MONITOR YOUR KUBERNETES APPLICATIONS LIKE A BOSS

Applications running on Kubernetes can be difficult to manage and monitor. A single point of failure anywhere throughout your distributed environment can stop the entire process and have a significant impact on user experience, revenue, and brand reputation. Autonomous monitoring is the most effective way to stay on top of your Kubernetes ecosystem.

anodot

NEW TECHNOLOGY, NEW MONITORING CHALLENGES

If you're already running Kubernetes, you are on the right track with your organization's progress towards IT automation. Today, Kubernetes is the de facto standard for container orchestration. It enables you to automate the microservices lifecycle, scale on demand, and self-remediate. From a business perspective, Kubernetes can help you reduce operational costs and increase the efficiency of your engineering teams.

Monitoring distributed environments has never been easy. While solving some of the key challenges involved in running distributed microservices at large, Kubernetes has also introduced some new ones. The growing adoption of microservices makes logging and monitoring more trying since it involves a large number of distributed and diversified applications constantly communicating with each other. On one hand, a single glitch can kill the entire process. On the other hand, identifying failures is becoming more difficult. It's not surprising that engineers list monitoring as a major obstacle for adopting Kubernetes.

Multi-layered Complexity

Monitoring Kubernetes environments involves staying on top of the Kube system, Kubernetes (infra), and the applications themselves. At the system level, monitoring pertains to the health of the entire Kubernetes system, including resource utilization status, how many applications are running on each node, and whether all the nodes are working properly and at what capacity.

Additional monitoring involves three separate layers: Kubernetes metrics, container metrics, and application metrics. Kubernetes metrics relate how a specific microservice and its deployment are being handled by the orchestrator, and provide information on network status, on-progress deployment status, the number of instances a service is utilizing, etc. Container metrics include CPU, network, and memory usage. Application metrics are specific to the app itself, and differ widely between a gaming app, a storage app, and an ecomm app.

That's a lot of ground to cover. A glitch anywhere throughout these layers can have a significant impact on user experience, revenue, and brand reputation, so monitoring your Kubernetes environment is non-negotiable.

If you're already using Kubernetes, you've clearly made a commitment to digital transfor-

-mation. You're probably using the auto-healing mechanism to resolve issues. So it hardly makes sense to be manually setting alerts, a key process in your AIOps workflow. Manual alerts and thresholds are a non-starter when it comes to Kubernetes. If you happen to be running multiple clusters, each with a large number of services, you'll find that it's rather impractical to use static alerts, such as "number of pods < X" or "ingress requests > Y", or to simply measure the number of HTTP errors. Values fluctuate for every region, data center, cluster, etc. You either get way too many false-positives—or you could miss a key event.

Autonomous Monitoring is Key

For effective monitoring, it's critical that your system automatically understands the severity of an incident and whether it has the potential to become catastrophic. Decisions such as when to wake up at night, when to scale out, or when to back pressure must be made automatically.

BEST PRACTICES FOR MONITORING KUBERNETES

Given our experience running Kubernetes in large-scale production environments, we'd like to share our guidelines for getting the most efficient alerting process. By adopting an AI monitoring system, your organization can use machine learning to constantly track millions of events in real time and to alert you when needed. This will enable your team to refocus their efforts on mission-critical tasks.

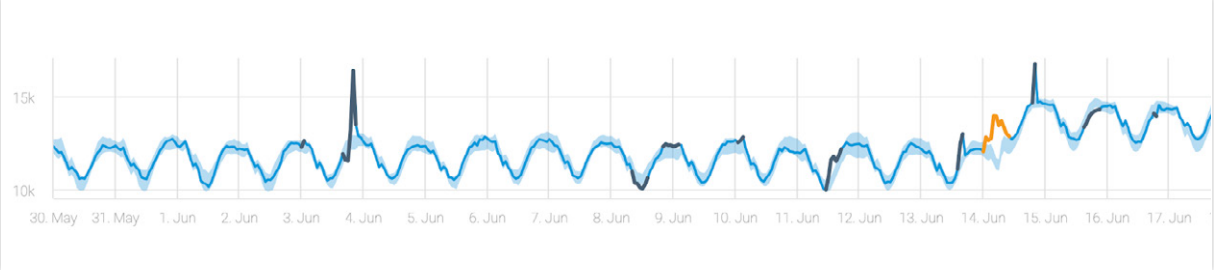
So whether you've already implemented AI monitoring, or you're thinking about doing so, here are our five best practices for managing autonomous alerts:

