

Bleemeo



kubernetes

Webinar

Monitoring Kubernetes

 10 Juin 2022

 11h00



Lionel Porcheron

CEO & co-fondateur de Bleemeo

bleemeo.com



Who am I?

Lionel Porcheron, CEO & co-founder Bleemeeo

- Ops background, managing 500+ machines in classical DC and in the Cloud
- DevOps for +15 years (started my monitoring journey with nagios-netsaint)
- Toulouse DevOps Meetup Leader, Capitole du Libre Leader

Bleemeo?

Observability & Monitoring as a service solution

Monitor your Servers, Containers, and applications in 30s

Prometheus, Graphite, StatsD, Nagios, compatible

2 Open Source projects (<https://github.com/bleemeo>):

- **Glouton**, universal monitoring agent written in Go with Prometheus, StatsD, Graphite, Nagios compatibility
- **SquirrelDB**, a scalable Prometheus compatible storage backend based on Cassandra



Kubernetes “k8s”

- Reference container orchestration platform
- Project has been initiated by Google and is now hosted by Cloud Native Computing Foundation (CNCF)
- Written in Go
- You can deploy it on your infrastructure, but vendors also offer managed Kubernetes: AWS, Azure, Google Cloud Platform, OVH, Scaleway, ...



Kubernetes Monitoring Challenges

- Kubernetes rely on a lot of components: API, kubelets, scheduler, controller manager, etc ...
- Nodes number in a Kubernetes cluster can be increasing/decreasing on demand
- Container in Kubernetes are short life: you can start/stop project, upgrade them, ...
- Workload running on Kubernetes are moving parts. K8s decide the best place at each deployment
- Kubernetes has a built-in container self healing solution
- Resources running in Kubernetes are usually only accessible from the Kubernetes



Kubernetes is complex...
... monitoring Kubernetes is a project itself

Key Metrics / “Golden Signals”

The RED Method

- (Request) **R**ate - the number of requests, per second, your services are serving.
- (Request) **E**rrors - the number of failed requests per second.
- (Request) **D**uration - distributions of the amount of time each request takes.

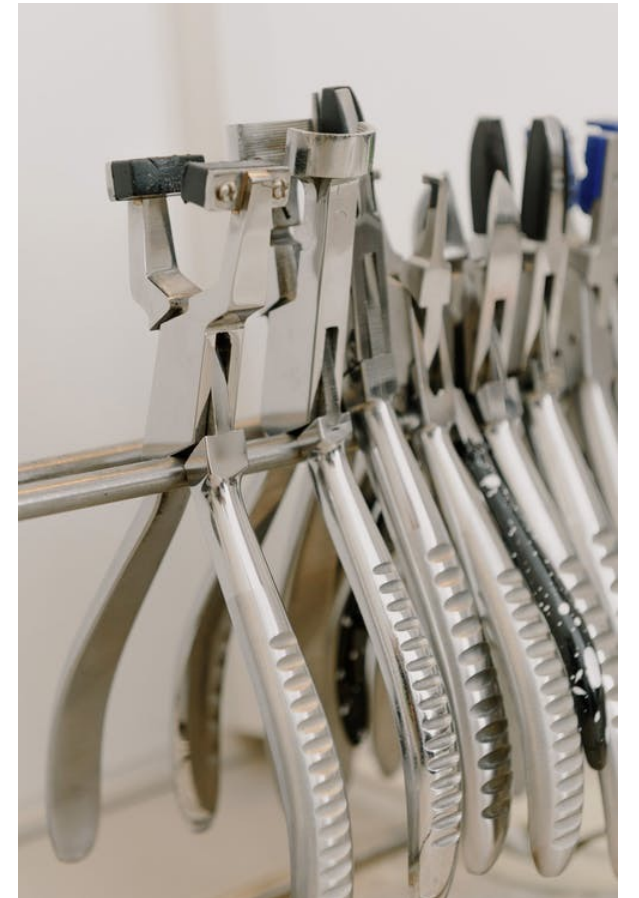
The USE Method

- (Resource) **U**tilization: as a percent over a time interval. e.g., "one disk is running at 90% utilization".
- (Resource) **S**aturation: as a queue length. e.g., "the CPUs have an average run queue length of four".
- (Resource) **E**rrors: scalar counts. e.g., "this network interface has had fifty late collisions".



