CONTAINER TELEMETRY OVERVIEW

Maryam Tahhan and Roman K.

Terminology

Host: some machine (physical or virtual)

Master: a host running Kubernetes API server and other master systems

Node: a host running kubelet + kube-proxy that pods can be scheduled onto. A node may be a VM or physical machine, depending on the cluster.

Cluster: a collection of one or masters + one or more nodes

Pod: a group of one or more 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.

Kubelet

The kubelet is the primary “node agent” that runs on each node.

The kubelet works in terms of a PodSpec.

A PodSpec is a YAML or JSON object that describes a pod.

The kubelet takes a set of PodSpecs that are provided through various mechanisms (primarily through the apiserver) and ensures that the containers described in those PodSpecs are running and healthy.

The kubelet doesn’t manage containers which were not created by Kubernetes.

https://kubernetes.io/docs/admin/kubelet/

STATS

Tools for Monitoring Compute, Storage, and Network Resources

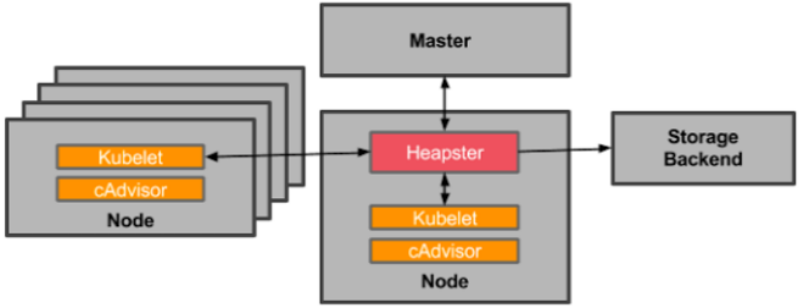
* Container events only

Heapster

- a cluster-wide aggregator of monitoring and event* data.
- runs as a pod in the cluster
- discovers all nodes in the cluster and queries usage information from the nodes' Kubelets, the on-machine Kubernetes agent.
- groups the information by pod

Kubelet fetches the data from cAdvisor (which is integrated into the Kubelet binary).

cAdvisor is a container resource usage and performance analysis agent.



https://kubernetes.io/docs/tasks/debug-application-cluster/resource-usage-monitoring/

Heapster is not a monitoring solution!

Heapster can retrieve metrics, in particular from Kubernetes components and write them into sinks. The sinks are the actual monitoring systems.

To support system components of Kubernetes Heapster calculates aggregated metrics and long term statistics, keeps them in memory and exposes via Heapster API.

This API is mainly used by Horizontal Pod Autoscaler which asks for the most recent performance related metrics to adjust the number of pods to the incoming traffic. The API is also used by KubeDash and will be used by the new UI (which will replace KubeDash) as well.

Additionally Heapster API allows to list all active nodes, namespaces, pods, containers etc. present in the system.

Heapster Metric Model

Cluster-level Metrics

Node-level Metrics

Namespace-level Metrics

Pod-level Metrics

Container-level Metrics

Heapster Metrics exposed to backend

Metric Name	Description
cpu/limit	CPU hard limit in millicores.
cpu/node_capacity	Cpu capacity of a node.
cpu/node_allocatable	Cpu allocatable of a node.
cpu/node_reservation	Share of cpu that is reserved on the node allocatable.
cpu/node_utilization	CPU utilization as a share of node allocatable.
cpu/request	CPU request (the guaranteed amount of resources) in millicores.
cpu/usage	Cumulative CPU usage on all cores.
cpu/usage_rate	CPU usage on all cores in millicores.
filesystem/usage	Total number of bytes consumed on a filesystem.
filesystem/limit	The total size of filesystem in bytes.
filesystem/available	The number of available bytes remaining in a the filesystem
filesystem/inodes	The number of available inodes in a the filesystem
filesystem/inodes_free	The number of free inodes remaining in a the filesystem
memory/limit	Memory hard limit in bytes.
memory/major_page_faults	Number of major page faults.
memory/major_page_faults_rate	Number of major page faults per second.
memory/node_capacity	Memory capacity of a node.
memory/node_allocatable	Memory allocatable of a node.
memory/node_reservation	Share of memory that is reserved on the node allocatable.