1. Track the API Gateway for Microservices in Order to Automatically Detect Application Issues

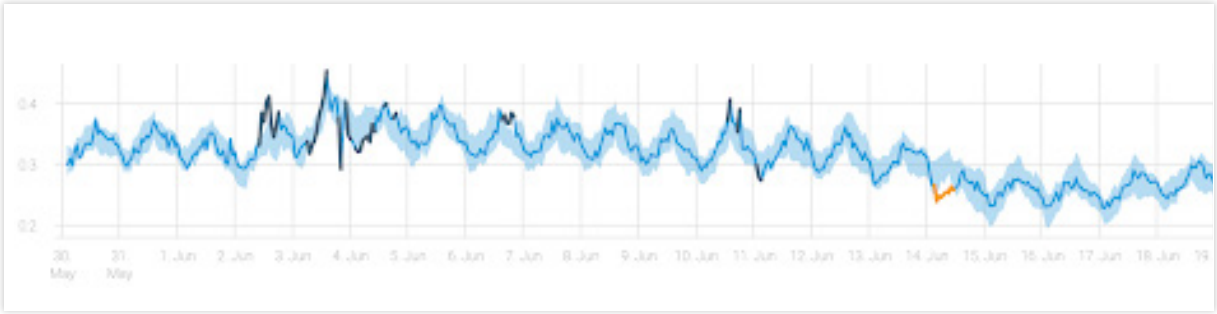
Although granular resource metrics such as CPU, load and memory are important to identify Kubernetes microservice issues, it's hard to use these convoluted metrics. In order to quickly understand when a microservice has issues, there are no better KPIs than API metrics like call error, request rate and latency. These will immediately point to a degradation in some component within the specific service.

The easiest way to learn service-level metrics is by automatically detecting anomalies on REST API requests on the service's load balancer, ideally over Ingress Controller such as Nginx or Istio. These metrics are agnostic to actual service and can easily measure all Kubernetes services in the same way. Alerts can be set at any level of the REST API (or any API), including customer-specific behavior in a multi-tenant SaaS.

Here is an example of an increase in request count to one of the microservices that was detected automatically:



An example of a latency drop for one of the services, identified automatically:



Look at how the baseline changes in the graph above. If there had been a static threshold here, the user would have had to manually adjust for this new state, or baseline.

There are several reasons rule-based alerts (aka static alerts) would not work here. With hundreds of different APIs, each with their own behavioral patterns, there is no feasible way to maintain a static alert for each of them. AI monitoring automatically adjusts to the new state baseline, which reduces quite a bit of maintenance.

Now, if you're more interested in dramatic changes – sure, anyone can identify when metrics drop to zero, but by that point, you're just trying to put out the fire. With the scale and granularity of AI analytics, pattern changes are detected far before they drop to zero. And the ability to adapt to fluctuating patterns reduces false-positives and the ensuing maintenance, while also shortening time to detection.

2. Stop Measuring Individual Containers

Due to the dynamic nature of Kubernetes resources, and the assumption that replicas in Deployment are symmetric, it has become very noisy to monitor container resources individually. Metrics change on a daily or hourly basis within the short life cycle. For example, ReplicaSets metrics change with every deployment as a new ReplicaSet ID is generated. Typically, we want to learn about pattern changes on the entire set of containers. For that we use cAdvisor, which can provide container-based metrics on CPU, memory and network usage. Key metrics include:

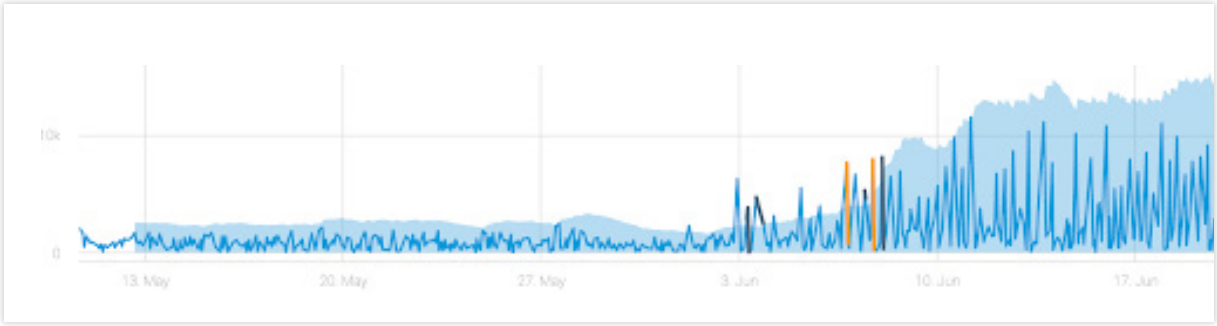
`Container_cpu_usage_seconds_total` - Total CPU used in both User space and System (Kernel)

`Container_memory_usage_bytes` - Total Memory used RSS+SWP

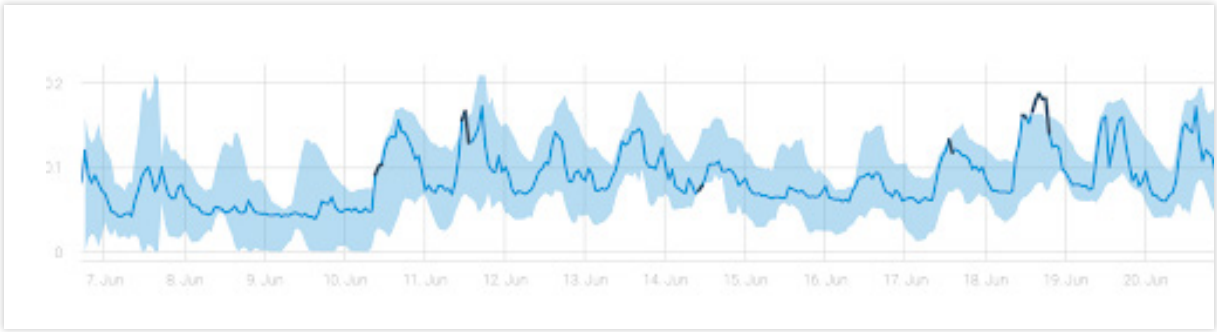
`Container_memory_max_usage_bytes` - Max Memory Recorded

From the above, it's clear that a change in the average 'max memory usage' for a given container set indicates a need to fix the default limits and requests for memory.

The anomalies in the graph below show an increase in average memory usage by some set of containers in a specific cluster. In this case, the Etcd cluster:



In the next example, the system learns the Kubernetes Deployment CPU usage and understands what constitutes normal behavior. The baseline in this graph indicates the metric's normal behavior, learned over time. Establishing this baseline ensures an alert isn't fired each and every time a CPU peaks. The system recognizes the dark blue peaks in the previous graph as less significant anomalies and those marked in orange as critical.



3. Get to the Heart of Potential Problems by Tracking "Status" or "Reason" Dimensions

Kube-State-Metrics is a nice add-on that keeps tabs on the API server and generates Kubernetes resource metrics. Typically the ones that merit alerts are related to the Pod's lifecycle and services state. Take, for example, a container state metric that includes the property "reason":"OOMKilled". This error may occur occasionally due to HW issues or rare events, but will usually not require immediate attention. But by following "status" and "reason" metrics over time, you can really start to understand if these "hiccups" are

actually trends. In that case, you'll likely need to adjust the allocated Limits and Requests to alleviate memory pressure.

As for issues related to services, the number of replicas per service usually depends on the scale of events (in an auto-scaled environment) and the roll-out strategy. It is important to track unavailable replicas and make sure that issues are not persistent.

