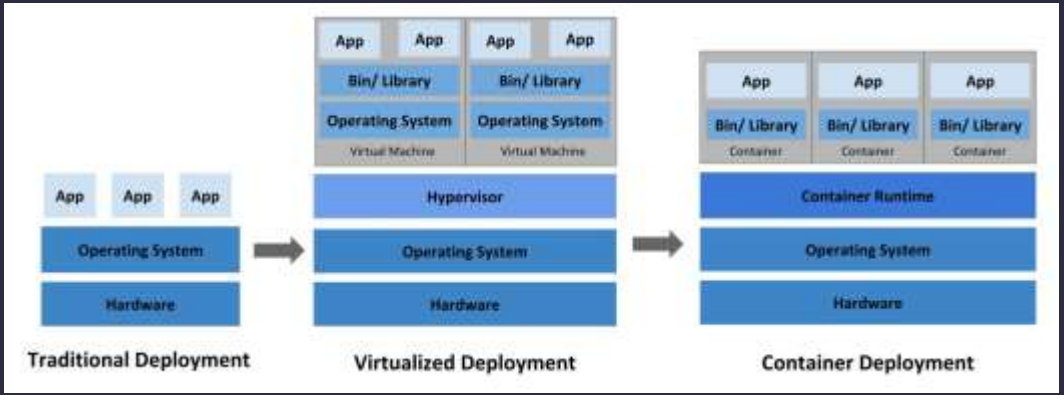
Optimization of the collaboration between kubernetes and etcd based on their network infrastructure

Kotsiaridis Konstantinos
Supervisor: Athanasios Korakis
Volos, February 2023

Evolution of Applications Deployment

Apps run on physical servers → Apps run on Virtual Machines → Apps run on Containers

- Apps run on physical servers: One app per server, else apps underperorm
- Apps run on Virtual Machines: Multiple apps per server but resource intensive
- Apps run on Containers: Multiple apps per server - portable and lightweight



Kubernetes

Open-source platform for automating the deployment, scaling, and management of containerized applications

Provides a consistent and efficient way to orchestrate and manage containerized applications in a cluster of machines

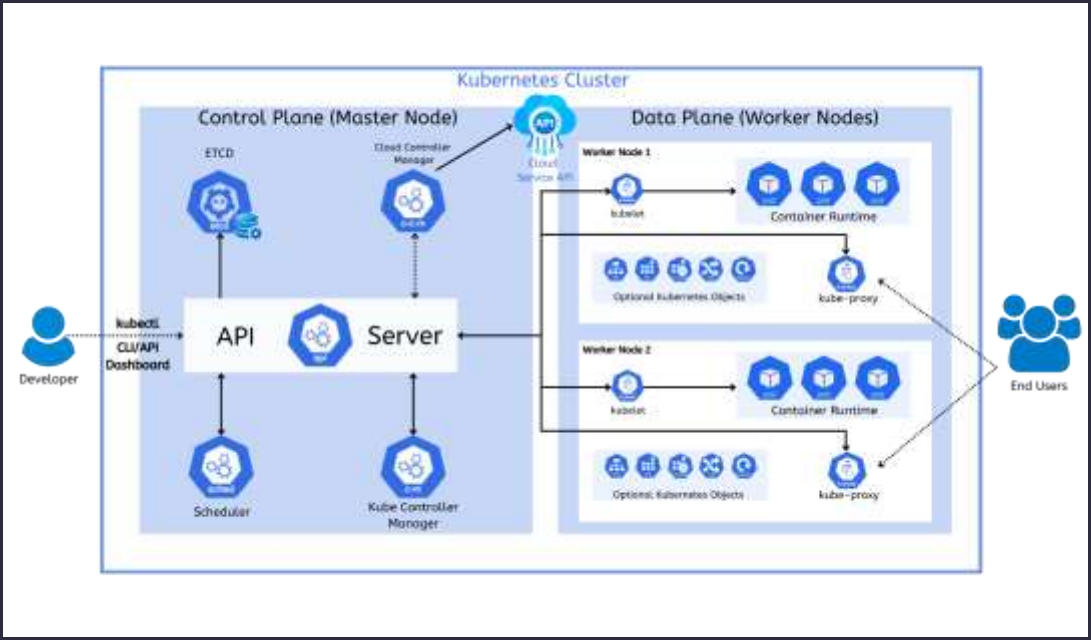
Built-in support for service discovery, automatic scaling, and self-healing mechanisms

