

# Kostenabschätzung

---

## Google vs Mapbox

Google bietet keine Möglichkeit an einzelne Tiles zu laden, sondern rechnet lediglich mit map loads. Interaktionen wie Zoomen, Bewegen der Karte oder andere Arten neue Tiles zu laden werden nicht berücksichtigt. Daher gibt es auch keine Preisdaten die wir gut vergleichen könnten.

Mapbox bietet zwar auch diese Möglichkeit an nur für einen map load zu zahlen, jedoch nur wenn man deren eigene - auf [leaflet.js](#) basierende - [Bibliothek](#) benutzt.

### Vergleich

Current requests	per month
Map Loads	55.000
Geocoding	40.000

## google vs mapbox

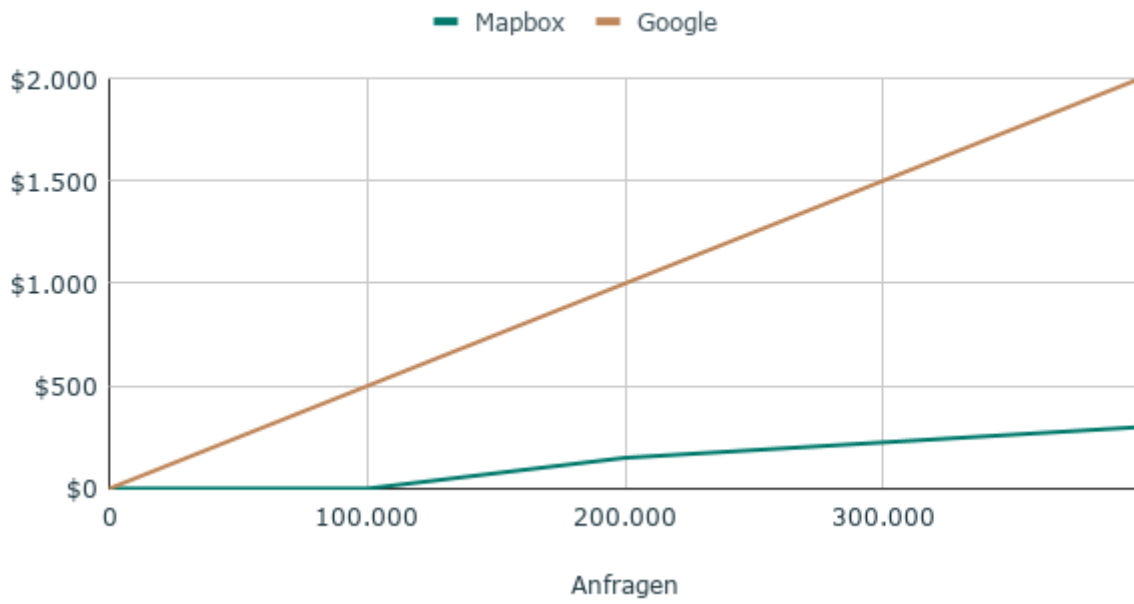
Google bietet keine Möglichkeit an einzelne Tiles zu laden, sondern rechnet lediglich mit map loads. Interaktionen wie Zoomen, Bewegen der Karte oder andere Arten neue Tiles zu laden werden nicht berücksichtigt. Daher gibt es auch keine Preisdaten die wir gut vergleichen könnten.

Mapbox bietet zwar auch diese Möglichkeit an nur für einen map load zu zahlen, jedoch nur wenn man deren eigene - auf [leaflet.js](#) basierende - [Bibliothek](#) benutzt.