- **Termination Status**

  `Kube_pod_container_status_terminated_reasonq`

  `Error`

  `OOMKilled`

  `ErrImagePull`

  `ImagePullBackoff`

  `CrashLoopBackoff`

  `ContainerCannotRun`

- **Services / Deployment Status**

  `Kube_endpoint_address_not_ready`

  `Kube_deployment_status_replicas_unavailable`

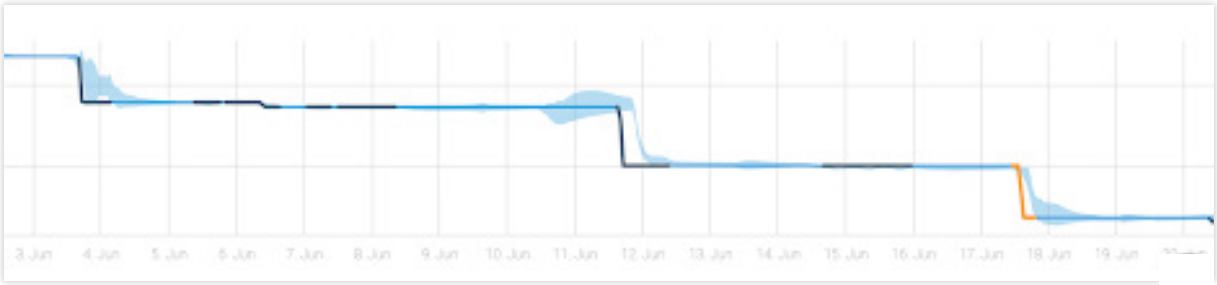
4. Always Alert on High Disk Usage!

High disk usage (HDU) is by far the most common issue for every system. There is no magic for getting around this or automatic healing for StatefulSet resources and statically attached volumes. The HDU alert will always require attention, and usually indicates an application issue. Make sure to monitor ALL disk volumes, including the root file system. Kubernetes Node Exporter provides a nice metric for tracking devices:

```
node_filesystem_avilable_bytes
```

Usually, you will set an alert for 75-80 percent utilization. Nevertheless, identifying pattern

changes earlier can reduce your headaches. The chart below shows real disk utilization over time and triggers anomaly alerts on meaningful drops.



5. Don't Ignore the Kube-System

By far the most complex issues stem from the actual Kubernetes System. Issues will typically occur due to DNS bottlenecks, network overload, and, the sum of all fears – Etcd. It is critical to track degradation of master nodes and identify issues before they happen, particularly load average, memory and disk size. To make sure that we do not crash the clusters, we need to monitor specific kube-system patterns, in addition to the aggregated resource pattern detection mentioned in tip #2.

Most of the components on kube-system provide metrics on "/metrics" resource that can be scraped directly automatically. We recommend tracking application-level metrics for each kube-system component, such as:

- **DNS KPI**
  - Total DNS requests - Whether you are using CoreDNS or KubeDNS, this metric may indicate a resources issue, scaling limits or even an application bug.
  - DNS request time - High latency may break some applications and request times will take longer.
- **ETCD KPI**
  - Failed proposals (`proposals_failed_total`) typically stem from either 1) longer duration outages caused by quorum loss in the cluster, or 2) temporary failures related to a leader election.
  - An unusually high snapshot duration (`snapshot_save_total_duration_seconds`) indicates there's something going on with your disk that may cause cluster instability.
  - A message failure increase (`message_sent_failed_total`) suggests that there are more critical issues happening in your network that may also cause the cluster to be unstable.

MONITORING KUBERNETES WITH ANODOT

As opposed to other off-the-shelf or homegrown monitoring solutions, Anodot's deep 360 technology is built for 100% autonomous monitoring of 100% of your Kubernetes data—in real-time. By leveraging AI to constantly monitor and correlate performance, Anodot provides mission-critical alerts in context, enabling the shortest time to detection and resolution.

Anodot's industry leading monitoring technology includes:

Unsupervised machine learning built for monitoring. Learning every metric's normal behavior is a prerequisite to identifying anomalous behavior. However, container and microservices metrics run a wide gamut of signal types and behaviors. Anodot uses sequential adaptive learning algorithms that initialize a model of what is normal on the fly, and then compute the relation of each new data point going forward.

