Monitor the World

Meaningful Metrics for Kubernetes Applications and Clusters

Nick Turner, Amazon EKS

November 21, 2018

About Me

- SDE at Amazon EKS
- Twitter: @Nck_T
- Github: nckturner
- I enjoy spending time outdoors.

Agenda

- Monitoring Overview
- Tools Overview
- Metrics Sources
- Key Metrics
- Correcting Problems
- The Control Plane

Monitoring Microservices

Why do we monitor?

- To detect problems so that we can fix them
- To prevent outages
- Because we are nosy

The Difficulties of Monitoring

- Microservices are hard to monitor
  - Wealth of potential metrics to monitor, selecting actionable metrics is difficult
  - Debugging can be more difficult: “We replaced our monolith with micro services so that every outage could be more like a murder mystery.” – Honest Status Update (@honest_update)
- Containers are hard to monitor
  - Containers are generally more transient

A Method to the Madness

USE – Brendan Gregg

- For every resource, check:
  - Utilization
  - Saturation
  - Errors

RED – Tom Wilkie

- For every service, monitor request:
  - Rate
  - Errors
  - Duration

Tools Overview

Tools

Monitoring

- Prometheus
- Cloudwatch
- Metrics Server
- Node Exporter
- Node Problem Detector
- Kube State Metrics
- cAdvisor
- Kibana

Logging

- fluentd
- ELK
- Cloudwatch Logs

Alerting

- AlertManager
- Cloudwatch Alarms

And many more!

Prometheus

- Comprehensive Open Source Monitoring Framework
- Rich querying language
- Pull based Model
- Multi-dimensional data model (each metric value has a name and key-value dimensions)