Monitoring Tools for Kubernetes

- Nagios & similar tools work with a “static” inventory and are not designed for containers and dynamic workloads
- Prometheus has been designed for this & heavily rely on labels usage
- SaaS tools like Bleemeo & others are also designed for Containers & Kubernetes
- Kubernetes monitoring is usually deployed inside Kubernetes (except Cloud solutions)
- Principle: measure everything, get metrics for everything you can
- Don't forget notifications (for what require human attention !)



Use Prometheus (format)

Prometheus in a nutshell

- Prometheus was "initiated" in 2012 at Soundcloud and is now a (graduated) CNCF project
- Prometheus became de-facto standard for monitoring
- A Time Series Database where data is identified by metric name and labels (key/value pairs)
- A powerful PromQL query language
- No complex storage: designed to store multiple days (not weeks) of data
- Data are collected via a poll over HTTP
- A rich ecosystem with exporters (to get metrics) and web panels (query & display)

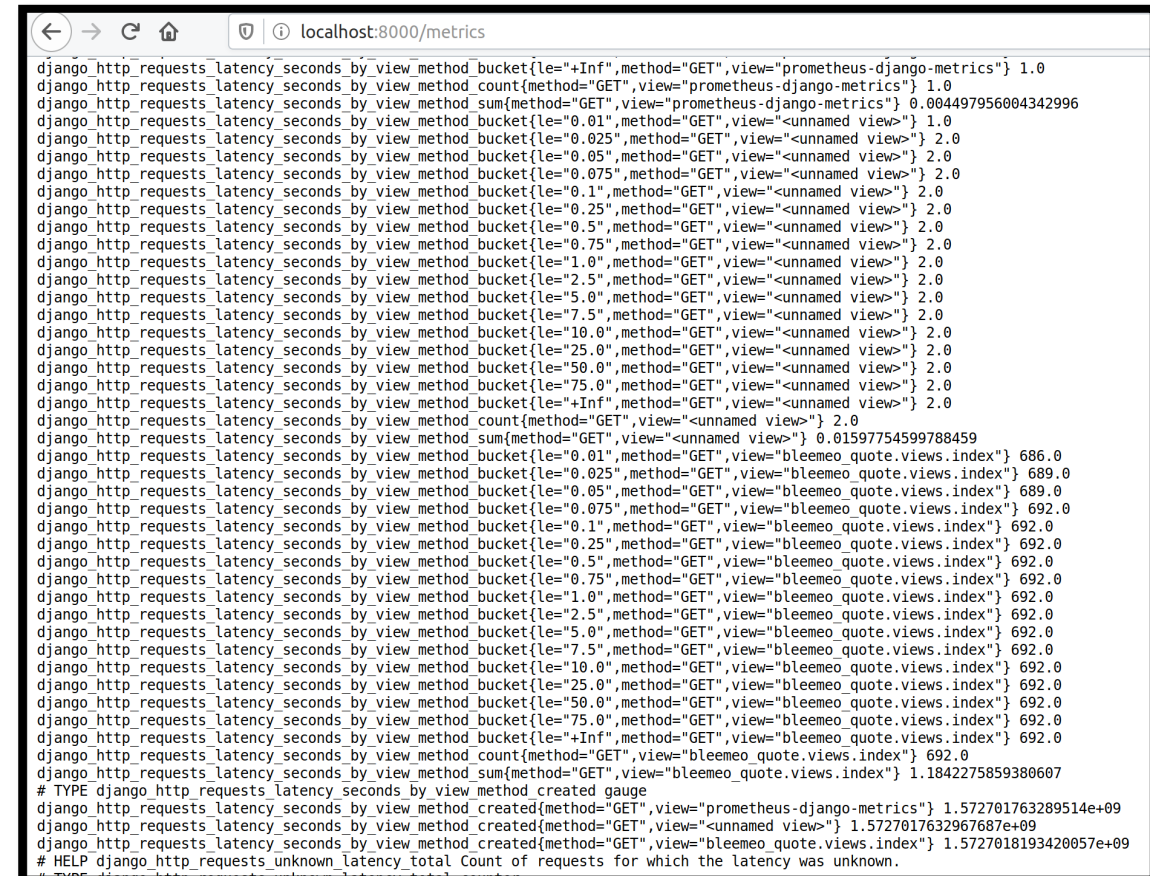


Prometheus

Prometheus metrics

- Prometheus metrics endpoint is a plain text "web page"
- Human readable
- Scraped by a Prometheus server
- Data can be queried with PromQL
- Can be used by Prometheus ecosystem: Alert Manager, Grafana...

```
global:
  scrape_interval: 5s
  scrape_configs:
    - job_name: "node-application-monitoring-app"
      static_configs:
        - targets: ["docker.host:8080"]
```



```
← → ↻ 🏠 ⓘ localhost:8000/metrics
django_http_requests_latency_seconds_by_view_method_bucket{le="+Inf",method="GET",view="prometheus-django-metrics"} 1.0
django_http_requests_latency_seconds_by_view_method_count{method="GET",view="prometheus-django-metrics"} 1.0
django_http_requests_latency_seconds_by_view_method_sum{method="GET",view="prometheus-django-metrics"} 0.004497956004342996
django_http_requests_latency_seconds_by_view_method_bucket{le="0.01",method="GET",view="<unnamed view>"} 1.0
django_http_requests_latency_seconds_by_view_method_bucket{le="0.025",method="GET",view="<unnamed view>"} 2.0
django_http_requests_latency_seconds_by_view_method_bucket{le="0.05",method="GET",view="<unnamed view>"} 2.0