Supports multiple network and storage solutions, and can be integrated with various add-ons

Organizations of all sizes use it. Ideal platform for running modern, cloud-native applications

Kubernetes pods: fundamental unit of scaling and deployment in a Kubernetes cluster, consist of one or more containers running together

Kubernetes Architecture



Master-Worker Architecture

Worker Node: all containerized applications run here

API server: handles requests to the Kubernetes API

etcd: key-value store
Stores cluster configuration data for Kubernetes

etcd

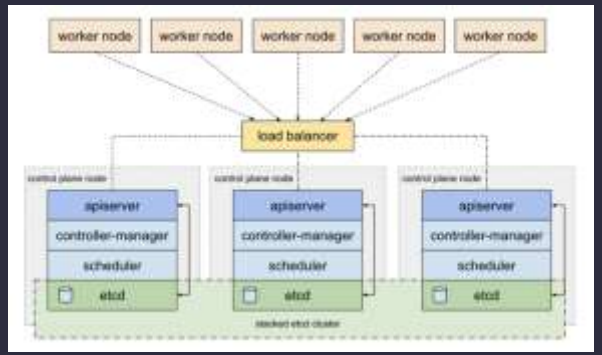
Distributed, consistent key-value store

Kubernetes' storage for cluster configuration data(ex. pods' state)

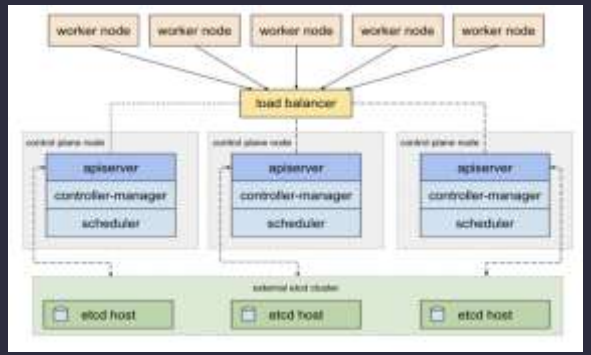
Each node contains a write ahead log(wal) with all the stored data

