, the Pod in question will remain in the Pending state until a matching node is found. .
Insufficient free CPU	Indicates the number of times during the last measurement period the InsufficientFreeCpu event was triggered.	Number	If a Pod is stuck in the Pending state, it means that it can not be scheduled onto a node. Generally this is because there are insufficient resources of one type or another that prevent scheduling. The scheduler triggers an InsufficientFreeCpu or an InsufficientFreeMemory event at around such times. In this case you can try several things: • Add more nodes to the cluster. • Terminate unneeded pods to make room for pending pods. • Check that the pod is not larger than your nodes. For example, if all nodes have a capacity of

Measurement	Description	Measurement Unit	Interpretation
			cpu:1, then a pod with a request of cpu: 1.1 will never be scheduled. You can use the detailed diagnosis of these measures to identify the nodes on which the events occurred, when it occurred, which Pods were impacted, and why.
Insufficient free memory	Indicates the number of times during the last measurement period the InsufficientFreeMemory event was triggered.	Number	
Out of disk nodes	Indicates the number of OutofDisk events that occurred during the last measurement period.	Number	OutOfDisk indicates that the file system on the worker node is full. Kubernetes begins migrating pods off the node until the situation is fixed and the status of the node moves back to Ready. Use the detailed diagnosis of this measure to know when this event occurred, why, on which nodes, and which Pods were impacted.
Host network not supported	Indicates the number of HostNetworkNotSupported events that occurred during the last measurement period.	Number	If you use the host network mode for a container, that container's network stack is not isolated from the node's (the container shares the host's networking namespace), and the container does not get its own IP-address allocated. For instance, if you run a container which binds to port 80 and you use host networking, the container's application is available on port 80 on the node's IP address. Host mode networking can be useful to optimize performance, and in situations where a container needs to handle a large range of ports, as it does not require network address translation (NAT), and no "userland-proxy" is created for each port.
Undefined	Indicates the number of times the	Number	If Pod requests bandwidth shaping,

Measurement	Description	Measurement Unit	Interpretation
shaper	NilShaper event occurred during the last measurement period.		but the shaper is undefined, then this event occurs.
Nodes rebooted	Indicates the number of times the Rebooted event occurred during the last measurement period.	Number	Use the detailed diagnosis of this measure to know which nodes were rebooted, when, why, and which Pods were impacted.
Node has sufficient disk	Indicates the number of times the NodeHasSufficientDisk event was triggered during the last measurement period.	Number	
Nodes out of disk space	Indicates the number of times the NodeOutofDisk event occurred since the last measurement period.	Number	OutOfDisk indicates that the file system on the worker node is full. Kubernetes begins migrating pods off the node until the situation is fixed and the status of the node moves back to Ready. Use the detailed diagnosis of this measure to know when this event occurred, why, on which nodes, and which Pods were impacted.
Invalid disk capacity	Indicates the number of times the InvalidDiskCapacity event occurred since the last measurement period.	Number	Use the detailed diagnosis of this measure to know when this event occurred, why, on which nodes, and which Pods were impacted.
Free disk space failed	Indicates the number of times the FreeDiskSpaceFailed event occurred during the last measurement period.	Number	This event occurs if the host file system is full. One of the common reasons for this is the garbage collector's failure to delete any image.
Pulling images	Indicates the number of times the Pulling event occurred during the last measurement period.	Number	During the deployment of an application to a Kubernetes cluster, you will typically want one or more images to be pulled from a Docker registry. In the application's manifest file you specify the images to pull,

Measurement	Description	Measurement Unit	Interpretation
			the registry to pull them from, and the credentials to use when pulling the images. Use the detailed diagnosis of this measure to know when this event occurred, which image was being pulled, and which nodes and Pods were impacted by the event.
Images pulled	Indicates the number of times the Pulled event occurred during the last measurement period.	Number	Images are pulled based on the ImagePullPolicy. The default pull policy is IfNotPresent which causes the Kubelet to skip pulling an image if it already exists. If you would like to always force a pull, you can do one of the following: • set the imagePullPolicy of the container to Always. • omit the imagePullPolicy and use :latest as the tag for the image to use. • omit the imagePullPolicy and the tag for the image to use. • enable the AlwaysPullImages admission controller.
Images created	Indicates the number of times the Created event occurred during the last measurement period.	Number	
Images started	Indicates the number of times the Started event occurred during the last measurement period.	Number	
Failed to pull images	Indicates the number of times the Failed event occurred during the	Number	Common causes for failure to pull images are:

Measurement	Description	Measurement Unit	Interpretation
	last measurement period.		• Network connectivity issues • Incorrect image tag • The image does not exist • Kubernetes does not have permission to pull the image Use the detailed diagnosis of this measure to know when this event occurred, why, on which nodes, and which Pods were impacted.
Images neverpull policy violated	Indicates the number of times during the last measurement period the ErrImageNeverPull event occurred.	Number	The never pull policy disables images pulling completely. If this policy is set, then the image is assumed to exist locally. No attempt is made to pull the image. This pull policy should be used if you want or need to have a full control on which images are used. It is a good choice for containers that are dedicated to a project where only specific images can be used. If, when attempting to pull an image, the kubelet finds that the image is not present locally, then this policy is violated. Use the detailed diagnosis of this measure to know when this event occurred, why, on which nodes, and which Pods were impacted.
Back off pulling images	Indicates the number of times the BackOff event occurred in the last measurement period.	Number	This event is triggered, if: • There is an invalid container image tag; • Kubernetes does not have

Measurement	Description	Measurement Unit	Interpretation
			permissions to access the image; - The image does not exist; Use the detailed diagnosis of this measure to know when this event occurred, why, on which nodes, and which Pods were impacted.
Containers created	Indicates the number of times the Created event occurred in the last measurement period.	Number	This event is triggered every time a container is created.
Killing containers	Indicates the number of times the Killing event occurred in the last measurement period.	Number	This event is triggered every time a container is killed. Use the detailed diagnosis of this measure to know when this event occurred, why, on which nodes, and which Pods were impacted.
Containers unhealthy	Indicates the number of times the Unhealthy event occurred in the last measurement period.	Number	There is a single main process that is running in a container. Such a process can start other child processes within a container, if necessary. Every such process, including the main process, can have its own lifecycle – but if the main process stops, the container stops as well. A container is healthy, by the most general definition, if its main process is running. If the container's main process is terminated unexpectedly, then the container is considered unhealthy. Use the detailed diagnosis of this measure to know when this event occurred, why, on which nodes, and which Pods were impacted.
Pods sync	Indicates the number of times the	Number	Use the detailed diagnosis of this

Measurement	Description	Measurement Unit	Interpretation
failed	FailedSync event occurred in the last measurement period.		measure to know when this event occurred, why, on which nodes, and which Pods were impacted.
Failed pods config validation	Indicates the number of times the FailedValidation event occurred in the last measurement period.	Number	Use the detailed diagnosis of this measure to know when this event occurred, why, on which nodes, and which Pods were impacted.
Out of disk in pods	Indicates the number of times the OutOfDisk event occurred in the last measurement period.	Number	Sometimes, the container(s) running in a Pod can fill up disk space, triggering an OutOfDisk event. Use the detailed diagnosis of this measure to know when this event occurred, why, on which nodes, and which Pods were impacted.
Host/Port conflict	Indicates the number of times the HostPortConflict event occurred in the last measurment period.	Number	
Pods created	Indicates the number of times the SuccessfulCreate event occurred in the last measurement period.	Number	
Failed replicaset	Indicates the number of times the FailedCreate event occurred in the last measurement period.	Number	This event is triggered if a ReplicationController fails to create Pods. In such a case, use the detailed diagnosis of this measure to know when this event occurred, why, on which nodes, and which Pods were impacted. Typically, if a ReplicationController cannot create Pods, you may have to debug the Pods. The first step in debugging a Pod is taking a look at it. Check the current state of the Pod and recent events: Look at the state of the containers in the Pod. Are they all running? Have there been recent restarts? Then, continue debugging

Measurement	Description	Measurement Unit	Interpretation
			depending on the state of the Pods. Are Pods stuck in a Pending state? Then, check for resource inadequacies. Are the Pods in Waiting state? Then, check if there are any issues in image pulling. Are the Pods crashing? Then, study the container logs for troubleshooting the same.
Pods deleted	Indicates the number of times the SuccessfulDelete event occurred in the last measurement period.	Number	
Pods delete failed	Indicates the number of times the FailedDelete event occurred in the last measurement period.	Number	Use the detailed diagnosis of this measure to know when this event occurred, why, on which nodes, and which Pods were impacted.
Preempting containers	Indicates the number of times the PreemptContainer event occurred in the last measurement period.	Number	
Containers exceeded grace period	Indicates the number of times the ExceededGracePeriod event occurred in the last measurement period.	Number	As part of the graceful termination lifecycle, Kubernetes first sends a SIGTERM signal to the containers in a Pod to let the containers know that they are going to be shut down soon. At this point, Kubernetes waits for a specified time called the termination grace period. This is 30 seconds by default. If the containers in the Pod are still running after the grace period, Kubernetes triggers the ExceededGracePeriod event, and sends the SIGKILL signal to forcibly remove the containers. Use the detailed diagnosis of this measure to know when this event occurred, why, on which nodes, and which Pods were impacted.

Measurement	Description	Measurement Unit	Interpretation
Failed to attach volume	Indicates the number of times the FailedAttachVolume event occurred in the last measurement period.	Number	The FailedAttachVolume is an error that occurs when Persistent Volume (PV) is unable to be detached from a node. This means it can no longer be attached to another node and happens because Kubernetes will not force detatch PVs from nodes. In other words, the FailedAttachVolume event is an outcome of a fundamental failure to unmount and detach the volume from the failed node. The FailedMount event typically follows the FailedAttachVolume event because the mount operation happens after the attach operation and because the attach has already failed, it means that the mount operation is not possible. The FailedMount and FailedAttachVolume events can help you debug issues with storage. These events will prevent Pods from starting correctly. You may think that your Pods are just slow to start, but if there are permissions or networking issues when creating network volumes, you will need to rectify them to get your Pods working again. Use the detailed diagnosis of these measures to know when these events occurred, why, on which nodes, and which Pods were impacted.
Failed mount	Indicates the number of times the FailedMount event occurred in the last measurement period.	Number	
Volume resize failed	Indicates the number of times the VolumeResizeFailed event occurred in the last measurement period.	Number	Typically, if a PVC is already attached to a Pod, then resizing that PVC would fail with the

Measurement	Description	Measurement Unit	Interpretation
			VolumeResizeFailed event. In such cases, update the size of the PV, then edit the PVC accordingly, and delete the Pod to get it to the detached state. Then, recreate that Pod. Use the detailed diagnosis of this measure to know when this event occurred, why, on which nodes, and which Pods were impacted.
File system resize failed	Indicates the number of times the FileSystemResizeFailed event occurred in the last measurement period.	Number	This event is triggered if errors are encountered when expanding the file system. Use the detailed diagnosis of this measure to know when this event occurred, why, on which nodes, and which Pods were impacted.
Failed map volume	Indicates the number of times the FailedMapVolume event occurred in the last measurement period.	Number	Use the detailed diagnosis of this measure to know when this event occurred, why, on which nodes, and which Pods were impacted.
Container GC failed	Indicates the number of times the ContainerGCFailed event occurred in the last measurement period.	Number	Typically, whenever Pod eviction thresholds are too close to the node's physical memory limits, one of these events will be triggered. Use the detailed diagnosis of these measures to know when these events occurred, why, on which nodes, and which Pods were impacted.
Image GC failed	Indicates the number of times the ImageGCFailed event occurred in the last measurement period.		
Failed node allocatable enforcement	Indicates the number of times the FailedNodeAllocatableEnforcement event occurred in the last measurement period.	Number	The kubelet exposes a feature named *Node Allocatable* that helps to reserve compute resources for system daemons. *Allocatable* on a Kubernetes node is defined as the amount of compute

Measurement	Description	Measurement Unit	Interpretation
			resources that are available for Pods. To properly enforce node allocatable constraints on the node, you must enable the new cgroup hierarchy via the --*cgroups-per-qos* flag. This flag is enabled by default. When enabled, the kubelet will parent all end-user pods under a cgroup hierarchy managed by the kubelet. Following is the recommended cgroup configuration for Kubernetes nodes. All OS system daemons are expected to be placed under a top level *SystemReserved* cgroup. Kubelet and Container Runtime are expected to be placed under *KubeReserved* cgroup. *kube-reserved* is meant to capture resource reservation for kubernetes system daemons like the kubelet, container runtime, node problem detector, etc. *system-reserved* is meant to capture resource reservation for OS system daemons like sshd, udev, etc To optionally enforce *system-reserved*on system daemons, specify the parent control group for OS system daemons as the value for --*system-reserved-cgroup* kubelet flag. If this specification includes an invalid cgroup, then Kubelet will fail to enforce system-reserved, and will trigger the FailedNodeAllocatableEnforcement

Measurement	Description	Measurement Unit	Interpretation
			event.
Sandbox changed	Indicates the number of times the SandboxChanged event occurred in the last measurement period.	Number	Whenever the config map or any other part of a Pod setup changes, the SandboxChanged event is triggered.
Failed to create pod sandbox	Indicates the number of times the FailedCreatePodSandBox event occurred in the last measurement period.	Number	At the lowest layers of a Kubernetes node is the software that, among other things, starts and stops containers. We call this the “Container Runtime”. The plugin API for container runtimes in Kubernetes is called Container Runtime Interface (CRI). A Pod is composed of a group of application containers in an isolated environment with resource constraints. In CRI, this environment is called PodSandbox. Before starting a Pod, kubelet calls RuntimeService.RunPodSandbox to create the environment. This includes setting up networking for a pod (e.g., allocating an IP). If kubelet is unable to create the environment for running a Pod, the FailedCreatePodSandBox event is triggered. Use the detailed diagnosis of this measure to know when this event occurred, why, on which nodes, and which Pods were impacted.
Failed pod sandbox status	Indicates the number of times the FailedPodSandBoxStatus event occurred in the last measurement period.	Number	If kubelet is unable to get the Pod sandbox status, then the FailedPodSandBoxStatus event is triggered. Use the detailed diagnosis of this

Measurement	Description	Measurement Unit	Interpretation
			measure to know when this event occurred, why, on which nodes, and which Pods were impacted.
Container probe warnings	Indicates the number of times the ContainerProbeWarning event occurred in the last measurement period.	Number	A Probe is a diagnostic performed periodically by the kubelet on a Container. The kubelet can optionally perform and react to three kinds of probes on running Containers: • livenessProbe: Indicates whether the Container is running. If the liveness probe fails, the kubelet kills the Container, and the Container is subjected to its restart policy. • readinessProbe: Indicates whether the Container is ready to service requests. If the readiness probe fails, the endpoints controller removes the Pod's IP address from the endpoints of all Services that match the Pod. • startupProbe: Indicates whether the application within the Container is started. All other probes are disabled if a startup probe is provided, until it succeeds. If the startup probe fails, the kubelet kills the Container, and the Container is subjected to its restart policy.

Measurement	Description	Measurement Unit	Interpretation
			A ContainerProbeWarning event is triggered when any of these probes fail. Use the detailed diagnosis of this measure to know when this event occurred, why, on which nodes, and which Pods were impacted.
Failed post start hook	Indicates the number of times the FailedPostStartHook event occurred during the last measurement period.	Number	Hooks enable Containers to be aware of events in their management lifecycle and run code implemented in a handler when the corresponding lifecycle hook is executed. There are two hooks that are exposed to Containers: • PostStart: This hook executes immediately after a container is created. However, there is no guarantee that the hook will execute before the container ENTRYPOINT. No parameters are passed to the handler. • PreStop: This hook is called immediately before a container is terminated due to an API request or management event such as liveness probe failure, preemption, resource contention and others. A call to the preStop hook fails if the container is already in terminated or completed state. It is blocking, meaning it is synchronous, so it

Measurement	Description	Measurement Unit	Interpretation
			must complete before the call to delete the container can be sent. No parameters are passed to the handler. If a hook handler fails, it broadcasts an event. While failure of the PostStart hook handler triggers the FailedPostStartHook event, the failure of the PreStop hook handler triggers the FailedPreStopHook event. Use the detailed diagnosis of these measures to know when these events occurred, why, on which nodes, and which Pods were impacted.
Failed pre stop hook	Indicates the number of times the FailedPreStopHook event occurred during the last measurement period.	Number	
Node has sufficient memory	Indicates the number of times the NodeHasSufficientMemory event occurred in the last measurement period.	Number	If the MemoryPressure condition of a node is False, it implies that that node has sufficient memory. In such cases, the NodeHasSufficientMemory event is generated. Use the detailed diagnosis of this measure to know when this event occurred, on which nodes, and which Pods were impacted.
Failed resource metric	Indicates the number of times the FailedGetResourceMetric event occurred in the last measurement period.	Number	The Horizontal Pod Autoscaler automatically scales the number of Pods in a replication controller, deployment or replica set based on observed CPU utilization (or, with custom metrics support, on some other application-provided metrics). At configured intervals, the controller manager queries the resource

Measurement	Description	Measurement Unit	Interpretation
			utilization against the metrics specified in each HorizontalPodAutoscaler definition. The controller manager obtains the metrics from either the resource metrics API (for per-pod resource metrics), or the custom metrics API (for all other metrics). Typically, metrics are fetched from a series of aggregated APIs - metrics.k8s.io, custom.metrics.k8s.io, and external.metrics.k8s.io. The controller then calculates the actual utilization value of the resource, considers the target/desired utilization value that is set, and computes the ratio between the desired and actual metric value. The autoscaler then scales the desired number of replicas up or down based on this ratio. One of the common reasons for the failure of auto-scaling is the inability of the controller to fetch the resource metrics from the API. Without the metrics, scales cannot be computed, and consequently, the count of replicas cannot be scaled up/down. The FailedGetResourceMetric is broadcast everytime the controller fails to get resource metrics. Use the detailed diagnosis of this measure to know when this event occurred, why, on which nodes, and which Pods were impacted.
Node has no disk pressure	Indicates the number of times the NodeHasNoDiskPressure event occurred in the last measurement	Number	If the DiskPressure condition of a node is False, it implies that that node has sufficient disk space. In

Measurement	Description	Measurement Unit	Interpretation
	period.		such cases, the NodeHasNoDiskPressure event is generated. Use the detailed diagnosis of this measure to know when this event occurred, on which nodes, and which Pods were impacted.
Node has sufficient PID	Indicates the number of times the NodeHasSufficientPID event occurred in the last measurement period.	Number	If the PIDPressure condition of a node is False, it implies that that node has sufficient processes. In such cases, the NodeHasSufficientPID event is generated. Use the detailed diagnosis of this measure to know when this event occurred, on which nodes, and which Pods were impacted.
Provisioning failed	Indicates the number of times the ProvisioningFailed event occurred in the last measurement period.	Number	This event is triggered if Kubernetes fails to provision a volume for a PVC. If a PV belonging to a StorageClass needs to be dynamically provisioned for a PVC, then a key field that your StorageClass definition should contain is the Provisioner. A Provisioner determines what volume plugin is to be used for provisioning PVs dynamically. Likewise, the definition should also include mountOptions. In this case, if the Provisioner - i.e., volume plugin - in use does not support mount options, then volume provisioning will fail. Where multiple mountOptions are provided, provisioning failures will also occur if even one of the mount options is found to be invalid.

Measurement	Description	Measurement Unit	Interpretation
			Provisioning failures may also occur if: • The volume plugin does not match any of the supported plugins; • The application is requesting more storage space than is available in the underlying volumes that have been provisioned. Use the detailed diagnosis of this measure to know when this event occurred, why, on which nodes, and which Pods were impacted.
Back-off restarting failed containers	Indicates the number of times the BackOff event occurred in the last measurement period.	Number	If this event is triggered, it means that Kubernetes started your container, then the container subsequently exited. This forced Kubernetes to restart the container. After restarting it a few times, Kubernetes declares that the container is in the BackOff state. However, Kubernetes will keep on trying to restart it. Common causes for this are: • The application inside the container keeps crashing • Some type of parameters of the pod or container have been configured incorrectly • An error has been made when deploying Kubernetes Use the detailed diagnosis of this measure to know when this event occurred, why, on which nodes, and which Pods were impacted.

Measurement	Description	Measurement Unit	Interpretation
Completed jobs	Indicates the number of times the SawCompletedJob event occurred in the last measurement period.	Number	A Job creates one or more Pods and ensures that a specified number of them successfully terminate. As pods successfully complete, the Job tracks the successful completions. When a specified number of successful completions is reached, the task (ie, Job) is complete. This event is triggered every time a Job completes.
Error creating pods	Indicates the number of times the FailedCreate event occurred, with the message "Error creating: pods", in the last measurement period	Number	If a Job fails to create Pods, then this event is triggered. An entire Pod can fail for a number of reasons, such as when the Pod is kicked off the node (node is upgraded, rebooted, deleted, etc.), or if a container of the Pod fails and the .spec.template.spec.restartPolicy = "Never". When a Pod fails, then the Job controller starts a new Pod. Use the detailed diagnosis of this measure to know when this event occurred, why, on which nodes, and which Pods were impacted.
Successfully schedule pods	Indicates the number of times the Scheduled event occurred, in the last measurement period.	Number	This event is generated if a Pod is successfully scheduled to a node.
Failed to schedule pods	Indicates the number of times the FailedScheduling event occurred in the last measurement period.	Number	This event is generated if a Pod could not be scheduled to any node in a cluster. One of the common causes for scheduling failures is the lack of adequate memory and/or CPU resources in the nodes to accommodate the Pods. Use the detailed diagnosis of this measure to know when this event occurred, why, on which nodes, and

Measurement	Description	Measurement Unit	Interpretation
			which Pods were impacted.
Failed to stop pods	Indicates the number of times the FailedKillPod event occurred in the last measurement period.	Number	This event occurs if a Pod is stuck in the Terminating state. This is detected by finding Pods where every container has been terminated, but the Pod is still running. Usually, this is caused when a node in the cluster gets taken out of service abruptly, and the cluster scheduler and controller-manager do not clean up all of the pods on that node. Use the detailed diagnosis of this measure to know when this event occurred, why, on which nodes, and which Pods were impacted.
Failed to create a pod container	Indicates the number of times the FailedCreatePodContainer event occurred in the last measurement period.	Number	This event is generated if Kubernetes fails to create a container in a Pod. Use the detailed diagnosis of this measure to know when this event occurred, why, on which nodes, and which Pods were impacted.
Network is not ready	Indicates the number of times the NetworkNotReady event occurred in the last measurement period.	Number	This event is triggered if the Pod's runtime network is not ready.
Failed to place pods on node	Indicates the number of times the FailedPlacement event occurred in the last measurement period.	Number	This event is triggered if the Daemonset Controller fails to place a Pod on a node. Common reasons for this are: • Insufficient resources on the node; • The node has been marked as Unschedulable Use the detailed diagnosis of this

Measurement	Description	Measurement Unit	Interpretation
			measure to know when this event occurred, why, on which nodes.
Found failed daemon pods on node	Indicates the number of times the FailedDaemonPod event occurred in the last measurement period.	Number	This event is often associated with the cluster health rather than issues with the daemon set. Use the detailed diagnosis of this measure to know when this event occurred, why, on which nodes.
Failed to cancel deployments	Indicates the number of times the DeploymentCancellationFailed event occurred in the last measurement period.	Number	While a running Deployment can be canceled, most often, it is the stuck Deployments that are canceled. The cancellation is a best-effort operation, and may take some time to complete. The replication controller may partially or totally complete its deployment before the cancellation is effective. If a Deployment is successfully canceled, then the DeploymentCancelled event is triggered. When canceled, the deployment configuration will be automatically rolled back by scaling up the previous running replication controller. On the other hand, if cancellation of a Deployment fails, then the DeploymentCancellationFailed event is triggered. You can use the detailed diagnosis of these measures to know when each of these events occurred, why, and on which nodes.
Cancelled deployments	Indicates the number of times the DeploymentCancelled event occurred in the last measurement period.	Number	
Created new replication controllers	Indicates the number of times the DeploymentCreated event occurred in the last measurement period.	Number	This event is triggered every time a new Deployment is created. You can use the detailed diagnosis

Measurement	Description	Measurement Unit	Interpretation
			of this measure to know when this event occurred, on which nodes, and which Pods were created in the process.
No available ingress IP to allocate to service	Indicates the number of times the IngressIPRangeFull event occurred in the last measurement period.	Number	Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster. Traffic routing is controlled by rules defined on the Ingress resource. An Ingress can be configured to give Services externally-reachable URLs, load balance traffic, terminate SSL / TLS, and offer name based virtual hosting. An Ingress controller is responsible for fulfilling the Ingress, usually with a load balancer, though it may also configure your edge router or additional frontends to help handle the traffic. When an Ingress is created, typically, an IP address is allocated by the Ingress Controller to satisfy the Ingress. Ingress controllers and load balancers may take a minute or two to allocate an IP address. Until that time, you often see the address listed as <pending>. Sometimes, the Ingress Controller may not find any IP address to allocate to the service for which the Ingress was created. In this case, the Ingress will fail with the event IngressIPRangeFull. Use the detailed diagnosis of this measure to know when this event occurred, why, and which nodes and Pods were impacted by the event.

Measurement	Description	Measurement Unit	Interpretation
Failed to detach volumes	Indicates the number of times the FailedDetachVolume event occurred in the last measurement period.	Number	This event is triggered if a volume fails to be detached from a node. A Persistent Volume that cannot be detached poses a problem if you try to create another Pod using the same PVC. Use the detailed diagnosis of this measure to know when this event occurred, why, and which nodes and Pods were impacted by the event.
Failed to unmount volumes	Indicates the number of times the FailedUnMount event occurred in the last measurement period.	Number	This event is triggered if Kubernetes failed to unmount a volume from a node. Use the detailed diagnosis of this measure to know when this event occurred, why, and which nodes and Pods were impacted by the event.
Failed unmapped devices	Indicates the number of times the FailedUnmapDevice event occurred in the last measurement period.	Number	If a Pod mounted with a storage device - i.e., a volume - is deleted, then the tear down process should be able to unmap the device. If it fails to do so, then the FailedUnmapDevice event is triggered. Use the detailed diagnosis of this measure to know when this event occurred, why, and which nodes and Pods were impacted by the event.
Unsupported mount option	Indicates the number of times the UnsupportedMountOptionevent occurred in the last measurement period.	Number	If a PV belonging to a StorageClass needs to be dynamically provisioned for a PVC, then a key field that your StorageClass definition should contain is the Provisioner. A Provisioner determines what volume plugin is to be used for provisioning PVs dynamically.

Measurement	Description	Measurement Unit	Interpretation
			Additionally, the definition may also include mountOptions. In this case, if the Provisioner - i.e., volume plugin - in use does not support mount options, then the UnsupportedMountOption event will be triggered, resulting in the failure of volume provisioning. Where multiple mountOptions are provided, provisioning failures will also occur if even one of the mount options is found to be invalid. Use the detailed diagnosis of this measure to know when this event occurred, why, and which nodes and Pods were impacted by the event.
Invalid selector	Indicates the number of times the InvalidSelector event occurred in the last measurement period.	Number	If this event occurs, it implies that the target scalable's selector could not be parsed. Use the detailed diagnosis of this measure to know when this event occurred, why, and which nodes and Pods were impacted by the event.
Unknown metric source type	Indicates the number of times the InvalidMetricSourceType event occurred in the last measurement period.	Number	This event occurs if the HPA controller encounters an unknown metric source type. Use the detailed diagnosis of this measure to know when this event occurred, why, and which nodes and Pods were impacted by the event. One of the common
Failed to convert the given HPA	Indicates the number of times the FailedConvertHPA event occurred in the last measurement period.	Number	This event is fired if the the HPA controller was unable to convert the given HPA to the v2alpha1 version. Use the detailed diagnosis of this measure to know when this event

Measurement	Description	Measurement Unit	Interpretation
			occurred, why, and which nodes and Pods were impacted by the event.
HPA controller was unable to get the targets	Indicates the number of times the FailedGetScale event occurred in the last measurement period.	Number	This event is triggered if the Horizontal Pod Autoscaler (HPA) was not able to get the scale for the given scalable resource. If this event occurs, then the HPA will be unable to perform up/down scaling. Therefore, the AbleToScale status condition of the HPA will become False. Use the detailed diagnosis of this measure to know when this event occurred, why, and which nodes and Pods were impacted by the event.
Failed to compute desired number of replicas	Indicates the number of times the FailedComputeMetricsReplicas event occurred in the last measurement period.	Number	This event is triggered if the Horizontal Pod Autoscaler is unable to compute the replica count. This can happen if the controller is unable to connect to the custom/resource metrics API, for any reason. Because of this, the controller will not be able to compute the resource utilization value. Without the resource utilization, the controller will not be able to compute the replica count. Use the detailed diagnosis of this measure to know when this event occurred, why, and which nodes and Pods were impacted by the event.
Failed rescale	Indicates the number of times the FailedRescale event occurred in the last measurement period.	Number	A scale update was needed and the HPA controller was unable to actually update the scale subresource of the target scalable, then this event is fired. Use the detailed diagnosis of this

Measurement	Description	Measurement Unit	Interpretation
			measure to know when this event occurred, why, and which nodes and Pods were impacted by the event.
Failed to update status	Indicates the number of times the FailedUpdateStatus event occurred in the last measurement period.	Number	The event is triggered if the HPA controller fails to update the status of the HPA object. Use the detailed diagnosis of this measure to know when this event occurred, why, and which nodes and Pods were impacted by the event.
No persistent volumes available	Indicates the number of times the FailedBinding event occurred in the last measurement period.	Number	To associate a Pod with storage, a cluster administrator should first create a PersistentVolume (PV) that is backed by physical storage. A cluster user should then create a PersistentVolumeClaim (PVC), which gets automatically bound to a PV. Finally, the user creates a Pod that uses the PVC as storage. If a PVC is created, but no PersistentVolumes are available for the PVC to be bound to, then the FailedBinding event gets fired. In such cases, Pods that use unbound PVCs will stay in the Pending state, until the problem is resolved. Use the detailed diagnosis of this measure to know when this event occurred, why, and which nodes and Pods were impacted by the event.
Volume size or class is different	Indicates the number of times the VolumeMismatch event occurred in the last measurement period.	Number	This event is triggered if the volume size or class is different from what is requested in the claim. Typically, a user creates a PersistentVolumeClaim (PVC) with

Measurement	Description	Measurement Unit	Interpretation
			a specific amount of storage requested and with certain access modes. A control loop in the master watches for new PVCs, checks if any static PV (a PV manually created by the administrator) exactly matches the new PVC, and binds them together. Claims will remain unbound indefinitely if a matching volume does not exist. Claims will be bound as matching volumes become available. For example, a cluster provisioned with many 50Gi PVs would not match a PVC requesting 100Gi. The PVC can be bound when a 100Gi PV is added to the cluster. Until a 100Gi PV becomes available, the cluster will not bind the PVC with any of the existing PVs; instead, it will fail binding with the event VolumeMismatch. Use the detailed diagnosis of this measure to know when this event occurred, why, and which nodes and Pods were impacted by the event.
Error creating recycler pods	Indicates the number of times the VolumeFailedRecycle event occurred in the last measurement period.	Number	An administrator can configure a custom recycler Pod template using the Kubernetes controller manager command line arguments. The custom recycler Pod template must contain a *volumes* specification. You need to configure the path of the volume to be recycled in the *path* specification of the *volumes* section. Typically, when a Pod is deleted and the PV has to be freed up, the

Measurement	Description	Measurement Unit	Interpretation
			recycler Pod comes in and tries to make the PV available. But, sometimes, due to certain errors, the recycler POD may switch to the 'failed' state. For instance, if the recycler Pod fails to remove the .snapshot folder, the Pod will fail to be created. As a result, the PV too fails to become available - i.e., volume recycling fails. n such a situation, the VolumeFailedRecycle event is fired. Use the detailed diagnosis of this measure to know when this event occurred, why, and which nodes and Pods were impacted by the event.
Volume is recycled	Indicates the number of times the VolumeRecycled event occurred in the last measurement period.	Number	This event is triggered every time a volume is successfully recycled. Use the detailed diagnosis of this measure to know when this event occurred, why, and which nodes and Pods were impacted by the event.
Pod is recylced	Indicates the number of times the RecyclerPod event occurred in the last measurement period.	Number	This event is triggered every time a recycler pod is successfully recycled. Use the detailed diagnosis of this measure to know when this event occurred, why, and which nodes and Pods were impacted by the event.
Volume is deleted	Indicates the number of times the VolumeDelete event occurred in the last measurement period.	Number	Every time a volume is deleted, the VolumeDelete event is triggered. Use the detailed diagnosis of this measure to know when this event occurred, why, and which nodes and Pods were impacted by the event.
Error when	Indicates the number of times the	Number	This event is triggered if volume

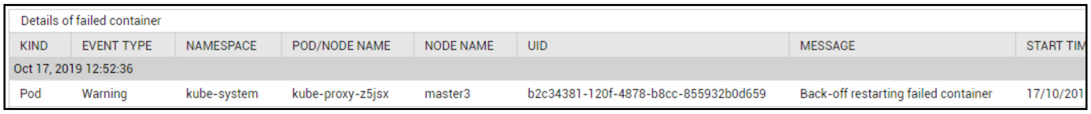
Measurement	Description	Measurement Unit	Interpretation
deleting the volume	VolumeFailedDelete event occurred in the last measurement period.		deletion fails. This can happen if the path specification in your PV does not match with the actual path of the volume being deleted. Use the detailed diagnosis of this measure to know when this event occurred, why, and which nodes and Pods were impacted by the event.
Error cleaning provisioned volume	Indicates the number of times the ProvisioningCleanupFailed event occurred in the last measurement period.	Number	This event is triggered if a provisioned volume is not automatically cleaned up, when the Pod mounting that volume is removed. In this case, you will have to manually delete the volume. Use the detailed diagnosis of this measure to know when this event occurred, why, and which nodes and Pods were impacted by the event.
Error creating load balancer	Indicates the number of times the CreatingLoadBalancerFailed event occurred in the last measurement period.	Number	If the static IP address defined in the loadBalancerIP property of the Kubernetes service manifest does not exist, or has not been created in the node resource group and no additional delegations are configured, the load balancer service creation fails with the event CreatingLoadBalancerFailed. Many load balancer issues around creating, updating, and deleting the load balancer can also be traced to a permissions issue with your cloud provider. Ensure that your Kubernetes nodes have the ability to create and modify load balancers in your cloud provider to avoid these issues. If your cloud provider

Measurement	Description	Measurement Unit	Interpretation
			provides Identity & Access Management (IAM) double-check the permissions that your nodes and pods have. Use the detailed diagnosis of this measure to know when this event occurred, why, and which nodes and Pods were impacted by the event.
Deleting load balancer	Indicates the number of times the DeletingLoadBalancer event occurred in the last measurement period.	Number	Use the detailed diagnosis of this measure to know when this event occurred, why, and which nodes and Pods were impacted by the event.
Not available nodes for Load Balancer service	Indicates the number of times the UnAvailableLoadBalancer event occurred in the last measurement period.	Number	Load balancers require at least one server to send traffic to for the load balancing. This can be an issue if the service is not able to target any pods, or if the load balancer is unable to health check any servers in your cluster. In such a situation, the UnAvailableLoadBalancer event gets fired. To troubleshoot this issue, check the endpoints registered with the service using *kubectl describe service <service>*, figure out which nodes those pods run on, and compare it to the servers registered to the load balancer in your cloud provider. Use the detailed diagnosis of this measure to know when this event occurred, why, and which nodes and Pods were impacted by the event.
Error updating load balancer with new hosts	Indicates the number of times the LoadBalancerUpdateFailed event occurred in the last measurement period.	Number	Many load balancer issues around creating, updating, and deleting the load balancer can be traced to a permissions issue with your cloud provider. Ensure that your

Measurement	Description	Measurement Unit	Interpretation
			Kubernetes nodes have the ability to create and modify load balancers in your cloud provider to avoid these issues. If your cloud provider provides Identity & Access Management (IAM) double-check the permissions that your nodes and pods have. Use the detailed diagnosis of this measure to know when this event occurred, why, and which nodes and Pods were impacted by the event.
Error deleting load balancer	Indicates the number of times the DeletingLoadBalancerFailed event occurred in the last measurement period.	Number	Many load balancer issues around creating, updating, and deleting the load balancer can be traced to a permissions issue with your cloud provider. Ensure that your Kubernetes nodes have the ability to create and modify load balancers in your cloud provider to avoid these issues. If your cloud provider provides Identity & Access Management (IAM) double-check the permissions that your nodes and pods have. Use the detailed diagnosis of this measure to know when this event occurred, why, and which nodes and Pods were impacted by the event.
Deleted load balancer	Indicates the number of times the DeletedLoadBalancer event occurred in the last measurement period.	Number	This event occurs if a load balancer is deleted. Use the detailed diagnosis of this measure to know when this event occurred, why, and which nodes and Pods were impacted by the event.
System out of memory	Indicates the number of times the SystemOOM event occurred in the	Number	This event is triggered if a node runs out of memory. Such an event can

Measurement	Description	Measurement Unit	Interpretation
	last measurement period.		happen if the kubelet is unable to reclaim memory by proactively failing one or more Pods on the node. Use the detailed diagnosis of this measure to know when this event occurred, why, and which nodes and Pods were impacted by the event.
Evicted pods	Indicates the number of times the Evicted event occurred in the last measurement period.	Number	This event happens when a node determines that Pods need to be evicted, or terminated, to free up some resource such as CPU, memory, or disk. This can have devastating consequences if the kubelet is unable to determine the best resources to evict. For instance, if a kubelet detecting disk pressure on a node evicted Pods that have no effect on disk usage, then such an eviction will not ease the disk space crunch on that node. Moreover, since the evicted Pods would get scheduled on other nodes, they will also overload the other nodes, thus causing more evictions. Knowing when evictions happened, and being able to correlate it with other events in that time frame, can help avoid the issue. Use the detailed diagnosis of this measure to know when this event occurred, why, and which nodes and Pods were impacted by the event.