django_http_requests_latency_seconds_by_view_method_bucket{le="0.075",method="GET",view="<unnamed view>"} 2.0
django_http_requests_latency_seconds_by_view_method_bucket{le="0.1",method="GET",view="<unnamed view>"} 2.0
django_http_requests_latency_seconds_by_view_method_bucket{le="0.25",method="GET",view="<unnamed view>"} 2.0
django_http_requests_latency_seconds_by_view_method_bucket{le="0.5",method="GET",view="<unnamed view>"} 2.0
django_http_requests_latency_seconds_by_view_method_bucket{le="0.75",method="GET",view="<unnamed view>"} 2.0
django_http_requests_latency_seconds_by_view_method_bucket{le="1.0",method="GET",view="<unnamed view>"} 2.0
django_http_requests_latency_seconds_by_view_method_bucket{le="2.5",method="GET",view="<unnamed view>"} 2.0
django_http_requests_latency_seconds_by_view_method_bucket{le="5.0",method="GET",view="<unnamed view>"} 2.0
django_http_requests_latency_seconds_by_view_method_bucket{le="7.5",method="GET",view="<unnamed view>"} 2.0
django_http_requests_latency_seconds_by_view_method_bucket{le="10.0",method="GET",view="<unnamed view>"} 2.0
django_http_requests_latency_seconds_by_view_method_bucket{le="25.0",method="GET",view="<unnamed view>"} 2.0
django_http_requests_latency_seconds_by_view_method_bucket{le="50.0",method="GET",view="<unnamed view>"} 2.0
django_http_requests_latency_seconds_by_view_method_bucket{le="75.0",method="GET",view="<unnamed view>"} 2.0
django_http_requests_latency_seconds_by_view_method_bucket{le="+Inf",method="GET",view="<unnamed view>"} 2.0
django_http_requests_latency_seconds_by_view_method_count{method="GET",view="<unnamed view>"} 2.0
django_http_requests_latency_seconds_by_view_method_sum{method="GET",view="<unnamed view>"} 0.01597754599788459
django_http_requests_latency_seconds_by_view_method_bucket{le="0.01",method="GET",view="bleemo quote.views.index"} 686.0
django_http_requests_latency_seconds_by_view_method_bucket{le="0.025",method="GET",view="bleemo quote.views.index"} 689.0
django_http_requests_latency_seconds_by_view_method_bucket{le="0.05",method="GET",view="bleemo quote.views.index"} 689.0
django_http_requests_latency_seconds_by_view_method_bucket{le="0.075",method="GET",view="bleemo quote.views.index"} 692.0
django_http_requests_latency_seconds_by_view_method_bucket{le="0.1",method="GET",view="bleemo quote.views.index"} 692.0