Metric Name	Description
memory/node_utilization	Memory utilization as a share of memory allocatable.
memory/page_faults	Number of page faults.
memory/page_faults_rate	Number of page faults per second.
memory/request	Memory request (the guaranteed amount of resources) in bytes.
memory/usage	Total memory usage.
memory/cache	Cache memory usage.
memory/rss	RSS memory usage.
memory/working_set	Total working set usage. Working set is the memory being used and not easily dropped by the kernel.
network/rx	Cumulative number of bytes received over the network.
network/rx_errors	Cumulative number of errors while receiving over the network.
network/rx_errors_rate	Number of errors while receiving over the network per second.
network/rx_rate	Number of bytes received over the network per second.
network/tx	Cumulative number of bytes sent over the network
network/tx_errors	Cumulative number of errors while sending over the network
network/tx_errors_rate	Number of errors while sending over the network
network/tx_rate	Number of bytes sent over the network per second.
uptime	Number of milliseconds since the container was started.

Kubernetes Custom Metrics

In Kubernetes there is a way for a container to expose arbitrary application-specific metrics to cAdvisor, which are in turn consumed by kubelet's stats/summary API.

To expose custom metrics, do the following:

- Turn on *--enable-custom-metrics* on each kubelet.
- Instrument your application using the Prometheus client library, so that metrics are exported via the */metrics* HTTP endpoint.
- Map the container's port to a hostPort in the pod definition.
- In the pod definition, mount a volume at */etc/custom-metrics* for the container, and the volume should contain a file named *definition.json*.
- definition.json should have the following content:

```
{
  "endpoint": "http://localhost:<hostport>/metrics",
  "metrics_config": [
    "<metric_name_foo>",
    "<metric_name_bar>",
    "..."
  ]
}
```

Kubernetes Custom Metrics

How it works:

- When you turn on *--enable-custom-metrics*, kubelet looks for the following when starting a container:
  - Does it have a volume mounted at */etc/custom-metrics*?
  - Is there a file named *definition.json* in the volume?
- If both are true, it adds docker label *io.cadvisor.metric.prometheus=/etc/custom-metrics/definition.json* to the container. You can verify by looking at the output of *docker inspect <container_id>*.
- This label signals to cAdvisor that this container has custom metrics in prometheus format, and that the detail for how to retrieve the metrics are defined in that file inside the container.
- cAdvisor parses the file, and periodically issues HTTP requests to the endpoint defined in the file. Since cAdvisor runs in the host network namespace, the HTTP queries will only work if the port is open on the host, hence the need for hostPort.
- **Per-pod, can be consumed by the horizontal pod autoscaler**

kube-state-metrics service

kube-state-metrics is a service that provides additional metrics that Heapster does not. Heapster exposes metrics about the resource utilization of components such as CPU, memory, or network. On the other hand, it listens to the Kubernetes API and generates metrics about the state of Kubernetes logical objects: node status, node capacity (CPU and memory), number of desired/available/unavailable/updated replicas per deployment, pod status (e.g. waiting, running, ready), and so on.

cAdvisor REST API

Machine Information *</api/vX.Y/machine>*

- Number of schedulable logical CPU cores
- Memory capacity (in bytes)
- Maximum supported CPU frequency (in kHz)
- Available filesystems: major, minor numbers and capacity (in bytes)
- Network devices: mac addresses, MTU, and speed (if available)
- Machine topology: Nodes, cores, threads, per-node memory, and caches

Container Information *</api/v1.0/containers/<absolute container name>*

- Absolute container name
- List of subcontainers
- ContainerSpec which describes the resource isolation enabled in the container
- Detailed resource usage statistics of the container for the last N seconds (N is globally configurable in cAdvisor)
- Histogram of resource usage from the creation of the container

cAdvisor REST API

Subcontainer Information *</api/v1.1/subcontainers/<absolute container name>*

- Returns the information of the specified container and all subcontainers (recursively).