You can use the detailed diagnosis of the *Back-off restarting failed containers* measure to know when the BackOff event occurred, the message that was displayed when the event occurred, and the nodes and Pods impacted by the event.

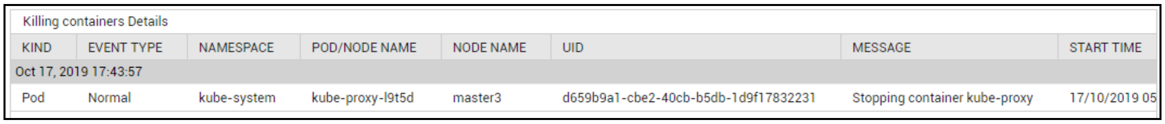


Details of failed container

KIND	EVENT TYPE	NAMESPACE	POD/NODE NAME	NODE NAME	UID	MESSAGE	START TIM
Oct 17, 2019 12:52:36							
Pod	Warning	kube-system	kube-proxy-z5jsx	master3	b2c34381-120f-4878-b8cc-855932b0d659	Back-off restarting failed container	17/10/201

Figure 3.3: The detailed diagnosis of the Back-off restarting failed containers measure

You can use the detailed diagnosis of the *Killing containers* measure to know when the Killing event occurred, the message that was displayed when the event occurred, and the nodes and Pods impacted by the event.

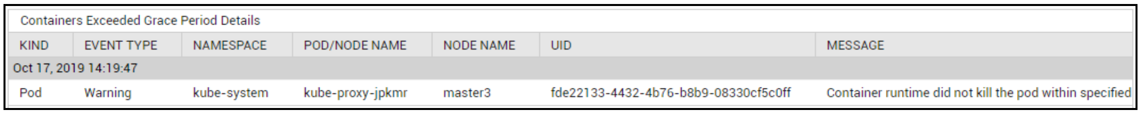


Killing containers Details

KIND	EVENT TYPE	NAMESPACE	POD/NODE NAME	NODE NAME	UID	MESSAGE	START TIME
Oct 17, 2019 17:43:57							
Pod	Normal	kube-system	kube-proxy-l9t5d	master3	d659b9a1-cbe2-40cb-b5db-1d9f17832231	Stopping container kube-proxy	17/10/2019 05

Figure 3.4: The detailed diagnosis of the Killing containers measure

Using the detailed diagnosis of the *Containers exceeded grace period* measure, you can quickly determine when the ExceededGracePeriod event occurred, why the event was triggered, and which nodes and Pods were impacted by the event.

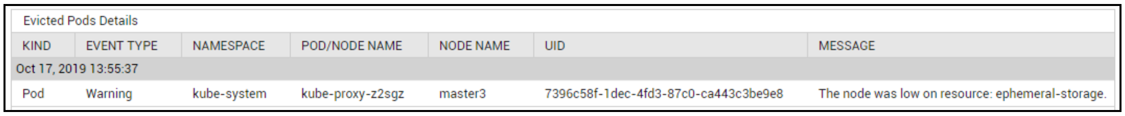


Containers Exceeded Grace Period Details

KIND	EVENT TYPE	NAMESPACE	POD/NODE NAME	NODE NAME	UID	MESSAGE
Oct 17, 2019 14:19:47						
Pod	Warning	kube-system	kube-proxy-jpkmr	master3	fde22133-4432-4b76-b8b9-08330cf5c0ff	Container runtime did not kill the pod within specified

Figure 3.5: The detailed diagnosis of the Containers exceeded grace period measure

Using the detailed diagnosis of the *Evicted pods* measure, you can quickly determine when the Evicted event occurred, why the event was triggered, and which nodes and Pods were impacted by the event.

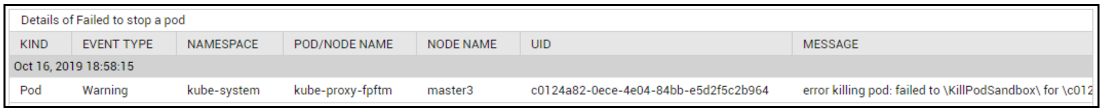


Evicted Pods Details

KIND	EVENT TYPE	NAMESPACE	POD/NODE NAME	NODE NAME	UID	MESSAGE
Oct 17, 2019 13:55:37						
Pod	Warning	kube-system	kube-proxy-z2sgz	master3	7396c58f-1dec-4fd3-87c0-ca443c3be9e8	The node was low on resource: ephemeral-storage.

Figure 3.6: The detailed diagnosis of the Evicted pods measure

With the help of the detailed diagnosis of the *Failed to stop pods* measure, you can at-a-glance figure out when the FailedKillPod event occurred, and which nodes and Pods were impacted by that event. You can also view the error message that Kubernetes throws when firing this event, so you can troubleshoot easily.

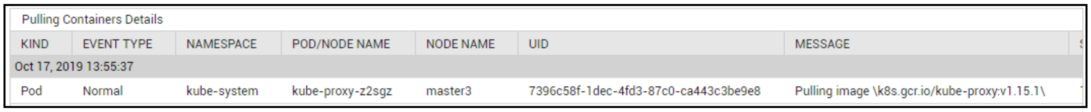


Details of Failed to stop a pod

KIND	EVENT TYPE	NAMESPACE	POD/NODE NAME	NODE NAME	UID	MESSAGE
Oct 16, 2019 18:58:15						
Pod	Warning	kube-system	kube-proxy-fpftm	master3	c0124a82-0ece-4e04-84bb-e5d2f5c2b964	error killing pod: failed to \KillPodSandbox\ for \c012

Figure 3.7: The detailed diagnosis of the Failed to stop pods measure

Use the detailed diagnosis of the *Pulling images* measures to know when the Pulling event occurred, which image was being pulled, and which nodes and Pods were impacted by the event.

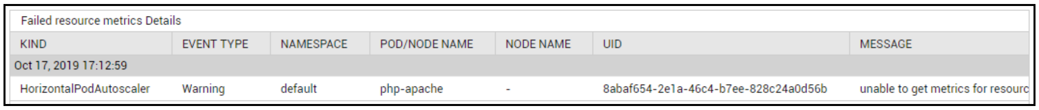


Pulling Containers Details

KIND	EVENT TYPE	NAMESPACE	POD/NODE NAME	NODE NAME	UID	MESSAGE
Oct 17, 2019 13:55:37						
Pod	Normal	kube-system	kube-proxy-z2sgz	master3	7396c58f-1dec-4fd3-87c0-ca443c3be9e8	Pulling image \k8s.gcr.io/kube-proxy:v1.15.1\

Figure 3.8: The detailed diagnosis of the Pulling images measure

Use the detailed diagnosis of the *Failed resource* measure to determine when the FailedGetResourceMetric event occurred, what could have caused the event to be triggered, and which Pods were impacted by the event.

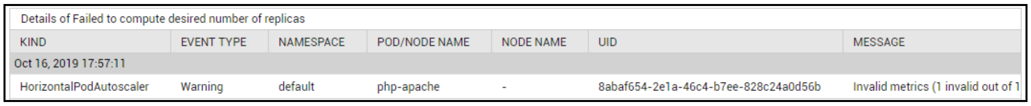


Failed resource metrics Details

KIND	EVENT TYPE	NAMESPACE	POD/NODE NAME	NODE NAME	UID	MESSAGE
Oct 17, 2019 17:12:59						
HorizontalPodAutoscaler	Warning	default	php-apache	-	8abaf654-2e1a-46c4-b7ee-828c24a0d56b	unable to get metrics for resourc

Figure 3.9: The detailed diagnosis of the Failed resource metric measure

Using the detailed diagnosis of the *Failed to compute desired number of replicas* measure to know when the FailedComputeMetricsReplicas event occurred, why, and which Pods were impacted.

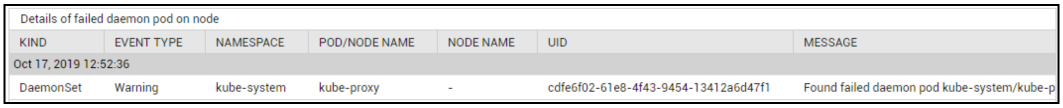


Details of Failed to compute desired number of replicas

KIND	EVENT TYPE	NAMESPACE	POD/NODE NAME	NODE NAME	UID	MESSAGE
Oct 16, 2019 17:57:11						
HorizontalPodAutoscaler	Warning	default	php-apache	-	8abaf654-2e1a-46c4-b7ee-828c24a0d56b	Invalid metrics (1 invalid out of 1

Figure 3.10: The detailed diagnosis of the Failed to compute desired number of replicas measure

By viewing the detailed diagnosis of the *Found failed daemon pods on node* measure, you will be able to ascertain when the FailedDaemonPod event occurred and which Pod was impacted by the event. The detailed diagnosis also reveals the error message that the event throws, so you can troubleshoot easily.



Details of failed daemon pod on node

KIND	EVENT TYPE	NAMESPACE	POD/NODE NAME	NODE NAME	UID	MESSAGE
Oct 17, 2019 12:52:36						
DaemonSet	Warning	kube-system	kube-proxy	-	cdfe6f02-61e8-4f43-9454-13412a6d47f1	Found failed daemon pod kube-system/kube-p

Figure 3.11: The detailed diagnosis of the Found failed daemon pods on node measure

3.1.2 API Server Connectivity Test

The API server is the component on the master that exposes the Kubernetes API. It is the front-end for the Kubernetes control plane. All communication paths from the cluster to the master terminate at the API server. The cluster receives Object specs from users via the API server on the master node. While the master uses the API server to forward the Object specs to the scheduler and to the kubelets, the kubelets also update the master with the Object status via the API server. Instructions for creating, starting, destroying objects on a worker node are also sent to worker nodes via the API server only.

This implies that the non- availability of the API server can bring the entire cluster to a standstill! Administrators may no longer be able to stop, update, or start new pods, services, or the replication controller. Moreover, users will be denied access to the cluster, and consequently, to the business-critical applications/services running within! To avoid this, administrators must periodically check if the API server is available and promptly detect its unavailability. This is exactly what the API Server Connectivity test does!

At configured intervals, this test checks whether/not the API server is available, and instantly alerts administrators if it is not. This way, the test urges administrators to investigate the reason for the non-availability and fix it, so that cluster operations resume quickly.

Target of the test : A Kubernetes Cluster

Agent deploying the test : A remote agent

Outputs of the test : One set of results for the Kubernetes cluster being monitored

Configurable parameters for the test

Parameter	Description
Test Period	How often should the test be executed.
Host	The IP address of the host for which this test is to be configured.
Port	Specify the port at which the specified Host listens. By default, this is *6443*.
Load Balancer / Master Node IP	To run this test and report metrics, the eG agent needs to connect to the Kubernetes API on the master node and run API commands. To enable this connection, the eG agent has to be configured with either of the following: • If only a single master node exists in the cluster, then configure the eG agent with the IP address of the master node.

Parameter	Description
	• If the target cluster consists of more than one master node, then you need to configure the eG agent with the IP address of the load balancer that is managing the cluster. In this case, the load balancer will route the eG agent's connection request to any available master node in the cluster, thus enabling the agent to connect with the API server on that node, run API commands on it, and pull metrics. By default, this parameter will display the**Load Balancer / Master Node IP** that you configured when manually adding the Kubernetes cluster for monitoring, using the **Kubernetes Cluster Preferences** page in the eG admin interface (see Figure 2.3). The steps for managing the cluster using the eG admin interface are discussed elaborately in How to Monitor the Kubernetes Cluster Using eG Enterprise? Whenever the eG agent runs this test, it uses the IP address that is displayed (by default) against this parameter to connect to the Kubernetes API. If there is any change in this IP address at a later point in time, then make sure that you update this parameter with it, by overriding its default setting.
SSL	By default, the Kubernetes cluster is SSL-enabled. This is why, the eG agent, by default, connects to the Kubernetes API via an HTTPS connection. Accordingly, this flag is set to **Yes** by default. If the cluster is not SSL-enabled in your environment, then set this flag to **No**.
Authentication Token	The eG agent requires an authentication bearer token to access the Kubernetes API, run API commands on the cluster, and pull metrics of interest. The steps for generating this token have been detailed in Section **1.1** Typically, once you generate the token, you can associate that token with the target Kubernetes cluster, when manually adding that cluster for monitoring using the eG admin interface. The steps for managing the cluster using the eG admin interface are discussed elaborately in Section **Chapter 2** By default, this parameter will display the **Authentication Token** that you provided in the **Kubernetes Cluster Preferences page** of the eG admin interface, when manually adding the cluster for monitoring (see Figure 2.3). Whenever the eG agent runs this test, it uses the token that is displayed (by default) against this parameter for accessing the API and pulling metrics. If for any reason, you generate a new authentication token for the target cluster at a later point in time, then make sure you update this parameter with the change. For that, copy the new token and paste it against this parameter.
Proxy Host	If the eG agent connects to the Kubernetes API on the master node via a proxy server, then provide the IP address of the proxy server here. If no proxy is used, then the

Parameter	Description
	default setting -*none* - of this parameter, need not be changed,
Proxy Port	If the eG agent connects to the Kubernetes API on the master node via a proxy server, then provide the port number at which that proxy server listens here. If no proxy is used, then the default setting -*none* - of this parameter, need not be changed,
Proxy Username, Proxy Password, Confirm Password	**These parameters are applicable only if the eG agent uses a proxy server to connect to the Kubernetes cluster, and that proxy server requires authentication.** In this case, provide a valid user name and password against the **Proxy Username** and **Proxy Password** parameters, respectively. Then, confirm the password by retyping it in the **Confirm Password** text box. If no proxy server is used, or if the proxy server used does not require authentication, then the default setting - *none* - of these parameters, need not be changed.
DD Frequency	Refers to the frequency with which detailed diagnosis measures are to be generated for this test. The default is *1:1*. This indicates that, by default, detailed measures will be generated every time this test runs, and also every time the test detects a problem. You can modify this frequency, if you so desire. Also, if you intend to disable the detailed diagnosis capability for this test, you can do so by specifying *none* against DD frequency.
Detailed Diagnosis	To make diagnosis more efficient and accurate, the eG Enterprise suite embeds an optional detailed diagnostic capability. With this capability, the eG agents can be configured to run detailed, more elaborate tests as and when specific problems are detected. To enable the detailed diagnosis capability of this test for a particular server, choose the **On** option. To disable the capability, click on the **Off** option. The option to selectively enable/disable the detailed diagnosis capability will be available only if the following conditions are fulfilled: • The eG manager license should allow the detailed diagnosis capability • Both the normal and abnormal frequencies configured for the detailed diagnosis measures should not be 0.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Availability	Indicates whether/not the API server is available.	Percent	If the value of this measure is 0, it indicates that the API server is unavailable. The value 100 on the other

Measurement	Description	Measurement Unit	Interpretation
			hand indicates that the API server is available. In the event of the non-availability of the API server, you can use the detailed diagnosis of this measure to figure out the reason for the non-available. You can also use the /var/log/kube-apiserver.log file on the master node to figure out what could have caused the failure of the server. The common causes for the unavailability of the API server are as follows: • API server VM shutting down or crashing; • API server losing access to its backing storage To ensure the high availability of the API server, you may want to consider the following courses of action: • Use IaaS provider's automatic VM restarting feature for IaaS VMs; • Use IaaS providers reliable storage (e.g. GCE PD or AWS EBS volume) for VMs with apiserver+etcd

3.1.3 Kube Garbage Collection Test

The Kubernetes project is written in the Go programming language (also known as Golang). Go is a statically typed, compiled programming language designed at Google. Go is syntactically similar to C, but with memory safety, garbage collection, structural typing, and communicating sequential processes (CSP)-style concurrency.

Garbage collectors have the responsibility of tracking heap memory allocations, freeing up allocations that are no longer needed, and keeping allocations that are still in- use. The Go programming language uses a non-generational concurrent tri-color mark and sweep collector.

When a collection starts, the collector runs through four phases of work:

- Mark Setup
- Marking
- Mark Termination
- Sweeping

The **Mark Setup** phase is where the Write Barrier is turned on. The purpose of the Write Barrier is to allow the collector to maintain data integrity on the heap during a collection since both the collector and application goroutines will be running concurrently. In order to turn the Write Barrier on, every application goroutine running must be stopped. The only way to do that is for the collector to watch and wait for each goroutine to make a function call. Function calls guarantee the goroutines are at a safe point to be stopped.

Once the Write Barrier is turned on, the collector commences with the **Marking** phase. The Marking phase consists of marking values in heap memory that are still in-use. This work starts by inspecting the stacks for all existing goroutines to find root pointers to heap memory. Then the collector must traverse the heap memory graph from those root pointers. The first thing the collector does in this phase is take 25% of the available CPU capacity for itself. For example, if an application uses 4 CPUs, then the collector will hog an entire CPU while at this phase. In this case typically, the collector will use the 25% CPU capacity that it has set aside for this phase, to do the marking work, allowing normal application work to continue on the remaining 75%.

Once the Marking work is done, the next phase is **Mark Termination**. This is when the Write Barrier is turned off, various clean up tasks are performed, and the next collection goal is calculated.

Once the collection is finished, the full CPU capacity is released for the use of the application Goroutines again, thus bringing the application back to full throttle.

Sweeping typically happens after the collection is finished. Sweeping is when the memory associated with values in heap memory that were not marked as in-use are reclaimed. This activity occurs when application Goroutines attempt to allocate new values in heap memory.

In summary, by performing garbage collection, Golang ensures that applications make optimal use of available heap memory. While this improves application performance at one end, at the other, every collection also inflicts certain latencies on the running application that may slow down application work. For instance, at the Mark Setup phase, the garbage collector stops all application Goroutines, so it can turn on the Write Barrier. This imposes a Stop the World (STW) latency on the running application. Likewise, the application Goroutines are stopped at the Mark Termination phase as well, once again inflicting an STW latency on the applications. Also, sometimes, garbage collection steals CPU capacity to stay alive, and degrades application performance in the bargain. For instance, in the Marking phase, if the Goroutine dedicated to the collector is unable to finish the marking work before the heap memory in- use reaches its limit, the collector will recruit the application Goroutines to assist with the Marking work. This is called a Mark Assist. When this happens, the application will be forced to compete with the collector for the available CPU resources. This contention can occasionally choke application performance!

To optimize garbage collection and eliminate its ill effects, administrators must ensure that the collector does more work, while consuming minimum time and resources. For this purpose, administrators must first study the garbage collection activity closely, and figure out how much time and resources the collector typically invests in this process. This is where, the Kube Garbage Collection test helps!

This test monitors the garbage collection activity of Golang, and reports the time the Golang collector spends collecting garbage. Administrators will be alerted if too much time is being spent in garbage collection. The test also reveals the number of threads and Goroutines presently engaged in garbage collection, thus revealing how resource-intensive the garbage collection is. This way, the test enables administrators to periodically review the garbage collection activity, assess its impact on application performance, and figure out if it needs to be fine-tuned to reduce application latencies.

Target of the test : A Kubernetes Cluster

Agent deploying the test : A remote agent

Outputs of the test : One set of results for the Kubernetes cluster being monitored

Configurable parameters for the test

Parameter	Description
Test Period	How often should the test be executed.
Host	The IP address of the host for which this test is to be configured.
Port	Specify the port at which the specified Host listens. By default, this is *6443*.
Load Balancer / Master Node IP	To run this test and report metrics, the eG agent needs to connect to the Kubernetes API on the master node and run API commands. To enable this connection, the eG agent has to be configured with either of the following:

Parameter	Description
	- If only a single master node exists in the cluster, then configure the eG agent with the IP address of the master node. - If the target cluster consists of more than one master node, then you need to configure the eG agent with the IP address of the load balancer that is managing the cluster. In this case, the load balancer will route the eG agent's connection request to any available master node in the cluster, thus enabling the agent to connect with the API server on that node, run API commands on it, and pull metrics. By default, this parameter will display the **Load Balancer / Master Node IP** that you configured when manually adding the Kubernetes cluster for monitoring, using the **Kubernetes Cluster Preferences** page in the eG admin interface (see Figure 2.3). The steps for managing the cluster using the eG admin interface are discussed elaborately in How to Monitor the Kubernetes Cluster Using eG Enterprise? Whenever the eG agent runs this test, it uses the IP address that is displayed (by default) against this parameter to connect to the Kubernetes API. If there is any change in this IP address at a later point in time, then make sure that you update this parameter with it, by overriding its default setting.
SSL	By default, the Kubernetes cluster is SSL-enabled. This is why, the eG agent, by default, connects to the Kubernetes API via an HTTPS connection. Accordingly, this flag is set to **Yes** by default. If the cluster is not SSL-enabled in your environment, then set this flag to **No**.
Authentication Token	The eG agent requires an authentication bearer token to access the Kubernetes API, run API commands on the cluster, and pull metrics of interest. The steps for generating this token have been detailed in Section **1.1** Typically, once you generate the token, you can associate that token with the target Kubernetes cluster, when manually adding that cluster for monitoring using the eG admin interface. The steps for managing the cluster using the eG admin interface are discussed elaborately in Section **Chapter 2** By default, this parameter will display the **Authentication Token** that you provided in the **Kubernetes Cluster Preferences page** of the eG admin interface, when manually adding the cluster for monitoring (see Figure 2.3). Whenever the eG agent runs this test, it uses the token that is displayed (by default) against this parameter only accessing the API and pulling metrics. If for any reason, you generate a new authentication token for the target cluster at a later point in time, then make sure you update this parameter with the change. For that, copy the new

Parameter	Description
	token and paste it against this parameter.
Proxy Host	If the eG agent connects to the Kubernetes API on the master node via a proxy server, then provide the IP address of the proxy server here. If no proxy is used, then the default setting -*none* - of this parameter, need not be changed,
Proxy Port	If the eG agent connects to the Kubernetes API on the master node via a proxy server, then provide the port number at which that proxy server listens here. If no proxy is used, then the default setting -*none* - of this parameter, need not be changed,
Proxy Username, Proxy Password, Confirm Password	**These parameters are applicable only if the eG agent uses a proxy server to connect to the Kubernetes cluster, and that proxy server requires authentication.** In this case, provide a valid user name and password against the **Proxy Username** and **Proxy Password** parameters, respectively. Then, confirm the password by retyping it in the **Confirm Password** text box. If no proxy server is used, or if the proxy server used does not require authentication, then the default setting - *none* - of these parameters, need not be changed.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Avg GC invocation duration	Indicates the average time spent in garbage collection.	Seconds	A low value is desired for this measure. A very high value or a consistent increase in the value of this measure is a cause for concern, as it indicates that the garbage collector is probably taking too long to complete collections. Since garbage collection often triggers stop-the-world latencies in applications, prolonged garbage collection activities can adversely impact application availability and performance. In short, the longer GC runs, poorer will be application performance. One way to reduce GC time, is to fine-

Measurement	Description	Measurement Unit	Interpretation
			tune the configuration option called GC Percentage at runtime. This is set to 100 by default. This value represents a ratio of how much new heap memory can be allocated before the next collection has to start. Setting the GC Percentage to 100 means, based on the amount of heap memory marked as live after a collection finishes, the next collection has to start at or before 100% more new allocations are added to heap memory. You could decide to change the GC Percentage value to something larger than 100. This will increase the amount of heap memory that has to be allocated before the next collection can start, thus delaying the start of the next collection. On the flip side though, increasing the GC percentage will slow down the pace of the collector. The collector has a pacing algorithm which is used to determine when a collection is to start. The algorithm depends on a feedback loop that the collector uses to gather information about the running application and the stress the application is putting on the heap. Stress can be defined as how fast the application is allocating heap memory within a given amount of time. It’s that stress that determines the pace at which the collector needs to run. One misconception is thinking that slowing down the pace of the collector is a way to improve performance. In

Measurement	Description	Measurement Unit	Interpretation
			reality though, application performance truly improves only when more work is getting done between collections or during a collection. This can be achieved only by reducing the amount or the number of allocations any piece of work is adding to heap memory. Increasing the GC percentage in fact, increases the workload of collections by adding more to the heap memory after every collection. In the long run, this may degrade application performance than improve it.
OS threads created	Indicates the number of threads spawned by the garbage collection process.	Number	A large value for this measure is indicative of resource-intensive garbage collections.
Goroutines	Indicates the number of Goroutines used for garbage collection.	Number	An unusually high value for this measure could indicate that the garbage collector is probably recruiting application Goroutines as well to do the Marking work on the collections. This in turn could be because of of Marking workloads that the collector is unable to complete using just its dedicated Goroutines. Such workloads are usually imposed by applications that consume heap memory significantly.

3.1.4 Kube Master Services Test

Master components/services make global decisions about the cluster (for example, scheduling), and detect and respond to cluster events. These services are as follows:

- **kube-apiserver:** This exposes the Kubernetes API and front-ends the control pane.
- **kube-scheduler:** This watches newly created pods that have no node assigned, and selects a node for them to run on.
- **kube-controller-manager:** This runs processes called controllers. These controllers include:
  - *Node controller:* Responsible for noticing and responding when nodes go down.
  - *Replication controller:* Responsible for maintaining the correct number of pods for every replication controller object in the system.
  - *Endpoints controller:* Populates the Endpoints object (that is, joins Services & Pods).
  - *Service Account and Token controllers:* Creates default accounts and API access tokens for new namespaces
- **etcd:** Consistent and highly-available key value store used as Kubernetes' backing store for all cluster data.

The failure of any of these services can be business-impacting! For instance, if the kube-scheduler is not running, then pods will have no nodes to run on. Without the kube-controller-manager, cluster state cannot be managed. Such anomalies can threaten the availability of the cluster and deny users access to critical applications/services running on the cluster. To avoid this, administrators must keep track of the state of each of the master services. This is where, eG Enterprise helps!

Using the **API Server Connectivity** test, administrators can periodically check if the kube-api-server service is running or not. With the help of the **Kube Master Services** test, administrators can keep tabs on the running state of the other master services, namely - the scheduler, the etcd, and the controller-manager. If any of these services is down, then the **Kube Master Services** test promptly alerts administrators to the failure of the corresponding service. This way, the test enables administrators to rapidly troubleshoot the abnormal state of a critical master service, restore the service to normalcy, and assure users of uninterrupted access to containerized business applications.

Target of the test : A Kubernetes Cluster

Agent deploying the test : A remote agent

Outputs of the test : One set of results for the Kubernetes cluster being monitored

Configurable parameters for the test

Parameter	Description
Test Period	How often should the test be executed.
Host	The IP address of the host for which this test is to be configured.
Port	Specify the port at which the specified Host listens. By default, this is *6443*.
Load Balancer / Master Node IP	To run this test and report metrics, the eG agent needs to connect to the Kubernetes API on the master node and run API commands. To enable this connection, the eG agent has to be configured with either of the following: • If only a single master node exists in the cluster, then configure the eG agent with the IP address of the master node. • If the target cluster consists of more than one master node, then you need to configure the eG agent with the IP address of the load balancer that is managing the cluster. In this case, the load balancer will route the eG agent's connection request to any available master node in the cluster, thus enabling the agent to connect with the API server on that node, run API commands on it, and pull metrics. By default, this parameter will display the **Load Balancer / Master Node IP** that you configured when manually adding the Kubernetes cluster for monitoring, using the **Kubernetes Cluster Preferences** page in the eG admin interface (see Figure 2.3). The steps for managing the cluster using the eG admin interface are discussed elaborately in How to Monitor the Kubernetes Cluster Using eG Enterprise? Whenever the eG agent runs this test, it uses the IP address that is displayed (by default) against this parameter to connect to the Kubernetes API. If there is any change in this IP address at a later point in time, then make sure that you update this parameter with it, by overriding its default setting.
SSL	By default, the Kubernetes cluster is SSL-enabled. This is why, the eG agent, by default, connects to the Kubernetes API via an HTTPS connection. Accordingly, this flag is set to **Yes** by default. If the cluster is not SSL-enabled in your environment, then set this flag to **No**.
Authentication Token	The eG agent requires an authentication bearer token to access the Kubernetes API, run API commands on the cluster, and pull metrics of interest. The steps for generating this token have been detailed in Section **1.1** Typically, once you generate the token, you can associate that token with the target Kubernetes cluster, when manually adding that cluster for monitoring using the

Parameter	Description
	eG admin interface. The steps for managing the cluster using the eG admin interface are discussed elaborately in Section **Chapter 2** By default, this parameter will display the **Authentication Token** that you provided in the **Kubernetes Cluster Preferences page** of the eG admin interface, when manually adding the cluster for monitoring (see Figure 2.3). Whenever the eG agent runs this test, it uses the token that is displayed (by default) against this parameter for accessing the API and pulling metrics. If for any reason, you generate a new authentication token for the target cluster at a later point in time, then make sure you update this parameter with the change. For that, copy the new token and paste it against this parameter.
Proxy Host	If the eG agent connects to the Kubernetes API on the master node via a proxy server, then provide the IP address of the proxy server here. If no proxy is used, then the default setting -*none* - of this parameter, need not be changed,
Proxy Port	If the eG agent connects to the Kubernetes API on the master node via a proxy server, then provide the port number at which that proxy server listens here. If no proxy is used, then the default setting -*none* - of this parameter, need not be changed,
Proxy Username, Proxy Password, Confirm Password	**These parameters are applicable only if the eG agent uses a proxy server to connect to the Kubernetes cluster, and that proxy server requires authentication.** In this case, provide a valid user name and password against the **Proxy Username** and **Proxy Password** parameters, respectively. Then, confirm the password by retyping it in the **Confirm Password** text box. If no proxy server is used, or if the proxy server used does not require authentication, then the default setting - *none* - of these parameters, need not be changed.
DD Frequency	Refers to the frequency with which detailed diagnosis measures are to be generated for this test. The default is *1:1*. This indicates that, by default, detailed measures will be generated every time this test runs, and also every time the test detects a problem. You can modify this frequency, if you so desire. Also, if you intend to disable the detailed diagnosis capability for this test, you can do so by specifying *none* against DD frequency.
Detailed Diagnosis	To make diagnosis more efficient and accurate, the eG Enterprise suite embeds an optional detailed diagnostic capability. With this capability, the eG agents can be configured to run detailed, more elaborate tests as and when specific problems are detected. To enable the detailed diagnosis capability of this test for a particular server, choose the **On** option. To disable the capability, click on the **Off** option. The option to selectively enable/disable the detailed diagnosis capability will be

Parameter	Description
	available only if the following conditions are fulfilled: • The eG manager license should allow the detailed diagnosis capability • Both the normal and abnormal frequencies configured for the detailed diagnosis measures should not be 0.