django_http_requests_latency_seconds_by_view_method_bucket{le="0.25",method="GET",view="bleemo quote.views.index"} 692.0
django_http_requests_latency_seconds_by_view_method_bucket{le="0.5",method="GET",view="bleemo quote.views.index"} 692.0
django_http_requests_latency_seconds_by_view_method_bucket{le="0.75",method="GET",view="bleemo quote.views.index"} 692.0
django_http_requests_latency_seconds_by_view_method_bucket{le="1.0",method="GET",view="bleemo quote.views.index"} 692.0
django_http_requests_latency_seconds_by_view_method_bucket{le="2.5",method="GET",view="bleemo quote.views.index"} 692.0
django_http_requests_latency_seconds_by_view_method_bucket{le="5.0",method="GET",view="bleemo quote.views.index"} 692.0
django_http_requests_latency_seconds_by_view_method_bucket{le="7.5",method="GET",view="bleemo quote.views.index"} 692.0
django_http_requests_latency_seconds_by_view_method_bucket{le="10.0",method="GET",view="bleemo quote.views.index"} 692.0
django_http_requests_latency_seconds_by_view_method_bucket{le="25.0",method="GET",view="bleemo quote.views.index"} 692.0
django_http_requests_latency_seconds_by_view_method_bucket{le="50.0",method="GET",view="bleemo quote.views.index"} 692.0
django_http_requests_latency_seconds_by_view_method_bucket{le="75.0",method="GET",view="bleemo quote.views.index"} 692.0
django_http_requests_latency_seconds_by_view_method_bucket{le="+Inf",method="GET",view="bleemo quote.views.index"} 692.0
django_http_requests_latency_seconds_by_view_method_count{method="GET",view="bleemo quote.views.index"} 692.0
django_http_requests_latency_seconds_by_view_method_sum{method="GET",view="bleemo quote.views.index"} 1.1842275859380607
# TYPE django_http_requests_latency_seconds_by_view_method_created gauge
django_http_requests_latency_seconds_by_view_method_created{method="GET",view="prometheus-django-metrics"} 1.572701763289514e+09
django_http_requests_latency_seconds_by_view_method_created{method="GET",view="<unnamed view>"} 1.5727017632967687e+09
django_http_requests_latency_seconds_by_view_method_created{method="GET",view="bleemo quote.views.index"} 1.5727018193420057e+09
# HELP django_http_requests_unknown_latency_total Count of requests for which the latency was unknown.
```