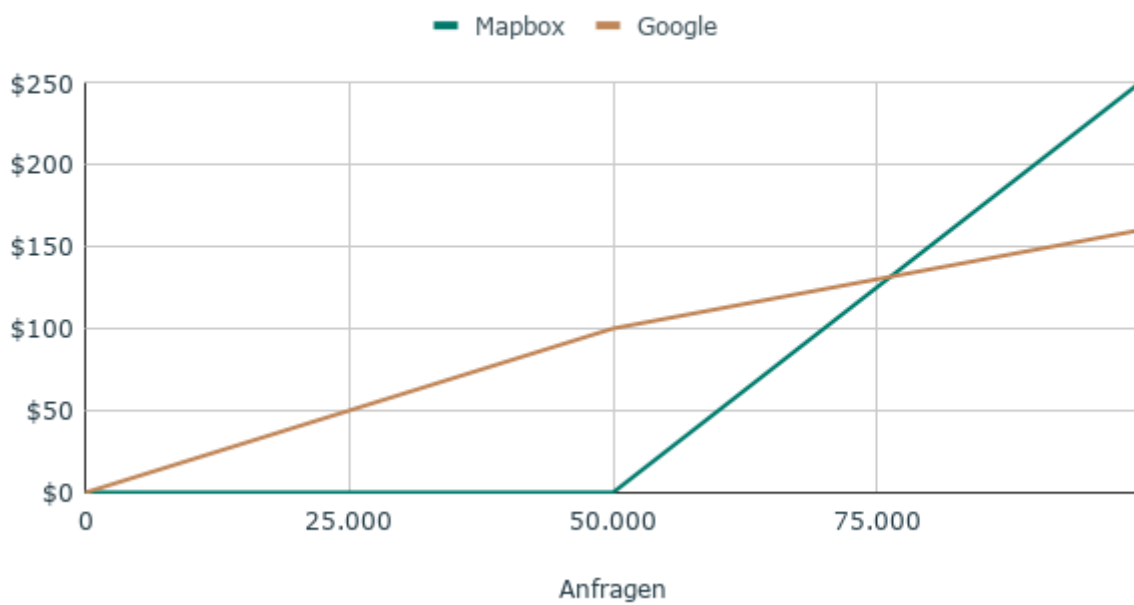
### Vergleich

Current requests	per month
Mapload	40.000
Geocoding	55.000

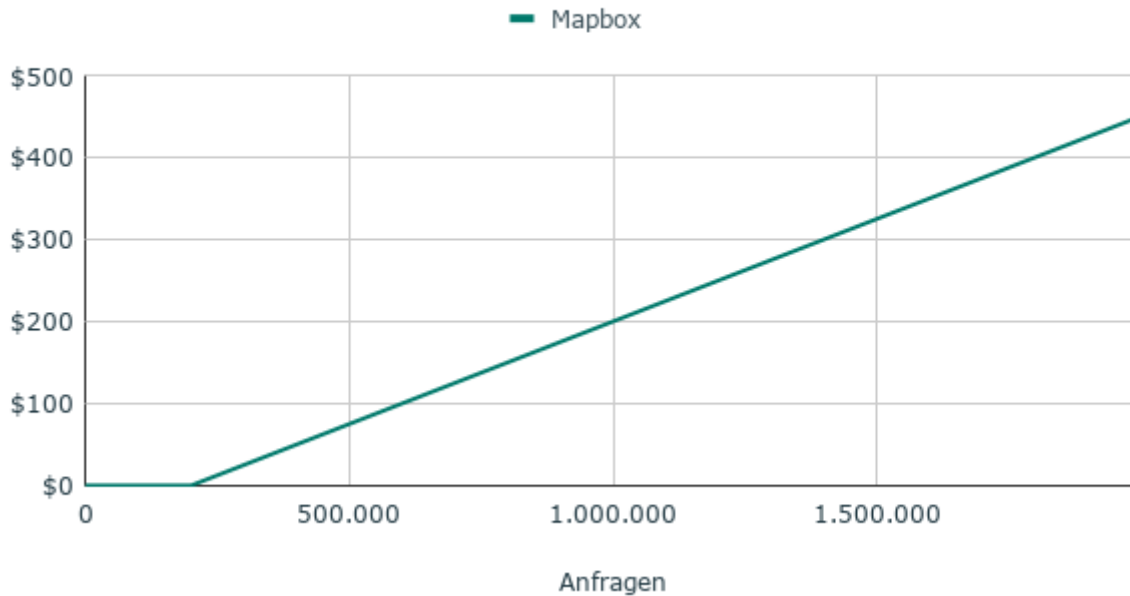
## Geocoding



## MapLoads



## TileLoads



### Umrechnung

Ich habe leider nirgendwo einen guten Vergleich gefunden, auch keine Erfahrungsberichte von Leuten die gewechselt und ihre Zahlen veröffentlicht haben.