Measurements made by the test

<table>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
<tr><td>State</td><td>Indicates the current state of this service.</td><td></td><td>The values that this measure can report and their corresponding numeric values are listed in the table below:
<table>
<tr><th>Measure Value</th><th>Numeric Value</th></tr>
<tr><td>Running</td><td>1</td></tr>
<tr><td>Not Running</td><td>0</td></tr>
<tr><td>Unknown</td><td>2</td></tr>
</table>
If this measure reports the value <i>Not Running</i> or <i>Unknown</i>, then use the detailed diagnosis of this measure to determine why. You can also use the /var/log/kube-scheduler.log file on the master to troubleshoot issues with the scheduler. Likewise, use the /var/log/kube-controller-manager.log file on the master to troubleshoot issues with the controller-manager.
<b>Note:</b>
By default, this measure reports the <b>Measure Value</b>s discussed above to indicate the state of a master service. In the graph of this measure however, the same is represented using the numeric equivalents only.</td></tr>
</table>

3.1.5 The Kube Cluster Layer

Using the test mapped to this layer, you can:

- Understand the composition and receive an overview of the health of the Kubernetes cluster;
- Track the status of each of the nodes in a Kubernetes cluster, monitor resource allocations to Pods and containers on each node, and identify overcommitted nodes;
- Determine whether/not the resource allocations to Pods/containers in each namespace align with resource quota or LimitRange settings;
- Identify Persistent Volumes that are unbound, and those that have failed automatic reclamation;

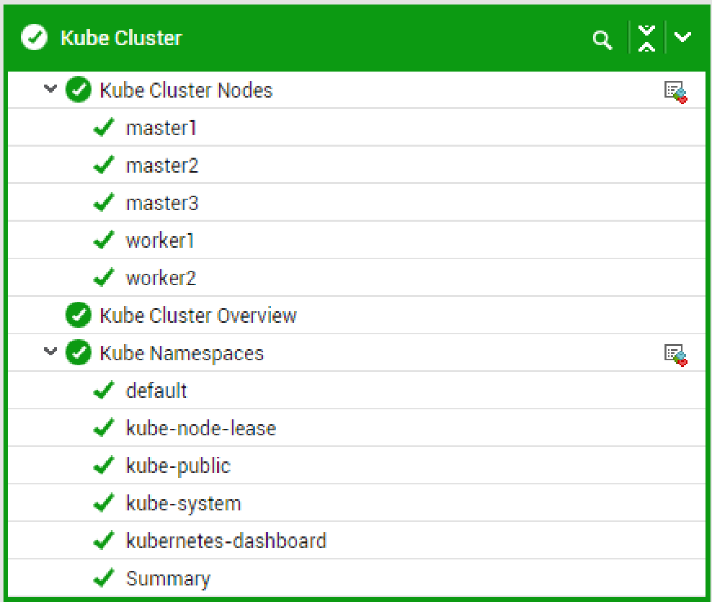


Figure 3.12: The tests mapped to the Kube Cluster layer

3.1.6 Kube Cluster Nodes Test

A node is a worker machine in Kubernetes. A node may be a VM or physical machine, depending on the cluster. Each node contains the services necessary to run pods and is managed by the master components. The services on a node include the container runtime, kubelet and kube-proxy.

A node's status contains information such as the addresses (hostname, external IP address, internal IP address of the node), conditions describing the status of running nodes, the total resource capacity of the node and the usable (allocatable) capacity, and general information pertaining to the node (eg., kernel version, Kubernetes version etc.).

Nodes are automatically managed by the Node controller. If a node is unreachable beyond a configured duration, then the node controller automatically deletes all the Pods on that node. However, sometimes, manual administration/management of nodes may become necessary. For instance, administrators may have to manually delete unreachable node objects, if the node controller is unable to do so. Likewise, if a node is to be rebooted, then the administrator will have to manually mark that node as "unschedulable", so that new Pods do not get scheduled to that node.

While the Node controller manages the node 'condition', the Kubernetes scheduler manages Pod placements by automatically comparing the resource requirement of the containers in the Pods with the total and allocatable resource capacity of the nodes, and scheduling Pods on those nodes that fit their resource profile. Sometimes, a node may run Pods that oversubscribe to the node's resources - i.e., the sum of limits of the containers on the node may exceed the total resource capacity of the node. In an overcommitted environment, it is possible that the Pods on the node will attempt to use more compute resource than is available at any given point in time. If this happens, it can degrade the performance of containerized applications, as you may have a single Pod hogging the node's resources! Administrators may hence want to be promptly alerted to a resource overcommitment, so they can quickly identify which Pod is guilty of overcommitment and determine how resource allocations and usage priorities can be tweaked to ensure performance does not suffer! Additionally, administrators may also want to track resource usage across containers on a node, so they can proactively isolate a potential resource contention and instantly initiate pre-emptive action. The Kube Cluster Nodes test does all this and more!

The test auto-discovers the nodes in a Kubernetes cluster and clearly distinguishes between the master nodes and the workers. The test then monitors the condition of each node and points administrators to those nodes whose condition is 'unhealthy' or have been marked as 'unschedulable'. Additionally, the test reports the total CPU and memory capacity of every node, tracks the sum of resource requests/limits of the containers on each node, and accurately pinpoints those nodes where containers have oversubscribed to the node's capacity. Detailed diagnostics of the test lead administrators to the exact Pods that have oversubscribed to the node's resources. With the help of this information, administrators may decide to resize containers or reset resource usage priorities of containers, so that cluster performance is not compromised. Furthermore, the test reveals the percentage of a node's resources that are being utilized by the containers, thereby warning administrators of a probable contention for resources on a node.

Target of the test : A Kubernetes Cluster

Agent deploying the test : A remote agent

Outputs of the test : One set of results for each node in the Kubernetes cluster being monitored

Configurable parameters for the test

Parameter	Description
Test Period	How often should the test be executed.
Host	The IP address of the host for which this test is to be configured.
Port	Specify the port at which the specified Host listens. By default, this is *6443*.
Load Balancer / Master Node IP	To run this test and report metrics, the eG agent needs to connect to the Kubernetes API on the master node and run API commands. To enable this connection, the eG agent has to be configured with either of the following: • If only a single master node exists in the cluster, then configure the eG agent with the IP address of the master node. • If the target cluster consists of more than one master node, then you need to configure the eG agent with the IP address of the load balancer that is managing the cluster. In this case, the load balancer will route the eG agent's connection request to any available master node in the cluster, thus enabling the agent to connect with the API server on that node, run API commands on it, and pull metrics. By default, this parameter will display the **Load Balancer / Master Node IP** that you configured when manually adding the Kubernetes cluster for monitoring, using the **Kubernetes Cluster Preferences** page in the eG admin interface (see Figure 2.3). The steps for managing the cluster using the eG admin interface are discussed elaborately in How to Monitor the Kubernetes Cluster Using eG Enterprise? Whenever the eG agent runs this test, it uses the IP address that is displayed (by default) against this parameter to connect to the Kubernetes API. If there is any change in this IP address at a later point in time, then make sure that you update this parameter with it, by overriding its default setting.
SSL	By default, the Kubernetes cluster is SSL-enabled. This is why, the eG agent, by default, connects to the Kubernetes API via an HTTPS connection. Accordingly, this flag is set to **Yes** by default. If the cluster is not SSL-enabled in your environment, then set this flag to **No**.
Authentication Token	The eG agent requires an authentication bearer token to access the Kubernetes API, run API commands on the cluster, and pull metrics of interest. The steps for generating this token have been detailed in Section **1.1** Typically, once you generate the token, you can associate that token with the target Kubernetes cluster, when manually adding that cluster for monitoring using the

Parameter	Description
	eG admin interface. The steps for managing the cluster using the eG admin interface are discussed elaborately in Section **Chapter 2** By default, this parameter will display the **Authentication Token** that you provided in the **Kubernetes Cluster Preferences page** of the eG admin interface, when manually adding the cluster for monitoring (see Figure 2.3). Whenever the eG agent runs this test, it uses the token that is displayed (by default) against this parameter for accessing the API and pulling metrics. If for any reason, you generate a new authentication token for the target cluster at a later point in time, then make sure you update this parameter with the change. For that, copy the new token and paste it against this parameter.
Proxy Host	If the eG agent connects to the Kubernetes API on the master node via a proxy server, then provide the IP address of the proxy server here. If no proxy is used, then the default setting -*none* - of this parameter, need not be changed,
Proxy Port	If the eG agent connects to the Kubernetes API on the master node via a proxy server, then provide the port number at which that proxy server listens here. If no proxy is used, then the default setting -*none* - of this parameter, need not be changed,
Proxy Username, Proxy Password, Confirm Password	**These parameters are applicable only if the eG agent uses a proxy server to connect to the Kubernetes cluster, and that proxy server requires authentication.** In this case, provide a valid user name and password against the **Proxy Username** and **Proxy Password** parameters, respectively. Then, confirm the password by retyping it in the **CONFIRM PASSWORD** text box. If no proxy server is used, or if the proxy server used does not require authentication, then the default setting - *none* - of these parameters, need not be changed.
DD Frequency	Refers to the frequency with which detailed diagnosis measures are to be generated for this test. The default is *3:1*. This indicates that, by default, detailed measures will be generated every third time this test runs, and also every time the test detects a problem. You can modify this frequency, if you so desire. Also, if you intend to disable the detailed diagnosis capability for this test, you can do so by specifying *none* against DD frequency.
Detailed Diagnosis	To make diagnosis more efficient and accurate, the eG Enterprise suite embeds an optional detailed diagnostic capability. With this capability, the eG agents can be configured to run detailed, more elaborate tests as and when specific problems are detected. To enable the detailed diagnosis capability of this test for a particular server, choose the **On** option. To disable the capability, click on the **Off** option. The option to selectively enable/disable the detailed diagnosis capability will be available only if the following conditions are fulfilled:

Parameter	Description
	• The eG manager license should allow the detailed diagnosis capability • Both the normal and abnormal frequencies configured for the detailed diagnosis measures should not be 0.

Measurements made by the test

<table>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
<tr><td>Status</td><td>Indicates whether/not this node is running.</td><td></td><td>The values that this measure reports and their corresponding numeric values are detailed in the table below:
<table>
<tr><th>Measure Value</th><th>Numeric Value</th></tr>
<tr><td>Running</td><td>1</td></tr>
<tr><td>Not running</td><td>0</td></tr>
<tr><td>Unknown</td><td>2</td></tr>
</table>
<b>Note:</b>
By default, this test reports the <b>Measure Value</b>s listed in the table above to indicate the state of a node. In the graph of this measure however, the state is indicated using the numeric equivalents only.
In the event that this measure reports the value <i>Not running</i> or <i>Unknown</i> for a node, then you can use the detailed diagnosis of this measure to know the reason for the abnormal status.</td></tr>
<tr><td>Is node unschedulable?</td><td>Indicates whether/not this node is unschedulable.</td><td></td><td>By default, healthy nodes with a <i>Ready</i> status are marked as schedulable, meaning that new pods are allowed for placement on the node. Manually marking a node as unschedulable blocks any new pods from being scheduled on the node. Typically, nodes from which Pods need to be migrated/evacuated are candidates for</td></tr>
</table>

<table>
<thead>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
</thead>
<tbody>
<tr><td></td><td></td><td></td><td>being marked as 'unschedulable' status. Sometimes, nodes that have been unhealthy for a long time are also set as 'unschedulable'.
The values that this measure reports and their corresponding numeric values are detailed in the table below:
<table>
<thead><tr><th>Measure Value</th><th>Numeric Value</th></tr></thead>
<tbody>
<tr><td>Yes</td><td>1</td></tr>
<tr><td>No</td><td>0</td></tr>
<tr><td>Unknown</td><td>2</td></tr>
</tbody>
</table>
<b>Note:</b>
By default, this test reports the <b>Measure Value</b>s listed in the table above to indicate whether/not a node has been manually set as unschedulable. In the graph of this measure however, the same is indicated using the numeric equivalents only.</td></tr>
<tr><td>Maintenance mode</td><td>Indicates whether/not this node is in the maintenance mode.</td><td></td><td>By putting a node into maintenance mode, all existing workloads will be restarted on other nodes to ensure availability, and no new workloads will be started on the node. Maintenance mode allows you to perform operations such as security updates or rebooting machines without the loss of availability.
The values that this measure reports and their corresponding numeric values are detailed in the table below:
<table>
<thead><tr><th>Measure Value</th><th>Numeric Value</th></tr></thead>
<tbody>
<tr><td>Enabled</td><td>1</td></tr>
<tr><td>Disabled</td><td>0</td></tr>
</tbody>
</table>
</td></tr>
</tbody>
</table>

<table>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
<tr><td></td><td></td><td></td><td><b>Note:</b>
By default, this test reports the <b>Measure Value</b>s listed in the table above to indicate whether/not a node is in the maintenance mode. In the graph of this measure however, the same is indicated using the numeric equivalents only.</td></tr>
<tr><td>Age</td><td>Indicates how old this node is.</td><td></td><td>The value of this measure is expressed in number of days, hours, and minutes.
Use the detailed diagnosis of this measure to know more about a particular node.</td></tr>
<tr><td>Is node network available?</td><td>Indicates whether/not the network of this node is correctly configured.</td><td></td><td>The values that this measure reports and their corresponding numeric values are detailed in the table below:
<table>
<tr><th>Measure Value</th><th>Numeric Value</th></tr>
<tr><td>Yes</td><td>1</td></tr>
<tr><td>No</td><td>0</td></tr>
<tr><td>Unknown</td><td>2</td></tr>
</table>
<b>Note:</b>
By default, this test reports the <b>Measure Value</b>s listed in the table above to indicate the availability of a node's network. In the graph of this measure however, the same is indicated using the numeric equivalents only.
If this measure reports the value <i>Yes</i> for a node - i.e., if the network of a node is indeed unavailable - then you can use the detailed diagnosis of this measure to figure out the reason for the unavailability.</td></tr>
<tr><td>Is node out of disk?</td><td>Indicates whether/not there is insufficient free disk space on this node</td><td></td><td>The values that this measure reports and their corresponding numeric values are detailed in the table below:</td></tr>
</table>

<table>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
<tr><td></td><td>for adding new Pods.</td><td></td><td>
<table>
<tr><th>Measure Value</th><th>Numeric Value</th></tr>
<tr><td>Yes</td><td>1</td></tr>
<tr><td>No</td><td>0</td></tr>
<tr><td>Unknown</td><td>2</td></tr>
</table>
<b>Note:</b>
By default, this test reports the <b>Measure Value</b>s listed in the table above to indicate whether/not a node has run out of disk space. In the graph of this measure however, the same is indicated using the numeric equivalents only.
If this measure reports the value <i>Yes</i> for a node - i.e., if a node has indeed run out of free disk space - then you can use the detailed diagnosis of this measure to figure out the reason for the anomaly.
</td></tr>
<tr><td>Does node have memory pressure?</td><td>Indicates whether/not this node is running low on memory.</td><td></td><td>
The values that this measure reports and their corresponding numeric values are detailed in the table below:
<table>
<tr><th>Measure Value</th><th>Numeric Value</th></tr>
<tr><td>Yes</td><td>1</td></tr>
<tr><td>No</td><td>0</td></tr>
<tr><td>Unknown</td><td>2</td></tr>
</table>
<b>Note:</b>
By default, this test reports the <b>Measure Value</b>s listed in the table above to indicate whether/not a node has sufficient memory. In the graph of this measure however, the same is indicated using the numeric equivalents only.
If this measure reports the value <i>Yes</i> for a node - i.e., if a node is running out of
</td></tr>
</table>

<table>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
<tr><td></td><td></td><td></td><td>memory - then you can use the detailed diagnosis of this measure to figure out the reason for the anomaly.</td></tr>
<tr><td>Does node have disk pressure?</td><td>Indicates whether/not this node's disk capacity is low.</td><td></td><td>The values that this measure reports and their corresponding numeric values are detailed in the table below:
<table>
<tr><th>Measure Value</th><th>Numeric Value</th></tr>
<tr><td>Yes</td><td>1</td></tr>
<tr><td>No</td><td>0</td></tr>
<tr><td>Unknown</td><td>2</td></tr>
</table>
<b>Note:</b>
By default, this test reports the <b>Measure Value</b>s listed in the table above to indicate whether/not a node is low on disk capacity. In the graph of this measure however, the same is indicated using the numeric equivalents only.
If this measure reports the value <i>Yes</i> for a node - i.e., if a node is low on disk capacity - then you can use the detailed diagnosis of this measure to figure out the reason for the anomaly.</td></tr>
<tr><td>Is node under PID pressure?</td><td>Indicates whether/not too many processes are running on the node.</td><td></td><td>The values that this measure reports and their corresponding numeric values are detailed in the table below:
<table>
<tr><th>Measure Value</th><th>Numeric Value</th></tr>
<tr><td>Yes</td><td>1</td></tr>
<tr><td>No</td><td>0</td></tr>
<tr><td>Unknown</td><td>2</td></tr>
</table>
<b>Note:</b>
By default, this test reports the <b>Measure</b></td></tr>
</table>

<table>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
<tr><td></td><td></td><td></td><td><p><b>Value</b>s listed in the table above to indicate whether/not a node is under PID pressure. In the graph of this measure however, the same is indicated using the numeric equivalents only.</p><p>If this measure reports the value <i>Yes</i> for a node - i.e., if too many processes are running on a node - then you can use the detailed diagnosis of this measure to figure out the reason for the anomaly.</p></td></tr>
<tr><td>Is node ready?</td><td>Indicates whether/not a node is healthy and ready to accept Pods.</td><td></td><td><p>This measure reports the value <i>Yes</i>, if a node is healthy and is ready to accept Pods. The value <i>No</i> is reported if a node is not healthy and is not accepting Pods. The value <i>Unknown</i> is reported if the node controller has not heard from the node in the last <b>node-monitor-grace-period</b> (default is 40 seconds).</p><p>The values that this measure reports and their corresponding numeric values are detailed in the table below:</p>
<table>
<tr><th>Measure Value</th><th>Numeric Value</th></tr>
<tr><td>Yes</td><td>1</td></tr>
<tr><td>No</td><td>0</td></tr>
<tr><td>Unknown</td><td>2</td></tr>
</table>
<p><b>Note:</b></p><p>By default, this test reports the <b>Measure Value</b>s listed in the table above to indicate whether/not a node is under PID pressure. In the graph of this measure however, the same is indicated using the numeric equivalents only.</p><p>If this measure reports the value <i>No</i> or <i>Unknown</i> for a node, then you can use the</p></td></tr>
</table>

Measurement	Description	Measurement Unit	Interpretation
			detailed diagnosis of this measure to figure out the reason for the anomaly.
Total CPUs	Indicates the total CPU capacity of this node, in terms of the number of CPU cores it supports.	Number	
Memory capacity	Indicates the total memory capacity of this node.	GB	
Pods capacity	Indicates the maximum number of Pods that can be scheduled on this node.	Number	
Running pods	Indicates the number of Pods currently running on this node.	Number	If the value of this measure for a node is equal to or is growing closer to the value of the Pods capacity measure, it indicates that that node has or is about to exhaust its Pod capacity. You can use the detailed diagnosis of this measure to know which Pods are running on the node and which containers are running within each Pod.
Pods percent	Indicates the percentage of the Pod capacity of this node that is currently being utilized.	Percent	The formula used to compute the value of this measure is as follows: *(Running pods/Pods capacity)*100* A value equal to or close to 100% indicates that the node has or is about to exhaust its Pod capacity. In such circumstances, you may want to consider increasing the Pod capacity of the node or freeing the node of unused/inactive Pods.
Total containers	Indicates the total number of containers running on this node.	Number	To know which containers are running on the node, use the detailed diagnosis of this measure.
CPU capacity	Indicates the CPU	Millicpu	

Measurement	Description	Measurement Unit	Interpretation
	capacity of this node.		
CPU limits	Indicates the total amount of CPU resources that containers on this node are allowed to use.	Millicpu	The value of this measure is the sum of CPU limits set for the individual containers across all the Pods running on this node. If the value of this measure is greater than the value of the *CPU capacity* measure, it could mean that one/more Pods have oversubscribed to the node's CPU capacity.
CPU requests	Indicates the minimum amount of CPU resources guaranteed to all the containers on this node.	Millicpu	The value of this measure is the sum of CPU requests configured for the individual containers across all the Pods running on this node.
Memory limits	Indicates the total amount of memory resources that containers on this node are allowed to use.	GB	The value of this measure is the sum of memory limits set for the individual containers across all the Pods running on this node. If the value of this measure is greater than the value of the *Memory capacity* measure, it could mean that one/more Pods have oversubscribed to the node's memory capacity.
Memory requests	Indicates the minimum amount of memory resources guaranteed to all the containers on this node.	GB	The value of this measure is the sum of memory requests configured for the individual containers across all the Pods running on this node.
CPU limits allocation	Indicates what percentage of the capacity of this node is allocated as CPU limits to containers. In other words, this is the percentage of a node's CPU capacity that the containers on that node	Percent	The formula used for computing this measure is as follows: *(CPU limits/CPU capacity)*100* If the value of this measure exceeds 100%, it means that the node is overcommitted. In other words, it means that the Pods on the node have been allowed to use more resources than the node's capacity. In such

Measurement	Description	Measurement Unit	Interpretation
	are allowed to use.		a situation, you may want to look up the detailed diagnostics of this measure to identify the Pods that are contributing to the overcommitment.
Memory limits allocation	Indicates what percentage of the memory capacity of this node is allocated as memory limits to containers. In other words, this is the percentage of a node's memory capacity that the containers on that node are allowed to use.	Percent	The formula used for computing this measure is as follows: *(Memory limits/Memory capacity)*100* If the value of this measure exceeds 100%, it means that the node is overcommitted. In other words, it means that the Pods on the node have been allowed to use more resources than the node's capacity. In such a situation, you may want to look up the detailed diagnostics of this measure to identify the Pods that are contributing to the overcommitment.
CPU requests allocation	Indicates what percentage of the total CPU capacity of this node is set as CPU requests for the containers on that node. In other words, this is the percentage of a node's CPU capacity that the containers on that node are guaranteed to receive.	Percent	The formula used for computing this measure is as follows: *(CPU requests/CPU capacity)*100* Compare the value of this measure across nodes to know which node has been guaranteed the maximum CPU resources. You can even use the detailed diagnosis of this measure to identify the specific Pods in that node with the maximum CPU requests.
Memory requests allocation	Indicates what percentage of the total memory capacity of this node is set as memory requests for the containers on that node. In other words, this is the percentage of a node's memory	Percent	The formula used for computing this measure is as follows: *(Memory requests/Memory capacity)*100* Compare the value of this measure across nodes to know which node has been guaranteed the maximum memory resources. You can even use the detailed diagnosis of this measure to identify the

<table>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
<tr><td></td><td>capacity that the containers on that node are guaranteed to receive.</td><td></td><td>specific Pods in that node with the maximum memory requests.</td></tr>
<tr><td>CPU allocation overcommited</td><td>Indicates whether/not this node is overcommitted in terms of CPU resources.</td><td></td><td>If the CPU limits allocation measure reports a value greater than 100% for a node, then this measure will report the value <i>True</i> for that node. This implies that the node's CPU resources are overcommitted. On the other hand, if the CPU limits allocation measure of a node reports a value lesser than 100%, then this measure will report the value <i>False</i> for that node. This implies that the node's CPU resources are not overcommitted.
The values that this measure reports and their corresponding numeric values are detailed in the table below:
<table><tr><th>Measure Value</th><th>Numeric Value</th></tr><tr><td>True</td><td>1</td></tr><tr><td>False</td><td>0</td></tr></table>
<b>Note:</b>
By default, this test reports the <b>Measure Value</b>s listed in the table above to indicate whether/not a node's CPU resources are overcommitted. In the graph of this measure however, the same is indicated using the numeric equivalents only.
In an overcommitted environment, it is possible that the Pods on the node will attempt to use more compute resource than is available at any given point in time. To know which Pods are using more resources than the node's capacity, use the detailed diagnosis of this measure.</td></tr>
</table>

Measurement	Description	Measurement Unit	Interpretation
			When an overcommitment occurs, the node must give priority to one Pod over another. The facility used to make this decision is referred to as a Quality of Service (QoS) Class. By assigning a QOS class to each container, administrators can make sure that the performance of mission-critical applications does not suffer owing to insufficient resources. For each compute resource, a container is divided into one of three QoS classes with decreasing order of priority: • Priority 1 (highest) - Guaranteed - If limits and optionally requests are set (not equal to 0) for all resources and they are equal, then the container is classified as Guaranteed. Guaranteed containers are considered top priority, and are guaranteed to only be terminated if they exceed their limits, or if the system is under resource pressure and there are no lower priority containers that can be evicted. • Priority 2 - Burstable - If requests and optionally limits are set (not equal to 0) for all resources, and they are not equal, then the container is classified as Burstable. Burstable containers under resource pressure are more likely to be terminated once they exceed their requests and no other BestEffort containers exist.

<table>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
<tr><td></td><td></td><td></td><td>• Priority 3 (lowest) - BestEffort - If requests and limits are not set for any of the resources, then the container is classified as BestEffort. BestEffort containers are treated with the lowest priority. Processes in these containers are first to be terminated if the system runs out of resources.
Administrators can also control the level of overcommit and manage container density on nodes. For this, masters can be configured to override the ratio between request and limit set on developer containers. In conjunction with a per-project LimitRange specifying limits and defaults, this adjusts the container limit and request to achieve the desired level of overcommit.</td></tr>
<tr><td>Memory allocation overcommitted</td><td>Indicates whether/not this node is overcommitted in terms of memory resources.</td><td></td><td>If the <i>Memory limits allocation</i> measure reports a value greater than 100% for a node, then this measure will report the value <i>True</i> for that node. This implies that the node's memory resources are overcommitted. On the other hand, if the <i>Memory limits allocation</i> measure of a node reports a value lesser than 100%, then this measure will report the value <i>False</i> for that node. This implies that the node's memory resources are not overcommitted.
The values that this measure reports and their corresponding numeric values are detailed in the table below:
<table><tr><th>Measure Value</th><th>Numeric Value</th></tr></table></td></tr>
</table>

<table>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
<tr><td></td><td></td><td></td><td>
<table>
<tr><td>True</td><td>1</td></tr>
<tr><td>False</td><td>0</td></tr>
</table>

Note:

By default, this test reports the **Measure Value**s listed in the table above to indicate whether/not a node's memory resources are overcommitted. In the graph of this measure however, the same is indicated using the numeric equivalents only.

In an overcommitted environment, it is possible that the Pods on the node will attempt to use more compute resource than is available at any given point in time. To know which Pods may attempt to use more resources than the node's capacity, use the detailed diagnosis of this measure.

When an overcommitment occurs, the node must give priority to one pod over another. The facility used to make this decision is referred to as a Quality of Service (QoS) Class. By assigning a QOS class to each container, administrators can make sure that the performance of mission-critical applications does not suffer owing to insufficient resources.

For each compute resource, a container is divided into one of three QoS classes with decreasing order of priority:

- Priority 1 (highest) - Guaranteed - If limits and optionally requests are set (not equal to 0) for all resources and they are equal, then the container is classified as Guaranteed. Guaranteed containers
</td></tr>
</table>

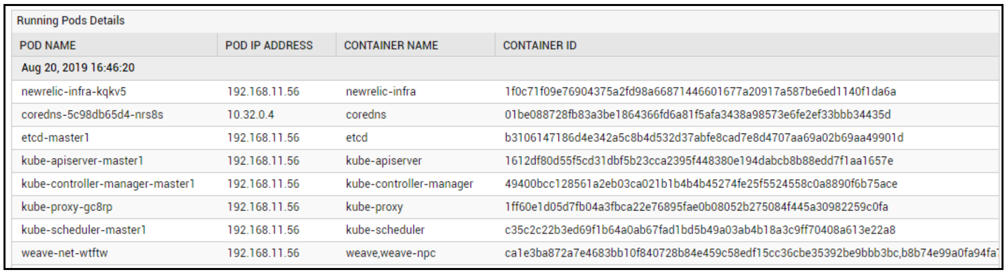
Measurement	Description	Measurement Unit	Interpretation
			are considered top priority, and are guaranteed to only be terminated if they exceed their limits, or if the system is under resource pressure and there are no lower priority containers that can be evicted. • Priority 2 - Burstable - If requests and optionally limits are set (not equal to 0) for all resources, and they are not equal, then the container is classified as Burstable. Burstable containers under resource pressure are more likely to be terminated once they exceed their requests and no other BestEffort containers exist. • Priority 3 (lowest) - BestEffort - If requests and limits are not set for any of the resources, then the container is classified as BestEffort. BestEffort containers are treated with the lowest priority. Processes in these containers are first to be terminated if the system runs out of resources. Administrators can also control the level of overcommit and manage container density on nodes. For this, masters can be configured to override the ratio between request and limit set on developer containers. In conjunction with a per-project LimitRange specifying limits and defaults, this adjusts the container limit and request to achieve the desired level of overcommit.

<table>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
<tr><td>Total images</td><td>Indicates the total number of images on this node.</td><td>Number</td><td>Use the detailed diagnosis of this measure to know which images are on the node.</td></tr>
<tr><td>Used images</td><td>Indicates the total number of images currently used by the containers on this node.</td><td>Number</td><td>To view the used images, use the detailed diagnosis of this measure.</td></tr>
<tr><td>Not used images</td><td>Indicates the number of images still to be used by the containers on this node.</td><td>Number</td><td>To view the unused images, use the detailed diagnosis of this measure.</td></tr>
<tr><td>Images size</td><td>Indicates the total size of images on this node.</td><td>GB</td><td></td></tr>
<tr><td>Node type</td><td>Indicates the node type.</td><td></td><td>A node can be a Master node or a Worker node in a cluster. A cluster has at least one worker node and at least one master node.The worker node(s) host the pods that are the components of the application. The master node(s) manages the worker nodes and the pods in the cluster. Multiple master nodes are used to provide a cluster with failover and high availability.
If a node is the master node in a cluster, then this measure will report the value Master. For a worker node, this measure will report the value Worker.
The numeric values that correspond to these measure values are as follows:
<table>
<tr><th>Measure Value</th><th>Numeric Value</th></tr>
<tr><td>Master</td><td>1</td></tr>
<tr><td>Worker</td><td>2</td></tr>
</table>
<b>Note:</b>
By default, this test reports the <b>Measure Value</b>s listed in the table above to indicate</td></tr>
</table>

Measurement	Description	Measurement Unit	Interpretation
			the node type. In the graph of this measure however, the same is indicated using the numeric equivalents only.
CPU usage	Indicates the amount of CPU resources used by this node.	Millicpu	Ideally, the value of this measure should be much lesser than the value of the *CPU capacity* measure. If the value of this measure is equal to or is rapidly approaching the value of the *CPU capacity* measure, it means that the node is running out of CPU resources.
CPU utilization	Indicates the percentage of CPU resources utilized by this node.	Percent	A value close to 100% is indicative of excessive CPU usage by a node, and hints at a potential CPU contention on the node. A value greater than 100% implies that one/more Pods have probably over-subscribed to the node's capacity. To know which Pod on the node is contributing to the contention/overcommitment, use the detailed diagnosis of this measure.
Memory usage	Indicates the amount of memory resources used by this node.	Millicpu	Ideally, the value of this measure should be much lesser than the value of the *Memory capacity* measure. If the value of this measure is equal to or is rapidly approaching the value of the *Memory capacity* measure, it means that the node is running out of memory resources.
Memory utilization	Indicates the percentage of memory resources utilized by this node.	Percent	A value close to 100% is indicative of excessive memory usage by a node, and signals a potential memory contention on the node. A value greater than 100% implies that one/more Pods have probably over-subscribed to the node's capacity. To know which Pod on the node is contributing to the

Measurement	Description	Measurement Unit	Interpretation
			contention/overcommitment, use the detailed diagnosis of this measure.

The detailed diagnosis of the *Running pods* measure reveals which Pods are running on the node and which containers are running within each Pod.

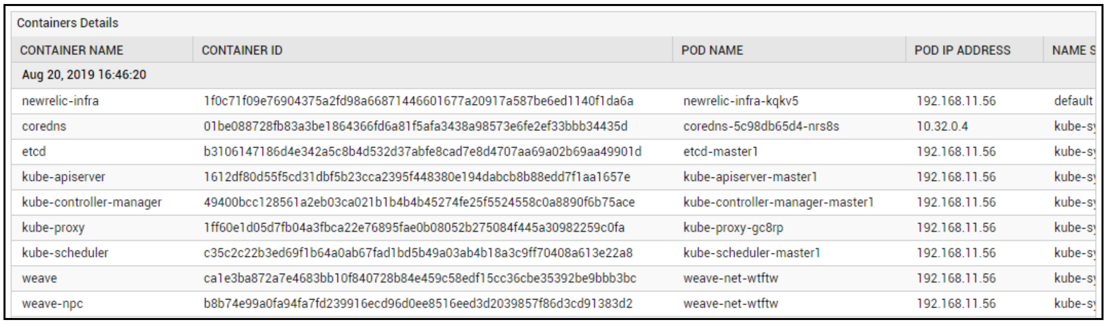


Running Pods Details

POD NAME	POD IP ADDRESS	CONTAINER NAME	CONTAINER ID
Aug 20, 2019 16:46:20			
newrelic-infra-kqkv5	192.168.11.56	newrelic-infra	1f0c71f09e76904375a2fd98a66871446601677a20917a587be6ed1140f1da6a
coredns-5c98db65d4-nrs8s	10.32.0.4	coredns	01be088728fb83a3be1864366fd6a81f5afa3438a98573e6fe2ef33bbb34435d
etcd-master1	192.168.11.56	etcd	b3106147186d4e342a5c8b4d532d37abfe8cad7e8d4707aa69a02b69aa49901d
kube-apiserver-master1	192.168.11.56	kube-apiserver	1612df80d55f5cd31dbf5b23cca2395f448380e194dabcb8b88edd7f1aa1657e
kube-controller-manager-master1	192.168.11.56	kube-controller-manager	49400bcc128561a2eb03ca021b1b4b4b45274fe25f5524558c0a8890f6b75ace
kube-proxy-gc8rp	192.168.11.56	kube-proxy	1ff60e1d05d7fb04a3fbca22e76895fae0b08052b275084f445a30982259c0fa
kube-scheduler-master1	192.168.11.56	kube-scheduler	c35c2c22b3ed69f1b64a0ab67fad1bd5b49a03ab4b18a3c9ff70408a613e22a8
weave-net-wtftw	192.168.11.56	weave,weave-npc	ca1e3ba872a7e4683bb10f840728b84e459c58edf15cc36cbe35392be9bbb3bc,b8b74e99a0fa94fa

Figure 3.13: The detailed diagnosis of the Running pods measure

The detailed diagnosis of the *Total containers* measure reveals the names of containers running on a node, the Pod to which each container belongs, and the namespace to which the Pod belongs.