Uses Raft Consensus Algorithm for consistency and recoverability of the data

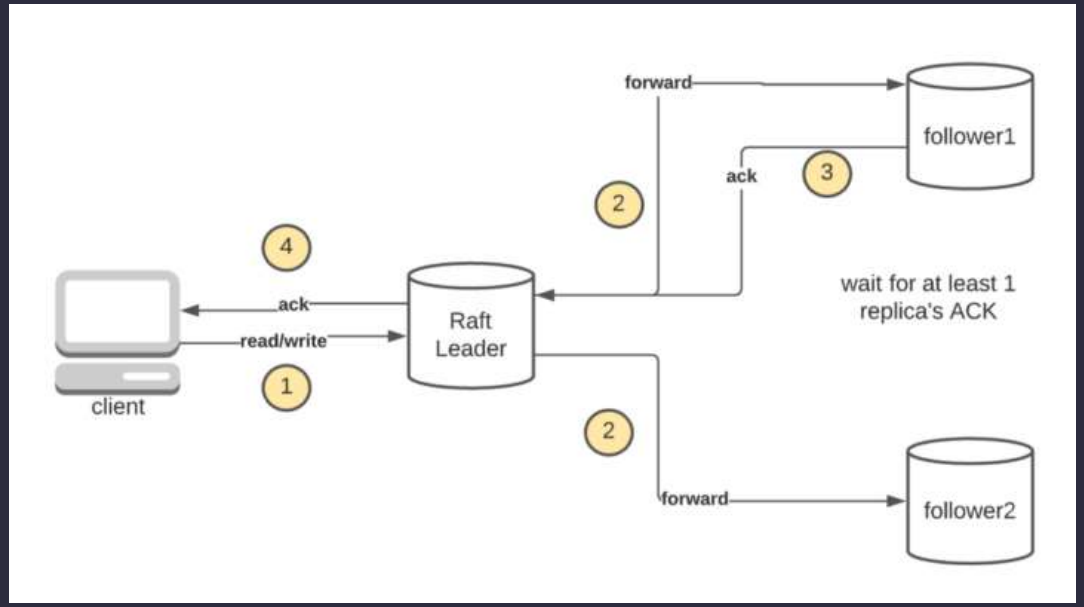
Stacked etcd cluster



External etcd Cluster



Raft Algorithm



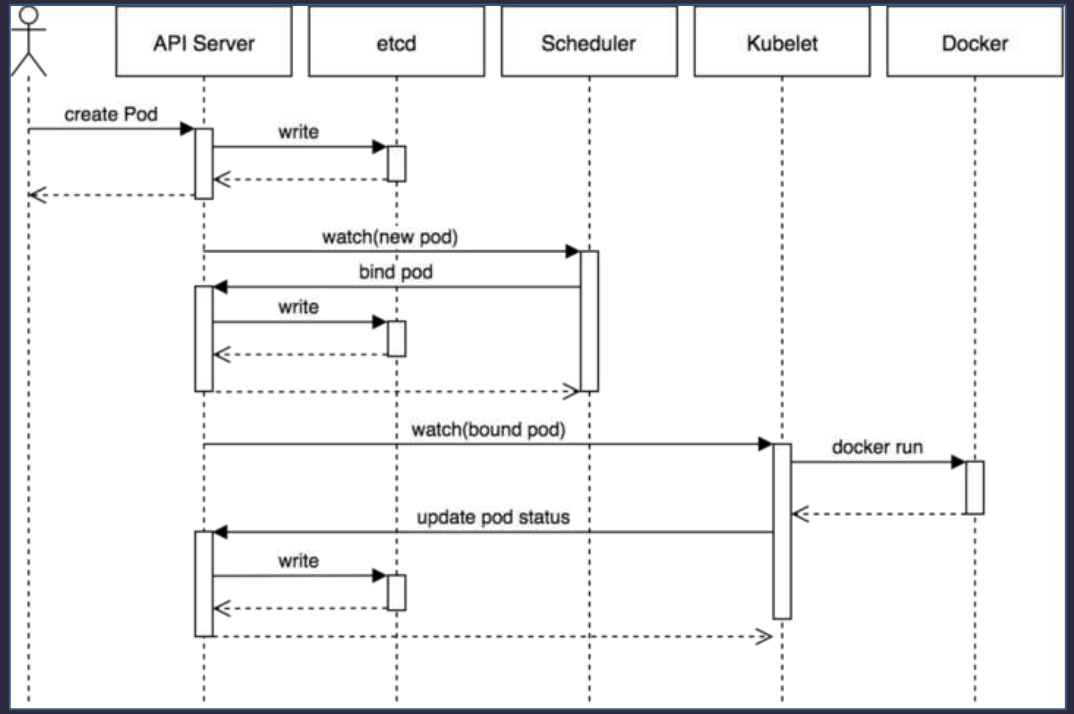
Raft is a consensus-shared agreement algorithm for distributed systems

Using Raft all etcd nodes have the same copy of write ahead log

Only Raft leader appends wal

Raft followers forward write requests to the leader

Kubernetes and etcd



Kubernetes API server uses gRPC for communication with the external etcd cluster

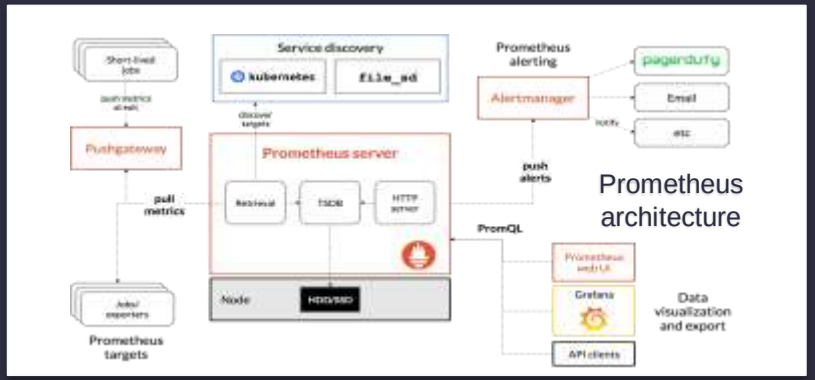
gRPC uses the Round Robin algorithm for load balancing