Derzeit erreichen wir ca. 40.000 Maploads pro Monat, also ca \$80.

Für den selben Preis erhalten wir bei Mapbox 320.000 Tile requests.

Bei 40.000 Usern sind das dann 8 Tiles pro User.

Für eine detaillierte Suche ist das sicherlich nicht ausreichend, aber gleichzeitig gibt es vermutlich auch viele, die einfach nur auf die Seite gehen und nicht mit der Karte interagieren.

Meine Vermutung ist, dass wir mehr für die Tiles zahlen, jedoch die Kosten für Forward und Reverse Geocoding gegen 0 gehen werden.

Um genau herauszufinden wie viele Tiles der durchschnittliche User benötigt, sollten wir das ganze am besten einfach austesten. [>Tests](#)

### Free

Für die Beschaffung der geojson Polygone für z.B. Länderumrisse werden wir sowieso weiterhin [Nominatim](#) verwenden. Diese Anfragen müssen allerdings gecached werden, da Nominatim ein Ratelimit von 1 req/s angibt. [>Charon](#)

Hier kann man sich dann überlegen ob es nicht sinnvoller wäre alle Geocoding Anfragen über Nominatim zu erledigen.

Solange die Anfragen gecached werden, sehe ich hier keine Nachteile.

Matrix oder Isochrone Anfragen zu einem späteren Zeitpunkt müssten dann allerdings trotzdem über Mapbox oder einen

anderen Anbieter gehen.

## Tests

---

Wie mit Birgit kurz besprochen werde ich dem Server noch ein paar Logging Funktionen hinzufügen, sodass wir nach dem Test abschätzen können, wie die Karte genutzt wird.

Hierfür wäre natürlich wichtig zu definieren, was wir alles wissen wollen.

- Requested Tiles
- Suchanfragen
- ???

## Projekte

---

### Atlas

[github.com/chronark/atlas](https://github.com/chronark/atlas)

Frontend Applikation zur Visualisierung der Jobs.

Die Darstellung der Karte selbst benutzt [openlayers](#) und das Clientseitige Statemanagement wird realisiert mit [redux](#).

### Einbau

Der 1.0 Einbau soll möglichst reibungslos verlaufen, daher wird Atlas zu diesem Zeitpunkt noch keine Jobsuchen selbst durchführen. Eine Liste an aktuellen Jobs muss daher von außen mit `map.setJobs(jobs)` gesetzt werden. Diese Funktion sorgt dafür, dass alle Jobs aktualisiert werden.

## API

Auszug aus [customTypes.ts](#):

```
interface Location {
  /**
   * Latitude of the location.
   */
  lat: number
  /**
   * Longitude of the location.
   */
  lon: number
}

interface Job {
  /**
   * Name of the corporation offering the job.
   */
  corp: string
  /**
   * An array of locations where the job is offered.
   */
  locations: Location[]
  /**
   * The entrydate for the job.
   */
  date: string
  /**
   * Internal id for each job.
   */
  id: number
  /**
   * URL to the job's or company's logo.
   */
  logo: string
  /**
   * Calculated matching score for the user and job.
   * Must be between 0.0 and 1.0 included
   */
  score: number
  /**
   * Job title description.
   */
  title: string
  /**
   * Job classification.
   */
  type: string
  /**
   * URL for more information about this job or company's page.
   */
}
```

```
*/  
url: string  
}
```

Wie von Heiko gewünscht, ermöglicht dies auch die Darstellung von Jobs, die mehrere Orte angegeben haben ohne, dass mehrer Job-Objekte erstellt werden müssen.

## Charon

[github.com/chronark/charon](https://github.com/chronark/charon)

Ich habe vor 1-2 Monaten aus Spaß zu Hause an einem File-cache in [go](#) gebastelt, weil ich dachte, dass man sich damit eigentlich sämtliche Kosten sparen könnte, da die Anfragen immer unterhalb des Gratis-Limits der Provider liegen.

Leider verstößt das gegen die [TOS](#) (2.8) von Mapbox.

