Title: Go Server that Responds to GET Requests on Path "/metrics"

Description: This document describes a Go server that listens on port 12345 and responds to GET requests on the path "/metrics". It uses caching to limit disk I/O by storing the metrics data in memory and only re-reading the file if the cache is stale. The document also includes a script to create a dummy metrics file and an example systemd service configuration file to start the server as a service.

Go Server Code

```
package main

import (
	"fmt"
	"io/ioutil"
	"net/http"
	"time"
)

// filePath is the path to the file containing the metrics data
const filePath = "data/metrics_from_special_app.txt"

// cacheTTL is the amount of time the metrics data should be cached in memory
const cacheTTL = 30 * time.Second

var (
	// lastCacheTime is the time the cache was last updated
	lastCacheTime time.Time
	// metricsCache contains the cached metrics data
	metricsCache string
)

// getMetrics reads the metrics data from the file specified in filePath.
// It uses caching to limit disk I/O by storing the metrics data in memory
// and only re-reading the file if the cache is stale (based on the cacheTTL constant).
func getMetrics() (string, error) {
	now := time.Now()
	if now.Sub(lastCacheTime) < cacheTTL {
		return metricsCache, nil
	}
	data, err := ioutil.ReadFile(filePath)
	if err != nil {
		return "", err
	}
	metrics := string(data)
	metricsCache = metrics
	lastCacheTime = now
	return metrics, nil
}
```

```
// metricsHandler is the handler function for GET requests on the path "/metrics".
// It reads the metrics data using the getMetrics function and returns it as a string
// with key-value fields separated by line breaks.
func metricsHandler(w http.ResponseWriter, r *http.Request) {
	if r.Method != http.MethodGet {
		http.Error(w, "Method not allowed", http.StatusMethodNotAllowed)
		return
	}
	if r.URL.Path != "/metrics" {
		http.NotFound(w, r)
		return
	}
	metrics, err := getMetrics()
	if err != nil {
		http.Error(w, "Failed to read metrics data", http.StatusInternalServerError)
		return
	}
	fmt.Fprint(w, metrics)
}

func main() {
	// Register the metricsHandler function to handle requests on the path "/metrics".
	http.HandleFunc("/metrics", metricsHandler)
	// Start the server and listen on port 12345.
	http.ListenAndServe(":12345", nil)
}
```

The code defines the getMetrics function that reads the metrics data from the file specified in the filePath constant. It uses caching to limit disk I/O by storing the metrics data in memory and only re-reading the file if the cache is stale (based on the cacheTTL constant).

The metricsHandler function is the handler function for GET requests on the path "/metrics". It reads the metrics data using the getMetrics function and returns it as a string with key-value fields separated by line breaks.

The main function registers the metricsHandler function to handle requests on the path "/metrics". It starts the server and listens on port 12345.

Script to Create a Dummy Metrics File

```
#!/bin/sh

mkdir -p data  # Create the "data" directory if it doesn't exist
echo "metric1 value1" > data/metrics_from_special_app.txt
echo "metric2 value2" >> data/metrics_from_special_app.txt
echo "metric3 value3" >> data
```

this is a bash script for createing the matrix

You need to run this bash script using ./script-name

To create the systemd service configuration file, you can create a file named go-server.service in the systemd-config folder with the following contents:

makefile

```
[Unit]
Description=Go server

[Service]
Type=simple
ExecStart=/path/to/go-server-binary
Restart=always

[Install]
WantedBy=multi-user.target
```

Replace /path/to/go-server-binary with the path to your compiled Go server binary.

To install the systemd service, you can run the following command:

```
$ sudo cp systemd-config/go-server.service /etc/systemd/system/
$ sudo systemctl daemon-reload
$ sudo systemctl enable go-server.service
$ sudo systemctl start go-server.service
```

Finally, you can create a startup script that runs the Go server. Here's an example script:

bash

```
#!/bin/bash
cd /path/to/go-server/
./go-server &
```

Replace /path/to/go-server/ with the path to your Go server directory. Make the script executable and add it to your system's startup processes