Docker Container Information *</api/v1.2/docker/<Docker container name or blank for all Docker containers>*

- Returns the information of the specified container(s).

Events *</api/v1.3/events/<absolute container name>*

Parameter	Description	Default
start_time	Start time of events to query (for stream=false)	Beginning of time
end_time	End time of events to query (for stream=false)	Now
stream	Whether to stream new events as they occur. If false returns historical events	false
subcontainers	Whether to also return events for all subcontainers	false
max_events	The max number of events to return (for stream=false)	10
all_events	Whether to include all supported event types	false
oom_events	Whether to include OOM events	false
oom_kill_events	Whether to include OOM kill events	false
creation_events	Whether to include container creation events	false
deletion_events	Whether to include container deletion events	false

cAdvisor REST API v2.0

- Machine Information < */api/v2.0/machine* >
- Attributes < */api/v2.0/attributes* >
  Attributes endpoint provides hardware and software attributes of the running machine.
- Container Stats < */api/v2.0/stats/<container identifier* >
- Container Stats Summary < */api/v2.0/summary/<container identifier* >
  It provides the latest collected stats and percentiles (max, average, and 90%ile) values for usage in last minute and hour.
- Container Spec < */api/v2.0/spec/<container identifier* >

Container Monitoring Alternative: Prometheus

Prometheus is an open source monitoring and alerting toolkit made for monitoring applications in clusters. Its main features are:

- a multi-dimensional data model (time series identified by metric name and key/value pairs)
- a flexible query language to leverage this dimensionality
- no reliance on distributed storage; single server nodes are autonomous
- time series collection happens via a pull model over HTTP
- pushing time series is supported via an intermediary gateway
- targets are discovered via service discovery or static configuration
- multiple modes of graphing and dashboarding support
- Support for using the Kubernetes API itself to discover cluster services and monitoring targets.

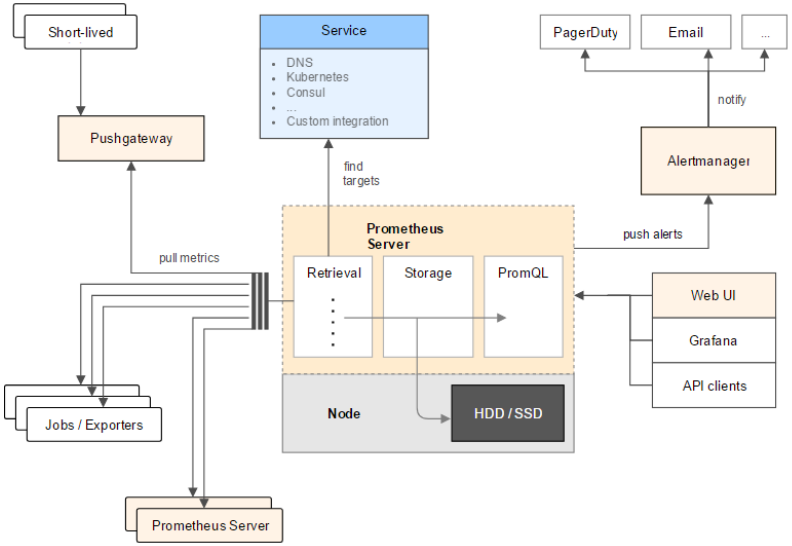
When does it fit?/When does it not fit?

Prometheus works well for recording any purely numeric time series. It fits both machine-centric monitoring as well as monitoring of highly dynamic service-oriented architectures

Prometheus is designed for reliability, to be the system you go to during an outage to allow you to quickly diagnose problems.