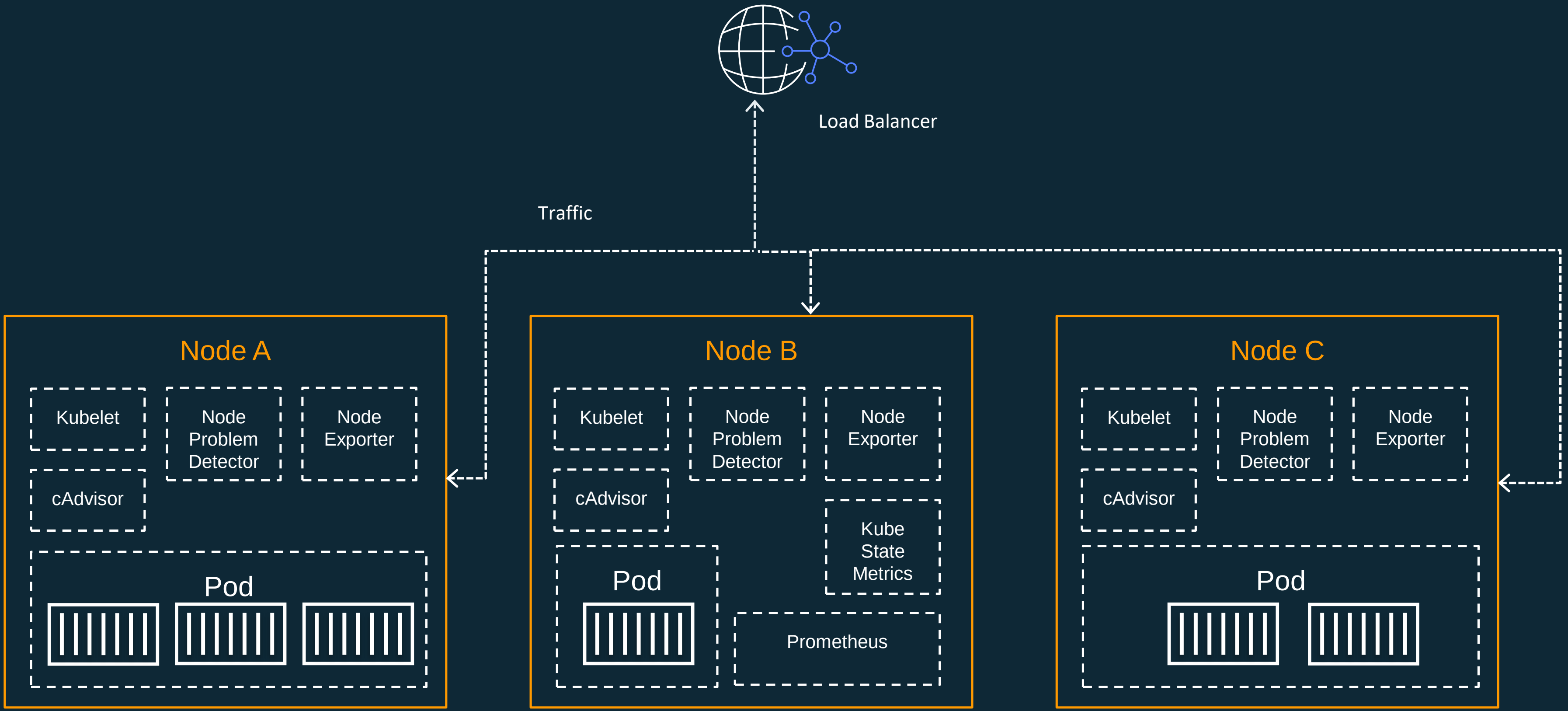
Amazon Cloudwatch

- Metrics, Logging and Alerting framework fully managed by Amazon
- Highly Available
- You may want to export Cloudwatch metrics into Prometheus, or vice versa

Amazon CloudWatch

Metrics Sources

Metrics Sources



Metrics Sources – Node Exporter

- Exposes node hardware/OS metrics
- Can be run as a Daemonset
- Requires access to the host filesystem
- github.com/prometheus/node_exporter

- Rich built in collectors
  - cpu
  - meminfo
  - filesystem
  - loadavg
  - diskstats
  - arp
  - boottime
  - ipvs

Metrics Sources – Node Problem Detector

- Reports problems up the stack with:
  - Events (temporary)
  - NodeConditions (permanent)
- Can be run as a Daemonset
- github.com/kubernetes/node-problem-detector

Metrics Sources – cAdvisor

- Collects and exports container-level metrics
- Includes:
  - Resource isolation parameters
  - Historical resource usage
- Can be run as a daemonset, also linked inside Kubelet
- github.com/google/cadvisor

Metrics Sources – Kube State Metrics

- Generates metrics based on Kubernetes objects that are present in the cluster
- Be cautious of memory usage for large deployments
- For example, generated deployment metrics include:
  - kube_deployment_status_replicas
  - kube_deployment_status_replicas_available
  - kube_deployment_status_replicas_unavailable
  - kube_deployment_status_replicas_updated
- github.com/kubernetes/kube-state-metrics

Metrics Sources – Metrics Server

- Gets data from kubelet
- Stores only current values of core metrics (pods and nodes) – does not give you historical metrics
- Used by the Horizontal Pod Autoscaler to make decisions
- Run as an aggregated API server
  - /apis/metrics.k8s.io/v1beta1/nodes
  - /apis/metrics.k8s.io/v1beta1/pods
- github.com/kubernetes-incubator/metrics-server

Metrics Sources – Instrumented Application

- Expose a metrics endpoint from your application, i.e. http://localhost:9090/metrics
- Configure Prometheus to scrape the endpoint
- 4 metric types:
  - Counter
  - Gauge
  - Histogram
  - Summary
- Client libraries available in:
  - Official:
    - Go, Java or Scala, Python, Ruby
  - Unofficial third-party client libraries:
    - Bash, C++, Common Lisp, Elixir, Erlang, Haskell, Lua for Nginx, Lua for Tarantool, .NET / C#, Node.js, Perl, PHP, Rust

Instrumenting Applications with Prometheus

```
func main() {
    http.Handle("/store", promhttp.InstrumentHandlerCounter(
        promauto.NewCounterVec(
            prometheus.CounterOpts{
                Name: ”store_requests",
                Help: "User store requests",
            },
            []string{"code"},
        ),
        http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
            // handle request
        }),
    ))

    http.Handle("/metrics", promhttp.Handler())
    http.ListenAndServe(":9000", nil)
}
```

Metrics Sources – Cloudwatch Exporter

- Prometheus exporter for cloudwatch
- Export cloudwatch metrics to prometheus
- All metrics exported as gauges
- github.com/prometheus/cloudwatch_exporter

Key Metrics

Key Metrics - Resources

What are resources in a Kubernetes Cluster?

