In this exercise, you will familiarise yourself with a structured logging library and a monitoring tool, as well as create a client that implements retries and circuit breakers.

First, let's clone the https://github.com/kelindar/http-echo repository that we have prepared for you. It contains an HTTP server that echoes a request (snippet below).

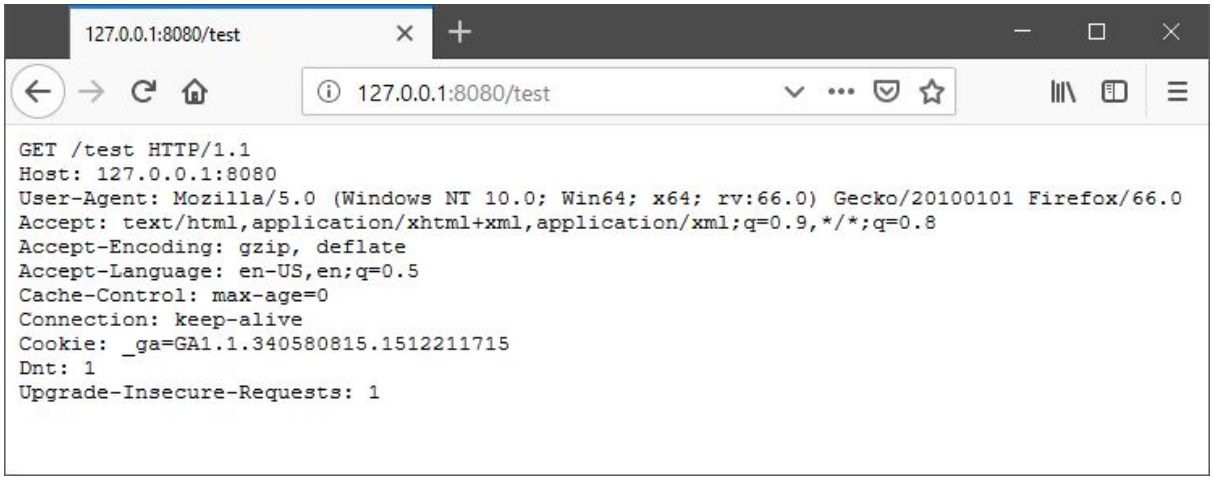
```
// ListenAndServe starts the server
func (s *Server) ListenAndServe() {
    s.logger.Info("echo server is starting on port 8080...")
    http.HandleFunc("/", s.echo)
    http.ListenAndServe(":8080", nil)
}

// Echo echoes back the request as a response
func (s *Server) echo(writer http.ResponseWriter, request *http.Request) {
    writer.Header().Set("Access-Control-Allow-Origin", "*")
    writer.Header().Set("Access-Control-Allow-Headers", "Content-Range,
Content-Disposition, Content-Type, ETag")

    // 30% chance of failure
    if rand.Intn(100) < 30 {
        writer.WriteHeader(500)
        writer.Write([]byte("a chaos monkey broke your server"))
        return
    }

    // Happy path
    writer.WriteHeader(200)
    request.Write(writer)
}
```

In the happy path, you should be seeing the headers of the request printed back out. The HTTP status code will be 200.



However, sometimes you will see an error with the HTTP status code of 500.

What you will need to do:

1. Add logging on failures.
2. Instrument requests and their latency.
3. Create a Datadog dashboard.
4. Create a Go client which continuously calls the endpoint and automatically retries (or opens the circuit)

First, let's start by adding **logging**. Following the advice we've given earlier, add logs to the failed requests.

Next, let's **instrument** the server by adding request latency monitoring along with the count and create a dashboard for it. We will use Datadog for this exercise.

1. Sign up with Datadog and create a free account.

2. Install the Datadog agent on your machine (https://docs.datadoghq.com/agent/) and make sure it runs.
3. Integrate your cloned HTTP server application with the Datadog client (https://github.com/DataDog/datadog-go)
4. Add request monitoring and create a Datadog dashboard.

Lastly, let's write a Go client which calls this endpoint continuously and implements retries and/or circuit breakers.

Feel free to use these libraries:

- Retry-Go for retries: https://github.com/avast/retry-go
- Go-Hystrix for circuit breakers: https://github.com/myteksi/hystrix-go

Several Grab engineers have written an extensive and detailed article on how to use and properly configure both circuit breakers and retries. Please read through them.

- https://engineering.grab.com/designing-resilient-systems-part-1
- https://engineering.grab.com/designing-resilient-systems-part-2

Circuit breakers and retries are just the tip of the iceberg when it comes to building reliable systems. Read through the following articles that our fellow Grabbers have written to understand rate limiting, bulkheading, load balancing and chaos engineering:

- https://engineering.grab.com/beyond-retries-part-1
- https://engineering.grab.com/beyond-retries-part-2
- https://engineering.grab.com/beyond-retries-part-3