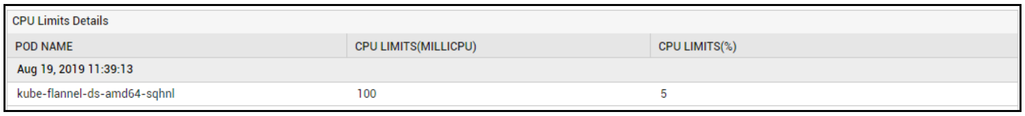


Containers Details

CONTAINER NAME	CONTAINER ID	POD NAME	POD IP ADDRESS	NAME S
Aug 20, 2019 16:46:20				
newrelic-infra	1f0c71f09e76904375a2fd98a66871446601677a20917a587be6ed1140f1da6a	newrelic-infra-kqkv5	192.168.11.56	default
coredns	01be088728fb83a3be1864366fd6a81f5afa3438a98573e6fe2ef33bbb34435d	coredns-5c98db65d4-nrs8s	10.32.0.4	kube-sy
etcd	b3106147186d4e342a5c8b4d532d37abfe8cad7e8d4707aa69a02b69aa49901d	etcd-master1	192.168.11.56	kube-sy
kube-apiserver	1612df80d55f5cd31dbf5b23cca2395f448380e194dabcb8b88edd7f1aa1657e	kube-apiserver-master1	192.168.11.56	kube-sy
kube-controller-manager	49400bcc128561a2eb03ca021b1b4b4b45274fe25f5524558c0a8890f6b75ace	kube-controller-manager-master1	192.168.11.56	kube-sy
kube-proxy	1ff60e1d05d7fb04a3fbca22e76895fae0b08052b275084f445a30982259c0fa	kube-proxy-gc8rp	192.168.11.56	kube-sy
kube-scheduler	c35c2c22b3ed69f1b64a0ab67fad1bd5b49a03ab4b18a3c9ff70408a613e22a8	kube-scheduler-master1	192.168.11.56	kube-sy
weave	ca1e3ba872a7e4683bb10f840728b84e459c58edf15cc36cbe35392be9bbb3bc	weave-net-wtftw	192.168.11.56	kube-sy
weave-npc	b8b74e99a0fa94fa7fd239916ecd96d0ee8516eed3d2039857f86d3cd91383d2	weave-net-wtftw	192.168.11.56	kube-sy

Figure 3.14: The detailed diagnosis of the Total containers measure

If the *CPU limits allocation* measure reports a value over 100%, it indicates an overcommitment of CPU resources on the node. In such a situation, you can use the detailed diagnostics of this measure to identify the Pods that are contributing to the overcommitment.

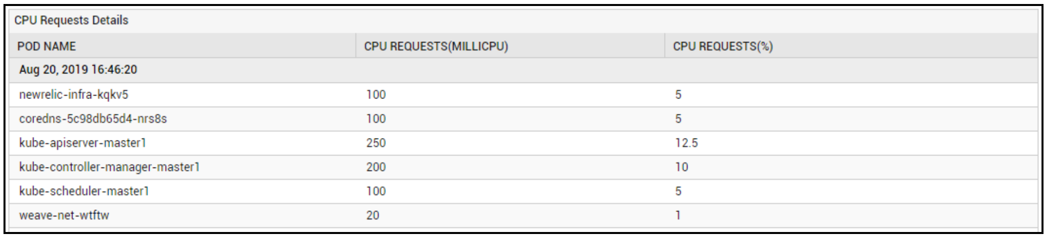


CPU Limits Details

POD NAME	CPU LIMITS(MILLICPU)	CPU LIMITS(%)
Aug 19, 2019 11:39:13		
kube-flannel-ds-amd64-sqhnl	100	5

Figure 3.15: The detailed diagnosis of the CPU limits allocation measure

Using the detailed diagnosis of the *CPU requests allocation* measure, you can quickly identify the specific Pods on the node with the maximum CPU requests. In the event of a CPU contention on the node, this information will lead you to the exact Pod that is hogging CPU resources.

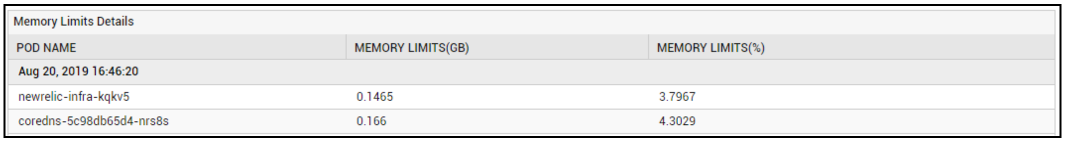


CPU Requests Details

POD NAME	CPU REQUESTS(MILLICPU)	CPU REQUESTS(%)
Aug 20, 2019 16:46:20		
newrelic-infra-kqkv5	100	5
coredns-5c98db65d4-nrs8s	100	5
kube-apiserver-master1	250	12.5
kube-controller-manager-master1	200	10
kube-scheduler-master1	100	5
weave-net-wtftw	20	1

Figure 3.16: The detailed diagnosis of the CPU requests allocation measure

If the *Memory limits allocation* measure reports a value over 100%, it indicates an overcommitment of memory resources on the node. In such a situation, you can use the detailed diagnostics of this measure to identify the Pods that are contributing to the overcommitment.

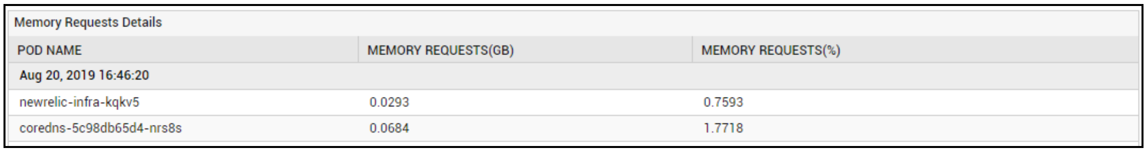


Memory Limits Details

POD NAME	MEMORY LIMITS(GB)	MEMORY LIMITS(%)
Aug 20, 2019 16:46:20		
newrelic-infra-kqkv5	0.1465	3.7967
coredns-5c98db65d4-nrs8s	0.166	4.3029

Figure 3.17: The detailed diagnosis of the Memory limits allocation measure

Using the detailed diagnosis of the *Memory requests allocation* measure, you can quickly identify the specific Pods on the node with the maximum memory requests. In the event of a memory contention on the node, this information will lead you to the exact Pod that is hogging memory resources.



Memory Requests Details

POD NAME	MEMORY REQUESTS(GB)	MEMORY REQUESTS(%)
Aug 20, 2019 16:46:20		
newrelic-infra-kqkv5	0.0293	0.7593
coredns-5c98db65d4-nrs8s	0.0684	1.7718

Figure 3.18: The detailed diagnosis of the Memory requests allocation measure

3.1.7 Kube Cluster Overview Test

A Kubernetes cluster is a set of machines, called nodes, that run containerized applications managed by Kubernetes. A cluster has at least one worker node and at least one master node.

The worker node(s) host the pods that are the components of the application. The master node(s) manages the worker nodes and the pods in the cluster. Multiple master nodes are used to provide a cluster with failover and high availability.

The kube-scheduler schedules Pods to a node, based on the resource capacity of the node and the resource requirements of the containers in the Pods. To ensure that no Pod hogs the node's resources, resource requests and limits can be set per container.

At any given point in time, an administrator needs to have a macro view of the composition of their Kubernetes cluster - i.e., the number of nodes and Pods in the cluster - and the operational state of the nodes and Pods. This will help them quickly spot nodes and Pods that have failed - i.e., it will help them quickly detect a mismatch between the actual state of the cluster and its desired state. By taking appropriate action on such mismatches, administrators can prevent any adverse impact on the availability and performance of containerized applications. Additionally, administrators also need to track how the Pods are utilizing the cluster's compute resources. This way, they can proactively detect probable resource contentions / over-subscriptions, and rapidly initiate measures to right-size the cluster components (i.e., Pods and containers), so that application performance is not affected by resource crunches. Administrators also require an overview of Deployments across the cluster, so that they can easily locate problem areas. The Kube Cluster Overview test provides administrators with all these useful high-level insights!

This test monitors a Kubernetes cluster, reports the total count of nodes in the cluster, and also precisely pinpoints the master and worker nodes of the cluster. The test also tracks the Pod capacity of the cluster alongside Pod allocations, and additionally highlights Pods and nodes in an abnormal state. This enables administrators to rapidly detect any glaring mismatch between the desired state and actual state of the cluster and initiate appropriate remedial measures. Furthermore, the test reveals how the Pods in the cluster are utilizing the cluster's compute resource capacity. In the process, the test brings to light irregularities such as resource over- subscription and current/potential resource contention. Detailed diagnostics provided by the test lead administrators to the exact Pods that are hogging cluster resources, or have been poorly sized. This way, the test points administrators to those Pods for which resource allocations need to be fine-tuned to ensure optimal cluster performance. In addition, the test helps administrators easily compare the desired state of Deployments with the actual state, so that they can instantly capture and resolve discrepancies (if any).

Target of the test : A Kubernetes Cluster

Agent deploying the test : A remote agent

Outputs of the test : One set of results for the Kubernetes cluster being monitored

Configurable parameters for the test

Parameter	Description
Test Period	How often should the test be executed.
Host	The IP address of the host for which this test is to be configured.
Port	Specify the port at which the specified Host listens. By default, this is *6443*.
Load Balancer / Master Node IP	To run this test and report metrics, the eG agent needs to connect to the Kubernetes API on the master node and run API commands. To enable this connection, the eG agent has to be configured with either of the following: • If only a single master node exists in the cluster, then configure the eG agent with the IP address of the master node. • If the target cluster consists of more than one master node, then you need to configure the eG agent with the IP address of the load balancer that is managing the cluster. In this case, the load balancer will route the eG agent's connection request to any available master node in the cluster, thus enabling the agent to connect with the API server on that node, run API commands on it, and pull metrics. By default, this parameter will display the **Load Balancer / Master Node IP** that you configured when manually adding the Kubernetes cluster for monitoring, using the **Kubernetes Cluster Preferences** page in the eG admin interface (see Figure 2.3). The steps for managing the cluster using the eG admin interface are discussed elaborately in How to Monitor the Kubernetes Cluster Using eG Enterprise? Whenever the eG agent runs this test, it uses the IP address that is displayed (by default) against this parameter to connect to the Kubernetes API. If there is any change in this IP address at a later point in time, then make sure that you update this parameter with it, by overriding its default setting.
SSL	By default, the Kubernetes cluster is SSL-enabled. This is why, the eG agent, by default, connects to the Kubernetes API via an HTTPS connection. Accordingly, this flag is set to **Yes** by default. If the cluster is not SSL-enabled in your environment, then set this flag to **No**.
Authentication Token	The eG agent requires an authentication bearer token to access the Kubernetes API, run API commands on the cluster, and pull metrics of interest. The steps for generating this token have been detailed in Section **1.1** Typically, once you generate the token, you can associate that token with the target Kubernetes cluster, when manually adding that cluster for monitoring using the

Parameter	Description
	eG admin interface. The steps for managing the cluster using the eG admin interface are discussed elaborately in Section **Chapter 2** By default, this parameter will display the **Authentication Token** that you provided in the **Kubernetes Cluster Preferences page** of the eG admin interface, when manually adding the cluster for monitoring (see Figure 2.3). Whenever the eG agent runs this test, it uses the token that is displayed (by default) against this parameter for accessing the API and pulling metrics. If for any reason, you generate a new authentication token for the target cluster at a later point in time, then make sure you update this parameter with the change. For that, copy the new token and paste it against this parameter.
Proxy Host	If the eG agent connects to the Kubernetes API on the master node via a proxy server, then provide the IP address of the proxy server here. If no proxy is used, then the default setting -*none* - of this parameter, need not be changed,
Proxy Port	If the eG agent connects to the Kubernetes API on the master node via a proxy server, then provide the port number at which that proxy server listens here. If no proxy is used, then the default setting -*none* - of this parameter, need not be changed,
Proxy Username, Proxy Password, Confirm Password	**These parameters are applicable only if the eG agent uses a proxy server to connect to the Kubernetes cluster, and that proxy server requires authentication.** In this case, provide a valid user name and password against the **Proxy Username** and **Proxy Password** parameters, respectively. Then, confirm the password by retyping it in the **Confirm Password** text box. If no proxy server is used, or if the proxy server used does not require authentication, then the default setting - *none* - of these parameters, need not be changed.
DD Frequency	Refers to the frequency with which detailed diagnosis measures are to be generated for this test. The default is *1:1*. This indicates that, by default, detailed measures will be generated every time this test runs, and also every time the test detects a problem. You can modify this frequency, if you so desire. Also, if you intend to disable the detailed diagnosis capability for this test, you can do so by specifying *none* against DD frequency.
Detailed Diagnosis	To make diagnosis more efficient and accurate, the eG Enterprise suite embeds an optional detailed diagnostic capability. With this capability, the eG agents can be configured to run detailed, more elaborate tests as and when specific problems are detected. To enable the detailed diagnosis capability of this test for a particular server, choose the **On** option. To disable the capability, click on the **Off** option. The option to selectively enable/disable the detailed diagnosis capability will be available only if the following conditions are fulfilled:

Parameter	Description
	• The eG manager license should allow the detailed diagnosis capability • Both the normal and abnormal frequencies configured for the detailed diagnosis measures should not be 0.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Total nodes	Indicates the total number of nodes in the cluster.	Number	
Master nodes	Indicates the count of master nodes in the cluster.	Number	Use the detailed diagnosis of this measure to know which are the master nodes in the cluster.
Worker nodes	Indicates the number of worker nodes in the cluster.	Number	Use the detailed diagnosis of this measure to know which are the worker nodes in the cluster.
Nodes added to cluster	Indicates the number of nodes that were added to the cluster since the last measurement period.	Number	Use the detailed diagnosis of this measure to know which nodes were recently added to the cluster.
Nodes removed from cluster	Indicates the number of nodes that were removed from the cluster since the last measurement period.	Number	Use the detailed diagnosis of this measure to know which nodes were recently removed from the cluster.
Running nodes	Indicates the number of nodes in the cluster that are currently running.	Number	
Not running nodes	Indicates the number of nodes in the cluster that are not running presently.	Number	Use the detailed diagnosis of this measure to know which nodes are not running and why.
Unknown nodes	Indicates the number of nodes in the cluster that	Number	Use the detailed diagnosis of this measure to know which nodes are in an

Measurement	Description	Measurement Unit	Interpretation
	are in the Unknown presently.		Unknown state and why.
Pods capacity	Indicates the maximum number of Pods that can be created on the nodes in the cluster.	Number	
Allocated pods	Indicates the number of Pods that have been scheduled to nodes in the cluster.	Number	If the value of this measure is equal to or close to the value of the *Pods capacity* measure, it indicates that the cluster has or is about to exhaust its capacity. In such a situation, you may want to add more nodes to your cluster or increase the Pod capacity of your cluster.
Running pods	Indicates the number of Pods in the cluster that are in the Running state currently.	Number	If a Pod is in the Running state, it means that the Pod has been bound to a node, and all of the Containers have been created. At least one Container is still running, or is in the process of starting or restarting. Use the detailed diagnosis of this measure to know which Pods are in the Running state.
Pending pods	Indicates the number of Pods in the cluster that are in the Pending state currently.	Number	If a Pod is in the Pending state, it means that the Pod has been accepted by the Kubernetes system, but one or more of the Container images has not been created. This includes time before being scheduled as well as time spent downloading images over the network, which could take a while. If a pod is stuck in Pending it means that it can not be scheduled onto a node. Generally this is because there are insufficient resources of one type or another that prevent scheduling. If this is the case, do the following:

Measurement	Description	Measurement Unit	Interpretation
			• Add more nodes to the cluster. • Terminate unneeded pods to make room for pending pods. • Check that the pod is not larger than your nodes. For example, if all nodes have a capacity of cpu:1, then a pod with a request of cpu: 1.1 will never be scheduled. Use the detailed diagnosis of this measure to know which Pods are in the Pending state.
Succeeded pods	Indicates the number of Pods in the cluster that are in the Succeeded state currently.	Number	If a Pod is in the Succeeded state, it means that all Containers in the Pod have terminated in success, and will not be restarted.
Failed pods	Indicates the number of Pods in the cluster that are in the Failed state currently.	Number	If a Pod is in the Failed state, it means that all Containers in the Pod have terminated, and at least one Container has terminated in failure. That is, the Container either exited with non-zero status or was terminated by the system. Use the detailed diagnosis of this measure to know which Pods are in the Failed state. Ideally, the value of this measure should be 0.
Unknown pods	Indicates the number of Pods in the cluster that are in the Unknown state currently.	Number	If a Pod is in the Unknown state, it means that the state of the Pod could not be obtained, probably due to an error in communicating with the host of the Pod. Ideally, the value of this measure should be 0.

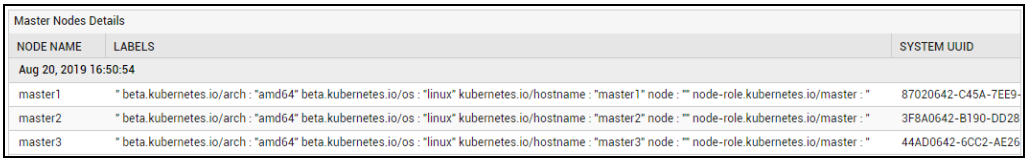
Measurement	Description	Measurement Unit	Interpretation
Running pods utilization	Indicates the percentage of Pods in the cluster that are in a Running state currently.	Percent	The formula used for computing this measure is as follows: *[Running pods/Pods capacity]*100* Ideally, the value of this measure should be high.
Total CPUs	Indicates the total number of CPU cores supported by the cluster.	Number	
CPU capacity	Indicates the total CPU capacity of the cluster.	Millicpu	
CPU requests	Indicates the minimum CPU resources guaranteed to the Pods in the cluster.	Millicpu	This is the sum of CPU requests configured for all containers in all Pods across nodes in the cluster. A request is the amount of that resource that the system will guarantee to a Pod.
CPU limits	Indicates that maximum amount of CPU resources that the Pods in the cluster can use.	Millicpu	This is the sum of CPU limits set for all containers in all Pods across nodes in the cluster. A limit is the maximum amount that the system will allow the Pod to use.
CPU limits allocation	Indicates what percentage of the CPU capacity of the cluster is allocated as CPU limits to containers. In other words, this is the percentage of a cluster's CPU capacity that the containers are allowed to use.	Percent	The formula used for computing this measure is as follows: *(CPU limits/CPU capacity)*100* If the value of this measure exceeds 100%, it means that one/more Pods are probably over-subscribing to the capacity of one/more nodes.
CPU requests allocation	Indicates what percentage of the total CPU capacity of the cluster is set as CPU requests for the	Percent	The formula used for computing this measure is as follows: *(CPU requests/CPU capacity)*100* If the value of this measure is unusually

Measurement	Description	Measurement Unit	Interpretation
	containers in the cluster. In other words, this is the percentage of a cluster's CPU capacity that the containers on the cluster are guaranteed to receive.		high, then you can use the detailed diagnosis of this measure to review the CPU requests configured for each Pod in the cluster. In the process, you can accurately identify the Pod for which the maximum amount of CPU resources in the cluster is guaranteed - i.e., the Pod that is hogging the CPU capacity of the cluster.
Memory capacity	Indicates the total memory capacity of the cluster.	GB	
Memory requests	Indicates the minimum memory resources guaranteed to the Pods in the cluster.	GB	This is the sum of memory requests configured for all containers in all Pods across nodes in the cluster. A request is the amount of that resource that the system will guarantee to the Pod.
Memory limits	Indicates the maximum amount of memory resources that the Pods in the cluster can use.	GB	This is the sum of memory limits set for all containers in all Pods across nodes in the cluster. A limit is the maximum amount that the system will allow the Pod to use.
Memory limits allocation	Indicates what percentage of the memory capacity of the cluster is allocated as memory limits to containers in the cluster. In other words, this is the percentage of a cluster's memory capacity that the containers on the cluster are allowed to use.	Percent	The formula used for computing this measure is as follows: *(Memory limits/Memory capacity)*100* If the value of this measure exceeds 100%, it means that one/more Pods are probably over-subscribing to the capacity of one/more nodes in the cluster.
Memory requests allocation	Indicates what	Percent	The formula used for computing this measure is as follows:

Measurement	Description	Measurement Unit	Interpretation
	percentage of the total memory capacity of the cluster is set as memory requests for the containers in the cluster. In other words, this is the percentage of a cluster's memory capacity that the containers in the cluster are guaranteed to receive.		*(Memory requests/Memory capacity)*100* If the value of this measure is unusually high, then you can use the detailed diagnosis of this measure to review the memory requests configured for each Pod in the cluster. In the process, you can accurately identify the Pod for which the maximum amount of memory resources in the cluster is guaranteed - i.e., the Pod that is hogging the memory capacity of the cluster.
Total pods with updated deployment	Indicates the total number of non-terminated Pod replicas in the cluster that have been updated with changes (if any) made to Pod template specifications.	Number	Typically, whenever changes are made to a Deplopyment's Pod template - say, labels or container images of the template are changed - then a Deployment rollout is triggered. A new ReplicaSet is created and the Deployment manages moving the Pods from the old ReplicaSet to the new one at a controlled rate. Ideally, the value of this measure should be the same as the value of the *Total pods with deployment* measure. If not, then it means that the desired number of Pod replicas are not yet fully updated with the changes to the Pod template.
Ready pods with deployment	Indicates the number of ready Pods created in the cluster across Deployments.	Number	
Total available pods with deployment	Indicates the number of available Pods created in the cluster across Deployments.	Number	A Pod is said to be Available, if it is ready without any containers crashing for at least the duration configured against minReadySeconds in the Pod specification. Ideally, the value of this measure should

Measurement	Description	Measurement Unit	Interpretation
			be the same as the value of the *Total pods with deployment* measure. This means that the desired state of the Deployments is not the same as their actual state.
Total unavailable pods with deployment	Indicates the total number of unavailable Pods created in the cluster across Deployments.	Number	Any Pod that is not ready, or is ready but has containers crashing for a period of time beyond the *minReadySeconds* duration, is automatically considered Unavailable. Ideally, the value of this measure should be 0. If this measure reports a non-zero value or a value equal to or close to the value of the *Total pods with deployment* measure, it means that the desired state of the Deployments is not the same as their actual state.

Use the detailed diagnosis of the *Master nodes* measure to know which are the master nodes in the cluster.

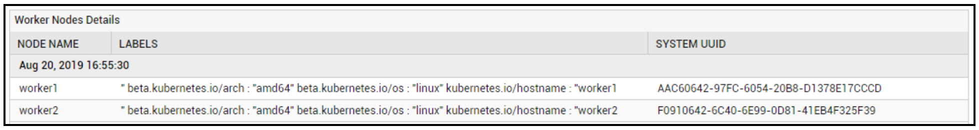


Master Nodes Details

NODE NAME	LABELS	SYSTEM UUID
Aug 20, 2019 16:50:54		
master1	" beta.kubernetes.io/arch : "amd64" beta.kubernetes.io/os : "linux" kubernetes.io/hostname : "master1" node : "" node-role.kubernetes.io/master : "	87020642-C45A-7EE9-
master2	" beta.kubernetes.io/arch : "amd64" beta.kubernetes.io/os : "linux" kubernetes.io/hostname : "master2" node : "" node-role.kubernetes.io/master : "	3F8A0642-B190-DD28
master3	" beta.kubernetes.io/arch : "amd64" beta.kubernetes.io/os : "linux" kubernetes.io/hostname : "master3" node : "" node-role.kubernetes.io/master : "	44AD0642-6CC2-AE26

Figure 3.19: The detailed diagnosis of the Master nodes measure

Use the detailed diagnosis of the *Worker nodes* measure to know which are the worker nodes in the cluster.

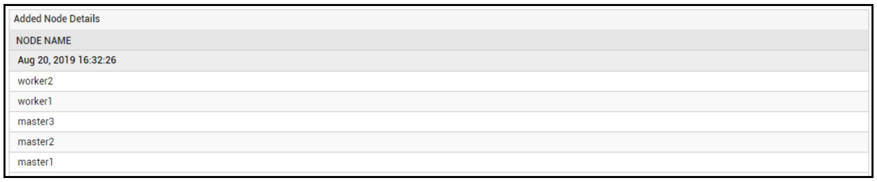


Worker Nodes Details

NODE NAME	LABELS	SYSTEM UUID
Aug 20, 2019 16:55:30		
worker1	" beta.kubernetes.io/arch : "amd64" beta.kubernetes.io/os : "linux" kubernetes.io/hostname : "worker1	AAC60642-97FC-6054-20B8-D1378E17CCCD
worker2	" beta.kubernetes.io/arch : "amd64" beta.kubernetes.io/os : "linux" kubernetes.io/hostname : "worker2	F0910642-6C40-6E99-0D81-41EB4F325F39

Figure 3.20: The detailed diagnosis of the Worker nodes measure

Use the detailed diagnosis of the *Nodes added to cluster* measure to know which nodes were recently added to the cluster.

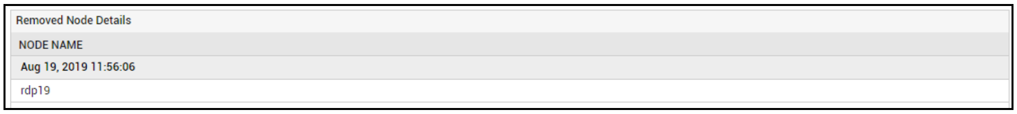


Added Node Details

NODE NAME
Aug 20, 2019 16:32:26
worker2
worker1
master3
master2
master1

Figure 3.21: The detailed diagnosis of the Nodes added to cluster measure

Use the detailed diagnosis of the *Nodes removed from cluster* measure to know which nodes were recently removed from the cluster.

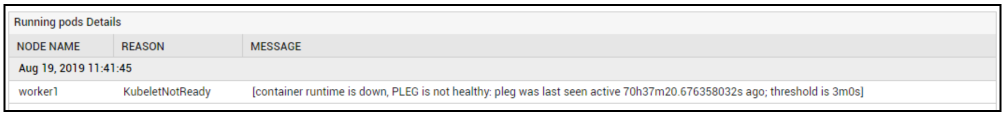


Removed Node Details

NODE NAME
Aug 19, 2019 11:56:06
rdp19

Figure 3.22: The detailed diagnosis of the Nodes removed from cluster measure

Use the detailed diagnosis of the *Nodes not running* measure to know which nodes are not running and why.

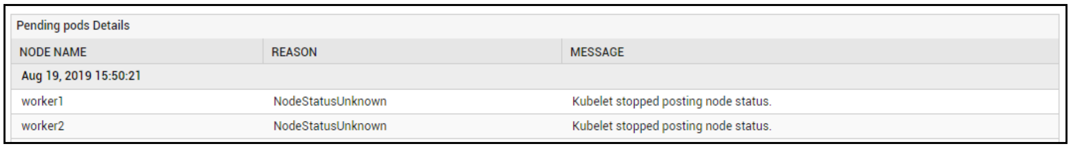


Running pods Details

NODE NAME	REASON	MESSAGE
Aug 19, 2019 11:41:45		
worker1	KubeletNotReady	[container runtime is down, PLEG is not healthy: pleg was last seen active 70h37m20.676358032s ago; threshold is 3m0s]

Figure 3.23: The detailed diagnosis of the Nodes not running measure

Use the detailed diagnosis of the *Unknown nodes* measure to know which nodes are in an Unknown state and why.

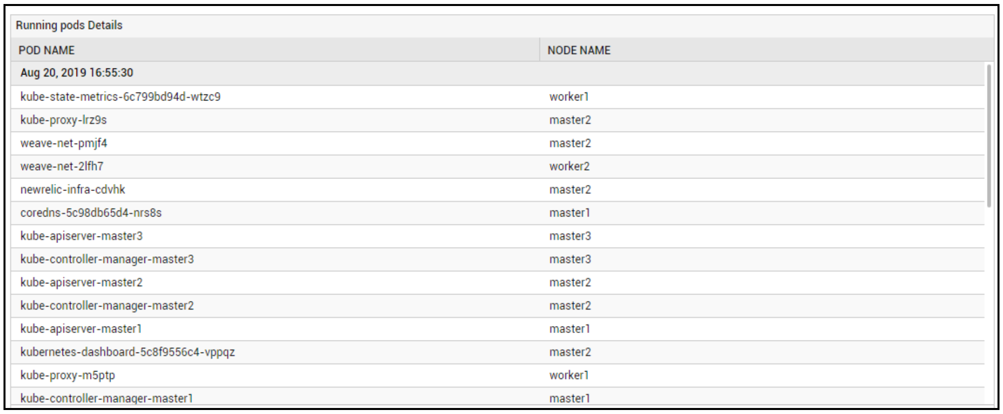


Pending pods Details

NODE NAME	REASON	MESSAGE
Aug 19, 2019 15:50:21		
worker1	NodeStatusUnknown	Kubelet stopped posting node status.
worker2	NodeStatusUnknown	Kubelet stopped posting node status.

Figure 3.24: The detailed diagnosis of the Unknown nodes measure

Use the detailed diagnosis of the *Running pods* measure to know which Pods are in the Running state and which node each running Pod is scheduled to.

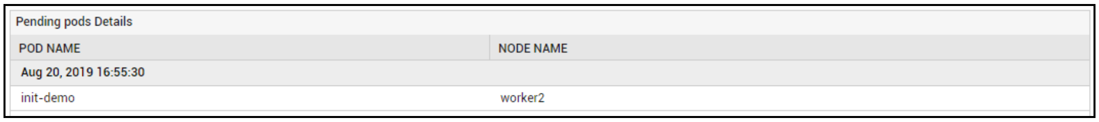


Running pods Details

POD NAME	NODE NAME
Aug 20, 2019 16:55:30	
kube-state-metrics-6c799bd94d-wtzc9	worker1
kube-proxy-lrz9s	master2
weave-net-pmjf4	master2
weave-net-2lfh7	worker2
newrelic-infra-cdvhk	master2
coredns-5c98db65d4-nrs8s	master1
kube-apiserver-master3	master3
kube-controller-manager-master3	master3
kube-apiserver-master2	master2
kube-controller-manager-master2	master2
kube-apiserver-master1	master1
kubernetes-dashboard-5c8f9556c4-vppqz	master2
kube-proxy-m5ptp	worker1
kube-controller-manager-master1	master1

Figure 3.25: The detailed diagnosis of the Running pods measure reported by the Kube Cluster Overview test

Use the detailed diagnosis of the *Pending pods* measure to know which Pods are in the Pending state and which node each pending Pod is scheduled to.

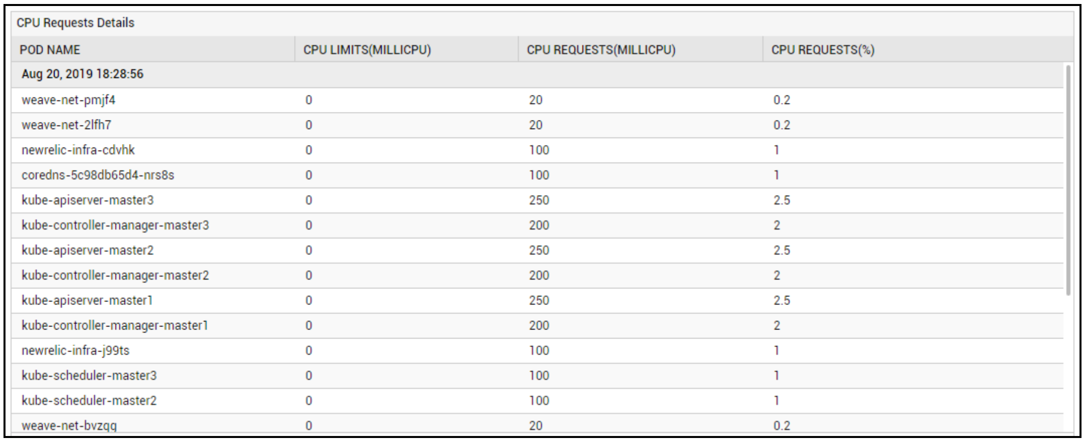


Pending pods Details

POD NAME	NODE NAME
Aug 20, 2019 16:55:30	
init-demo	worker2

Figure 3.26: The detailed diagnosis of the Pending pods measure reported by the Kube Cluster Overview test

If the value of the *CPU requests allocation* measure is unusually high, then you can use the detailed diagnosis of this measure to review the CPU requests configured for each Pod in the cluster. In the process, you can accurately identify the Pod that is guaranteed to receive the maximum amount of CPU resources in the cluster - i.e., the Pod that is hogging the CPU capacity of the cluster.

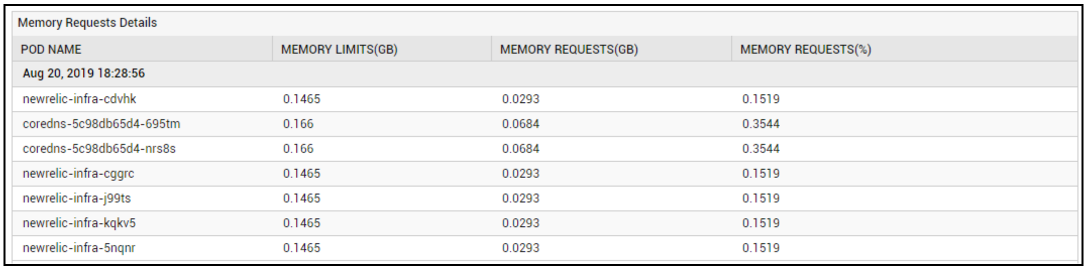


CPU Requests Details

POD NAME	CPU LIMITS(MILLICPU)	CPU REQUESTS(MILLICPU)	CPU REQUESTS(%)
Aug 20, 2019 18:28:56			
weave-net-pmjf4	0	20	0.2
weave-net-2lfh7	0	20	0.2
newrelic-infra-cdvhk	0	100	1
coredns-5c98db65d4-nrs8s	0	100	1
kube-apiserver-master3	0	250	2.5
kube-controller-manager-master3	0	200	2
kube-apiserver-master2	0	250	2.5
kube-controller-manager-master2	0	200	2
kube-apiserver-master1	0	250	2.5
kube-controller-manager-master1	0	200	2
newrelic-infra-j99ts	0	100	1
kube-scheduler-master3	0	100	1
kube-scheduler-master2	0	100	1
weave-net-bvzqq	0	20	0.2

Figure 3.27: The detailed diagnosis of the CPU requests allocation measure reported by the Kube Cluster Overview test

If the value of the *Memory requests allocation* measure is unusually high, then you can use the detailed diagnosis of this measure to review the memory requests configured each Pod in the cluster. In the process, you can accurately identify the Pod that is guaranteed to receive the maximum amount of memory resources in the cluster - i.e., the Pod that is hogging the memory capacity of the cluster.