Auch wenn wir Tiles und Geocoding von Mapbox nicht cachen dürfen, gibt es trotzdem einige Vorteile:

- Secrets für diverse APIs werden erst beim Server hinzugefügt, damit diese nicht im javascript code an den Endnutzer weiter gegeben werden.
- Nominatim's Ratelimit Problem wird gelöst
- Datenschutz der Endnutzer [>Datenschutz](#)
- Möglichkeit selbst aus dem Verhalten der Nutzer zu lernen [>Statistik](#)

### Datenschutz

Durch einen Proxyserver gelangen die IP-Adresse und sonstige Daten der Endnutzer nicht an den Karten-Provider.

Zwar behaupten diese, dass sie nichts mit den Daten anfangen, aber sicher ist sicher.

### Statistik

Durch den Server können außerdem einige nützliche Daten erhoben werden um genauer heraus zu finden, wie der Kartenservice genutzt wird.

Wie suchen nutzer nach jobs?

Alle Requests an Charon werden automatisch anonym(!) protokolliert und können somit später zur Preisabschätzung und Forschung genutzt werden.

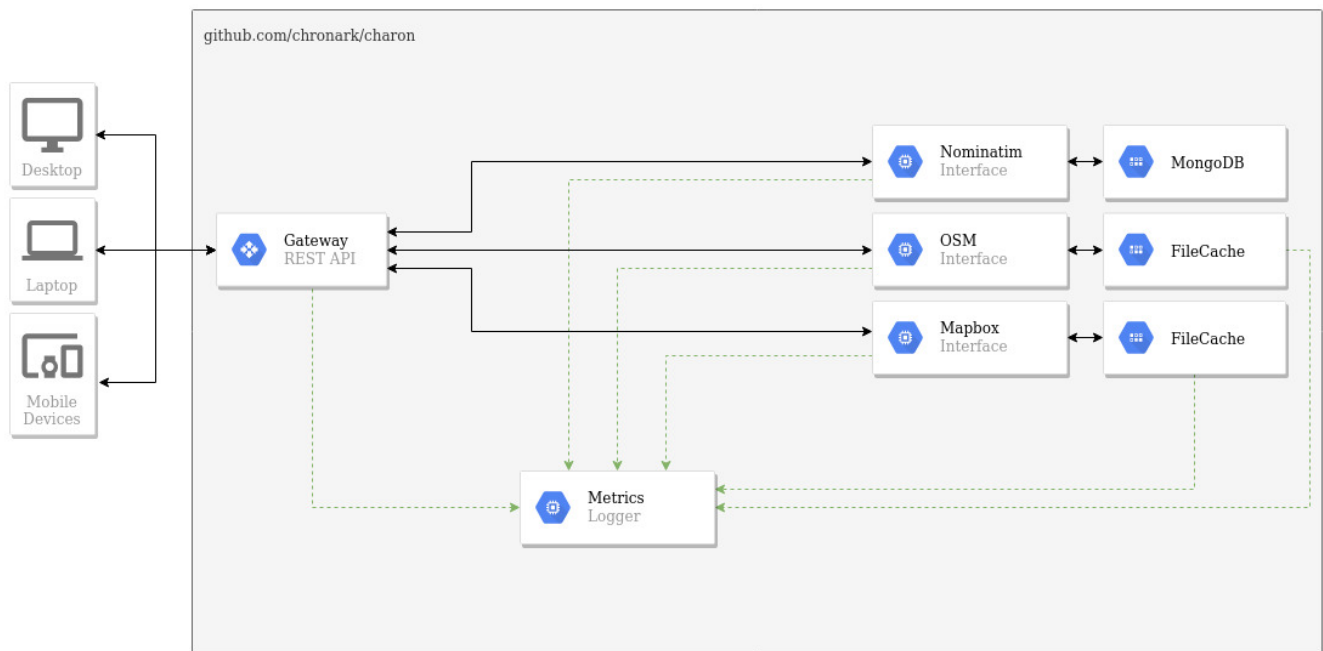
### Aufbau

Charon ist eine kleine Gruppe an Microservices zum Cachen und Protokollieren von Tiles und GeoJSON Daten.