Monitor Kubernetes
from inside
&
from outside

Monitoring Kubernetes itself

You should monitor in your k8s cluster nodes:

- Resource utilization: CPU, memory, I/O, disk space
- k8s components are running correctly: kubelet, api
- Certificates validity (by default, certificates are valid 1 year)
- Number of nodes in the cluster
- Number of pods in the cluster
- Usually rely on a DaemonSet container
- kube-state-metric project from Kubernetes is a good base for k8s metrics

Monitoring Services in k8s (Pods)

- Your monitoring should be using Service Discovery to find Pods
- Monitor resource utilization: CPU, memory, I/O, disk space
- Monitor usage (rate, response time, errors code)
- Application metrics
- Use Prometheus ecosystem to monitor your services & applications
- Instrument your code using Prometheus SDK
- Define proper tags in your metrics to be able to query them later
- Don't forget in Prometheus paradigm that a label modification implies new metric

Monitor service globally

- Monitor service availability from outside of Kubernetes
- Use external probes to monitor your service
- Monitor service usage globally
- Use open source project: Blackbox exporter (for e.g.) or Cloud solutions: Uptime Robot, Bleemeo



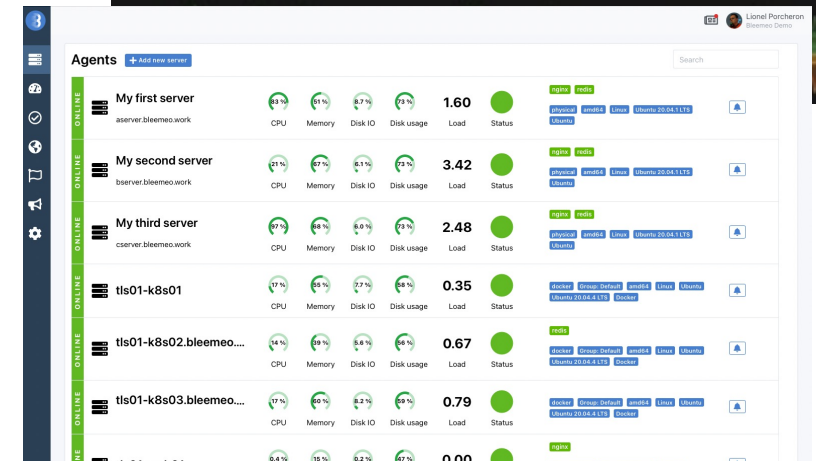
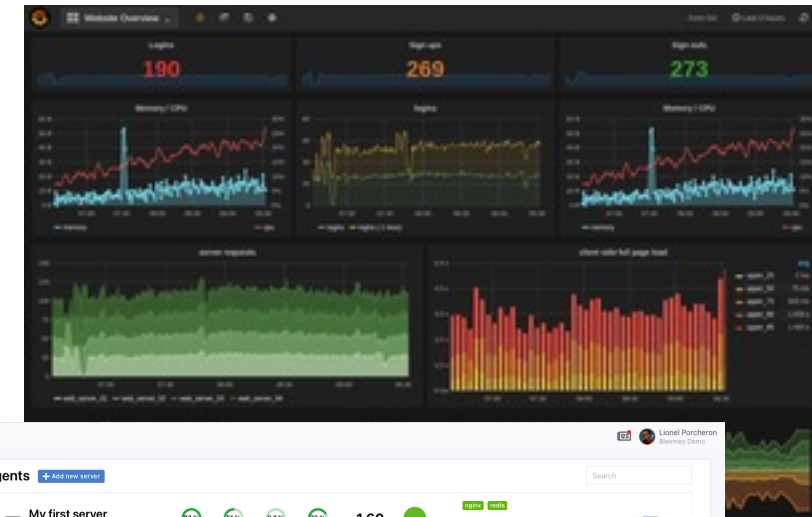
Overloading your Kubernetes

- When something wrong happens, all team members open monitoring hosted in k8s and generate extra load...
- Monitoring can be resource consuming
- Focus on golden signals and important metrics
- Pay attention when designing your dashboards



Building your dashboards

- If you use Prometheus, usual solution is Grafana
- You can find dashboards templates
- You can build your dashboards
- Prioritize golden signals of all cluster for your default dashboard
- Have detailed dashboards to go deeper for each node/Pod
- Cloud tools like Bleemeo offer automatic dashboarding



Agent Name	CPU	Memory	Disk IO	Disk usage	Load	Status	Details
My first server aserver.bleemeo.work	63%	61%	6.3%	73%	1.60	Online	physical amd64 Linux Ubuntu 20.04.4 LTS
My second server bserver.bleemeo.work	61%	67%	6.1%	71%	3.42	Online	physical amd64 Linux Ubuntu 20.04.4 LTS
My third server csserver.bleemeo.work	67%	64%	6.0%	71%	2.48	Online	physical amd64 Linux Ubuntu 20.04.4 LTS
tts01-k8s01	17%	64%	2.7%	4%	0.35	Online	docker amd64 Linux Ubuntu 20.04.4 LTS
tts01-k8s02.bleemeo....	14%	69%	6.8%	4%	0.67	Online	docker amd64 Linux Ubuntu 20.04.4 LTS
tts01-k8s03.bleemeo....	17%	60%	6.2%	4%	0.79	Online	docker amd64 Linux Ubuntu 20.04.4 LTS
tts01-work01	6.4%	15%	0.2%	0%	0.00	Online	