Memory Requests Details

POD NAME	MEMORY LIMITS(GB)	MEMORY REQUESTS(GB)	MEMORY REQUESTS(%)
Aug 20, 2019 18:28:56			
newrelic-infra-cdvhk	0.1465	0.0293	0.1519
coredns-5c98db65d4-695tm	0.166	0.0684	0.3544
coredns-5c98db65d4-nrs8s	0.166	0.0684	0.3544
newrelic-infra-cggrc	0.1465	0.0293	0.1519
newrelic-infra-j99ts	0.1465	0.0293	0.1519
newrelic-infra-kqkv5	0.1465	0.0293	0.1519
newrelic-infra-5nqnr	0.1465	0.0293	0.1519

Figure 3.28: The detailed diagnosis of the Memory request allocation measure reported by the Kube Cluster Overview test

3.1.8 Kube Namespaces Test

Kubernetes supports multiple virtual clusters backed by the same physical cluster. These virtual clusters are called namespaces.

Namespaces are intended for use in environments with many users spread across multiple teams, or projects. Namespaces are a way to divide cluster resources between multiple users (via resource quota). A resource quota, defined by a ResourceQuota object, provides constraints that limit aggregate resource consumption per namespace. It can limit the API resources - i.e., the quantity of objects (eg., pods, services, deployments etc.) that can be created in a namespace , as well as the total amount of compute resources - i.e., CPU and memory - that may be consumed by the API resources in that namespace.

If quota is enabled in a namespace for compute resources like CPU and memory, users must specify requests and/or limits for those values. A request is the amount of that resource that the system will guarantee to the namespace, and a limit is the maximum amount that the system will allow the namespace to use. Typically, within a namespace, a Pod or Container can consume as much CPU and memory as defined by the namespace's resource quota. You can also define compute resource usage limits for individual containers, during their creation. However, to ensure that no single pod/container in a namespace hogs the resources of that namespace, you can define a Limit Range. Limit Range is a policy to constrain resource by Pod or Container in a namespace. A limit range provides constraints that can:

- Enforce minimum and maximum compute resources usage per Pod or Container in a namespace.
- Enforce minimum and maximum storage request per PersistentVolumeClaim in a namespace.
- Enforce a ratio between request and limit for a resource in a namespace.
- Set default request/limit for compute resources in a namespace and automatically inject them to Containers at runtime. The default requests/limits will apply to those containers for which requests and/or limits have not been specifically defined.

If creating or updating an API resource in a namespace violates a quota constraint / limit range, then that create/update request will fail. For instance, say a namespace is configured with a resource quota that restricts the number of Pods that can be created in that namespace to 2. In this case, if the creation of a third Pod is attempted within that namespace, then the pod creation will fail. Likewise, if you are attempting to create a container with a memory limit of 2Gi within a namespace that has a resource quota constraint of 1Gi, then Kubernetes will not allow the container to be created. This can eventually result in a mismatch between the cluster's desired state and its actual state. To avoid this, administrators must first be well- aware of the resource request/limit that has been set per namespace and also for the pods and containers in each namespace. Then, administrators should track how the containers in each namespace are using the allocated compute resources, and determine whether any namespace is likely to violate its quota, well before an actual violation happens. The Kube Namespaces enables administrators to pre-empt any adverse impact to cluster health by monitoring namespaces, their quota definitions, and their resource usage!

This test auto-discovers the namespaces configured in a Kubernetes cluster, and reports the current state of each namespace, thus bringing inactive/terminating namespaces to light. Additionally, the test also reports the request/limit settings for each namespace and the requests/limits that apply to the pods and the containers in every namespace. Furthermore, the test also measures how much of the allowed / guaranteed compute resources each namespace is currently utilizing, thus enabling administrators to accurately identify the namespaces that are currently experiencing or may potentially experience a contention for resources. The resource quota of such namespaces may require rework. This way, the test proactively alerts administrators to problem conditions that may be caused by poor resource quota definitions, and prompts them to initiate preventive action.

Target of the test : A Kubernetes Cluster

Agent deploying the test : A remote agent

Outputs of the test : One set of results for each namespace configured in the Kubernetes cluster being monitored

Configurable parameters for the test

Parameter	Description
Test Period	How often should the test be executed.
Host	The IP address of the host for which this test is to be configured.
Port	Specify the port at which the specified Host listens. By default, this is *6443*.
Load Balancer / Master Node IP	To run this test and report metrics, the eG agent needs to connect to the Kubernetes API on the master node and run API commands. To enable this connection, the eG agent has to be configured with either of the following: • If only a single master node exists in the cluster, then configure the eG agent with the IP address of the master node. • If the target cluster consists of more than one master node, then you need to configure the eG agent with the IP address of the load balancer that is managing the cluster. In this case, the load balancer will route the eG agent's connection request to any available master node in the cluster, thus enabling the agent to connect with the API server on that node, run API commands on it, and pull metrics. By default, this parameter will display the **Load Balancer / Master Node IP** that you configured when manually adding the Kubernetes cluster for monitoring, using the **Kubernetes Cluster Preferences** page in the eG admin interface (see Figure 2.3). The steps for managing the cluster using the eG admin interface are discussed elaborately in How to Monitor the Kubernetes Cluster Using eG Enterprise? Whenever the eG agent runs this test, it uses the IP address that is displayed (by default) against this parameter to connect to the Kubernetes API. If there is any change

Parameter	Description
	in this IP address at a later point in time, then make sure that you update this parameter with it, by overriding its default setting.
SSL	By default, the Kubernetes cluster is SSL-enabled. This is why, the eG agent, by default, connects to the Kubernetes API via an HTTPS connection. Accordingly, this flag is set to **Yes** by default. If the cluster is not SSL-enabled in your environment, then set this flag to **No**.
Authentication Token	The eG agent requires an authentication bearer token to access the Kubernetes API, run API commands on the cluster, and pull metrics of interest. The steps for generating this token have been detailed in Section **1.1** Typically, once you generate the token, you can associate that token with the target Kubernetes cluster, when manually adding that cluster for monitoring using the eG admin interface. The steps for managing the cluster using the eG admin interface are discussed elaborately in Section **Chapter 2** By default, this parameter will display the **Authentication Token** that you provided in the **Kubernetes Cluster Preferences page** of the eG admin interface, when manually adding the cluster for monitoring (see Figure 2.3). Whenever the eG agent runs this test, it uses the token that is displayed (by default) against this parameter for accessing the API and pulling metrics. If for any reason, you generate a new authentication token for the target cluster at a later point in time, then make sure you update this parameter with the change. For that, copy the new token and paste it against this parameter.
Proxy Host	If the eG agent connects to the Kubernetes API on the master node via a proxy server, then provide the IP address of the proxy server here. If no proxy is used, then the default setting -*none* - of this parameter, need not be changed,
Proxy Port	If the eG agent connects to the Kubernetes API on the master node via a proxy server, then provide the port number at which that proxy server listens here. If no proxy is used, then the default setting -*none* - of this parameter, need not be changed,
Proxy Username, Proxy Password, Confirm Password	**These parameters are applicable only if the eG agent uses a proxy server to connect to the Kubernetes cluster, and that proxy server requires authentication.** In this case, provide a valid user name and password against the **Proxy Username** and **Proxy Password** parameters, respectively. Then, confirm the password by retyping it in the **Confirm Password** text box. If no proxy server is used, or if the proxy server used does not require authentication, then the default setting - *none* - of these parameters, need not be changed.

Parameter	Description
DD Frequency	Refers to the frequency with which detailed diagnosis measures are to be generated for this test. The default is *3:1*. This indicates that, by default, detailed measures will be generated every third time this test runs, and also every time the test detects a problem. You can modify this frequency, if you so desire. Also, if you intend to disable the detailed diagnosis capability for this test, you can do so by specifying *none* against DD frequency.
Detailed Diagnosis	To make diagnosis more efficient and accurate, the eG Enterprise suite embeds an optional detailed diagnostic capability. With this capability, the eG agents can be configured to run detailed, more elaborate tests as and when specific problems are detected. To enable the detailed diagnosis capability of this test for a particular server, choose the **On** option. To disable the capability, click on the **Off** option. The option to selectively enable/disable the detailed diagnosis capability will be available only if the following conditions are fulfilled: • The eG manager license should allow the detailed diagnosis capability • Both the normal and abnormal frequencies configured for the detailed diagnosis measures should not be 0.

Measurements made by the test

<table>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
<tr><td>Status</td><td>Indicates the current status of this namespace</td><td></td><td>The values that this measure reports and their corresponding numeric values are detailed in the table below:
<table>
<tr><th>Measure Value</th><th>Numeric Value</th></tr>
<tr><td>Active</td><td>1</td></tr>
<tr><td>Terminating</td><td>0</td></tr>
</table>
<b>Note:</b>
By default, this test reports the <b>Measure Value</b>s listed in the table above to indicate the state of a namespace. In the graph of this measure however, the state is indicated using the numeric equivalents only.</td></tr>
</table>

Measurement	Description	Measurement Unit	Interpretation
			Use the detailed diagnosis of this measure to view the labels that have been configured for the objects in an namespace. Labels are key/value pairs that are attached to objects, such as pods. Labels are intended to be used to specify identifying attributes of objects that are meaningful and relevant to users, but do not directly imply semantics to the core system. Labels can be used to organize and to select subsets of objects. Labels can be attached to objects at creation time and subsequently added and modified at any time. Each object can have a set of key/value labels defined. Example of labels: "release" : "stable", "release" : "canary"
Age	Indicates how old the namespace is		The value of this measure is expressed in number of days, hours, and minutes.
Total pods	Indicates the number of pods in this namespace.	Number	A Pod is the basic execution unit of a Kubernetes application–the smallest and simplest unit in the Kubernetes object model that you create or deploy. A Pod encapsulates an application’s container (or, in some cases, multiple containers), storage resources, a unique network IP, and options that govern how the container (s) should run. A Pod represents a unit of deployment: a single instance of an application in Kubernetes, which might consist of either a single container or a small number of containers that are tightly coupled and that share resources. To know which are the pods in a namespace, use the detailed diagnosis of this measure. If the resource quota enforced on a

Measurement	Description	Measurement Unit	Interpretation
			namespace restricts the number of pods you can create in a namespace, then you can use this measure to ascertain how much of that quota is being used currently, and how many more pods you can create before the quota is fully exhausted. If the node on which the existing pods are running has the resource capacity to support more pods, you may want to change your quota accordingly.
Total services	Indicates the number of services in this namespace.	Number	In Kubernetes, a Service is an abstraction which defines a logical set of Pods and a policy by which to access them (sometimes this pattern is called a micro-service). To know which services are in a namespace, use the detailed diagnosis of this measure. If the resource quota enforced on a namespace restricts the number of services you can create in a namespace, then you can use this measure to ascertain how much of that quota is being used currently, and how many more services you can create before the quota is fully exhausted.
Maximum CPU limits in container	Indicates the CPU resource limit set in the Limit Range for containers in this namespace.	Millicpu	**These measures will be reported only if a Limit Range has been configured and enabled for the containers in a namespace, and CPU limits/requests have been configured in that Limit Range.** Typically, to limit consumption by individual containers in a namespace, a Limit Range specification is used. If a Limit Range is enforced on the containers in a namespace, then the

Measurement	Description	Measurement Unit	Interpretation
			resource consuming capacity of each container in that namespace will be determined by the min/max limits defined within that Limit Range. In this case therefore, the max and min limit settings in the Limit Range will be reported as values of these measures, respectively. Moreover, in this case, Kubernetes will automatically foil any attempt to create/update a container, if that operation ends up violating the limit set in the Limit Range. For instance, say that the Limit Range specification enforced on a namespace rules that no single container in that namespace should consume CPU over 800m (max limit) or lesser than 100m (min limit). In this case, if an attempt is made to create a single container with a CPU limit of 900m, then Kubernetes will automatically foil that attempt. This is because, that container violates the max limit of 800m that is set per container in the enforced Limit Range. The container creation fails, even if that container does not violate the total CPU consumption limit set in the resource quota of that namespace.
Minimum CPU requests in container	Indicates the minimum CPU request limit set in the Limit Range for containers in this namespace.	Millicpu	
Default CPU limits in container	Indicates the default CPU limit defined in the Limit Range for containers in this namespace.	Millicpu	**These measures will be reported only if a if default CPU requests/limits are set in the Limit Range for the containers in a namespace.** Default CPU requests and limits can be set for the containers in a pod, using the Limit Range specification. These default settings apply only when minimum and/or maximum CPU limits are not defined at the individual container-level (during container creation).

Measurement	Description	Measurement Unit	Interpretation
			For instance, if a container being created is not configured with a CPU limit, but is configured with a CPU request, then the default CPU limit configured in the Limit Range will apply to that container. If a container being created is not configured with a CPU request, but is configured with a CPU limit instead, then Kubernetes does not automatically apply the default CPU request configured in the Limit Range to that container. Instead, the limits set for that container during creation are automatically set as its requests. On the other hand, if a container is being created with neither CPU limits nor CPU requests defined, then the default CPU limit and request defined in the Limit Range will automatically apply.
Default CPU requests in container	Indicates the default CPU request setting defined in the Limit Range for containers in this namespace.	Millicpu	
Maximum memory limits in container	Indicates the memory resource limit set in the Limit Range for containers in this namespace,	MB	**These measures will be reported only if a Limit Range has been configured and enabled for the containers in a namespace, and memory limits/requests have been configured in that Limit Range.** Typically, to limit consumption by individual containers in a namespace, a Limit Range specification is used. If a Limit Range is enforced on a namespace, then the resource consuming capacity of each container in that namespace will be determined by the min/max limits defined within that Limit Range. In this case therefore, the max and min memory limit settings in the Limit Range will be reported as values of these measures, respectively. Moreover, in this case, Kubernetes will automatically foil any attempt to create/update a container that

Measurement	Description	Measurement Unit	Interpretation
			violates the limit set in the Limit Range. For instance, say that the Limit Range specification enforced on a namespace rules that no single container in that namespace should consume memory over 500 MB (max limit) or lesser than 100 MB (min limit). In this case, if an attempt is made to create a single container with a memory limit of 800 MB, then Kubernetes will automatically foil that attempt. This is because, that container violates the max limit of 500 MB that is set per container in the enforced Limit Range. The container creation fails, even if that operation does not violate the total memory consumption limit set in the resource quota of that namespace. If the value of the *Maximum memory limits in container* measure reports the value 0, it means that the containers in the namespace have no upper bound on the amount of memory they use. In such situations, the Container could use all of the memory available on the Node where it is running which in turn could invoke the OOM Killer. Further, in case of an OOM Kill, a container with no resource limits will have a greater chance of being killed.
Minimum memory requests in container	Indicates the minimum memory request limit set in the Limit Range for containers in this namespace.	MB	
Default memory limits in container	Indicates the default memory limit defined in the Limit Range for containers in this namespace.	MB	**These measures will be reported only if a if default memory requests/limits are set in the Limit Range enforced for the containers in a namespace.** Default memory requests and limits can be set for the containers in a pod, using the Limit Range specification. These default settings apply only when minimum and/or maximum memory limits are not defined at the individual container-level (during

Measurement	Description	Measurement Unit	Interpretation
			container creation). For instance, if a container being created is not configured with a memory limit, but is configured with a memory request, then the default memory limit configured in the Limit Range will apply to that container. If a container being created is not configured with a memory request, but is configured with a memory limit instead, then Kubernetes does not automatically apply the default memory request configured in the Limit Range to that container. Instead, the limits set for that container during creation are automatically set as its requests. On the other hand, if a container is being created with neither memory limits nor memory requests defined, then the default memory limit and request defined in the Limit Range will automatically apply.
Default memory requests in container	Indicates the default memory request setting defined in the Limit Range for containers in this namespace.	MB	
CPU limit	Indicates the total amount of CPU resources that containers in this namespace are allowed to use, as per the resource quota.	Millicpu	Resource requests/limits set using the ResourceQuota object govern the aggregate resource consumption of a namespace - i.e., the total resources that can be consumed/requested across all pods/containers in a namespace. A resource quota is violated only when the total consumption of a resource, across pods/containers in the namespace, exceeds the limits defined in the resource quota. For instance, say that the resource quota of a namespace enforces a CPU usage limit of 2 cores and a memory usage limit of 500Gi. In this case, Kubernetes will allow you to create 2 containers with a CPU core each and 100Gi of memory each. However,

Measurement	Description	Measurement Unit	Interpretation
			if an attempt is made to create another container configured with 1 CPU core and 200Gi of memory, then such an addition operation will fail. This is because, the addition increases the total CPU usage of the namespace to 3 CPU cores, which violates the 2 core limit set by the resource quota.
CPU requests	Indicates the minimum amount of CPU resources that is guaranteed to the containers in this namespace, as per the resource quota.	Millicpu	
Memory limit	Indicates the total amount of memory resources that containers in this namespace are allowed to use, as per the resource quota.	MB	
Memory requests	Indicates the minimum amount of memory resources that is guaranteed to the containers in this namespace, as per the resource quota.	MB	
Used CPU limits	Indicates the sum of the CPU limits configured for the containers in this namespace.	Millicpu	If a resource quota is enabled for a namespace , you may want to compare the value of this measure with that of the *CPU limits* for that namespace. If this comparison reveals that the value of this measure is equal to or close to that of the *CPU limits* measure, it implies that that namespace has or is about to exhaust its quota of CPU resources. If the node on which the containers are running is resource-thick, you may want to reconfigure the resource quota and increase the aggregate CPU consumption capacity of the namespace, so as to prevent a resource quota violation and consequent throttling of creation/updation operations on

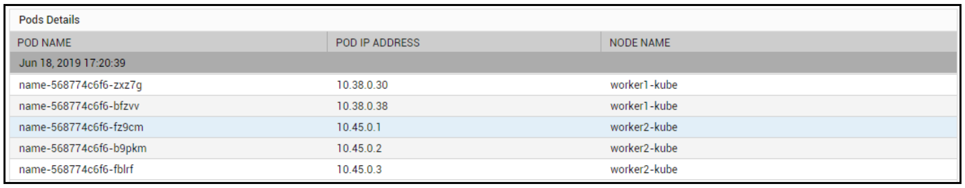
Measurement	Description	Measurement Unit	Interpretation
			the namespace.
Used memory limits	Indicates the sum of the memory limits configured for the containers in this namespace.	MB	If a resource quota is enabled for a namespace, you may want to compare the value of this measure with that of the *Memory limits* measure for that namespace. If this comparison reveals that the value of this measure is equal to or close to that of the *Memory limits* measure, it implies that that namespace has or is about to exhaust its quota of memory resources. If the node on which the containers are running is resource-thick, you may want to reconfigure the resource quota and increase the aggregate memory consumption capacity of the namespace, so as to prevent a resource quota violation and consequent throttling of creation/updation operations on the namespace.
Used requests CPU	Indicates the sum of the CPU requests configured for the containers in this namespace.	Millicpu	If a resource quota is enabled for a namespace, you may want to compare the value of this measure with that of the *CPU requests* measure for that namespace. If this comparison reveals that the value of this measure is equal to or close to that of the *CPU requests* measure, it implies that that namespace is rapidly utilizing the CPU resources guaranteed to it. If the node on which the containers are running is resource-thick, you may want to reconfigure the resource quota and increase the aggregate requests for the namespace, so as to prevent a resource quota violation and consequent throttling of creation/updation operations on the namespace.

Measurement	Description	Measurement Unit	Interpretation
Used requests memory	Indicates the sum of the memory requests configured for the containers in this namespace.	MB	If a resource quota is enabled for a namespace , you may want to compare the value of this measure with that of th *Memory requests* measure for that namespace. If this comparison reveals that the value of this measure is equal to or close to that of the *Memory requests* measure, it implies that that namespace is rapidly utilizing the memory resources guaranteed to it. If the node on which the containers are running is resource-thick, you may want to reconfigure the resource quota and increase the aggregate memory requests for the namespace, so as to prevent a resource quota violation and consequent throttling of creation/updation operations on the namespace.
Maximum CPU limits in pod	Indicates the maximum CPU usage limit set in the Limit Range for pods in this namespace.	Millicpu	**These measures will be reported only if a Limit Range has been configured and enabled for the pods in a namespace, and CPU limits/requests have been configured in that Limit Range.** Typically, to limit consumption by individual pods in a namespace, a Limit Range specification is used. If a Limit Range is enforced on a namespace, then the resource consuming capacity of each pod in that namespace will be determined by the min/max limits defined within that Limit Range. In this case therefore, the max and min limit settings in the Limit Range will be reported as values of these measures, respectively. Moreover, in this case, Kubernetes will automatically foil any attempt to create/update a container, if that operation ends up violating the limit set in the Limit Range for pods in

Measurement	Description	Measurement Unit	Interpretation
			the namespace. For instance, say that the Limit Range specification enforced on a namespace rules that no single pod in that namespace should consume CPU over 800m (max limit) or lesser than 100m (min limit). In this case, if an attempt is made to create a single pod containing containers with a total CPU limit of 900m, then Kubernetes will automatically foil that attempt. This is because, that pods violates the max limit of 800m that is set per pod in the enforced Limit Range. The pod creation fails, even if that pod does not violate the total CPU consumption limit set in the resource quota of that namespace.
Minimum CPU requests in pod	Indicates the minimum amount of CPU resources guaranteed to the pods in this namespace.	Millicpu	
Maximum memory limits in pod	Indicates the maximum memory usage limit set in the Limit Range for pods in this namespace.	MB	**These measures will be reported only if a Limit Range has been configured and enabled for the pods in a namespace, and memory limits/requests have been configured in that Limit Range.** To limit consumption by individual pods in a namespace, a Limit Range specification is used. If a Limit Range is enforced on a namespace, then the resource consuming capacity of each pod in that namespace will be determined by the min/max limits defined within that Limit Range. In this case therefore, the max and min memory limit settings in the Limit Range will be reported as values of these measures, respectively. Moreover, in this case, Kubernetes will automatically foil any attempt to create/update a pod, if that operation may potentially violate the limit set in the Limit Range. For instance, say that the Limit Range specification enforced on a

Measurement	Description	Measurement Unit	Interpretation
			namespace rules that no single pod in that namespace should consume memory over 500 MB (max limit) or lesser than 100 MB (min limit). In this case, if an attempt is made to create a single pod containing containers with a total memory limit of 800 MB, then Kubernetes will automatically foil that attempt. This is because, that pod violates the max limit of 500 MB that is set per pod in the enforced Limit Range. The pod creation fails, even if that operation does not violate the total memory consumption limit set in the resource quota of that namespace. If the value of the *Maximum memory limits* in pod measure reports the value 0, it means that the pods in the namespace have no upper bound on the amount of memory they use. In such situations, the pods could use all of the memory available on the Node where it is running which in turn could invoke the OOM Killer. Further, in case of an OOM Kill, a pod with no resource limits will have a greater chance of being killed.
Minimum memory requests in container	Indicates the minimum amount of memory resources guaranteed to the containers in this namespace.	MB	

The detailed diagnosis of the Total pods measure reveals the names of the pods in the namespace, the IP address of the pods, and the node on which each pod is running.

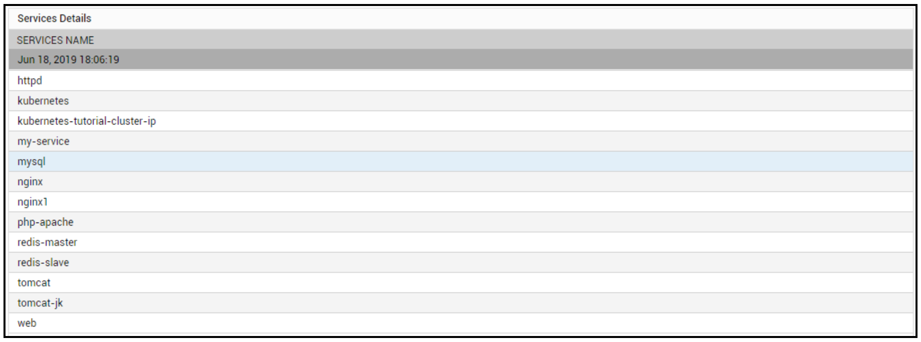


Pods Details

POD NAME	POD IP ADDRESS	NODE NAME
Jun 18, 2019 17:20:39		
name-568774c6f6-zxz7g	10.38.0.30	worker1-kube
name-568774c6f6-bfzvv	10.38.0.38	worker1-kube
name-568774c6f6-fz9cm	10.45.0.1	worker2-kube
name-568774c6f6-b9pkm	10.45.0.2	worker2-kube
name-568774c6f6-fblrf	10.45.0.3	worker2-kube

Figure 3.29: The detailed diagnosis of the Total Pods measure

The detailed diagnosis of the Total services measure reveals the names of the services in the namespace.



Services Details

SERVICES NAME
Jun 18, 2019 18:06:19
httpd
kubernetes
kubernetes-tutorial-cluster-ip
my-service
mysql
nginx
nginx1
php-apache
redis-master
redis-slave
tomcat
tomcat-jk
web

Figure 3.30: The detailed diagnosis of the Total services measure

3.1.9 Kube Persistent Volumes Test

A PersistentVolume (PV) is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using Storage Classes. It is a resource in the cluster just like a node is a cluster resource. PVs are volume plugins like Volumes, but have a lifecycle independent of any individual pod that uses the PV.

A PersistentVolumeClaim (PVC) is a request for storage by a user. It is similar to a pod. Pods consume node resources and PVCs consume PV resources. Pods can request specific levels of resources (CPU and Memory). Claims can request specific size and access modes (e.g., can be mounted once read/write or many times read-only).

Typically, a user creates a PersistentVolumeClaim with a specific amount of storage requested and with certain access modes. A control loop in the master watches for new PVCs, checks if any static PV (a PV manually created by the administrator) matches the new PVC, and binds them together. When none of the static PVs the administrator created matches a user's PVC, the cluster may try to dynamically provision a volume specially for the PVC. If a PV was dynamically provisioned for a new PVC, the loop will always bind that PV to the PVC. Claims will remain unbound indefinitely if a matching volume does not exist. Claims will be bound as matching volumes become available. For example, a cluster provisioned with many 50Gi PVs would not match a PVC requesting 100Gi. The PVC can be bound when a 100Gi PV is added to the cluster.

When a user is done with their volume, they can delete the PVC objects from the API which allows reclamation of the resource.

If there are many unfulfilled PVCs, an administrator may quickly want to check the status of the existing PVs to determine why they could not be bound to any of the PVCs - is it because the PVs are already bound? is it because the PVs have been released, but cannot be reclaimed? or has reclamation failed for many PVs? The Kube Persistent Volumes test provides answers to these questions!

This test auto-discovers PVs and reports the bind status of each PV, thereby pointing administrators to those PVs that are unbound, bound, or released, and those that could not be reclaimed. This way, administrators can figure out if the bind/relcamation status of a PV is why it could not be bound to a PVC. Also, if there are one/more available/unbound PVs, then administrators can use this test to verify the configuration - i.e., the access mode and storage capacity - of such PVs. This will reveal if those PVs are unbound because their configuration does not match any open PVC.

Target of the test : A Kubernetes Cluster

Agent deploying the test : A remote agent

Outputs of the test : One set of results for each Persistent Volume in the Kubernetes cluster being monitored

Configurable parameters for the test

Parameter	Description
Test Period	How often should the test be executed.
Host	The IP address of the host for which this test is to be configured.
Port	Specify the port at which the specified Host listens. By default, this is *6443*.
Load Balancer / Master Node IP	To run this test and report metrics, the eG agent needs to connect to the Kubernetes API on the master node and run API commands. To enable this connection, the eG agent has to be configured with either of the following: • If only a single master node exists in the cluster, then configure the eG agent with the IP address of the master node. • If the target cluster consists of more than one master node, then you need to configure the eG agent with the IP address of the load balancer that is managing the cluster. In this case, the load balancer will route the eG agent's connection request to any available master node in the cluster, thus enabling the agent to connect with the API server on that node, run API commands on it, and pull metrics. By default, this parameter will display the **Load Balancer / Master Node IP** that you configured when manually adding the Kubernetes cluster for monitoring, using the **Kubernetes Cluster Preferences** page in the eG admin interface (see Figure 2.3). The steps for managing the cluster using the eG admin interface are discussed elaborately in How to Monitor the Kubernetes Cluster Using eG Enterprise? Whenever the eG agent runs this test, it uses the IP address that is displayed (by default) against this parameter to connect to the Kubernetes API. If there is any change

Parameter	Description
	in this IP address at a later point in time, then make sure that you update this parameter with it, by overriding its default setting.
SSL	By default, the Kubernetes cluster is SSL-enabled. This is why, the eG agent, by default, connects to the Kubernetes API via an HTTPS connection. Accordingly, this flag is set to **Yes** by default. If the cluster is not SSL-enabled in your environment, then set this flag to **No**.
Authentication Token	The eG agent requires an authentication bearer token to access the Kubernetes API, run API commands on the cluster, and pull metrics of interest. The steps for generating this token have been detailed in Section **1.1** Typically, once you generate the token, you can associate that token with the target Kubernetes cluster, when manually adding that cluster for monitoring using the eG admin interface. The steps for managing the cluster using the eG admin interface are discussed elaborately in Section **Chapter 2** By default, this parameter will display the **Authentication Token** that you provided in the **Kubernetes Cluster Preferences page** of the eG admin interface, when manually adding the cluster for monitoring (see Figure 2.3). Whenever the eG agent runs this test, it uses the token that is displayed (by default) against this parameter for accessing the API and pulling metrics. If for any reason, you generate a new authentication token for the target cluster at a later point in time, then make sure you update this parameter with the change. For that, copy the new token and paste it against this parameter.
Proxy Host	If the eG agent connects to the Kubernetes API on the master node via a proxy server, then provide the IP address of the proxy server here. If no proxy is used, then the default setting -*none* - of this parameter, need not be changed,
Proxy Port	If the eG agent connects to the Kubernetes API on the master node via a proxy server, then provide the port number at which that proxy server listens here. If no proxy is used, then the default setting -*none* - of this parameter, need not be changed,
Proxy Username, Proxy Password, Confirm Password	**These parameters are applicable only if the eG agent uses a proxy server to connect to the Kubernetes cluster, and that proxy server requires authentication.** In this case, provide a valid user name and password against the **Proxy Username** and **Proxy Password** parameters, respectively. Then, confirm the password by retyping it in the **Confirm Password** text box. If no proxy server is used, or if the proxy server used does not require authentication, then the default setting - *none* - of these parameters, need not be changed.

Parameter	Description
DD Frequency	Refers to the frequency with which detailed diagnosis measures are to be generated for this test. The default is *1:1*. This indicates that, by default, detailed measures will be generated every time this test runs, and also every time the test detects a problem. You can modify this frequency, if you so desire. Also, if you intend to disable the detailed diagnosis capability for this test, you can do so by specifying *none* against DD frequency.
Detailed Diagnosis	To make diagnosis more efficient and accurate, the eG Enterprise suite embeds an optional detailed diagnostic capability. With this capability, the eG agents can be configured to run detailed, more elaborate tests as and when specific problems are detected. To enable the detailed diagnosis capability of this test for a particular server, choose the **On** option. To disable the capability, click on the **Off** option. The option to selectively enable/disable the detailed diagnosis capability will be available only if the following conditions are fulfilled: • The eG manager license should allow the detailed diagnosis capability • Both the normal and abnormal frequencies configured for the detailed diagnosis measures should not be 0.

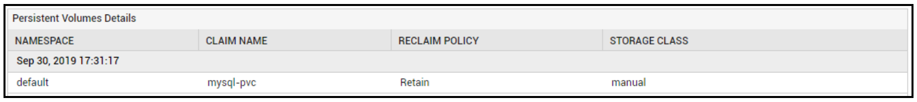
Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Status	Indicates the current status of this PV.		This measure can report any of the following values: • Available: A free resource that is yet bound to a claim • Bound: The volume is bound to a claim • Released: The claim has been deleted, but the resource is not reclaimed by the cluster. This depends upon the reclaim policy of the PV. For instance, if the reclaim policy is Retain, then the cluster will not automatically reclaim the resource once it is released; it can

<table>
<thead>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
</thead>
<tbody>
<tr><td></td><td></td><td></td><td>only be manually reclaimed.
• Failed: The volume has failed its automatic reclamation.
The numeric values that correspond to these measure values are as follows:
<table>
<thead>
<tr><th>Measure Value</th><th>Numeric Value</th></tr>
</thead>
<tbody>
<tr><td>Available</td><td>1</td></tr>
<tr><td>Bound</td><td>2</td></tr>
<tr><td>Released</td><td>3</td></tr>
<tr><td>Failed</td><td>4</td></tr>
</tbody>
</table>
<b>Note:</b>
By default, this test reports the <b>Measure Value</b>s listed in the table above to indicate the state of a PV. In the graph of this measure however, the same is indicated using the numeric equivalents only.
Using the detailed diagnosis of this measure, you can determine the namespace to which a PV belongs, the PVC that binds the PV (in case the PV is Bound), the reclaim policy configured for the PV, and the storage class (in case the PV is dynamically provisioned).</td></tr>
<tr><td>Age</td><td>Indicates how old this PV is.</td><td></td><td>The value of this measure is expressed in number of days, hours, and minutes.</td></tr>
<tr><td>Access modes</td><td>Indicates the access modes configured for this PV.</td><td></td><td>A PersistentVolume can be mounted on a host in any way supported by the resource provider. As shown in the table below, providers will have different capabilities and each PV’s access modes are set to the specific modes supported by that particular volume. For example, NFS can</td></tr>
</tbody>
</table>

<table>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
<tr><td></td><td></td><td></td><td>
<p>support multiple read/write clients, but a specific NFS PV might be exported on the server as read-only. Each PV gets its own set of access modes describing that specific PV’s capabilities.</p>
<p>The access modes are:</p>
<ul>
<li>ReadWriteOnce – the volume can be mounted as read-write by a single node</li>
<li>ReadOnlyMany – the volume can be mounted read-only by many nodes</li>
<li>ReadWriteMany – the volume can be mounted as read-write by many nodes</li>
</ul>
<p>The aforesaid access modes also represent the values that this measure can report. The numeric values that correspond to these measure values are as follows:</p>
<table>
<tr><th>Measure Value</th><th>Numeric Value</th></tr>
<tr><td>ReadOnlyMany</td><td>1</td></tr>
<tr><td>ReadWriteMany</td><td>2</td></tr>
<tr><td>ReadWriteOnce</td><td>3</td></tr>
</table>
<p><b>Note:</b></p>
<p>By default, this test reports the <b>Measure Value</b>s listed in the table above to indicate the access mode of a PV. In the graph of this measure however, the same is indicated using the numeric equivalents only.</p>
</td></tr>
<tr><td>Storage</td><td>Indicates the storage capacity configured for this PV.</td><td>GB</td><td></td></tr>
</table>

The detailed diagnosis of the Status measure reveals the namespace to which a PV belongs, the PVC that binds the PV (in case the PV is Bound), the reclaim policy configured for the PV, and the storage class (in case the PV is dynamically provisioned). If a PV is in the Released (but not reclaimed) state or in the Failed state, then you can use the detailed diagnosis to identify what reclaim policy applies to that PV, so you can easily troubleshoot the failure.