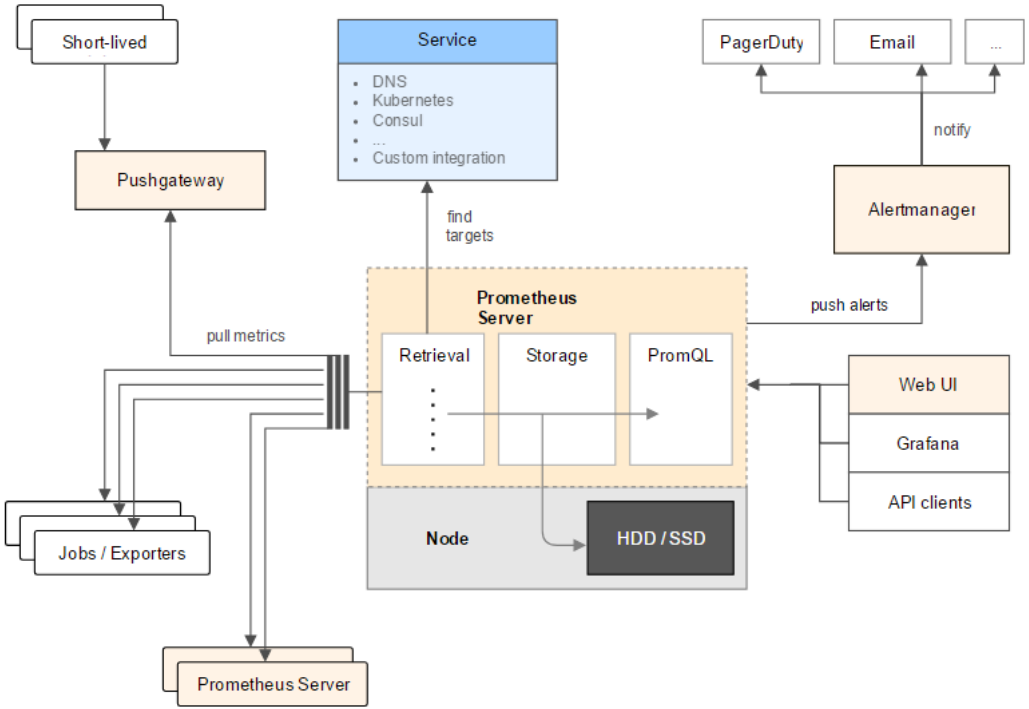
Prometheus is not a good choice for collecting billing data.

Prometheus Architecture



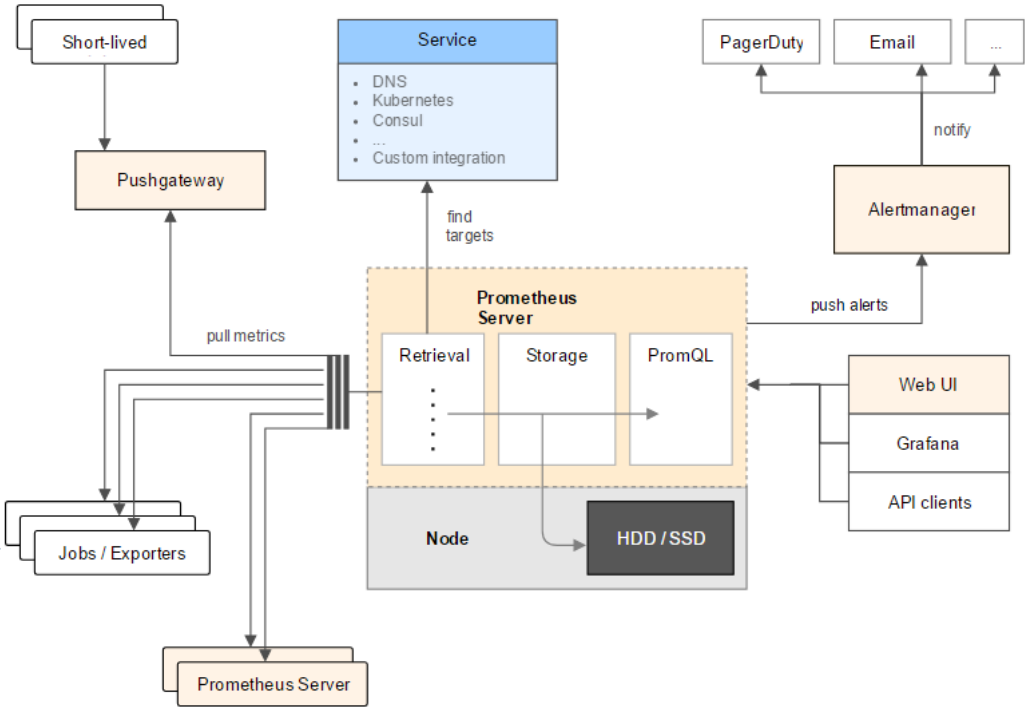
- The main Prometheus server which scrapes and stores time series data
- Client libraries for instrumenting application code
- A push gateway for supporting short-lived jobs
- Special-purpose exporters (for HAProxy, StatsD, Graphite, etc.)
- An alerts manager
- Various support tools