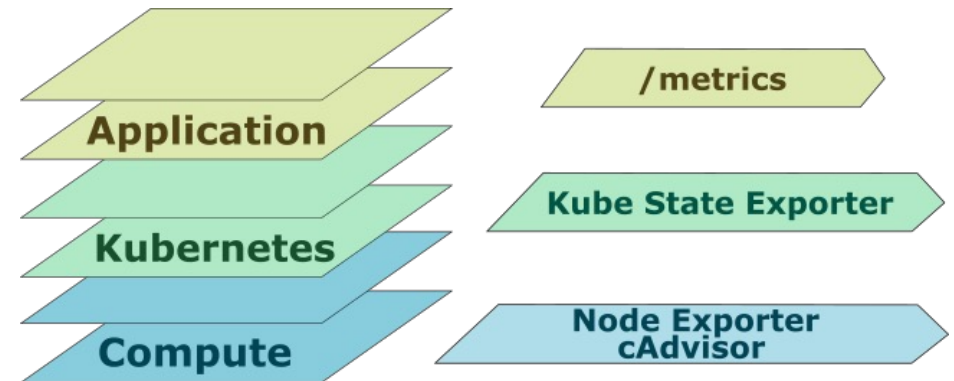
Don't forget alerting

- Golden signals are a good source of alerts
- Alerting means immediate attention is required
- Notify only when human action is required
- Check your dashboards



Server to apps coverage

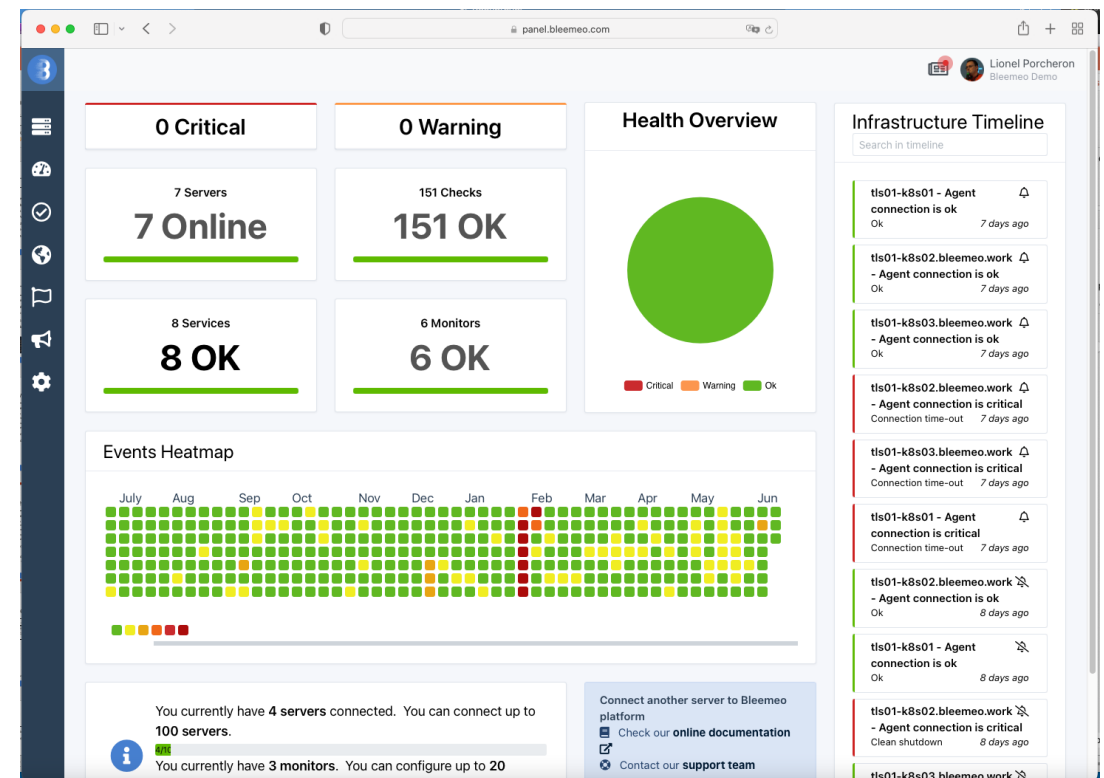
- Allow to correlate usage metrics and system metrics
- Allow to diagnose issue with business and technical data
- Prometheus can easily scrape metrics from:
 - Servers (node exporter, cAdvisor, Bleemeo Glouton)
 - Services (services exporters)
 - Kubernetes (kube-state-metric)
 - Application (instrumenting your code)