Persistent Volumes Details

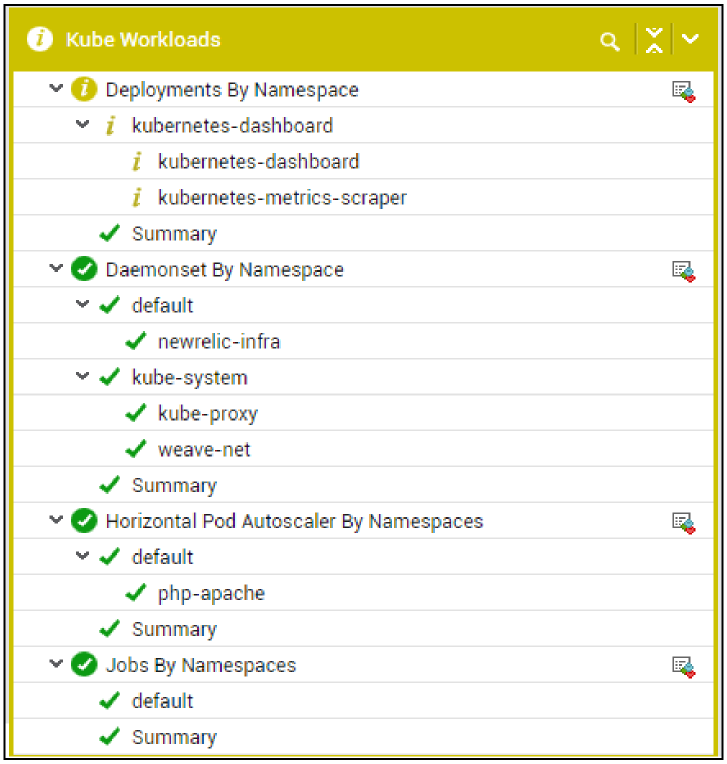
NAMESPACE	CLAIM NAME	RECLAIM POLICY	STORAGE CLASS
Sep 30, 2019 17:31:17			
default	mysql-pvc	Retain	manual

Figure 3.31: The detailed diagnosis of the Status measure reported by the Kube Persistent Volumes test

3.1.10 The Kube Workloads Layer

With the help of the tests mapped to this layer, you can:

- Pinpoint Pods that are in a Failed or Pending state, and those that are over-subscribing to a node's capacity;
- Spot Deployments where the actual state does not match the desired state;
- Identify Daemonsets that are running where they should not be and those that are not running where they should be;
- Promptly detect scaling issues experienced by a Horizonal Pod Autoscaler;
- Quickly capture failed jobs;



Kube Workloads

- Deployments By Namespace
  - kubernetes-dashboard
    - kubernetes-dashboard
    - kubernetes-metrics-scraper
  - ✓ Summary
- ✓ Daemonset By Namespace
  - ✓ default
    - ✓ newrelic-infra
  - ✓ kube-system
    - ✓ kube-proxy
    - ✓ weave-net
  - ✓ Summary
- ✓ Horizontal Pod Autoscaler By Namespaces
  - ✓ default
    - ✓ php-apache
  - ✓ Summary
- ✓ Jobs By Namespaces
  - ✓ default
  - ✓ Summary

Figure 3.32: The tests mapped to the Kube Workloads layer

3.1.11 Pods by Namespace Test

Pods are the smallest deployable uni ts of computing that can be created and managed in Kubernetes. A Pod (as in a pod of whales or pea pod) is a group of one or more containers (such as Docker containers), with shared storage/network, and a specification for how to run the containers. A Pod's contents are always co-located and co-scheduled, and run in a shared context.

Pods are created, assigned a unique ID (UID), and scheduled to nodes where they remain until termination (according to restart policy) or deletion. If a Node dies, the Pods scheduled to that node are scheduled for deletion, after a timeout period. At any given point in time, an administrator needs to know at which phase a Pod is in its life cycle, so they can promptly detect Pod failures or undue slowness in Pod creation and rapidly initiate investigations into the same. This is necessary because, if a Pod fails, then the cluster's actual state may go out of sync with its desired state.

Once a Pod is assigned to a node by scheduler, kubelet starts creating containers using container runtime. Alongside status of Pods, an administrator also needs to keep track of the status of containers at all times, as container failures impact the availability and performance of the containerized applications. This way, administrators can detect and resolve issues in containerized applications before end-users notice.

Typically, when Pods run containers, they use the CPU and memory resources on the node to which they are scheduled. By default, a Pod in Kubernetes will run with no limits on CPU and memory. This means that a single Pod can end up hogging the resources of the node! To avoid this, administrators can control the amount of CPU and memory resources each container in a Pod can use by setting resource requests and limits in the Pod configuration file. A Pod can use as much compute resources as represented by the sum of requests and limits of all containers in that Pod. This means that if the per container limits are not prudently set, then you could have Pods that over-subscribe to the node's capacity. Also, if containers are not sized according to their actual usage, then it can adversely impact the performance of the containerized applications. This is why, it is imperative that administrators track the actual resource usage of Pods, proactively detect potential resource contentions, and tweak usage limits and/or priorities to prevent such contentions. The Pods by Namespace test helps administrators perform all of the above!

This test auto-discovers the Pods in each Namespace, and reports the status of each Pod and that of the containers in every Pod. This leads administrators to Pods and containers in an abnormal state. Additionally, the test reports the resource requests and limits for each Pod, the resource capacity of the Node to which each Pod is scheduled, and actual resource utilization. In the process, the test accurately pinpoints those Pods that are over-subscribing to the node's capacity and those Pods that may potentially cause a contention for resources on the node. Since the test also reveals the QoS priority setting of each Pod, administrators can also figure out if a change in priority can help prevent probable resource contentions/overcommitment.

Target of the test : A Kubernetes Cluster

Agent deploying the test : A remote agent

Outputs of the test : One set of results for each Pod in every namespace in the Kubernetes cluster being monitored

Configurable parameters for the test

Parameter	Description
Test Period	How often should the test be executed.
Host	The IP address of the host for which this test is to be configured.
Port	Specify the port at which the specified Host listens. By default, this is *6443*.
Load Balancer / Master Node IP	To run this test and report metrics, the eG agent needs to connect to the Kubernetes API on the master node and run API commands. To enable this connection, the eG agent has to be configured with either of the following: • If only a single master node exists in the cluster, then configure the eG agent with

Parameter	Description
	the IP address of the master node. - If the target cluster consists of more than one master node, then you need to configure the eG agent with the IP address of the load balancer that is managing the cluster. In this case, the load balancer will route the eG agent's connection request to any available master node in the cluster, thus enabling the agent to connect with the API server on that node, run API commands on it, and pull metrics. By default, this parameter will display the **Load Balancer / Master Node IP** that you configured when manually adding the Kubernetes cluster for monitoring, using the **Kubernetes Cluster Preferences** page in the eG admin interface (see Figure 2.3). The steps for managing the cluster using the eG admin interface are discussed elaborately in How to Monitor the Kubernetes Cluster Using eG Enterprise? Whenever the eG agent runs this test, it uses the IP address that is displayed (by default) against this parameter to connect to the Kubernetes API. If there is any change in this IP address at a later point in time, then make sure that you update this parameter with it, by overriding its default setting.
SSL	By default, the Kubernetes cluster is SSL-enabled. This is why, the eG agent, by default, connects to the Kubernetes API via an HTTPS connection. Accordingly, this flag is set to **Yes** by default. If the cluster is not SSL-enabled in your environment, then set this flag to **No**.
Authentication Token	The eG agent requires an authentication bearer token to access the Kubernetes API, run API commands on the cluster, and pull metrics of interest. The steps for generating this token have been detailed in Section **1.1** Typically, once you generate the token, you can associate that token with the target Kubernetes cluster, when manually adding that cluster for monitoring using the eG admin interface. The steps for managing the cluster using the eG admin interface are discussed elaborately in Section **Chapter 2** By default, this parameter will display the **Authentication Token** that you provided in the **Kubernetes Cluster Preferences page** of the eG admin interface, when manually adding the cluster for monitoring (see Figure 2.3). Whenever the eG agent runs this test, it uses the token that is displayed (by default) against this parameter only accessing the API and pulling metrics. If for any reason, you generate a new authentication token for the target cluster at a later point in time, then make sure you update this parameter with the change. For that, copy the new token and paste it against this parameter.

Parameter	Description
Report System Namespace	The kube-system namespace consists of all objects created by the Kubernetes system. Monitoring such a namespace may not only increase the eG agent's processing overheads, but may also clutter the eG database. Therefore, to optimize agent performance and to conserve database space, this test, by default, excludes the kube-system namespace from monitoring. Accordingly, this flag is set to **No** by default. If required, you can set this flag to **Yes**, and enable monitoring of the kube-system namespace.
Proxy Host	If the eG agent connects to the Kubernetes API on the master node via a proxy server, then provide the IP address of the proxy server here. If no proxy is used, then the default setting -*none* - of this parameter, need not be changed,
Proxy Port	If the eG agent connects to the Kubernetes API on the master node via a proxy server, then provide the port number at which that proxy server listens here. If no proxy is used, then the default setting -*none* - of this parameter, need not be changed,
Proxy Username, Proxy Password, Confirm Password	**These parameters are applicable only if the eG agent uses a proxy server to connect to the Kubernetes cluster, and that proxy server requires authentication.** In this case, provide a valid user name and password against the **Proxy Username** and **Proxy Password** parameters, respectively. Then, confirm the password by retyping it in the **Confirm Password** text box. If no proxy server is used, or if the proxy server used does not require authentication, then the default setting - *none* - of these parameters, need not be changed.
DD Frequency	Refers to the frequency with which detailed diagnosis measures are to be generated for this test. The default is *1:1*. This indicates that, by default, detailed measures will be generated every time this test runs, and also every time the test detects a problem. You can modify this frequency, if you so desire. Also, if you intend to disable the detailed diagnosis capability for this test, you can do so by specifying *none* against DD frequency.
Detailed Diagnosis	To make diagnosis more efficient and accurate, the eG Enterprise suite embeds an optional detailed diagnostic capability. With this capability, the eG agents can be configured to run detailed, more elaborate tests as and when specific problems are detected. To enable the detailed diagnosis capability of this test for a particular server, choose the **On** option. To disable the capability, click on the **Off** option. The option to selectively enable/disable the detailed diagnosis capability will be available only if the following conditions are fulfilled: • The eG manager license should allow the detailed diagnosis capability

Parameter	Description
	• Both the normal and abnormal frequencies configured for the detailed diagnosis measures should not be 0.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Status	Indicates where this Pod is in its lifecycle.		A Pod can be in one of the following phases in its lifecycle: • Pending: The Pod has been accepted by the Kubernetes system, but one or more of the Container images has not been created. This includes time before being scheduled as well as time spent downloading images over the network, which could take a while. • Running: The Pod has been bound to a node, and all of the Containers have been created. At least one Container is still running, or is in the process of starting or restarting. • Succeeded: All Containers in the Pod have terminated in success, and will not be restarted. • Failed: All Containers in the Pod have terminated, and at least one Container has terminated in failure. That is, the Container either exited with non-zero status or was terminated by the system. • Unknown: For some reason the state of the Pod could not be obtained, typically

<table>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
<tr><td></td><td></td><td></td><td>due to an error in communicating with the host of the Pod.
<ul><li>CrashLoopBackoff: A Pod is starting, crashing, starting again, and then crashing again.</li></ul>
The numeric values that correspond to this are detailed in the table below:
<table>
<tr><th>Measure Value</th><th>Numeric Value</th></tr>
<tr><td>Running</td><td>1</td></tr>
<tr><td>Succeeded</td><td>2</td></tr>
<tr><td>Completed</td><td>3</td></tr>
<tr><td>Failed</td><td>4</td></tr>
<tr><td>Pending</td><td>5</td></tr>
<tr><td>CrashLoopBackOff</td><td>6</td></tr>
<tr><td>Unknown</td><td>7</td></tr>
</table>
<b>Note:</b>
By default, this test reports the <b>Measure Value</b>s listed in the table above to indicate the state of a Pod. In the graph of this measure however, the state is indicated using the numeric equivalents only.
Use the detailed diagnosis of this measure to know which containers are in the Pod, the images used by the containers, and the reason for the status.</td></tr>
<tr><td>Age</td><td>Indicates how old this Pod is.</td><td></td><td>The value of this measure is expressed in number of days, hours, and minutes.
Use the detailed diagnosis of this measure to know which node a Pod is scheduled to, the IP address of the Pod, and the images used by the containers in the Pod.</td></tr>
</table>

Measurement	Description	Measurement Unit	Interpretation
Termination grace period	Shows the optional duration in seconds the Pod needs to terminate gracefully.	Seconds	Because Pods represent running processes on nodes in the cluster, it is important to allow those processes to gracefully terminate when they are no longer needed (vs being violently killed with a KILL signal and having no chance to clean up). Users should be able to request deletion and know when processes terminate, but also be able to ensure that deletes eventually complete. When a user requests deletion of a Pod, the system records the intended grace period before the Pod is allowed to be forcefully killed, and a TERM signal is sent to the main process in each container. Once the grace period has expired, the KILL signal is sent to those processes, and the Pod is then deleted from the API server. The default grace period is 30 seconds. The *kubectl delete* command supports the *–grace-period=<seconds>* option which allows a user to override the default and specify their own value. The value 0 force deletes the Pod. You must specify an additional flag *--force* along with *--grace-period=0* in order to perform force deletions.
Quality of service	Indicates the Quality of Service (QOS) classification assigned to this Pod based on resource requirement.		Kubernetes provides different levels of Quality of Service to pods depending on what they request and what limits are set for them. Pods that need to stay up and consistently good can request guaranteed resources, while pods with less exacting requirements can use resources with less/no guarantee. For each resource, Kubernetes divide Pods into 3 QoS classes: Guaranteed, Burstable, and Best-Effort, in decreasing order of priority.

Measurement	Description	Measurement Unit	Interpretation
			• Guaranteed: Pods are considered top-priority and are guaranteed to not be killed until they exceed their limits. If limits and optionally requests (not equal to 0) are set for all resources across all containers and they are equal, then the pod is classified as Guaranteed. • Burstable: Pods have some form of minimal resource guarantee, but can use more resources when available. Under system memory pressure, these containers are more likely to be killed once they exceed their requests and no Best-Effort pods exist. If requests and optionally limits are set (not equal to 0) for one or more resources across one or more containers, and they are not equal, then the pod is classified as Burstable. • Best-Effort: Pods will be treated as lowest priority. Processes in these pods are the first to get killed if the system runs out of memory. These containers can use any amount of free memory in the node though. If requests and limits are not set for all of the resources, across all containers, then the pod is classified as Best-Effort. This test reports one of the above 3 QOS classes as the value of this measure. The numeric values that correspond to these measure values are as follows:

<table>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
<tr><td></td><td></td><td></td><td><table><tr><th>Measure Value</th><th>Numeric Value</th></tr><tr><td>Guaranteed</td><td>1</td></tr><tr><td>Burstable</td><td>2</td></tr><tr><td>Best Effort</td><td>3</td></tr></table>
<b>Note:</b>
By default, this test reports the <b>Measure Value</b>s listed in the table above to indicate the QOS class of a Pod. In the graph of this measure however, the same is indicated using the numeric equivalents only.</td></tr>
<tr><td>Restart policy</td><td>Indicates the restart policy of all containers within this Pod.</td><td></td><td>This measure reports one of the following values:
• Always: This means that the container will be restarted even if it exited with a zero exit code (i.e. successfully). This is useful when you do not care why the container exited, you just want to make sure that it is always running (e.g. a web server). This is the default.
• OnFailure: This means that the container will only be restarted if it exited with a non-zero exit code (i.e. something went wrong). This is useful when you want accomplish a certain task with the pod, and ensure that it completes successfully - if it does not it will be restarted until it does.
• Never: This means that the container will not be restarted regardless of why it exited.</td></tr>
</table>

<table>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
<tr><td></td><td></td><td></td><td>The numeric values that correspond to these measure values are as follows:
<table><tr><th>Measure Value</th><th>Numeric Value</th></tr><tr><td>Always</td><td>1</td></tr><tr><td>OnFailure</td><td>2</td></tr><tr><td>Never</td><td>3</td></tr></table>
Note:
By default, this test reports the **Measure Value**s listed in the table above to indicate the restart policy of the containers in a Pod. In the graph of this measure however, the same is indicated using the numeric equivalents only.</td></tr>
<tr><td>Are all init containers initialized?</td><td>Indicates whether/not the init containers (if any) in this Pod have started successfully.</td><td></td><td>Init containers are specialized containers that run before app containers in a Pod. Init containers can contain utilities or setup scripts not present in an app image.
The values that this measure reports and their corresponding numeric values are detailed in the table below:
<table><tr><th>Measure Value</th><th>Numeric Value</th></tr><tr><td>Yes</td><td>1</td></tr><tr><td>No</td><td>0</td></tr><tr><td>Unknown</td><td>2</td></tr></table>
Note:
By default, this test reports the **Measure Value**s listed in the table above to indicate the status of Init containers. In the graph of this measure however, the same is indicated using the numeric equivalents only.</td></tr>
</table>

<table>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
<tr><td></td><td></td><td></td><td>If this measure reports the value <i>No</i> or <i>Unknown</i> for a Pod, then you can use the detailed diagnosis of this measure to figure out the reason for the same.</td></tr>
<tr><td>Is Pod ready?</td><td>Indicates whether/not this Pod is ready.</td><td></td><td>If a Pod is in the Ready state, it means that the Pod is able to serve requests and should be added to the load balancing pools of all matching Services.
The values that this measure reports and their corresponding numeric values are detailed in the table below:
<table><tr><th>Measure Value</th><th>Numeric Value</th></tr><tr><td>Yes</td><td>1</td></tr><tr><td>No</td><td>0</td></tr><tr><td>Unknown</td><td>2</td></tr></table>
<b>Note:</b>
By default, this test reports the <b>Measure Value</b>s listed in the table above to indicate the Ready state of a Pod. In the graph of this measure however, the same is indicated using the numeric equivalents only.</td></tr>
<tr><td>Are all containers ready?</td><td>Indicates whether/not all containers in this Pod are ready.</td><td></td><td>If a container is in the Ready state, it means that the container is ready to service requests.
The values that this measure reports and their corresponding numeric values are detailed in the table below:
<table><tr><th>Measure Value</th><th>Numeric Value</th></tr><tr><td>Yes</td><td>1</td></tr><tr><td>No</td><td>0</td></tr><tr><td>Unknown</td><td>2</td></tr></table></td></tr>
</table>

<table>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
<tr><td></td><td></td><td></td><td><b>Note:</b>
By default, this test reports the <b>Measure Value</b>s listed in the table above to indicate whether/not the containers in a Pod are ready. In the graph of this measure however, the same is indicated using the numeric equivalents only.</td></tr>
<tr><td>Is pod scheduled?</td><td>Indicates whether/not this Pod has been scheduled to a node.</td><td></td><td>The values that this measure reports and their corresponding numeric values are detailed in the table below:
<table>
<tr><th>Measure Value</th><th>Numeric Value</th></tr>
<tr><td>Yes</td><td>1</td></tr>
<tr><td>No</td><td>0</td></tr>
<tr><td>Unknown</td><td>2</td></tr>
</table>
<b>Note:</b>
By default, this test reports the <b>Measure Value</b>s listed in the table above to indicate whether/not a Pod has been scheduled to a node. In the graph of this measure however, the same is indicated using the numeric equivalents only.
If this measure reports the value <i>No</i> for a Pod - i.e., if a Pod is not scheduled to a node - then you can use the detailed diagnosis of this measure to figure out the reason for the anomaly.</td></tr>
<tr><td>Total containers</td><td>Indicates the count of containers in this Pod.</td><td>Number</td><td></td></tr>
<tr><td>Volumes mounted</td><td>Indicates the count of volumes mounted in this Pod.</td><td>Number</td><td></td></tr>
<tr><td>Init containers</td><td>Indicates the total number of init</td><td>Number</td><td>Init containers are specialized containers that run before app containers in a Pod. Init</td></tr>
</table>

Measurement	Description	Measurement Unit	Interpretation
	containers (if any) in this Pod.		containers can contain utilities or setup scripts not present in an app image.
Priority	Indicates the priority class assigned to this Pod.		You can assign pods a priority class, which is a non-namespaced object that defines a mapping from a name to the integer value of the priority. The higher the value, the higher the priority. A priority class object can take any 32-bit integer value smaller than or equal to 1000000000 (one billion). Reserve numbers larger than one billion for critical pods that should not be preempted or evicted. There are two reserved priority classes for for critical system pods to have guaranteed scheduling. • System-node-critical: This priority class has a value of 2000001000 and is used for all pods that should never be evicted from a node. • System-cluster-critical: This priority class has a value of 2000000000 (two billion) and is used with pods that are important for the cluster. Pods with this priority class can be evicted from a node in certain circumstances. For example, pods configured with the system-node-critical priority class can take priority. However, this priority class does ensure guaranteed scheduling. This test reports one of the above two priority classes as the value of this measure. The numeric values that correspond to these measure values are as

<table>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
<tr><td></td><td></td><td></td><td>follows:
<table><tr><th>Measure Value</th><th>Numeric Value</th></tr><tr><td>System-cluster-critical</td><td>1</td></tr><tr><td>System-node-critical</td><td>2</td></tr></table>
<b>Note:</b>
By default, this test reports the <b>Measure Value</b>s listed in the table above to indicate the priority class assigned to a Pod. In the graph of this measure however, the same is indicated using the numeric equivalents only.</td></tr>
<tr><td>Running containers</td><td>Indicates the count of running containers in this Pod.</td><td>Number</td><td>If a container is in the Running state, it indicates that the container is executing without any issues.
Use the detailed diagnosis of this measure to know which containers in a Pod are in the Running state.</td></tr>
<tr><td>Terminated containers</td><td>Indicates the count of containers in this Pod that are in a Terminated state.</td><td>Number</td><td>If a container is in the Terminated state, it means that the container completed its execution and has stopped running. A container enters into this when it has successfully completed execution or when it has failed for some reason.
If the containers in a Pod entered this state because they have failed, then use the detailed diagnosis of this measure to know which are those containers, why the failure occurred, and the exit code.</td></tr>
<tr><td>Waiting containers</td><td>Indicates the count of containers in this Pod that are in a Waiting state.</td><td>Number</td><td>Waiting state is the default state of a container. If container is not in either Running or Terminated state, it is in Waiting state. A container in Waiting state still runs its required operations, like pulling images, applying Secrets, etc.</td></tr>
</table>

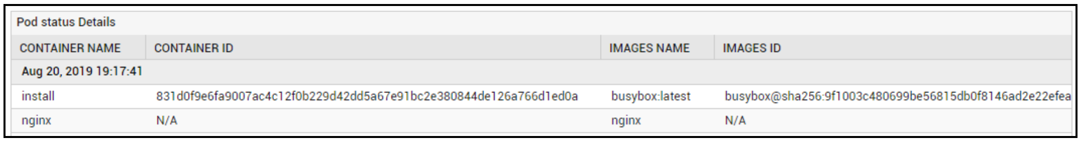
Measurement	Description	Measurement Unit	Interpretation
			Use the detailed diagnosis of this measure to know which containers are in the Waiting state and why.
Uptime of pods	Indicates the total time for which the containers in this Pod were up and running.	Seconds	
Number of times container has been restarted	Indicates the number of times the containers in this Pod have been restarted.	Number	Use the detailed diagnosis of this measure to identify the containers that were restarted and to determine the number of times each container was restarted. Frequently restarted containers can thus be isolated.
CPU requests	Indicates the minimum CPU resources guaranteed to this Pod.	Millicpu	This is the sum of CPU requests configured for all containers in a Pod. A request is the amount of that resource that the system will guarantee to the Pod.
CPU limits	Indicates that maximum amount of CPU resources that this Pod can use.	Millicpu	This is the sum of CPU limits set for all containers in a Pod. A limit is the maximum amount that the system will allow the Pod to use.
Total CPUs on node	Indicates the total number of CPU cores available to the node to which this Pod is scheduled.	Number	
CPU capacity on node	Indicates the CPU capacity of the node to which this Pod is scheduled.	Millicpu	
CPU limits allocation	Indicates what percentage of the capacity of the node is allocated as CPU limits to containers in this Pod. In other words,	Percent	The formula used for computing this measure is as follows: *(CPU limits/CPU capacity on node)*100* If the value of this measure exceeds 100%, it means that the Pod is oversubscribing to

Measurement	Description	Measurement Unit	Interpretation
	this is the percentage of a node's CPU capacity that the containers on this Pod are allowed to use.		the node's capacity. In other words, it means that the Pod has been allowed to use more resources than the node's capacity.
CPU requests allocation	Indicates what percentage of the total CPU capacity of the node is set as CPU requests for the containers on this Pod. In other words, this is the percentage of a node's CPU capacity that the containers on this Pod are guaranteed to receive.	Percent	The formula used for computing this measure is as follows: *(CPU requests/CPU capacity on node)*100* Compare the value of this measure across Pods to know which Pod has been guaranteed the maximum CPU resources.
CPU usage	Indicates the amount of CPU resources used by this Pod.	Millicpu	Ideally, the value of this measure should be much lesser than the value of the *CPU capacity on node* measure. If the value of this measure is equal to or is rapidly approaching the value of the *CPU capacity on node* measure, it means that the Pod is over-utilizing the CPU resources of the node.
CPU utilization	Indicates the percentage of CPU resources utilized by this Pod.	Percent	A value close to 100% is indicative of excessive CPU usage by a Pod, and hints at a potential CPU contention on the node. A value greater than 100% implies that the Pod has probably over-subscribed to the node's capacity.
Containers without CPU limits set	Indicates the number of containers in this Pod for which CPU limits are not set.	Number	If limit is not set, then if defaults to 0 (unbounded)
Containers without CPU requests set	Indicates the number of containers in this Pod	Number	In the case that request is not set for a container, it defaults to limit.

Measurement	Description	Measurement Unit	Interpretation
	for which CPU requests are not set.		
Memory requests	Indicates the minimum memory resources guaranteed to this Pod.	GB	This is the sum of memory requests configured for all containers in a Pod. A request is the amount of that resource that the system will guarantee to the Pod.
Memory limits	Indicates the maximum amount of memory resources that this Pod can use.	GB	This is the sum of memory limits set for all containers in a Pod. A limit is the maximum amount that the system will allow the Pod to use.
Memory capacity on node	Indicates the memory capacity of the node to which this Pod is scheduled.	GB	
Memory limits allocation	Indicates what percentage of the memory capacity of the node is allocated as memory limits to containers in this Pod. In other words, this is the percentage of a node's memory capacity that the containers on this Pod are allowed to use.	Percent	The formula used for computing this measure is as follows: *(Memory limits/Memory capacity on node)*100* If the value of this measure exceeds 100%, it means that the Pod is oversubscribing to the node's capacity. In other words, it means that the Pod has been allowed to use more resources than the node's capacity.
Memory requests allocation	Indicates what percentage of the total memory capacity of the node is set as memory requests for the containers on this Pod. In other words, this is the percentage of a node's memory capacity that the	Percent	The formula used for computing this measure is as follows: *(Memory requests/Memory capacity on node)*100* Compare the value of this measure across Pods to know which Pod has been guaranteed the maximum memory resources.

Measurement	Description	Measurement Unit	Interpretation
	containers on this Pod are guaranteed to receive.		
Memory usage	Indicates the amount of memory resources used by this Pod.	GB	Ideally, the value of this measure should be much lesser than the value of the *Memory capacity on node* measure. If the value of this measure is equal to or is rapidly approaching the value of the *Memory capacity on node* measure, it means that the Pod is over-utilizing the memory resources of the node.
Memory utilization	Indicates the percentage of memory resources utilized by this Pod.	Percent	A value close to 100% is indicative of excessive memory usage by a Pod, and hints at a potential memory contention on the node. A value greater than 100% implies that the Pod has probably over-subscribed to the node's capacity.
Containers without memory limits set	Indicates the number of containers in this Pod for which memory limits are not set.	Number	If limit is not set, then it defaults to 0 (unbounded)
Containers without memory requests set	Indicates the number of containers in this Pod for which memory requests are not set.	Number	In the case that request is not set for a container, it defaults to limit.

The detailed diagnosis of the *Status* measure reveals which containers are in the Pod, the images used by the containers, and the reason for the status.

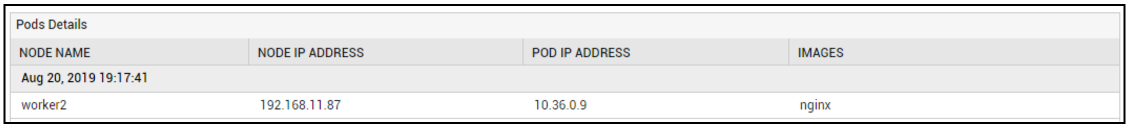


Pod status Details

CONTAINER NAME	CONTAINER ID	IMAGES NAME	IMAGES ID
Aug 20, 2019 19:17:41			
install	831d0f9e6fa9007ac4c12f0b229d42dd5a67e91bc2e380844de126a766d1ed0a	busybox:latest	busybox@sha256:9f1003c480699be56815db0f8146ad2e22efea
nginx	N/A	nginx	N/A

Figure 3.33: The detailed diagnosis of the Status measure reported by the Pods by Namespace test

The detailed diagnosis of the *Age* measure reveals which node a Pod is scheduled to, the IP address of the Pod, and the images used by the containers in the Pod.

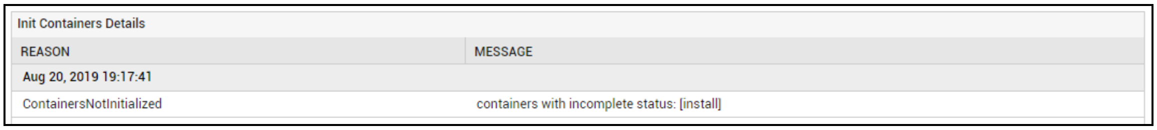


Pods Details

NODE NAME	NODE IP ADDRESS	POD IP ADDRESS	IMAGES
Aug 20, 2019 19:17:41			
worker2	192.168.11.87	10.36.0.9	nginx

Figure 3.34: The detailed diagnosis of the Age measure reported by the Pods by Namespace test

If the *Are all init containers initialized?* measure reports the value *Yes* or *Unknown*, then you can use the detailed diagnosis of this measure to figure out the reason why the init containers failed to initialize.

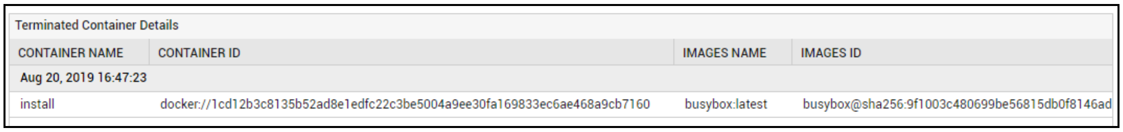


Init Containers Details

REASON	MESSAGE
Aug 20, 2019 19:17:41	
ContainersNotInitialized	containers with incomplete status: [install]

Figure 3.35: The detailed diagnosis of the Are all init containers initialized? measure

If the containers in a Pod entered the Terminated state, then use the detailed diagnosis of the *Terminated containers* measure to know which are those containers, why the failure occurred, and the exit code.

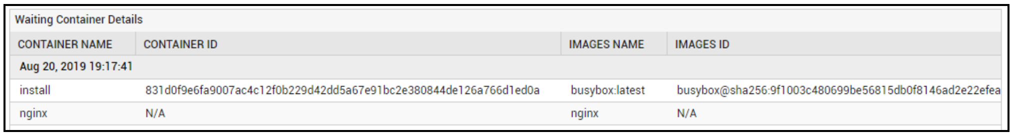


Terminated Container Details

CONTAINER NAME	CONTAINER ID	IMAGES NAME	IMAGES ID
Aug 20, 2019 16:47:23			
install	docker://1cd12b3c8135b52ad8e1edfc22c3be5004a9ee30fa169833ec6ae468a9cb7160	busybox:latest	busybox@sha256:9f1003c480699be56815db0f8146ad

Figure 3.36: The detailed diagnosis of the Terminated containers measure

Use the detailed diagnosis of the *Waiting containers* measure to know which containers are in the Waiting state and why.



Waiting Container Details

CONTAINER NAME	CONTAINER ID	IMAGES NAME	IMAGES ID
Aug 20, 2019 19:17:41			
install	831d0f9e6fa9007ac4c12f0b229d42dd5a67e91bc2e380844de126a766d1ed0a	busybox:latest	busybox@sha256:9f1003c480699be56815db0f8146ad2e22efea
nginx	N/A	nginx	N/A

Figure 3.37: The detailed diagnosis of the Waiting containers measure

3.1.12 Deployments by Namespace Test

A Deployment provides declarative updates for Pods and ReplicaSets. While a Pod encapsulates an application's container (or, in some cases, multiple containers), storage resources, a unique network IP, and options that govern how the container(s) should run, a ReplicaSet is used to maintain a stable set of replica (identical) Pods running at any given time.

Using a Deployment, you can easily:

- Deploy a ReplicaSet
- Update Pods (PodTemplateSpec)
- Rollback to older Deployment versions
- Scale Deployment up or down
- Pause and resume the Deployment
- Determine state of replicas
- Clean up older ReplicaSets
- Canary Deployment

Whenever a Deployment is used to perform such operations, it is only natural that administrators want to know the status of the deployment - whether it is paused or progressing. Most importantly, administrators will want to be alerted if the Deployment was unable to deliver the intended/desired result of the operation. For instance, an administrator would want to be alerted if any deployment fails to create the desired number of available replicas in a ReplicaSet, fails to update one/more replicas with changes to a Pod template, or does not have the adequate number of Pods to reach full capacity. This is because, such failures may result in a mismatch between the cluster's desired state and its actual state, which in turn may affect the availability and performance of the containerized applications that overlay the cluster. This is where the Deployments by Namespace test helps!