Prometheus Architecture



- Pull model only
- Storage is local
- Events are generated based on thresholds

Prometheus Architecture



Collectd has a native Prometheus plugin

All the platform stats can be pulled into Prometheus today without any additional development

Prometheus does not support

- raw log / event collection
- request tracing
- “magic” anomaly detection
- durable long-term storage
- automatic horizon scaling
- User / auth management

Monitoring cAdvisor with Prometheus

cAdvisor exposes container statistics as Prometheus metrics out of the box. By default, these metrics are served under the **/metrics** HTTP endpoint. This endpoint may be customized by setting the *-prometheus_endpoint* command-line flag.

To monitor cAdvisor with Prometheus, simply configure one or more jobs in Prometheus which scrape the relevant cAdvisor processes at that metrics endpoint.

cAdvisor Prometheus API

cAdvisor supports Prometheus natively.

http://localhost:4194/metrics - will have metrics for all your containers

If you're not already running it, you can do:

docker run -v /var/run/:/var/run -v /sys:/sys -p 4194:8080 google/cadvisor

Heapster vs Prometheus

	Prometheus	Heapster
Kubernetes Dashboard	-	+
Horizontal POD Autoscaler	-	+
Custom metrics	Include third-party metrics	Application metrics support is in Beta
Data model	Query language	REST API
cAdvisor integration	+	+
Alerts	Configurable rules – Can manually scale through webhooks	-
Storage and Visualization	+	-

Prometheus Alerts and notifications

Alerting with Prometheus is separated into two parts:

1. Alerting rules in Prometheus servers send alerts to an Alertmanager.
2. 2. The Alertmanager then manages those alerts, including silencing, inhibition, aggregation and sending out notifications via methods such as email, PagerDuty and HipChat.

Prometheus creates and sends alerts to the Alertmanager which then sends notifications out to different receivers based on their labels.

A receiver can be one of many integrations including: Slack, PagerDuty, email, or a custom integration via the generic webhook interface.

The Alertmanager handles alerts sent by client applications such as the Prometheus server. It takes care of deduplicating, grouping, and routing them to the correct receiver

EVENTS

Monitor Node Health with node problem detector

node-problem-detector aims to make various node problems visible to the upstream layers in cluster management stack.

It is a daemon which runs on each node, detects node problems and reports them to apiserver.

It collects node problems from various daemons and reports them to the apiserver as NodeCondition and Event.

Currently these problems are invisible to the upstream layers in cluster management stack, so Kubernetes will continue scheduling pods to the bad nodes.

node-problem-detector reports problems to apiserver.

Key Concepts

Node Problem: Node problems is the problem occurs to the node that may affect Kubernetes stability, includes:

- Hardware problem. (Bad cpu, bad memory, bad disk, bad gpu etc.)
- Kernel problem. (Kernel deadlock, corrupted filesystems etc.)
- Runtime issue. (Unresponsive docker daemon, zombie docker daemon etc.)
- Other problems. (Other problems need attention in the future)

Problem Daemon: Problem daemon monitors a specific kind of node problems, such as disk problem monitor, kernel problem monitor, runtime problem monitor etc. The problem daemon:

- could be tools dedicated for Kubernetes' use case.
- could also be an existing node health monitoring solution.

