# ToiletPaper #147

## In-memory Caching für AWS Lambda leicht gemacht

Autor: Robert Gruner / Software Engineer / Standort Leipzig

### ✕ Problem

Es ist bekannt, dass beim Aufruf einer AWS Lambda eine Runtime instanziiert wird, welche nach Bearbeitung der Anfrage nicht sofort wieder abgeräumt wird. AWS hält diese Runtime eine gewisse Zeit warm, um sie für spätere Aufrufe wiederverwenden zu können. AWS empfiehlt hier, die *Execution Environment* der Runtime zu nutzen, um die Lambda-Funktion performanter zu machen. Doch wie funktioniert das?

### ✓ Lösung

Einfach gesagt, gibt es in einer Node.js Lambda nur die Handler-Funktion selbst und einen globalen JavaScript-Kontext. Bei jedem Aufruf der Lambda wird die Handler-Funktion ausgeführt – ein Caching *innerhalb* des Funktionskontext ist damit nicht möglich. Allerdings hat diese Funktion Zugriff auf den globalen Kontext. Darin lassen sich beliebige Variablen oder Objekte speichern. Ein Folgeaufruf hat Zugriff auf diese Daten, solange diese Runtime existiert. Kombiniert man dieses Verhalten mit einer Dependency-Injection-Lösung wie tsyringe, bekommt man Instanz-Caching für Datenbank/HTTP Clients quasi geschenkt. Zudem ist es sehr simpel, statische Konfigurationen im Speicher zu cachen.

### ➔ Beispiel

Folgende 2 Varianten bietet tsyringe an, um die Instanz einer Dependency im globalen Kontext zu cachen. Für Klasseninstanzen eignet sich der *singleton* Decorator.

```
import { singleton } from "tsyringe";

@singleton()
class HttpClient {}
```

Wie der Name schon sagt, wird die Instanz dabei nur einmal erzeugt und dann im Dependency-Container vorgehalten. Für simple Objekte oder Dependencies, die mittels Factory erzeugt werden, eignet sich die Hilfsfunktion *instanceCachingFactory.*

```
import { container, instanceCachingFactory } from "tsyringe";

export interface CoffeeCache {
  configs?: CoffeeConfig[];
}

container.register("CoffeeCache", {
  useFactory: instanceCachingFactory<CoffeeCache>(() => ({
    /* "configs" key muss später befüllt werden */
  })),
});
```

Mit diesem Setup kann man sich z. B. durch ein **container.resolve(HttpClient)** die Instanz des Client heranziehen. Das geht überall im Code, somit auch innerhalb einer Handler-Funktion. Sobald man die Konfigurationen per HTTP abgerufen hat, kann man diese im CoffeeCache-Objekt speichern. Bei einem weiteren Aufruf der Lambda Funktion liegt die Konfigurationen bereits im Cache vor und man kann sich den HTTP-Request sparen.
Innerhalb anderer Klassen können die Dependencies auch via Constructor Injection herangezogen werden.

```
import { inject } from "tsyringe";

class CoffeeHandler {
  constructor(@inject("CoffeeCache") cache: CoffeeCache) {}
}
```

### + Weiterführende Aspekte

- Doku zu AWS Lambda Execution Context: https://docs.aws.amazon.com/lambda/latest/dg/runtimes-context.html
- AWS Lambda Best Practices: https://docs.aws.amazon.com/lambda/latest/dg/best-practices.html
- Source zur empfohlenen DI Lösung *tsyringe:* https://github.com/microsoft/tsyringe