This test auto-discovers Deployments in a Namespace, and for each Deployment, reports the overall status of the deployment - i.e., whether the desired state of the Deployment is the same as its actual state. If the state of the Deployment is Unhealthy, then you can use this test to figure out what could have caused the anomaly - is it because of a replica failure? is it because of unavailable Pods? or is it because of the Pods that are not yet up-to-date with changes made to the Pod template?

Target of the test : A Kubernetes Cluster

Agent deploying the test : A remote agent

Outputs of the test : One set of results for each Deployment in every namespace configured in the Kubernetes cluster being monitored

First-level Descriptor: Namespace

Second-level Descriptor: Deployment

Configurable parameters for the test

Parameter	Description
Test Period	How often should the test be executed.
Host	The IP address of the host for which this test is to be configured.
Port	Specify the port at which the specified Host listens. By default, this is *6443*.
Load Balancer / Master Node IP	To run this test and report metrics, the eG agent needs to connect to the Kubernetes API on the master node and run API commands. To enable this connection, the eG agent has to be configured with either of the following: • If only a single master node exists in the cluster, then configure the eG agent with the IP address of the master node. • If the target cluster consists of more than one master node, then you need to configure the eG agent with the IP address of the load balancer that is managing the cluster. In this case, the load balancer will route the eG agent's connection request to any available master node in the cluster, thus enabling the agent to connect with the API server on that node, run API commands on it, and pull metrics. By default, this parameter will display the **Load Balancer / Master Node IP** that you configured when manually adding the Kubernetes cluster for monitoring, using the **Kubernetes Cluster Preferences** page in the eG admin interface (see Figure 2.3). The steps for managing the cluster using the eG admin interface are discussed elaborately in How to Monitor the Kubernetes Cluster Using eG Enterprise? Whenever the eG agent runs this test, it uses the IP address that is displayed (by default) against this parameter to connect to the Kubernetes API. If there is any change in this IP address at a later point in time, then make sure that you update this parameter with it, by overriding its default setting.
SSL	By default, the Kubernetes cluster is SSL-enabled. This is why, the eG agent, by default, connects to the Kubernetes API via an HTTPS connection. Accordingly, this flag is set to **Yes** by default. If the cluster is not SSL-enabled in your environment, then set this flag to **No**.
Authentication Token	The eG agent requires an authentication bearer token to access the Kubernetes API, run API commands on the cluster, and pull metrics of interest. The steps for generating this token have been detailed in Section **1.1** Typically, once you generate the token, you can associate that token with the target Kubernetes cluster, when manually adding that cluster for monitoring using the

Parameter	Description
	eG admin interface. The steps for managing the cluster using the eG admin interface are discussed elaborately in Section **Chapter 2** By default, this parameter will display the **Authentication Token** that you provided in the **Kubernetes Cluster Preferences page** of the eG admin interface, when manually adding the cluster for monitoring (see Figure 2.3). Whenever the eG agent runs this test, it uses the token that is displayed (by default) against this parameter for accessing the API and pulling metrics. If for any reason, you generate a new authentication token for the target cluster at a later point in time, then make sure you update this parameter with the change. For that, copy the new token and paste it against this parameter.
Report System Namespace	The kube-system namespace consists of all objects created by the Kubernetes system. Monitoring such a namespace may not only increase the eG agent's processing overheads, but may also clutter the eG database. Therefore, to optimize agent performance and to conserve database space, this test, by default, excludes the kube-system namespace from monitoring. Accordingly, this flag is set to **No** by default. If required, you can set this flag to **Yes**, and enable monitoring of the kube-system namespace.
Proxy Host	If the eG agent connects to the Kubernetes API on the master node via a proxy server, then provide the IP address of the proxy server here. If no proxy is used, then the default setting -*none* - of this parameter, need not be changed,
Proxy Port	If the eG agent connects to the Kubernetes API on the master node via a proxy server, then provide the port number at which that proxy server listens here. If no proxy is used, then the default setting -*none* - of this parameter, need not be changed,
Proxy Username, Proxy Password, Confirm Password	**These parameters are applicable only if the eG agent uses a proxy server to connect to the Kubernetes cluster, and that proxy server requires authentication.** In this case, provide a valid user name and password against the **Proxy Username** and **Proxy Password** parameters, respectively. Then, confirm the password by retyping it in the **Confirm Password** text box. If no proxy server is used, or if the proxy server used does not require authentication, then the default setting - *none* - of these parameters, need not be changed.
DD Frequency	Refers to the frequency with which detailed diagnosis measures are to be generated for this test. The default is *1:1*. This indicates that, by default, detailed measures will be generated every time this test runs, and also every time the test detects a problem. You can modify this frequency, if you so desire. Also, if you intend to disable the

Parameter	Description
	detailed diagnosis capability for this test, you can do so by specifying *none* against DD frequency.
Detailed Diagnosis	To make diagnosis more efficient and accurate, the eG Enterprise suite embeds an optional detailed diagnostic capability. With this capability, the eG agents can be configured to run detailed, more elaborate tests as and when specific problems are detected. To enable the detailed diagnosis capability of this test for a particular server, choose the **On** option. To disable the capability, click on the **Off** option. The option to selectively enable/disable the detailed diagnosis capability will be available only if the following conditions are fulfilled: • The eG manager license should allow the detailed diagnosis capability • Both the normal and abnormal frequencies configured for the detailed diagnosis measures should not be 0.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Status	Indicates whether/not the desired state of this deployment is the same as its actual state.		This measure reports the value *Healthy* if the desired state of the Deployment is the same as its actual state. In other words, if the value of the *Total pods with deployment* measure is the same as the value of *Total available pods* with deployment measure, then this measure will report the value *Healthy*. If not, this measure will report the value *Unhealthy*. For instance, if the Deployment seeks to deploy a ReplicaSet with 3 replica (Pods) in it, and succeeds in creating such a ReplicaSet, then the value of this measure will be *Healthy*. On the other hand, if the Deployment created a ReplicaSet with only two available replica Pods, then the value of this measure will be *Unhealthy*. The numeric values that correspond to these measure values are as follows:

<table>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
<tr><td></td><td></td><td></td><td>
<table>
<tr><th>Measure Value</th><th>Numeric Value</th></tr>
<tr><td>Healthy</td><td>1</td></tr>
<tr><td>Unhealthy</td><td>0</td></tr>
</table>
<b>Note:</b>
By default, this test reports the <b>Measure Value</b>s listed in the table above to indicate the state of a Deployment. In the graph of this measure however, the same is indicated using the numeric equivalents only.</td></tr>
<tr><td>Is deployment paused?</td><td>Indicates whether/not this Deployment has been paused.</td><td></td><td>You can pause a Deployment before triggering one or more updates and then resume it. This allows you to apply multiple fixes in between pausing and resuming without triggering unnecessary rollouts.
The values that this measure can report and their corresponding numeric values are listed in the table below:
<table>
<tr><th>Measure Value</th><th>Numeric Value</th></tr>
<tr><td>Yes</td><td>1</td></tr>
<tr><td>No</td><td>0</td></tr>
</table>
<b>Note:</b>
By default, this test reports the <b>Measure Value</b>s listed in the table above to indicate the whether/not a Deployment has been paused. In the graph of this measure however, the same is indicated using the numeric equivalents only.</td></tr>
<tr><td>Age</td><td>Indicates how old this Deployment is.</td><td></td><td>The value of this measure is expressed in number of days, hours, and minutes.
You can use the detailed diagnosis of this</td></tr>
</table>

Measurement	Description	Measurement Unit	Interpretation
			measure to know the images that a Deployment pulled from the Container Registry to create containers on replica Pods.
Is progressing?	Indicates whether/not this Deployment is in progress.		A Deployment enters various states during its lifecycle. Kubernetes marks a Deployment as *progressing* when one of the following tasks is performed: • The Deployment creates a new ReplicaSet. • The Deployment is scaling up its newest ReplicaSet. • The Deployment is scaling down its older ReplicaSet(s). • New Pods become ready or available (ready for at least *MinReadySeconds*). Kubernetes marks a Deployment as *complete* when it has the following characteristics: • The Deployment has minimum availability. Minimum availability means that the Deployment's number of available replicas equals or exceeds the number required by the Deployment strategy. • All of the replicas associated with the Deployment have been updated to the latest version you have specified, meaning any updates you've requested have been completed.

<table>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
<tr><td></td><td></td><td></td><td>
<ul><li>No old pods for the Deployment are running.</li></ul>
<p>Your Deployment may get stuck trying to deploy its newest ReplicaSet without ever completing. This can occur due to some of the following factors:</p>
<ul>
<li>Insufficient quota</li>
<li>Readiness probe failures</li>
<li>Image pull errors</li>
<li>Insufficient permissions</li>
<li>Limit ranges</li>
<li>Application runtime misconfiguration</li>
</ul>
<p>Typically, a Deployment is considered to have <i>Failed</i>, if it is making progress for a duration beyond the progressDeadlineSeconds configuration.</p>
<p>This measure reports the value <i>Yes</i> for a Deployment, if it is in the <i>progressing</i> or <i>complete</i> state. The value <i>No</i> is reported, if the Deployment is in <i>fail to progress</i> state. The value <i>Unknown</i> is reported if the Deployment is not in any of the above-mentioned states - i.e., if the state cannot be determined.</p>
<p>The numeric values that correspond to these measure values are as follows:</p>
<table>
<tr><th>Measure Value</th><th>Numeric Value</th></tr>
<tr><td>Yes</td><td>1</td></tr>
<tr><td>No</td><td>0</td></tr>
<tr><td>Unknown</td><td>2</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
<tr><td></td><td></td><td></td><td><b>Note:</b>
By default, this test reports the <b>Measure Value</b>s listed in the table above to indicate whether/not a Deployment is progressing. In the graph of this measure however, the same is indicated using the numeric equivalents only.</td></tr>
<tr><td>Is available?</td><td>Indicates whether/not this Deployment is available.</td><td>Number</td><td>A deployment is said to be <i>Available</i>, if it has minimum availability. Minimum availability is dictated by the parameters specified in the deployment strategy. For instance, if the default Rolling Update strategy is used, then the <i>Max Unavailable</i> parameter of the strategy indicates the Minimum Availability during an upgrade. For example, if the <i>Max Unavailable</i> is set to 25% , then it means that a minimum of 75% of Pods should be available in the Deployment when an update is in progress.

If the Deployment is Available, then the value of this measure is <i>Yes</i>. If the Deployment is unavailable - i.e., if the Minimum Availability criteria is not met - then, the value of this measure is <i>No</i>. If the availability of the Deployment cannot be determined, then the value of this measure will be <i>Unknown</i>.
The numeric values that correspond to these measure values are as follows:
<table>
<tr><th>Measure Value</th><th>Numeric Value</th></tr>
<tr><td>Yes</td><td>1</td></tr>
<tr><td>No</td><td>0</td></tr>
<tr><td>Unknown</td><td>2</td></tr>
</table>
<b>Note:</b></td></tr>
</table>

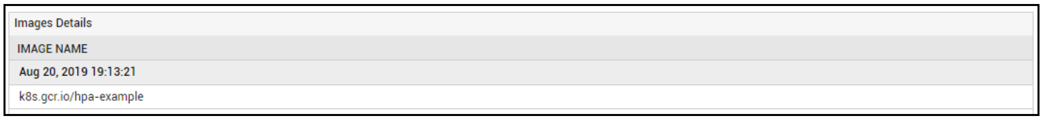
<table>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
<tr><td></td><td></td><td></td><td>By default, this test reports the <b>Measure Value</b>s listed in the table above to indicate whether/not a Deployment is Available. In the graph of this measure however, the same is indicated using the numeric equivalents only.
If the value of this measure for any Deployment is <i>No</i>, then you can use the detailed diagnosis of this measure to identify the unavailable Pods in the Deployment, and the reason for their unavailability. A Pod is said to be Available, if it is ready without any containers crashing for at least the duration configured against <i>minReadySeconds</i> in the Pod specification. Any Pod that is not ready, or is ready but has containers crashing for a period of time beyond the <i>minReadySeconds</i> duration, is automatically considered <i>Unavailable</i>.</td></tr>
<tr><td>Is replica failure?</td><td>Indicates whether/not any replica in this Deployment has failed.</td><td></td><td>The value <i>Yes</i> for this measure, indicates that one/more replicas in the Deployment could not be created. The value <i>No</i> indicates that all replicas were created. The value Unknown implies that the state of the replicas could not be determined.
The numeric values that correspond to these measure values are as follows:
<table><tr><th>Measure Value</th><th>Numeric Value</th></tr><tr><td>Yes</td><td>1</td></tr><tr><td>No</td><td>0</td></tr><tr><td>Unknown</td><td>2</td></tr></table>
<b>Note:</b>
By default, this test reports the <b>Measure</b></td></tr>
</table>

Measurement	Description	Measurement Unit	Interpretation
			Values listed in the table above to indicate whether/not any replicas in the Deployment could not be created. In the graph of this measure however, the same is indicated using the numeric equivalents only. If the failure of one/more replicas causes a a mismatch between the desired state and actual state of the Deployment, then the value of the *Status* measure of that Deployment will change to Unhealthy.
Observed generation	Indicates the generation observed after this Deployment.	Number	A generation is a sequence number representing a specific generation of the desired state. If the value of this measure for a Deployment matches with the desired generation sequence number of that Deployment, it implies that the Deployment is *complete*. If it does not match, then it means that a Deployment is *progressing or has failed*. In other words, if the value of the *Is progressing?* measure for a Deployment is *Yes* or *No*, then it means that the desired generation sequence number and the observed generation sequence number of that Deployment is not the same.
Total pods with deployment	Indicates the desired number of non-terminated Pod replicas targeted by this Deployment.	Number	
Total pods with updated deployment	Indicates the total number of non-terminated Pod replicas that have been updated by this Deployment with	Number	Typically, whenever changes are made to a Deployment's Pod template - say, labels or container images of the template are changed - then a Deployment rollout is triggered. A new ReplicaSet is created and

Measurement	Description	Measurement Unit	Interpretation
	changes (if any) made to the Pod template specification.		the Deployment manages moving the Pods from the old ReplicaSet to the new one at a controlled rate. Ideally, the value of this measure should be the same as the value of the *Total pods with deployment* measure. If not, then it means that the desired number of Pod replicas are not yet fully updated with the changes to the Pod template.
Ready pods with deployment	Indicates the number of ready Pods created by this Deployment.	Number	
Total available pods with deployment	Indicates the number of available Pods created by this Deployment.	Number	A Pod is said to be Available, if it is ready without any containers crashing for at least the duration configured against minReadySeconds in the Pod specification. Ideally, the value of this measure should be the same as the value of the *Total pods with deployment* measure. If not, then the *Status* measure of this test will report the value Unhealthy. This means that the desired state of the Deployment is not the same as its actual state.
Total unavailable pods with deployment	Indicates the total number of unavailable Pods created by this Deployment.	Number	Any Pod that is not ready, or is ready but has containers crashing for a period of time beyond the *minReadySeconds* duration, is automatically considered Unavailable. Ideally, the value of this measure should be 0. If this measure reports a non-zero value or a value equal to or close to the value of the *Total pods with deployment* measure, then the *Status* measure of this test will report the value *Unhealthy*. This means that the desired state of the Deployment is not the same as its actual

Measurement	Description	Measurement Unit	Interpretation
			state. In the event that this measure reports a non-zero value, then use the detailed diagnosis of this measure to identify the unavailable Pod replicas in the Deployment.
Collision count	Indicates the count of hash collisions for this deployment.	Number	The Deployment controller uses this field as a collision avoidance mechanism when it needs to create the name for the newest ReplicaSet.
Pods created by this deployment	Indicates the number of Pods created by this Deployment that are currently running.	Number	To know which Pods created by this Deployment are currently running, use the detailed diagnosis of this measure.
Retain old replica count	Indicates the number of old Replica Sets that this Deployment should retain to allow rollback.	Number	A Deployment's revision history is stored in the ReplicaSets it controls. When configuring a Deployment, you can optionally specify *.spec.revisionHistoryLimit*, where you can indicate the number of old ReplicaSets to retain to allow rollback. This value is reported as the value of this measure. Old ReplicaSets consume resources in etcd and crowd the output of *kubectl get rs*. The configuration of each Deployment revision is stored in its ReplicaSets; therefore, once an old ReplicaSet is deleted, you lose the ability to rollback to that revision of Deployment. By default, 10 old ReplicaSets will be kept, however its ideal value depends on the frequency and stability of new Deployments. More specifically, setting this field to zero means that all old ReplicaSets with 0 replicas will be cleaned up. In this case, a new Deployment rollout cannot be undone, since its revision history is cleaned up.

You can use the detailed diagnosis of the Age measure to know the images that a Deployment pulled from the Container Registry to create containers on replica Pods.

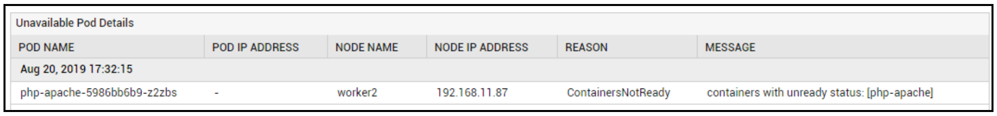


Images Details

IMAGE NAME
Aug 20, 2019 19:13:21
k8s.gcr.io/hpa-example

Figure 3.38: The detailed diagnosis of the Age measure of the Deployments by Namespace test

If the value of the *Is available?* measure for any Deployment is *No*, then you can use the detailed diagnosis of this measure to identify the unavailable Pods in the Deployment, and the reason for their unavailability. A Pod is said to be Available, if it is ready without any containers crashing for at least the duration configured against *minReadySeconds* in the Pod specification. Any Pod that is not ready, or is ready but has containers crashing for a period of time beyond the *minReadySeconds* duration, is automatically considered *Unavailable*.

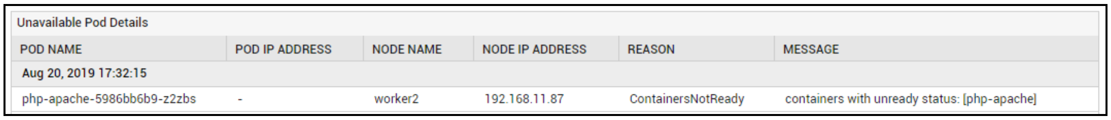


Unavailable Pod Details

POD NAME	POD IP ADDRESS	NODE NAME	NODE IP ADDRESS	REASON	MESSAGE
Aug 20, 2019 17:32:15					
php-apache-5986bb6b9-z2zbs	-	worker2	192.168.11.87	ContainersNotReady	containers with unready status: [php-apache]

Figure 3.39: The detailed diagnosis of the Is available? measure reported by the Deployments by Namespace test

In the event that the *Total unavailable pods with deployment* measure reports a non-zero value, then use the detailed diagnosis of this measure to identify the unavailable Pod replicas in the Deployment. You can also use the detailed diagnostics to figure what could have rendered a Pod unavailable. This information aids troubleshooting.



Unavailable Pod Details

POD NAME	POD IP ADDRESS	NODE NAME	NODE IP ADDRESS	REASON	MESSAGE
Aug 20, 2019 17:32:15					
php-apache-5986bb6b9-z2zbs	-	worker2	192.168.11.87	ContainersNotReady	containers with unready status: [php-apache]

Figure 3.40: The detailed diagnosis of the Total unavailable pods with deployment measure

3.1.13 Daemonset by Namespace

A DaemonSet ensures that all (or some) Nodes run a copy of a Pod. As nodes are added to the cluster, Pods are added to them. As nodes are removed from the cluster, those Pods are garbage collected. Deleting a DaemonSet will clean up the Pods it created.

Some typical uses of a DaemonSet are:

- running a cluster storage daemon, such as *glusterd*, *ceph*, on each node.
- running a logs collection daemon on every node, such as *fluentd* or *logstash*.
- running a node monitoring daemon on every node

Daemon pods are typically scheduled using one of the following:

- Daemonset Controller: Normally, the machine that a Pod runs on is selected by the Kubernetes scheduler. However, Pods created by the DaemonSet controller have the machine already selected .
- Default scheduler: You can also schedule DaemonSets using the default scheduler instead of the DaemonSet controller, by adding the *NodeAffinity* term to the DaemonSet pods, instead of the *.spec.nodeName* term. The default scheduler is then used to bind the pod to the target host.

Regardless of which scheduler (Daemonset Controller or default scheduler) schedules Daemon Pods, taints and tolerations are used to ensure that Daemon pods are not scheduled onto inappropriate nodes. One or more taints are applied to a node; this marks that the node should not accept any pods that do not tolerate the taints. Tolerations are applied to pods, and allow (but do not require) the pods to schedule onto nodes with matching taints.

Sometimes, a Daemon Pod may be 'misscheduled' on to a node where it does not belong. In other words, a Daemon Pod could be scheduled on to a node without 'matching taints'. This can cause certain cluster operations to run on nodes they should not run on, hampering cluster performance in the process. At some other times, a Daemon Pod may not run on the desired set of nodes. For instance, an anti-virus daemon, which should typically run on all nodes in a cluster/namespace, may run only on a few nodes. This is also detrimental to cluster performance. To ensure peak cluster performance, administrators should rapidly identify misscheduled DaemonSets and those that are not running on the desired nodes, and figure out what could have triggerred these anomalies. This is where the DaemonSet by Namespace test helps!

This test auto-discovers the DaemonSets in each namespace, and for each DaemonSet, reports the count of nodes scheduled to run that DaemonSet, the count of nodes on which it should run, and the count of nodes on which it should not. This way, the test promptly alerts administrators to incorrect scheduling of DaemonSets. Detailed diagnostics reveal which Daemon Pods are running on which node, thereby enabling administrators to quickly identify those nodes running Daemon Pods they should not be running. Additionally, the test also alerts administrators if a DaemonSet is updated.

Target of the test : A Kubernetes Cluster

Agent deploying the test : A remote agent

Outputs of the test : One set of results for each DaemonSet in every namespace configured in the Kubernetes cluster being monitored

First-level Descriptor: Namespace

Second-level Descriptor: DaemonSet

Configurable parameters for the test

Parameter	Description
Test Period	How often should the test be executed.
Host	The IP address of the host for which this test is to be configured.
Port	Specify the port at which the specified Host listens. By default, this is *6443*.
Load Balancer / Master Node IP	To run this test and report metrics, the eG agent needs to connect to the Kubernetes API on the master node and run API commands. To enable this connection, the eG agent has to be configured with either of the following: • If only a single master node exists in the cluster, then configure the eG agent with the IP address of the master node. • If the target cluster consists of more than one master node, then you need to configure the eG agent with the IP address of the load balancer that is managing the cluster. In this case, the load balancer will route the eG agent's connection request to any available master node in the cluster, thus enabling the agent to connect with the API server on that node, run API commands on it, and pull metrics. By default, this parameter will display the **Load Balancer / Master Node IP** that you configured when manually adding the Kubernetes cluster for monitoring, using the **Kubernetes Cluster Preferences** page in the eG admin interface (see Figure 2.3). The steps for managing the cluster using the eG admin interface are discussed elaborately in How to Monitor the Kubernetes Cluster Using eG Enterprise? Whenever the eG agent runs this test, it uses the IP address that is displayed (by default) against this parameter to connect to the Kubernetes API. If there is any change in this IP address at a later point in time, then make sure that you update this parameter with it, by overriding its default setting.
SSL	By default, the Kubernetes cluster is SSL-enabled. This is why, the eG agent, by default, connects to the Kubernetes API via an HTTPS connection. Accordingly, this flag is set to **Yes** by default. If the cluster is not SSL-enabled in your environment, then set this flag to **No**.

Parameter	Description
Authentication Token	The eG agent requires an authentication bearer token to access the Kubernetes API, run API commands on the cluster, and pull metrics of interest. The steps for generating this token have been detailed in Section **1.1** Typically, once you generate the token, you can associate that token with the target Kubernetes cluster, when manually adding that cluster for monitoring using the eG admin interface. The steps for managing the cluster using the eG admin interface are discussed elaborately in Section **Chapter 2** By default, this parameter will display the **Authentication Token** that you provided in the **Kubernetes Cluster Preferences page** of the eG admin interface, when manually adding the cluster for monitoring (see Figure 2.3). Whenever the eG agent runs this test, it uses the token that is displayed (by default) against this parameter for accessing the API and pulling metrics. If for any reason, you generate a new authentication token for the target cluster at a later point in time, then make sure you update this parameter with the change. For that, copy the new token and paste it against this parameter.
Proxy Host	If the eG agent connects to the Kubernetes API on the master node via a proxy server, then provide the IP address of the proxy server here. If no proxy is used, then the default setting -*none* - of this parameter, need not be changed,
Proxy Port	If the eG agent connects to the Kubernetes API on the master node via a proxy server, then provide the port number at which that proxy server listens here. If no proxy is used, then the default setting -*none* - of this parameter, need not be changed,
Proxy Username, Proxy Password, Confirm Password	**These parameters are applicable only if the eG agent uses a proxy server to connect to the Kubernetes cluster, and that proxy server requires authentication.** In this case, provide a valid user name and password against the **Proxy Username** and **Proxy Password** parameters, respectively. Then, confirm the password by retyping it in the **Confirm Password** text box. If no proxy server is used, or if the proxy server used does not require authentication, then the default setting - *none* - of these parameters, need not be changed.
DD Frequency	Refers to the frequency with which detailed diagnosis measures are to be generated for this test. The default is *3:1*. This indicates that, by default, detailed measures will be generated every third time this test runs, and also every time the test detects a problem. You can modify this frequency, if you so desire. Also, if you intend to disable the detailed diagnosis capability for this test, you can do so by specifying *none* against DD frequency.
Detailed Diagnosis	To make diagnosis more efficient and accurate, the eG Enterprise suite embeds an

Parameter	Description
	optional detailed diagnostic capability. With this capability, the eG agents can be configured to run detailed, more elaborate tests as and when specific problems are detected. To enable the detailed diagnosis capability of this test for a particular server, choose the **On** option. To disable the capability, click on the **Off** option. The option to selectively enable/disable the detailed diagnosis capability will be available only if the following conditions are fulfilled: • The eG manager license should allow the detailed diagnosis capability • Both the normal and abnormal frequencies configured for the detailed diagnosis measures should not be 0.

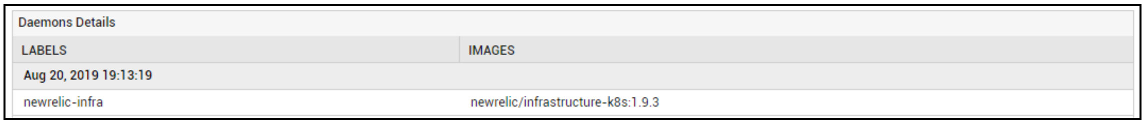
Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Age	Indicates how old this DaemonSet is.		The value of this measure is expressed in number of days, hours, and minutes. You can use the detailed diagnosis of this measure to know the labels and images used by the daemons run by the DaemonSet.
DaemonSet currently scheduled on nodes	Indicates the number of nodes (in this namespace) that are currently running this DaemonSet and are supposed to run this DaemonSet.	Number	Use the detailed diagnosis of this measure to know which Daemon Pods are running on which nodes in the namespace.
DaemonSets misscheduled on nodes	Indicates the number of nodes in this namespace, that are running this DaemonSet, but are not supposed to run this DaemonSet.	Number	Ideally, the value of this measure should be 0.
DaemonSet to run on nodes	Indicates the number of nodes (in this	Number	The value of this measure also includes the count of nodes that are already

Measurement	Description	Measurement Unit	Interpretation
	namespace) that should be running this DaemonSet.		running the DaemonSet. Ideally therefore, this value of this measure should be the same as the value of the *DaemonSet currently scheduled on nodes* measure. Any mismatch implies issues in scheduling, which in turn may impact cluster performance.
DaemonSet running on nodes	Indicates the number of nodes (in this namespace) that should be running this DaemonSet and have one or more of the Daemon Pods already running and ready.	Number	
DaemonSet updated on nodes	Indicates the number of nodes (in this namespace) that run the updated daemon pod spec.	Number	Updating a DaemonSet may involve: • **Changing node labels:** If node labels are changed, the DaemonSet will promptly add Pods to newly matching nodes and delete Pods from newly not-matching nodes. • **Changing a Daemon Pod:** You can modify the Pods that a DaemonSet creates. However, Pods do not allow all fields to be updated. Also, the DaemonSet controller will use the original template the next time a node (even with the same name) is created. • **Deleting a DaemonSet:** When deleting a DaemonSet, you can choose to leave the Daemon Pods on

Measurement	Description	Measurement Unit	Interpretation
			the nodes. In this case, if you subsequently create a new DaemonSet with the same selector, the new DaemonSet adopts the existing Pods. If any Pods need replacing the DaemonSet replaces them according to its *updateStrategy.* • **Performing a rolling update on a DaemonSet:** With *RollingUpdate* update strategy, after you update a DaemonSet template, old DaemonSet pods will be killed, and new DaemonSet pods will be created automatically, in a controlled fashion.
DaemonSet available on nodes	Indicates the number of nodes (in this namespace) that should be running this DaemonSet and have one or more of the Daemon Pods running and available.	Number	A Daemon Pod is considered to be 'available' if it is ready without any of its containers crashing for at least the duration specified against spec.minReadySeconds in the DaemonSet configuration (YAML) file.
DaemonSet unavailable on nodes	Indicates the number of nodes (in this namespace) that should be running this DaemonSet, but does not have it running and available.	Number	A Daemon Pod is considered to be 'unavailable' if it is not ready without any of its containers crashing for even the minimum duration specified against spec.minReadySeconds in the DaemonSet configuration (YAML) file. Ideally, the value of this measure should be 0.

Using the detailed diagnosis of the *Age* measure you can determine the label that has been assigned to a particular DaemonSet, and the images that the containers on the Daemon Pods are pulling from the Container Registry.

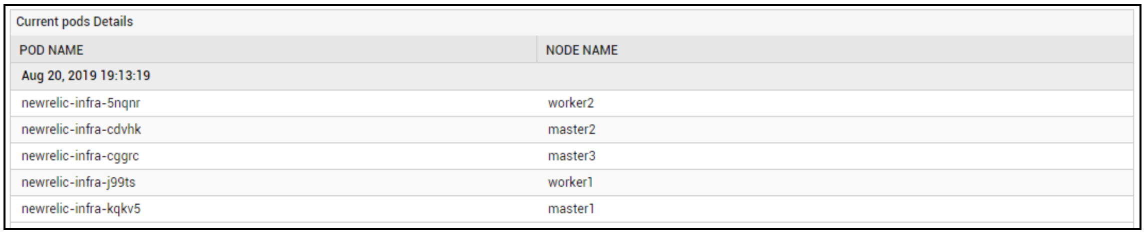


Daemons Details

LABELS	IMAGES
Aug 20, 2019 19:13:19	
newrelic-infra	newrelic/infrastructure-k8s:1.9.3

Figure 3.41: The detailed diagnosis of the Age measure of the DaemonSet by Namespace test

To know the Daemon Pods running a DaemonSet and the nodes on which these Pods are running, use the detailed diagnosis of the *DaemonSet currently scheduled on nodes* measure. Using this information, you can figure out if the DaemonSet is running on a node it is not supposed to run on and if it is not running on any node it should actually run on.



Current pods Details

POD NAME	NODE NAME
Aug 20, 2019 19:13:19	
newrelic-infra-5nqnr	worker2
newrelic-infra-cdvhk	master2
newrelic-infra-cggrc	master3
newrelic-infra-j99ts	worker1
newrelic-infra-kqkv5	master1

Figure 3.42: The detailed diagnosis of the DaemonSet currently scheduled on nodes measure

3.1.14 Horizontal Pod Autoscaler by Namespaces Test

Horizontal Pod Autoscaling allows you to define rules that will scale the numbers of replicas up or down in deployments, replica sets, or replication controllers, based on CPU utilization and optionally external and custom metrics. For instance, if you have a containerized application that uses up a lot of CPU under load, then you can configure a Horizontal Pod Autoscaler to automatically scale up the Deployment, so that additional replicas of this application (Pod) are automatically created to provide extra capacity when CPU utilization exceeds a target level. Likewise, you can configure the Horizontal Pod Autoscaler to scale down a Deployment, so that replica Pods are automatically terminated to release CPU resources when actual CPU utilization drops below a target level.

Typically, when creating a horizontal autoscaler, you can specify the target utilization value of the metric - this can be a raw value or an average value. Optionally, you can also specify the following:

- The maximum number of replicas the autoscaler can scale up to;
- The minimum number of replicas the autoscaler can scale down to

Whenever the autoscaler runs, the controller manager obtains the actual metrics from the resource metrics API (for per-pod resource metrics), or the custom metrics API (for metrics other than CPU and memory that are associated with a Pod), or the external metrics API (for metrics that are not associated with any object in the Kubernetes system - eg., an external queuing system, such as the AWS SQS service), as the case may be. Then, it does the following:

- For per-pod resource metrics (like CPU), the controller fetches the metrics from the resource metrics API for each Pod targeted by the HorizontalPodAutoscaler. Then, if a target utilization value is set, the controller calculates the utilization value as a percentage of the equivalent resource request on the containers in each pod. If a target raw value is set, the raw metric values are used directly. The controller then takes the mean of the utilization or the raw value (depending on the type of target specified) across all targeted pods, and produces a ratio, which will be used to scale the number of desired replicas.
- For per-pod custom metrics, the controller functions similarly to per-pod resource metrics, except that it works with raw values, not utilization values.
- For object metrics and external metrics, a single metric is fetched, which describes the object in question. This metric is compared to the target value, to produce a ratio as above.

If actual resource usage exceeds the targeted value, then the autoscaler uses the ratio it computes to scale up the replicas. On the other hand, if the actual resource usage falls below the targeted value, then the autoscaler uses the ratio it computes to scale down.

The efficiency of the autoscaler therefore relies on the successful computation of scales by the autoscaler, and how prudently you set the scaling limits (i.e., the minimum and maximum replica count for the autoscaler) and the target utilization values. Sometimes, the autoscaler may fail to compute scales. At some other times, user errors may restrict scalability or environmental issues may prevent scaling from even happening. At such times, the success of scaling hinges on the administrator's ability to promptly detect, diagnose, and fix the bottlenecks to scaling. With the Horizonal Pod Autoscaler by Namespaces test, administrators have the ability to achieve the above!