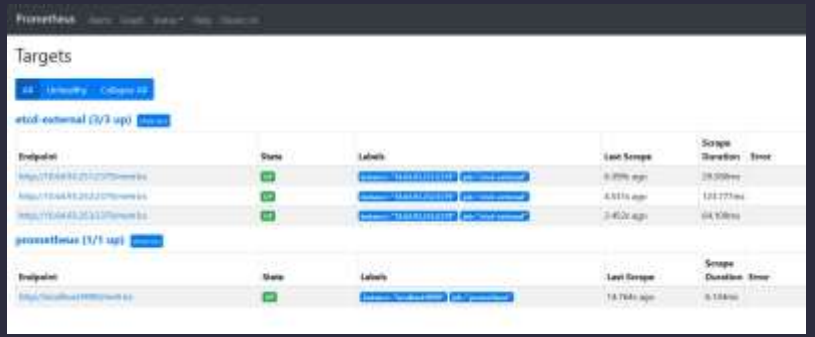
When Kubernetes pod state updates, API server sends request to etcd to update its state in wal

Pods states

Waiting	Pending	Unknown
Succeeded	Running	Completed
CrashLoop BackOff	Failed	

Prometheus Monitoring





Open-source monitoring and alerting system

Pull model for collecting metrics

Scrape metrics which services expose in specific format through HTTP endpoints

Default scrape interval 15 seconds

Grafana

Open-source data analytics and visualization tool

User friendly and with variety of customizable dashboards

Presentation of complex data in a clear and meaningful context

Many visualization options including graphs, heatmaps, histograms, and geo maps

Objective

Benefits of external etcd Cluster

Increased availability	Improved performance	Better resource utilization
Increased scalability	Communication latency	Greater Flexibility

TCP Proxy

Minimize Communication Latency

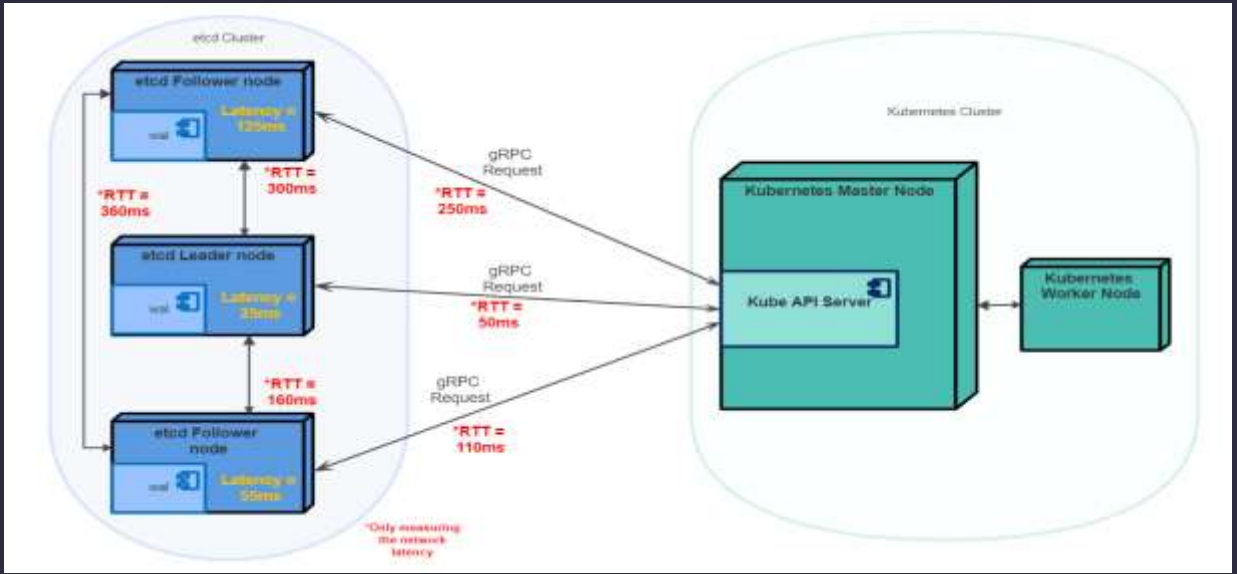
Prometheus and Grafana

Monitoring etcd nodes' metrics

Baseline Configuration

Average Response Time (excl. Processing time)

