Health checks (Terminus)

The **terminus** offers hooks to react on graceful shutdowns and supports you creating proper **Kubernetes** readiness / liveness checks for any HTTP application. The module **@nestjs/terminus** integrates the terminus library with the Nest ecosystem.

Getting started

To get started with `@nestjs/terminus` we need to install the required dependencies.

```
$ npm install --save @nestjs/terminus @godaddy/terminus
```

Setting up a health check

A health check represents a summary of **health indicators**. A health indicator executes a check of a service, whether it is in a healthy state or not. A health check is positive, if all the assigned health indicators are up and running. Because a lot of applications will need similar health indicators, **@nestjs/terminus** provides a set of predefined health indicators, such as:

- `DNSHealthIndicator`
- `TypeOrmHealthIndicator`
- `MongooseHealthIndicator`
- `MicroserviceHealthIndicator`

DNS Health Check

The first step to get started with our first health check, is to setup a service which will associate health indicators to an endpoint.

terminus-options.service.ts JS

```
import {
  TerminusEndpoint,
  TerminusOptionsFactory,
  DNSHealthIndicator,
  TerminusModuleOptions
} from '@nestjs/terminus';
import { Injectable } from '@nestjs/common';

@Injectable()
export class TerminusOptionsService implements TerminusOptionsFactory {
  constructor(
```

```
    private readonly dns: DNSHealthIndicator,
  ) {}

  createTerminusOptions(): TerminusModuleOptions {
    const healthEndpoint: TerminusEndpoint = {
      url: '/health',
      healthIndicators: [
        async () => this.dns.pingCheck('google', 'https://google.com'),
      ],
    };
    return {
      endpoints: [healthEndpoint],
    };
  }
}
```

Once we have set up our `TerminusOptionsService`, we can import the `TerminusModule` into the root `ApplicationModule`. The `TerminusOptionsService` will provide the settings, which in turn will be used by the `TerminusModule`.

app.module.ts JS

```
import { Module } from '@nestjs/common';
import { TerminusModule } from '@nestjs/terminus';
import { TerminusOptionsService } from './terminus-options.service';

@Module({
  imports: [
    TerminusModule.forRootAsync({
      useClass: TerminusOptionsService,
    }),
  ],
})
export class ApplicationModule { }
```

> **HINT**
> If done correctly, Nest will expose the defined health check(s), which are reachable through a GET request to the defined route. For example `curl -X GET 'http://localhost:3000/health'`

Custom health indicator

In some cases, the predefined health indicators provided by `@nestjs/terminus` do not cover all of your health check

requirements. In this case you can set up a custom health indicator according to your needs.

Let's get started by creating a service which will represent our custom health indicator. To get a basic understanding how a health indicator is structured, we will create an example `DogHealthIndicator`. This health indicator should have the state "up", if every `Dog` object has the type `goodboy`, otherwise it will throw an error, which then the health indicator will be seen as "down".

dog.health.ts JS

```
import { Injectable } from '@nestjs/common';
import { HealthCheckError } from '@godaddy/terminus';
import { HealthIndicatorResult } from '@nestjs/terminus';

export interface Dog {
  name: string;
  type: string;
}

@Injectable()
export class DogHealthIndicator extends HealthIndicator {
  private readonly dogs: Dog[] = [
    { name: 'Fido', type: 'goodboy' },
    { name: 'Rex', type: 'badboy' },
  ];

  async isHealthy(key: string): Promise<HealthIndicatorResult> {
    const badboys = this.dogs.filter(dog => dog.type === 'badboy');
    const isHealthy = badboys.length > 0;
    const result = this.getStatus(key, isHealthy, { badboys: badboys.length });

    if (isHealthy) {
      return result;
    }
    throw new HealthCheckError('Dogcheck failed', result);
  }
}
```

The next thing we need to do is registering the health indicator as a provider.

app.module.ts JS

```
import { Module } from '@nestjs/common';
import { TerminusModule } from '@nestjs/terminus';
import { TerminusOptions } from './terminus-options.service';
```

```
import { DogHealthIndicator } from './dog.health.ts';

@Module({
  imports: [
    TerminusModule.forRootAsync({
      imports: [ApplicationModule],
      useClass: TerminusOptionsService,
    }),
  ],
  providers: [DogHealthIndicator],
  exports: [DogHealthIndicator],
})
export class ApplicationModule { }
```

> **HINT**
> In a real world application the `DogHealthIndicator` should be provided in a separate module, for example `DogsModule`, which then will be imported by the `ApplicationModule`. But keep in mind to add the `DogHealthIndicator` to the `exports` array of the `DogModule` and add the `DogModule` in `imports` array of the `TerminusModule.forRootAsync()` parameter object.

The last required thing to do is to add the now available health indicator in the required health check endpoint. For that we go back to our `TerminusOptionsService` and implement it to the `/health` endpoint.

terminus-options.service.ts JS

```
import {
  TerminusEndpoint,
  TerminusOptionsFactory,
  DNSHealthIndicator,
  TerminusModuleOptions
} from '@nestjs/terminus';
import { Injectable } from '@nestjs/common';

@Injectable()
export class TerminusOptionsService implements TerminusOptionsFactory {
  constructor(
    private readonly dogHealthIndicator: DogHealthIndicator
  ) {}

  createTerminusOptions(): TerminusModuleOptions {
    const healthEndpoint: TerminusEndpoint = {
      url: '/health',
      healthIndicators: [
        async () => this.dogHealthIndicator.isHealthy('dog'),
```

```
      ],
    };
    return {
      endpoints: [healthEndpoint],
    };
  }
}
```

If everything has been done correctly, the `/health` endpoint should respond with a `503` response code and the following data.

```
{
  "status": "error",
  "error": {
    "dog": {
      "status": "down",
      "badboys": 1
    }
  }
}
```

You can view working examples in the `@nestjs/terminus` repository.

Support us

Nest is an MIT-licensed open source project. It can grow thanks to the support by these awesome people. If you'd like to join them, please read more here.

Principal Sponsor

Sponsors / Partners

Designed by Jakub Staroń, hosted by Netlify