The test auto-discovers the Horizonal Pod autoscalers defined in each namespace. For each autoscaler in a namespace, the test then reports whether/not that autoscaler can actually perform scaling, reveals if its scalability is constricted by its configuration, and alerts administrators if the autoscaler is unable to compute the scales. This way, the test enables administrators promptly capture problems impeding efficient autoscaling. If minimum and maximum replica counts were specified as part of the autoscaler definition, then the test also reports these numbers, so administrators can quickly figure out if changing these values can enhance scalability. Moreover, by enabling administrators to track current CPU utilization levels alongside the target utilization levels, the test not only helps them compute the scaling ratio themselves, but also helps them figure out if the target needs to be reset. Furthermore, by reporting the desired and current replica counts, the test reveals to administrators whether/not the autoscaler has successfully scaled up the replica count to the desired level.

Target of the test : A Kubernetes Cluster

Agent deploying the test : A remote agent

Outputs of the test : One set of results for each autoscaler in each namespace of the Kubernetes cluster being monitored

Configurable parameters for the test

Parameter	Description
Test Period	How often should the test be executed.
Host	The IP address of the host for which this test is to be configured.
Port	Specify the port at which the specified Host listens. By default, this is *6443*.
Load Balancer / Master Node IP	To run this test and report metrics, the eG agent needs to connect to the Kubernetes API on the master node and run API commands. To enable this connection, the eG agent has to be configured with either of the following: • If only a single master node exists in the cluster, then configure the eG agent with the IP address of the master node. • If the target cluster consists of more than one master node, then you need to configure the eG agent with the IP address of the load balancer that is managing the cluster. In this case, the load balancer will route the eG agent's connection request to any available master node in the cluster, thus enabling the agent to connect with the API server on that node, run API commands on it, and pull metrics. By default, this parameter will display the **Load Balancer / Master Node IP** that you configured when manually adding the Kubernetes cluster for monitoring, using the **Kubernetes Cluster Preferences** page in the eG admin interface (see Figure 2.3). The steps for managing the cluster using the eG admin interface are discussed elaborately in How to Monitor the Kubernetes Cluster Using eG Enterprise? Whenever the eG agent runs this test, it uses the IP address that is displayed (by default) against this parameter to connect to the Kubernetes API. If there is any change in this IP address at a later point in time, then make sure that you update this parameter with it, by overriding its default setting.
SSL	**By default, the Kubernetes cluster is SSL-enabled. This is why, the eG agent, by default, connects to the Kubernetes API via an HTTPS connection. Accordingly, this flag is set to Yes by default.** If the cluster is not SSL-enabled in your environment, then set this flag to **No**.
Authentication Token	The eG agent requires an authentication bearer token to access the Kubernetes API, run API commands on the cluster, and pull metrics of interest. The steps for generating this token have been detailed in Section **1.1**

Parameter	Description
	Typically, once you generate the token, you can associate that token with the target Kubernetes cluster, when manually adding that cluster for monitoring using the eG admin interface. The steps for managing the cluster using the eG admin interface are discussed elaborately in Section **Chapter 2** By default, this parameter will display the **Authentication token** that you provided in the **Kubernetes Cluster Preferences page** of the eG admin interface, when manually adding the cluster for monitoring (see Figure 2.3). Whenever the eG agent runs this test, it uses the token that is displayed (by default) against this parameter for accessing the API and pulling metrics. If for any reason, you generate a new authentication token for the target cluster at a later point in time, then make sure you update this parameter with the change. For that, copy the new token and paste it against this parameter.
Proxy Host	If the eG agent connects to the Kubernetes API on the master node via a proxy server, then provide the IP address of the proxy server here. If no proxy is used, then the default setting -*none* - of this parameter, need not be changed,
Proxy Port	If the eG agent connects to the Kubernetes API on the master node via a proxy server, then provide the port number at which that proxy server listens here. If no proxy is used, then the default setting -*none* - of this parameter, need not be changed,
Proxy Username, Proxy Password, Confirm Password	**These parameters are applicable only if the eG agent uses a proxy server to connect to the Kubernetes cluster, and that proxy server requires authentication.** In this case, provide a valid user name and password against the **Proxy Username** and **Proxy Password** parameters, respectively. Then, confirm the password by retyping it in the **Confirm Password** text box. If no proxy server is used, or if the proxy server used does not require authentication, then the default setting - *none* - of these parameters, need not be changed.
DD Frequency	Refers to the frequency with which detailed diagnosis measures are to be generated for this test. The default is *1:1*. This indicates that, by default, detailed measures will be generated every time this test runs, and also every time the test detects a problem. You can modify this frequency, if you so desire. Also, if you intend to disable the detailed diagnosis capability for this test, you can do so by specifying *none* against DD frequency.
Detailed Diagnosis	To make diagnosis more efficient and accurate, the eG Enterprise suite embeds an optional detailed diagnostic capability. With this capability, the eG agents can be configured to run detailed, more elaborate tests as and when specific problems are detected. To enable the detailed diagnosis capability of this test for a particular server, choose the **On** option. To disable the capability, click on the **Off** option.

Parameter	Description
	The option to selectively enable/disable the detailed diagnosis capability will be available only if the following conditions are fulfilled: • The eG manager license should allow the detailed diagnosis capability • Both the normal and abnormal frequencies configured for the detailed diagnosis measures should not be 0.

Measurements made by the test

<table>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
<tr><td>Age</td><td>Indicates the age of this autoscaler.</td><td></td><td>The value of this measure is expressed in number of days, hours, and minutes.</td></tr>
<tr><td>Is able to scale?</td><td>Indicates whether/not this autoscaler is allowed to scale.</td><td></td><td>This measure reports the value <i>Yes</i> if the autoscaler is able to fetch and update scales. The value <i>No</i> is reported if backoff conditions - eg., a CrashLoopBackOff that is causing a Pod to start and crashing in a loop - are preventing scaling. The value <i>Unknown</i> is reported if the state cannot be determined.
The numeric values that correspond to these measure values are as follows:
<table>
<tr><th>Measure Value</th><th>Numeric Value</th></tr>
<tr><td>Yes</td><td>1</td></tr>
<tr><td>No</td><td>2</td></tr>
<tr><td>Unknown</td><td>3</td></tr>
</table>
<b>Note:</b>
By default, this test reports the <b>Measure Value</b>s listed in the table above to indicate whether/not an autoscaler is allowed to scale. In the graph of this measure however, the same is indicated using the numeric equivalents only.
If this measure reports the value <i>No</i> or</td></tr>
</table>

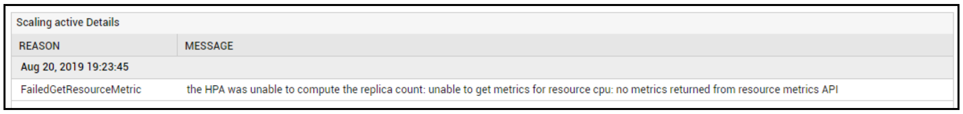
<table>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
<tr><td></td><td></td><td></td><td><i>Unknown</i>, then use the detailed diagnosis of this measure to know what prevented the autoscaler from performing scaling.</td></tr>
<tr><td>Is scaling active?</td><td>Indicates whether/not this autoscaler is enabled and is able to calculate the desired scales.</td><td></td><td>This measure reports the value <b>Yes</b> if the autoscaler is able to fetch metrics and compute the scales. The value <b>No</b> is reported if there are problems with fetching metrics. The value <b>Unknown</b> is reported if the state cannot be determined.
The numeric values that correspond to these measure values are as follows:
<table><tr><th>Measure Value</th><th>Numeric Value</th></tr><tr><td>Yes</td><td>1</td></tr><tr><td>No</td><td>2</td></tr><tr><td>Unknown</td><td>3</td></tr></table>
<b>Note:</b>
By default, this test reports the <b>Measure Value</b>s listed in the table above to indicate whether/not an autoscaler is able to fetch metrics. In the graph of this measure however, the same is indicated using the numeric equivalents only.
If this measure reports the value <b>No</b> or <b>Unknown</b>, then use the detailed diagnosis of this measure to know why the autoscaler could not fetch metrics.</td></tr>
<tr><td>Is scaling limited?</td><td>Indicates whether/not this autoscaler's ability to scale is restricted by a maximum / minimum replica count specification.</td><td></td><td>This measure reports the value <b>Yes</b> if you have to raise or lower the minimum or maximum replica count for the autoscaler to perform scaling. The value <b>No</b> is reported if the requested scaling is allowed. The value <b>Unknown</b> is reported if the state cannot be determined.</td></tr>
</table>

<table>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
<tr><td></td><td></td><td></td><td>The numeric values that correspond to these measure values are as follows:
<table>
<tr><th>Measure Value</th><th>Numeric Value</th></tr>
<tr><td>Yes</td><td>1</td></tr>
<tr><td>No</td><td>2</td></tr>
<tr><td>Unknown</td><td>3</td></tr>
</table>
<b>Note:</b>
By default, this test reports the <b>Measure Value</b>s listed in the table above to indicate whether/not an autoscaler is restricted by its minimum/maximum replica count specification. In the graph of this measure however, the same is indicated using the numeric equivalents only.
If this measure reports the value <b>No</b> or <b>Unknown</b>, then use the detailed diagnosis of this measure to know why the autoscaler could not scale.</td></tr>
<tr><td>Minimum replicas</td><td>Shows the lower limit for the number of Pods that can be set by this autoscaler. (Default: 1)</td><td>Number</td><td>If the value of this measure is the same as that of the <b>Current replicas</b> measure, then the autoscaler will not be able to scale down until the minimum replica count is decreased in the autoscaler definition. Under such circumstances, you will find that the <b>Is scaling limited?</b> measure reports the value <b>Yes</b>.</td></tr>
<tr><td>Maximum replicas</td><td>Shows the upper limit for the number of pods that can be set by this autoscaler.</td><td>Number</td><td>The value of this measure cannot be lesser than the value of the <b>Minimum replicas</b> measure.
If the value of this measure is the same as that of the <b>Current replicas</b> measure, then the autoscaler will not be able to scale up until the maximum replica count is increased</td></tr>
</table>

Measurement	Description	Measurement Unit	Interpretation
			in the autoscaler definition. Under such circumstances, you will find that the **Is scaling limited?** measure reports the value **Yes**
Target CPU utilization	Indicates the target average CPU utilization (represented as a percentage of requested CPU) set for this autoscaler.	Percent	If a target utilization is not set in the autoscaler's definition, then the default autoscaling policy will be used.
Current CPU utilization	Indicates the actual average CPU utilization across all Pods targeted by this autoscaler.	Percent	If the value of this measure is greater than that of the **Target CPU utilization** measure, the autoscaler will automatically scale up the replica Pod count to the desired level or up to the maximum replica count (whichever limit is reached first). If the value of this measure is lesser than that of the **Target CPU utilization** measure, the autoscaler will automatically scale down the replica pod count to the desired level or up to the minimum replica count (whichever limit is reached first).
Desired replicas	Indicates the number of replicas up to which this autoscaler can scale up or scale down.	Number	
Current replicas	Indicates the number of replicas currently managed by this autoscaler.	Number	If the value of this measure is not equal to that of the **Desired replicas** measure, it could mean one of the following: • Autoscaling has failed; • The minimum / maximum replica count specification in the autoscaler definition

Measurement	Description	Measurement Unit	Interpretation
			are restricting scalability. In the case of the former, you will have to investigate the reasons for the failure. In the case of the latter, check the value of the **Minimum replicas** and **Maximum replicas** measures and see if changing them will improve scalability of the autoscaler.

If the **Is scaling active?** measure reports the value **No** or **Unknown**, then use the detailed diagnosis of this measure to know why the autoscaler could not fetch metrics.



Scaling active Details

REASON	MESSAGE
Aug 20, 2019 19:23:45	
FailedGetResourceMetric	the HPA was unable to compute the replica count: unable to get metrics for resource cpu: no metrics returned from resource metrics API

Figure 3.43: The detailed diagnosis of the Is scaling active? measure

3.1.15 Jobs by Namespaces Test

A Job creates one or more Pods and ensures that a specified number of them successfully terminate. As pods successfully terminate, the Job tracks how many Pods completed their tasks successfully. When a specified number of successful completions is reached, the task (ie, Job) is complete.

Jobs are useful for large computation and batch-oriented tasks. Jobs can be used to support parallel execution of Pods. You can use a Job to run independent but related work items in parallel: sending emails, rendering frames, transcoding files, scanning database keys, etc.

In the real world, failure of such tasks can degrade the performance of business-critical applications managed by the Kubnernetes system. Likewise, delays in Job execution can also significantly delay the delivery of key business services that overlay the Kubernetes cluster. To ensure peak application/service performance at all times, it is imperative that administrators track the status and duration of each Job that is run on Kubernetes, promptly capture Job failures and slowness, rapidly determine the reason why a Job failed, and swiftly fix it. This is where the Jobs by Namespaces test helps!

This test auto-discovers the namespaces configured in the Kubernetes system, and for each namespace, reports the count of Jobs in different operational states. In the process, the test brings failed and slow Jobs to light. Detailed diagnostics of the test describes the failed and slow Jobs and also provides the reason why Jobs failed. Administrators can use this information to effectively troubleshoot the failure. Additionally, the test reports the status of Pods created by the Jobs, and alerts administrators if any Job resulted in Pod failures.

Target of the test : A Kubernetes Cluster

Agent deploying the test : A remote agent

Outputs of the test : One set of results for each namespace in the Kubernetes cluster being monitored

Configurable parameters for the test

Parameter	Description
Test Period	How often should the test be executed.
Host	The IP address of the host for which this test is to be configured.
Port	Specify the port at which the specified Host listens. By default, this is *6443*.
Load Balancer / Master Node IP	To run this test and report metrics, the eG agent needs to connect to the Kubernetes API on the master node and run API commands. To enable this connection, the eG agent has to be configured with either of the following: • If only a single master node exists in the cluster, then configure the eG agent with the IP address of the master node. • If the target cluster consists of more than one master node, then you need to configure the eG agent with the IP address of the load balancer that is managing the cluster. In this case, the load balancer will route the eG agent's connection request to any available master node in the cluster, thus enabling the agent to connect with the API server on that node, run API commands on it, and pull metrics. By default, this parameter will display the **Load Balancer / Master Node IP** that you configured when manually adding the Kubernetes cluster for monitoring, using the **Kubernetes Cluster Preferences** page in the eG admin interface (see Figure 2.3). The steps for managing the cluster using the eG admin interface are discussed elaborately in How to Monitor the Kubernetes Cluster Using eG Enterprise?

Parameter	Description
	Whenever the eG agent runs this test, it uses the IP address that is displayed (by default) against this parameter to connect to the Kubernetes API. If there is any change in this IP address at a later point in time, then make sure that you update this parameter with it, by overriding its default setting.
SSL	By default, the Kubernetes cluster is SSL-enabled. This is why, the eG agent, by default, connects to the Kubernetes API via an HTTPS connection. Accordingly, this flag is set to **Yes** by default. If the cluster is not SSL-enabled in your environment, then set this flag to **No**.
Authentication Token	The eG agent requires an authentication bearer token to access the Kubernetes API, run API commands on the cluster, and pull metrics of interest. The steps for generating this token have been detailed in Section **1.1** Typically, once you generate the token, you can associate that token with the target Kubernetes cluster, when manually adding that cluster for monitoring using the eG admin interface. The steps for managing the cluster using the eG admin interface are discussed elaborately in Section **Chapter 2** By default, this parameter will display the **Authentication Token** that you provided in the **Kubernetes Cluster Preferences page** of the eG admin interface, when manually adding the cluster for monitoring (see Figure 2.3). Whenever the eG agent runs this test, it uses the token that is displayed (by default) against this parameter for accessing the API and pulling metrics. If for any reason, you generate a new authentication token for the target cluster at a later point in time, then make sure you update this parameter with the change. For that, copy the new token and paste it against this parameter.
Job Age In Seconds	By default, this parameter is set to 300 seconds. This means that, by default, this test will count any Job that runs for a duration over 300 seconds as a *Longest running Job*. You can override this default setting by specifying a different duration (in seconds) value here.
Proxy Host	If the eG agent connects to the Kubernetes API on the master node via a proxy server, then provide the IP address of the proxy server here. If no proxy is used, then the default setting -*none* - of this parameter, need not

Parameter	Description
	be changed,
Proxy Port	If the eG agent connects to the Kubernetes API on the master node via a proxy server, then provide the port number at which that proxy server listens here. If no proxy is used, then the default setting -*none* - of this parameter, need not be changed,
Proxy Username, Proxy Password, Confirm Password	**These parameters are applicable only if the eG agent uses a proxy server to connect to the Kubernetes cluster, and that proxy server requires authentication.** In this case, provide a valid user name and password against the **PROXY USERNAME** and **PROXY PASSWORD** parameters, respectively. Then, confirm the password by retyping it in the **CONFIRM PASSWORD** text box. If no proxy server is used, or if the proxy server used does not require authentication, then the default setting - *none* - of these parameters, need not be changed.
DD Frequency	Refers to the frequency with which detailed diagnosis measures are to be generated for this test. The default is *1:1*. This indicates that, by default, detailed measures will be generated every time this test runs, and also every time the test detects a problem. You can modify this frequency, if you so desire. Also, if you intend to disable the detailed diagnosis capability for this test, you can do so by specifying *none* against DD frequency.
Detailed Diagnosis	To make diagnosis more efficient and accurate, the eG Enterprise suite embeds an optional detailed diagnostic capability. With this capability, the eG agents can be configured to run detailed, more elaborate tests as and when specific problems are detected. To enable the detailed diagnosis capability of this test for a particular server, choose the **On** option. To disable the capability, click on the **Off** option. The option to selectively enable/disable the detailed diagnosis capability will be available only if the following conditions are fulfilled: • The eG manager license should allow the detailed diagnosis capability • Both the normal and abnormal frequencies configured for the detailed diagnosis measures should not be 0.

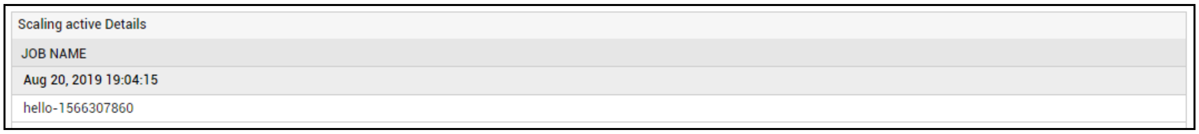
Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Completed jobs	Indicates the number of Jobs in this namespace that have completed execution.	Number	A non-parallel Job is one that creates only one Pod. Such a Job is said to have completed if that Pod terminates successfully. On the other hand, a parallel Job is one that creates multiple Pods. In the case of such Jobs, you need to specify the desired number of completions using the *completions* field in your Job specification. A parallel Job is said to have completed only if the desired number of Pods terminate successfully. A high value is desired for this measure.
Failed jobs	Indicates the number of Jobs in this namespace that failed.	Number	A Job is said to have failed if the specified number of Pods could not complete the tasks. By default, a Job will run uninterrupted unless a Pod fails (restartPolicy=Never) or a Container exits in error (restartPolicy=OnFailure). At which point, the Job will retry Pod creation. However, there are situations where you want to fail a Job after some amount of retries due to a logical error in configuration etc. To do so, set *.spec.backoffLimit* to specify the number of retries before considering a Job as failed. The back-off limit is set by default to 6. Once *.spec.backoffLimit* has been reached the Job will be marked as failed and any running Pods will be terminated. Another way to fail a Job is by setting an active deadline. Do this by setting the *.spec.activeDeadlineSeconds* field of the Job to a number of seconds. The *activeDeadlineSeconds* applies to the duration of the Job, no matter how many

Measurement	Description	Measurement Unit	Interpretation
			Pods are created. Once a Job reaches activeDeadlineSeconds, all of its running Pods are terminated and the Job status will become *type: Failed with reason: DeadlineExceeded*. Note that a Job's *.spec.activeDeadlineSeconds* takes precedence over its *.spec.backoffLimit*. Therefore, a Job that is retrying one or more failed Pods will not deploy additional Pods once it reaches the time limit specified by *activeDeadlineSeconds*, even if the *backoffLimit* is not yet reached. Ideally, the value of this measure should be 0. If the measure reports a non-zero value, then you can use the detailed diagnosis of this measure to know which Jobs failed and why.
Running pods	Indicates the number of Pods created by Jobs in this namespace, which are currently in the Running state.	Number	If a Pod is in the Running state, it means that the Pod has been bound to a node, and all of the Containers have been created. At least one Container is still running, or is in the process of starting or restarting.
Failed pods	Indicates the number of Pods created by Jobs in this namespace, which are currently in the Failed state.	Number	If a Pod is in the Failed state, it means that all Containers in the Pod have terminated, and at least one Container has terminated in failure. That is, the Container either exited with non-zero status or was terminated by the system.
Succeeded pods	Indicates the number of Pods created by Jobs in this namespace, which are currently in the Succeeded state.	Number	If a Pod is in the Succeeded state, it means that all Containers in the Pod have terminated in success, and will not be restarted.

Measurement	Description	Measurement Unit	Interpretation
Longest running jobs	Indicates the number of Jobs in this namespace that have been running for a duration greater than the value of the **JOB AGE SECONDS** parameter.	Number	Ideally, the value of this measure should be 0. If this measure reports a non-zero value, then use the detailed diagnosis of this measure to know which Jobs are executing for a long time.
Active cron jobs	Indicates the number of cron Jobs that are currently active in this namespace.	Number	A Cron Job creates Jobs on a time-based schedule. One CronJob object is like one line of a crontab (cron table) file. It runs a Job periodically on a given schedule, written in Cron format.

To know which Jobs in a namespace have been running for a long time, use the detailed diagnosis of the *Longest running jobs* measure.



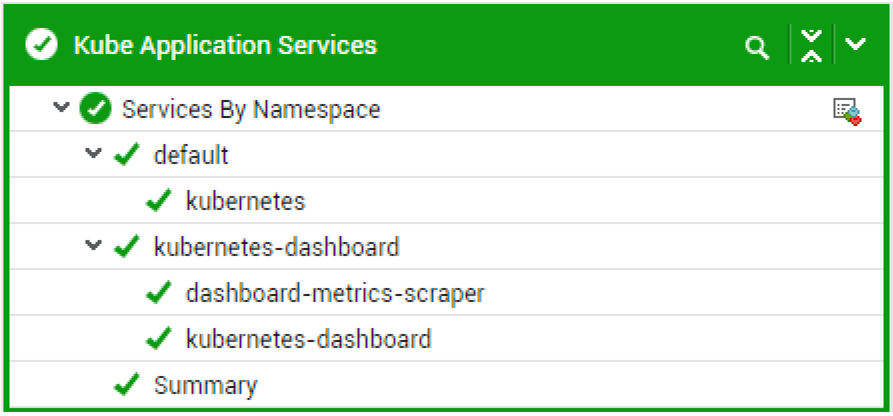
Scaling active Details

JOB NAME
Aug 20, 2019 19:04:15
hello-1566307860

Figure 3.44: The detailed diagnosis of the Longest running jobs measure

3.1.16 The Kube Application Services Layer

The tests mapped to this layer help you monitor Services, and quickly detect those Services that are not running currently.



Kube Application Services

- Services By Namespace
  - default
    - kubernetes
  - kubernetes-dashboard
    - dashboard-metrics-scraper
    - kubernetes-dashboard
  - Summary

Figure 3.45: The tests mapped to the Kube Application Services layer

3.1.17 Services by Namespace Test

In Kubernetes, a Service is an abstraction which defines a logical set of Pods and a policy by which to access them (sometimes this pattern is called a micro-service). Services enable a loose coupling between dependent Pods.

A Service is required because, Pods are mortal - they are born, and they die. In a deployment therefore, the set of Pods running in one moment in time could be different from the set of Pods running that application a moment later. This leads to a problem: if some set of Pods (call them “backends”) provides functionality to other Pods (call them “frontends”) inside your cluster, how do the frontends find out and keep track of which IP address to connect to, so that the frontend can use the backend part of the workload? This is where Services help! By associating a Service with a set of dependent pods, you can make sure that Kubernetes automatically reconciles changes among pods so that your applications continue to function.

A Service is defined using YAML (preferred) or JSON, like all Kubernetes objects. The set of Pods targeted by a Service is usually determined by a LabelSelector.

Although each Pod has a unique IP address, those IPs are not exposed outside the cluster without a Service. In fact, using Services, you can allow your applications to receive traffic from outside the cluster. By default however, a Service is accessible from within the cluster only. You can override this default setting using the *ServiceType* specification in the service definition. With the help of this specification, you can indicate where the Service should be exposed and what type of traffic (internal or external) it can receive. This means that if a Service is not up and running, then, depending upon the ServiceType, the unavailability of the Service can deny external users access to the application and can even hamper internal application operations. To assure users of continued access to their applications running in the Kubernetes cluster and to ensure peak application performance at all times, administrators should not only be able to promptly detect the non-availability of a Service, but should also be able to rapidly tell what type of Service it is and why it is not up. This is where the Services by Namespace test helps!

This test auto-discovers the Services defined within each namespace, and reports the current state, type, and age of each Service. This way, the test promptly alerts administrators if any Service is not up and running. Detailed diagnostics of the test also reveal the reason why the Service is so. Additionally, the test also reports the number and names of Pods that each Service targets and the LabelSelector used by each Service to identify the Pods. These details help in troubleshooting the abnormal state of a Service.

Target of the test : A Kubernetes Cluster

Agent deploying the test : A remote agent

Outputs of the test : One set of results for each Service in every namespace configured in the Kubernetes cluster being monitored

First-level Descriptor: Namespace

Second-level Descriptor: Service

Configurable parameters for the test

Parameter	Description
Test Period	How often should the test be executed.
Host	The IP address of the host for which this test is to be configured.
Port	Specify the port at which the specified Host listens. By default, this is *6443*.
Load Balancer / Master Node IP	To run this test and report metrics, the eG agent needs to connect to the Kubernetes API on the master node and run API commands. To enable this connection, the eG agent has to be configured with either of the following: • If only a single master node exists in the cluster, then configure the eG agent with the IP address of the master node. • If the target cluster consists of more than one master node, then you need to configure the eG agent with the IP address of the load balancer that is managing the cluster. In this case, the load balancer will route the eG agent's connection request to any available master node in the cluster, thus enabling the agent to connect with the API server on that node, run API commands on it, and pull metrics.

Parameter	Description
	By default, this parameter will display the **Load Balancer / Master Node IP** that you configured when manually adding the Kubernetes cluster for monitoring, using the **Kubernetes Cluster Preferences** page in the eG admin interface (see Figure 2.3). The steps for managing the cluster using the eG admin interface are discussed elaborately in How to Monitor the Kubernetes Cluster Using eG Enterprise? Whenever the eG agent runs this test, it uses the IP address that is displayed (by default) against this parameter to connect to the Kubernetes API. If there is any change in this IP address at a later point in time, then make sure that you update this parameter with it, by overriding its default setting.
SSL	By default, the Kubernetes cluster is SSL-enabled. This is why, the eG agent, by default, connects to the Kubernetes API via an HTTPS connection. Accordingly, this flag is set to **Yes** by default. If the cluster is not SSL-enabled in your environment, then set this flag to **No**.
Authentication Token	The eG agent requires an authentication bearer token to access the Kubernetes API, run API commands on the cluster, and pull metrics of interest. The steps for generating this token have been detailed in Section **1.1** Typically, once you generate the token, you can associate that token with the target Kubernetes cluster, when manually adding that cluster for monitoring using the eG admin interface. The steps for managing the cluster using the eG admin interface are discussed elaborately in Section **Chapter 2** By default, this parameter will display the **Authentication Token** that you provided in the **Kubernetes Cluster Preferences page** of the eG admin interface, when manually adding the cluster for monitoring (see Figure 2.3). Whenever the eG agent runs this test, it uses the token that is displayed (by default) against this parameter for accessing the API and pulling metrics. If for any reason, you generate a new authentication token for the target cluster at a later point in time, then make sure you update this parameter with the change. For that, copy the new token and paste it against this parameter.
Report System Namespace	The kube-system namespace consists of all objects created by the Kubernetes system. Monitoring such a namespace may not only increase the eG agent's processing overheads, but may also clutter the eG database. Therefore, to optimize agent performance and to conserve database space, this test, by default, excludes the kube-system namespace from monitoring. Accordingly, this flag is set to **No** by default. If required, you can set this flag to **Yes**, and enable monitoring of the kube-system namespace.

Parameter	Description
Proxy Host	If the eG agent connects to the Kubernetes API on the master node via a proxy server, then provide the IP address of the proxy server here. If no proxy is used, then the default setting -*none* - of this parameter, need not be changed,
Proxy Port	If the eG agent connects to the Kubernetes API on the master node via a proxy server, then provide the port number at which that proxy server listens here. If no proxy is used, then the default setting -*none* - of this parameter, need not be changed,
Proxy Username, Proxy Password, Confirm Password	**These parameters are applicable only if the eG agent uses a proxy server to connect to the Kubernetes cluster, and that proxy server requires authentication.** In this case, provide a valid user name and password against the **Proxy Username** and **Proxy Password** parameters, respectively. Then, confirm the password by retyping it in the **Confirm Password** text box. If no proxy server is used, or if the proxy server used does not require authentication, then the default setting - *none* - of these parameters, need not be changed.
DD Frequency	Refers to the frequency with which detailed diagnosis measures are to be generated for this test. The default is *1:1*. This indicates that, by default, detailed measures will be generated every time this test runs, and also every time the test detects a problem. You can modify this frequency, if you so desire. Also, if you intend to disable the detailed diagnosis capability for this test, you can do so by specifying *none* against DD frequency.
Detailed Diagnosis	To make diagnosis more efficient and accurate, the eG Enterprise suite embeds an optional detailed diagnostic capability. With this capability, the eG agents can be configured to run detailed, more elaborate tests as and when specific problems are detected. To enable the detailed diagnosis capability of this test for a particular server, choose the **On** option. To disable the capability, click on the **Off** option. The option to selectively enable/disable the detailed diagnosis capability will be available only if the following conditions are fulfilled: • The eG manager license should allow the detailed diagnosis capability • Both the normal and abnormal frequencies configured for the detailed diagnosis measures should not be 0.

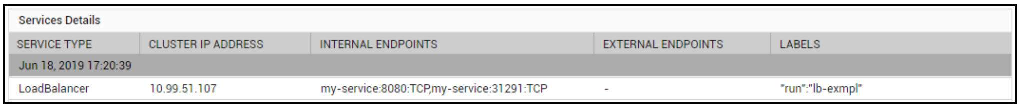
Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Service Type	Indicates the type of		The values that this measure reports and

<table>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
<tr><td></td><td>this Service.</td><td></td><td>their corresponding numeric values are detailed in the table below:
<table><tr><th>Measure Value</th><th>Numeric Value</th></tr><tr><td>ClusterIP</td><td>1</td></tr><tr><td>NodePort</td><td>2</td></tr><tr><td>ExternalName</td><td>3</td></tr><tr><td>LoadBalancer</td><td>4</td></tr></table>
Each of these types have been briefly described below:
• ClusterIP: Exposes the Service on an internal IP in the cluster. This type makes the Service only reachable from within the cluster.
• NodePort: Exposes the Service on the same port of each selected Node in the cluster using NAT. Makes a Service accessible from outside the cluster using <NodeIP>:<NodePort>. Superset of ClusterIP.
• ExternalName: Exposes the Service using an arbitrary name (specified by externalName in the spec) by returning a CNAME record with the name. No proxy is used. This type requires v1.7 or higher of kube-dns.
• LoadBalancer: Creates an external load balancer in the current cloud (if supported) and assigns a fixed, external IP to the Service. Superset of NodePort.</td></tr>
</table>

<table>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
<tr><td></td><td></td><td></td><td><b>Note:</b>
By default, this test reports the <b>Measure Value</b>s listed in the table above to indicate the Service type. In the graph of this measure however, the type is indicated using the numeric equivalents only.</td></tr>
<tr><td>Age</td><td>Indicates how old this Service is.</td><td></td><td>The value of this measure is expressed in number of days, hours, and minutes.
You can use the detailed diagnosis of this measure to know the Cluster IP on which the Service has been exposed, the LabelSelector using which the Service identifies the Pods, and the internal and external endpoints associated with the Service.</td></tr>
<tr><td>Total pods</td><td>Indicates the number of pods that this Service targets.</td><td>Number</td><td>Use the detailed diagnosis of this measure to know which Pods are targeted by the Service and which Node each Pod is running on.</td></tr>
<tr><td>Status</td><td>Indicates the current status of this Service.</td><td></td><td>The values that this measure reports and their corresponding numeric values are detailed in the table below:
<table><tr><th>Measure Value</th><th>Numeric Value</th></tr><tr><td>Running</td><td>1</td></tr><tr><td>Pending</td><td>0</td></tr></table>If the value of this measure is Pending, then you can use the detailed diagnosis of this measure to understand why the Service is in a Pending state.
<b>Note:</b>
By default, this test reports the <b>Measure Value</b>s listed in the table above to indicate the Service status. In the graph of this measure however, the status is indicated using the numeric equivalents only.</td></tr>
</table>

The detailed diagnosis of the Age measure reports the service type, the cluster IP address on which the service is exposed, the internal and external endpoints of the service, and the label selector.



Services Details

SERVICE TYPE	CLUSTER IP ADDRESS	INTERNAL ENDPOINTS	EXTERNAL ENDPOINTS	LABELS
Jun 18, 2019 17:20:39				
LoadBalancer	10.99.51.107	my-service:8080:TCP,my-service:31291:TCP	-	"run":"lb-exmpl"

Figure 3.46: The detailed diagnosis of the Age measure of the Services by Namespace test

About eG Innovations

eG Innovations provides intelligent performance management solutions that automate and dramatically accelerate the discovery, diagnosis, and resolution of IT performance issues in on-premises, cloud and hybrid environments. Where traditional monitoring tools often fail to provide insight into the performance drivers of business services and user experience, eG Innovations provides total performance visibility across every layer and every tier of the IT infrastructure that supports the business service chain. From desktops to applications, from servers to network and storage, from virtualization to cloud, eG Innovations helps companies proactively discover, instantly diagnose, and rapidly resolve even the most challenging performance and user experience issues.

eG Innovations is dedicated to helping businesses across the globe transform IT service delivery into a competitive advantage and a center for productivity, growth and profit. Many of the world's largest businesses use eG Enterprise to enhance IT service performance, increase operational efficiency, ensure IT effectiveness and deliver on the ROI promise of transformational IT investments across physical, virtual and cloud environments.

To learn more visit www.eginnovations.com.

Contact Us

For support queries, email support@eginnovations.com.

To contact eG Innovations sales team, email sales@eginnovations.com.