$$\frac{250+300}{3}+\frac{50+160}{3}+\frac{110+160}{3}\approx 343ms$$



TCP Proxy

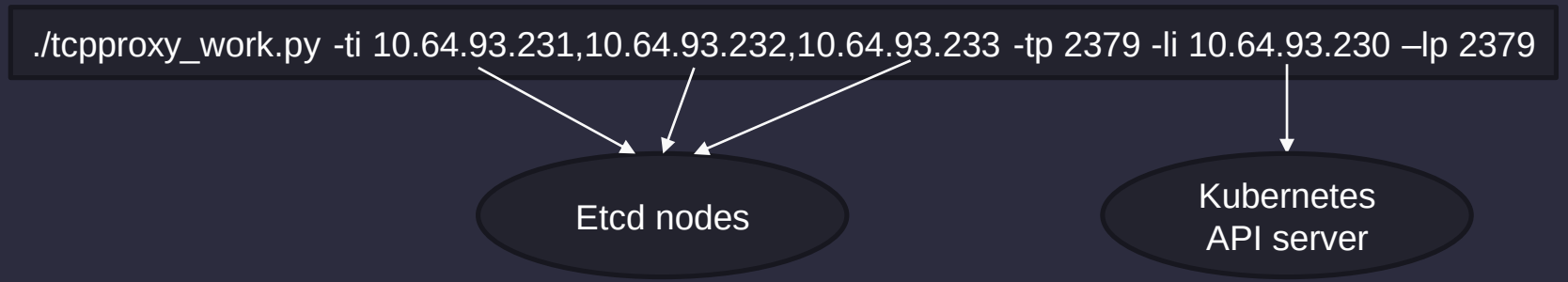
Modified version of a GitHub project, written in Python

Listens for incoming data on a socket and sends data to the target server.

Targets server's response is received and sent back to the client

Added ability to have multiple target servers, creating multiple pairs of sockets

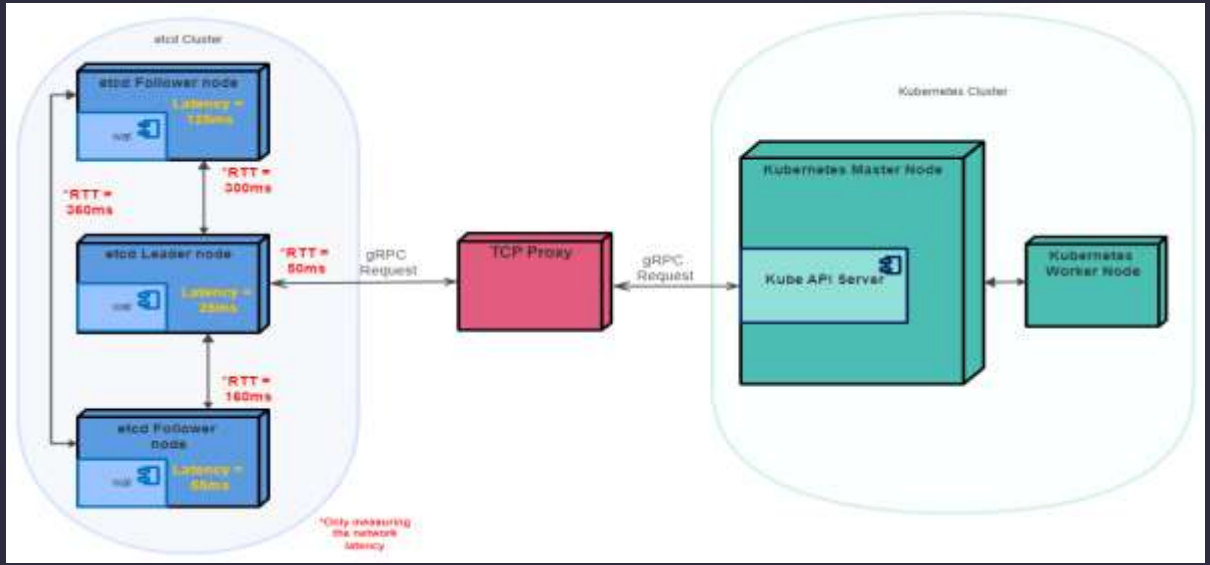
Added ability to request etcd API for nodes info, determine cluster's leader and target traffic only to this node