Problem Daemon

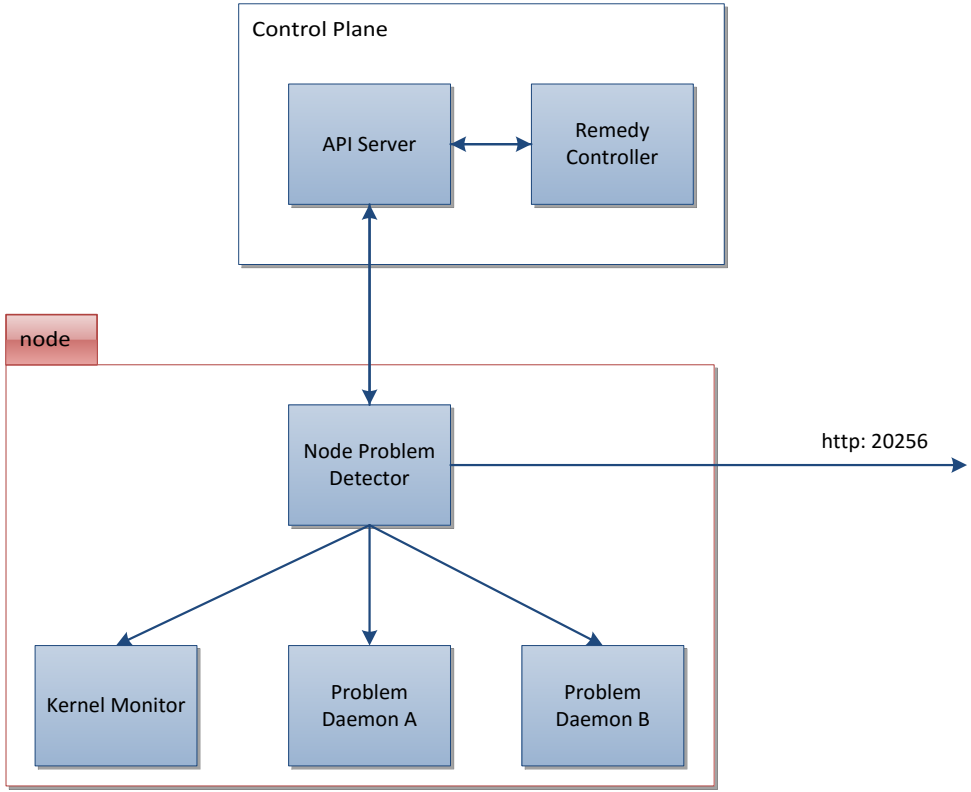
A problem daemon could be:

- A tiny daemon designed for dedicated usecase of Kubernetes.
- An existing node health monitoring daemon integrated with node-problem-detector.

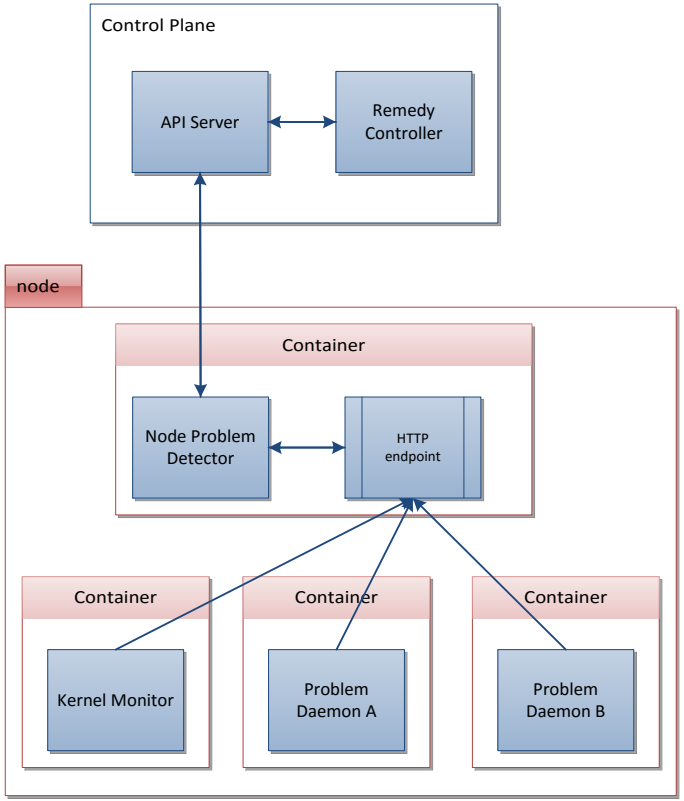
List of supported problem daemons:

- ***KernelMonitor*** a system log monitor monitors kernel log and reports problem according to predefined rules.
- ***AbrtAdaptor*** monitor ABRT log messages and report them further. ABRT (Automatic Bug Report Tool) is health monitoring daemon able to catch kernel problems as well as application crashes of various kinds occurred on the host.