Real-time monitoring of 100% of data. A cluster can consist of thousands of nodes, and an even greater amount of pods, and significant anomalies can occur in various metrics and depths of the architecture. In addition, disparate anomalies need to be constantly correlated to report on incidents in context, requiring complete data coverage. Anodot analyzes 100% of your Kubernetes metrics in real time and at scale by running machine learning algorithms on the live data stream itself, without reading and writing to a database.

Detection of seasonality. Kubernetes metric values fluctuate for every region, data center, cluster, etc, on a daily or hourly basis. Anodot's patented Vivaldi method for seasonality detects every kind of seasonal behavior, enabling the model to construct a true to life understanding of your metrics' dynamics. As opposed to manual or semi-autonomous thresholding, Anodot frees you from the false positives, false negatives and alert storms—while ensuring that you don't miss key events.

Metric correlation and root cause analysis. For each incident, Anodot groups correlated anomalies and identifies all events and contributing factors. Anodot uses a patented combination of algorithms to create an initial understanding of related metrics and consider them simultaneously in order to describe the whole incident.

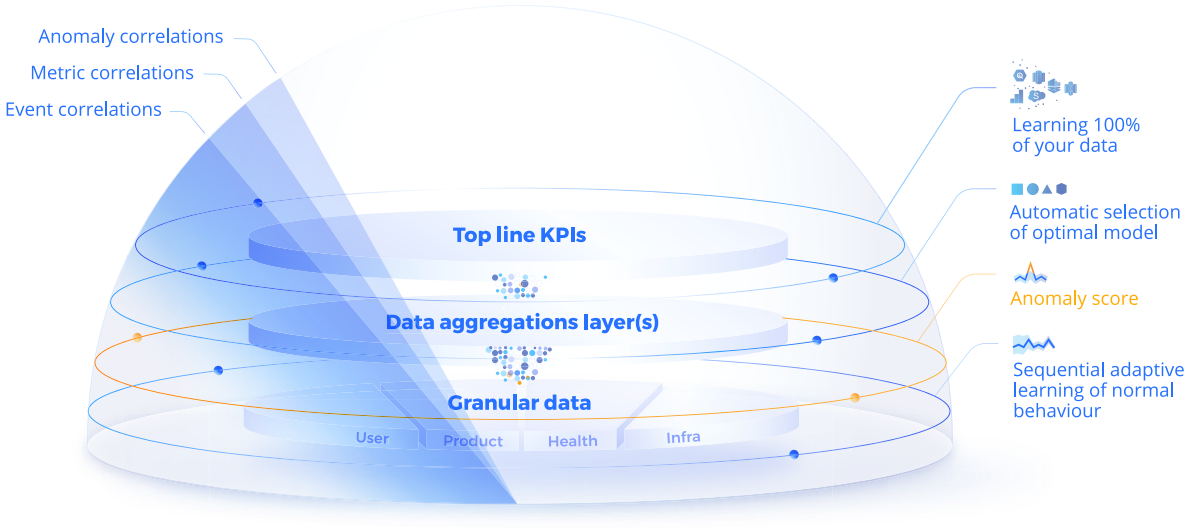
Anomaly Scoring. Grading anomalies is critical for filtering alerts by significance. Alerts are scored according to deviation, duration, frequency and other related conditions.

> Anodot's patented anomaly scoring method runs probabilistic Bayesian models to evaluate glitches both relative to normal based on their anomaly pattern, and relative to each other, to ensure that only—and all—significant incidents are detected and flagged.

By adopting an AI monitoring system, your organization can use machine learning to constantly track millions of your Kubernetes events in real time and to alert you when needed. Anodot's Autonomous Monitoring solution creates a comprehensive view by monitoring the Kubernetes environment and the applications themselves, to bulletproof your operations. It is vertical-agnostic, and ideal for companies in various industries including software, e-commerce, online retail, adtech, digital entertainment, fintech and more. Anodot automatically illuminates critical data blind spots for the shortest time to detection and resolution, so companies never miss another incident—and can rely on a system where every alert counts.

Deep 360 Monitoring™

Built for Autonomous Monitoring of All Data



www.anodot.com