Optimized Configuration

Average Response Time (excl. Processing time)

$$50 + 160 + tcpProxyLatency$$



Experiment

time helm install prometheus prometheus-community/kube-prometheus-stack

```
root@kmaster:~# kubectl get nodes
NAME       STATUS   ROLES    AGE   VERSION
kmaster    Ready    master   23m   v1.19.2
kworker1   Ready    <none>   21m   v1.19.2
root@kmaster:~# kubectl get pods
NAME                                                     READY   STATUS    RESTARTS   AGE
alertmanager-prometheus-kube-prometheus-alertmanager-0   2/2     Running   1          5m32s
alertmanager-prometheus1-kube-prometheu-alertmanager-0   2/2     Running   1          5m36s
prometheus-grafana-74d959d5-rwcvn                        3/3     Running   0          5m52s
prometheus-kube-prometheus-operator-5cd4b7b5f4-pv8gg     1/1     Running   0          5m52s
prometheus-kube-state-metrics-6997795855-5slzh           1/1     Running   0          5m52s
prometheus-prometheus-kube-prometheus-prometheus-0       2/2     Running   0          5m19s
prometheus-prometheus-node-exporter-ln6vp                0/1     Pending   0          5m52s
prometheus-prometheus-node-exporter-swwl6                0/1     Pending   0          5m52s
prometheus1-kube-prometheu-prometheus-0                  2/2     Running   0          5m32s
prometheus1-grafana-6cc58c75-nj7sm                       3/3     Running   0          6m
prometheus1-kube-prometheu-operator-7bf975c96d-cpxcf     1/1     Running   0          6m
prometheus1-kube-state-metrics-558bc8b497-gsq7m          1/1     Running   0          6m
prometheus1-prometheus-node-exporter-nvnck               1/1     Running   0          6m
prometheus1-prometheus-node-exporter-qppsj               1/1     Running   0          6m1s
root@kmaster:~#
```

Helm: open-source package manager for Kubernetes

Command installs helm chart for Prometheus monitoring stack

Command ran twice simultaneously to generate more traffic

Results

Baseline Configuration

```
kube-prometheus-stack has been installed.
  kubectl --namespace default get pods -l

Visit https://github.com/prometheus-opera

real    2m2.413s
user    0m14.378s
sys     0m0.783s
root@kmaster:~/myscripts/kmaster#
```

Optimized Configuration

```
kube-prometheus-stack has been installed.
  kubectl --namespace default get pods -l

Visit https://github.com/prometheus-opera

real    1m39.339s
user    0m14.303s
sys     0m0.872s
root@kmaster:~/myscripts/kmaster#
```

The optimized configuration reduced latency by

$$\left(\frac{122}{99} - 1\right) * 100\% = 23{,}23\%$$

Results

Upper chart: Baseline Configuration

Lower chart: Optimized Configuration

The results are also backed by the graphs in Grafana.

In green, yellow and cyan etcd nodes wal size

In orange, red and blue etcd node used bytes in wal

Clean installation of etcd cluster each time

Vertical red line: helm command finished in optimized configuration

Results

Upper chart: Baseline Configuration

Lower chart: Optimized Configuration

Graph: Network traffic between etcd cluster and Kubernetes API server

Yellow line: bytes etcd nodes sent to Kubernetes API server

Green line: bytes etcd nodes received from Kubernetes API server

Same amount of traffic in both configurations

Distributed in longer timeframe in baseline configuration

Thank you for your attention