- Disk
- CPU
- Memory
- Network Interfaces
- Load balancers

Key Metrics - CPU

Category	Metric
Utilization	CPU Used Time
Saturation	CPU load, throttled time / total time

- Useful both cluster wide and aggregated across pods by application and containers by image

CPU Query Examples

```
# container_cpu_usage_seconds_total is a counter (always increasing), so we need to take a rate

# Container cpu utilization per minute for myapp
rate(container_cpu_usage_seconds_total{container_name=”myapp"}[1m])

# Container cpu utilization by container
sum(rate(container_cpu_usage_seconds_total[1m])) by (container_name)

# CPU utilization by pod
sum(rate(container_cpu_usage_seconds_total[1m])) by (pod_name)

# CPU utilization at the cluster level
# id is a label for systemd slice (systemd’s hierarchical cgroups)
sum(rate(container_cpu_usage_seconds_total{id="/"}[1m])) / sum (machine_cpu_cores) * 100
```

Key Metrics - Memory

Category	Metric
Utilization	Memory Utilization (Memory Available / Memory Total)
Saturation	Swapping or Paging

- Useful both cluster wide and aggregated across pods by application and containers by image

Memory Query Examples

```
# Cluster utilization:
sum(node_memory_MemAvailable) / sum(node_memory_MemTotal) * 100

# Node utilization:
sum(node_memory_MemAvailable) by (instance) / sum(node_memory_MemTotal) by (instance) * 100
```

Key Metrics - Disk

Category	Metric
Utilization	Disk I/O time
Utilization	Disk Capacity Used / Disk Capacity Available
Saturation	Wait Queue Length

- Useful per node, cluster wide, and aggregated across pods by application

Disk Query Examples

```
# node disk utilization measured by io time per 1 minute:
avg(irate(node_disk_io_time_ms{device=~"(sd|xvd|nvme).+"}[1m]) / 1e3)
```

??

```
# Alert in 24 hours if disk will be full

((max by (namespace, pod, device) ((node_filesystem_size{fstype=~"ext[234]|btrfs|xfs|zfs"}
- node_filesystem_avail{fstype=~"ext[234]|btrfs|xfs|zfs"})
/ node_filesystem_size{fstype=~"ext[234]|btrfs|xfs|zfs"}))
> 0.85) and (predict_linear(node:node_filesystem_avail:[6h], 3600 * 24) < 0)
```

Key Metrics – Network Interfaces

Category	Metric
Utilization	Throughput / Instance Type Bandwidth

- Useful per node, cluster wide, and aggregated across pods by application

Key Metrics – Load Balancers

Category	Metric
Utilization	Requests Per Second
Saturation	Surge Queue Length

- Useful per Load Balancer (or aggregated by application if there are multiple per)

Key Metrics – Applications

Category	Metric
Rate	Requests per second
Errors	Status Code
Duration	Request Duration

- Aggregated across pods by application

Key Metrics – Applications

Category	Metric
Saturation	Pods available / Pods Total
Errors	Pod restarts

- Aggregated across pods by application

Correcting Problems

Correcting Problems

- Autoscale nodes with the cluster autoscaler
- Autoscale your service the HPA (Horizontal Pod Autoscaler)
- Detect an unhealthy node and terminate it
    - Node problem detector
    - Canary daemonset
- Rollback a deployment

The Control Plane

The Control Plane

- Apiserver
- Etcd
- Controller Manager
- Scheduler
- Other components

The Control Plane

- Mostly the same:
    - healthz
    - RED, USE
    - Running Pods / Desired
    - Pod Restarts
    - Scheduling – watch pod state changes (time in pending)

The Control Plane - Etcd

- Disk sync duration
- Leader Elections
- Quorum
- Corruption
  - (use --experimental-corrupt-check-time and --experimental-initial-corrupt-check)
- Disk Capacity
  - Occasional compaction might be necessary
- Latency (or just measure the API server)

Demo

References

Wilkie, T. (2017, Dec 15). *The RED Method: How To Instrument Your Services [B].* [Video File]. Retrieved from https://www.youtube.com/watch?v=TJLpYXbnfQ4

Gregg, B. (n.d.). *The USE Method.*

[Blog post]. Retrieved from http://www.brendangregg.com/usemethod.html

Cotton, B. (2018, May 4). *Reveal Your Deepest Kubernetes Metrics.* [Blog post]. Retrieved from https://www.youtube.com/watch?v=1oJXMdVi0mM&t=521s

Thank you!