Bleemeo simplifies and automates
monitoring deployments

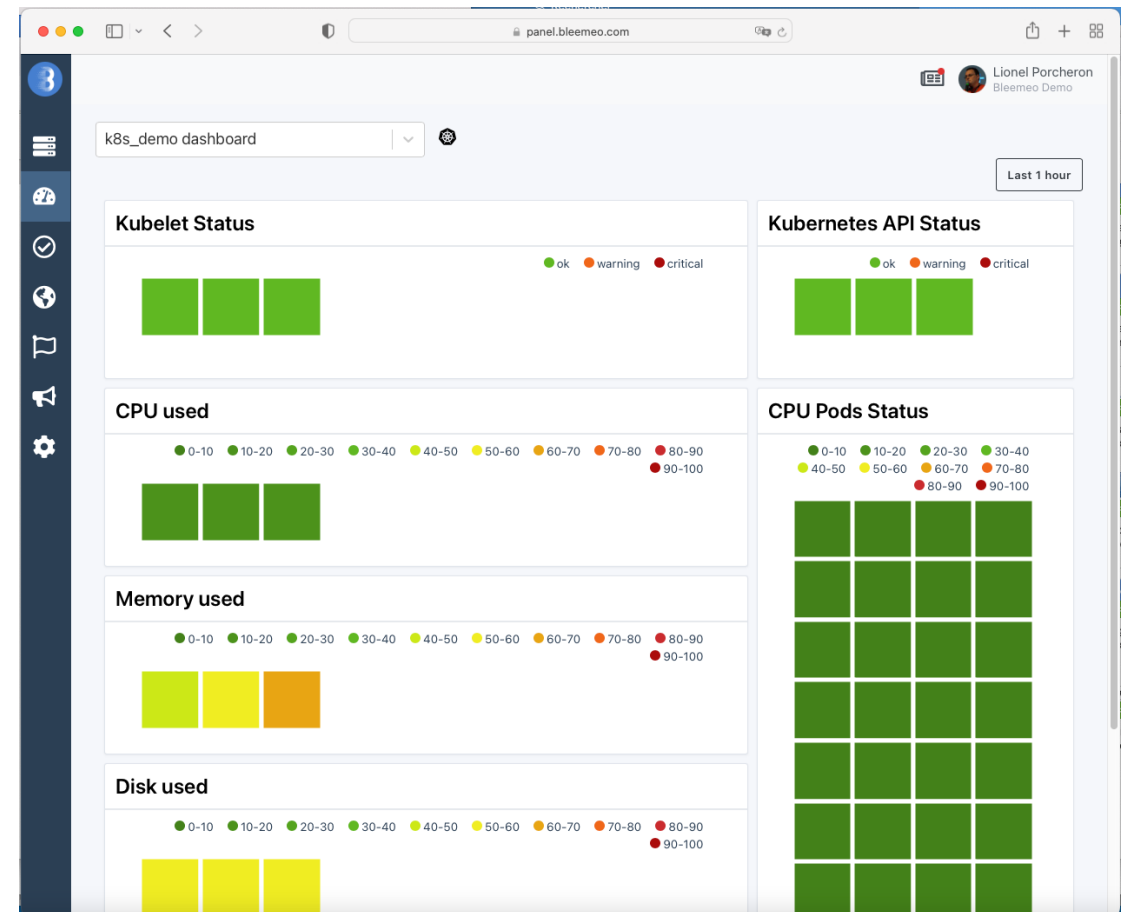
Bleemeo Monitoring Solution

- Cloud based solution
- Compatible with Prometheus and market standards (StatsD, Nagios, ...)
- Agent run on each server and discover services, containers and create dashboards for you
- Alerting with Slack, Teams, mail, SMS, ...
- Query data (on dashboards and alerting) with standard PromQL
- Mobile application for iOS and Android



k8s Monitoring with Bleemeo

- k8s dashboard is automatically build
- Health map of your cluster with:
 - Status of API & kubelets on each node
 - CPU utilization for all nodes of the cluster
 - Memory utilization for all nodes of the cluster
 - Disk utilization for all nodes of the cluster
 - CPU utilization for all Pods
- Configured by setting an environment variable
- Can be used in hosted & managed deployments



Conclusion

- When deploying Kubernetes, monitoring your cluster is a project by itself
- Use Prometheus format to collect and query your metrics
- Use golden signals metrics
- Do not underestimate load involved by deploying a monitoring solution on your k8s cluster
- Use managed solutions if you don't want/have the time to deploy & maintain complex stuff

Questions?

👉 Try for Free **Bleemeo**

<https://bleemeo.com/trial>

Voucher: WEBINARK8S2022 until next Friday