Node Problem Daemon Current Design



Node Problem Daemon Future Design



Problem Types

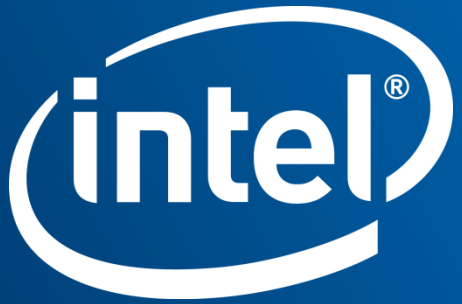
Based on the severity, node problems are currently divided into 2 categories:

- **Permanent**: Problems that impact long enough that the node is unavailable for hosting some kind of pods. NodeProblemDetector will report permanent problem as a **NodeCondition**.
- **Temporary**: Problems that have limited impact on pod hosting, but are informative and useful. NodeProblemDetector will report temporary problem as an **Event**.

Events and Conditions internals

Conditions	Description	Example
source	Source is the name of the problem daemon	collectd
type	Type is the condition type. It should describe the condition of node in problem.	KernelDeadlock, OutOfResource
status	Status indicates whether the node is in the condition or not.	true/false
transition	Transition is the time when the node transits to this condition.	
reason	Reason is a short reason of why node goes into this condition.	UnregisterNetDevice, TaskHung
message	Message is a human readable message of why node goes into this condition.	unable to handle kernel NULL pointer at

Events	Description	Example
source	Source is the name of the problem daemon	collectd
severity	Severity is the severity level of the event.	info, warning
timestamp	Timestamp is the time when the event is generated.	
reason	Reason is a short reason of why the event is generated.	XorgCrash, UncaughtException
message	Message is a human readable message of why the event is generated.	


intel®

https://github.com/prometheus/docs/blob/master/content/docs/introduction/media.md

Heapster Metric Model

- Cluster-level Metrics
  /api/v1/model/metrics/: Returns a list of available cluster-level metrics
- Node-level Metrics
  /api/v1/model/nodes/ : Returns a list of all available nodes.
  /api/v1/model/nodes/{node-name}/metrics/ : Returns a list of available node-level metrics.
- Namespace-level Metrics
  /api/v1/model/namespaces/ : Returns a list of all available namespaces.
  /api/v1/model/namespaces/{namespace-name}/metrics/ : Returns a list of available namespace-level metrics.
- Pod-level Metrics
  /api/v1/model/namespaces/{namespace-name}/pods/ : Returns a list of all available pods under a given namespace.
  /api/v1/model/namespaces/{namespace-name}/pods/{pod-name}/metrics/ : Returns a list of available pod-level metrics
- Container-level Metrics
  /api/v1/model/namespaces/{namespace-name}/pods/{pod-name}/containers/{container-name}/metrics/ : Returns a list of available container-level metrics
  /api/v1/model/nodes/{node-name}/freecontainers/{container-name}/metrics/ : Returns a list of available container-level metrics

Heapster Metric Labels

Heapster tags each metric with the following labels.

Label Name	Description
pod_id	Unique ID of a Pod
pod_name	User-provided name of a Pod
pod_namespace	The namespace of a Pod
container_base_image	Base image for the container
container_name	User-provided name of the container or full cgroup name for system containers
host_id	Cloud-provider specified or user specified Identifier of a node
hostname	Hostname where the container ran
labels	Comma-separated(Default) list of user-provided labels. Format is 'key:value'
namespace_id	UID of the namespace of a Pod
resource_id	A unique identifier used to differentiate multiple metrics of the same type.

Prometheus Data Model