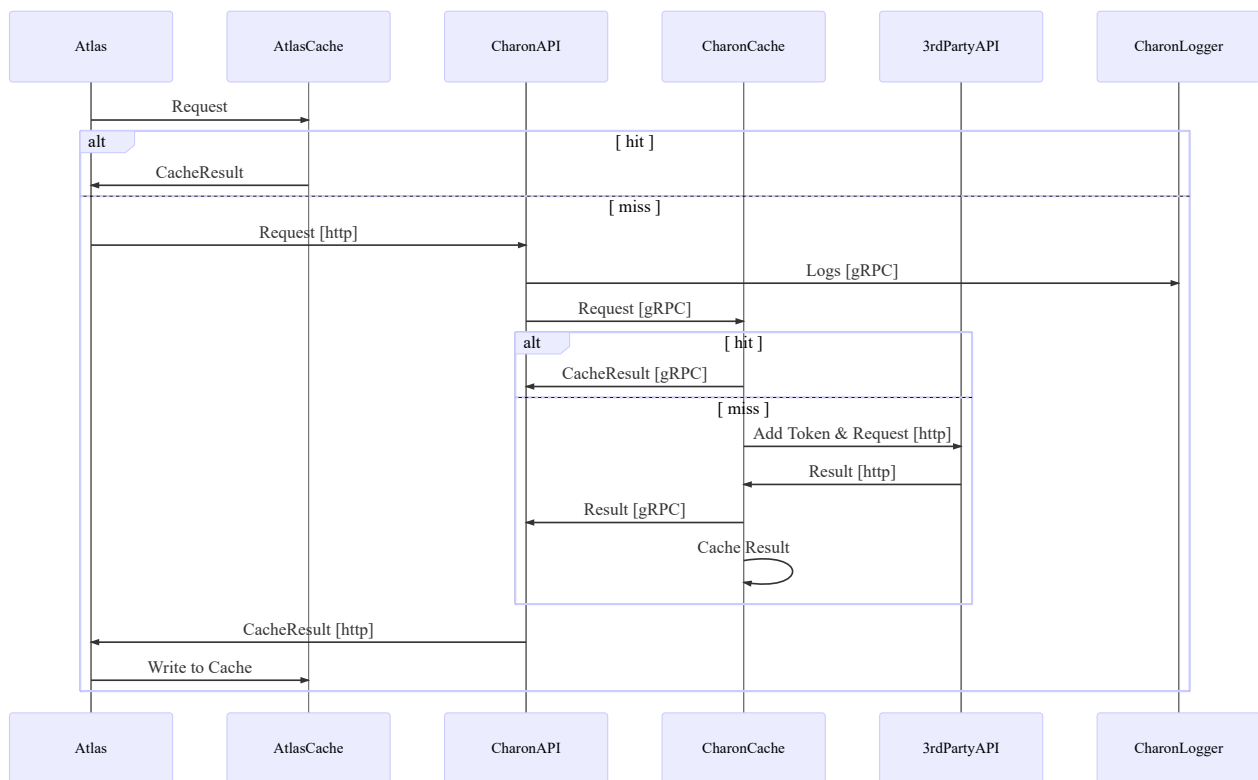
Veröffentlicht werden Endpoints für Tiles und Geocoding Anfragen über http. Die interne Kommunikation läuft über [gRPC](#).

Die einzelnen Services laufen innerhalb von Docker-Containern. Individuelles upgraden, skalieren und auch hot-reloads sind möglich.

### Architektur



## Geocoding/Tile Request Cycle



# Roadmap to 1.0

---

## TODO

### Map

#### React

Um schneller einen funktionierenden Prototypen zu erstellen, habe ich [react](#) benutzt. Das ist jedoch für eine einzelne Karte definitiv nicht nötig. Außerdem werden Frameworks hier ja nicht so gerne gesehen.

Der react Anteil ist allerdings sehr gering und kann wieder entfernt werden.

#### Vector-/Rastertiles

Da Vectortiles und Rastertiles ein bisschen unterschiedlich verarbeitet werden wollen und wir zumindest im Rahmen der Tests auch Rastertiles darstellen und cachen müssen, muss ein bisschen Code umgeschrieben werden, sodass man einfach wechseln kann.

#### Clickverhalten bei Clustern

Die Funktionalität zu entscheiden ob gezoomed wird oder ein Popup geöffnet wird, muss noch implementiert werden.

#### Popup integration

Wie werden diese integriert und dargestellt?

Gibt es ein Design?

Heiko wollte im Hintergrund eine Karte und vorne drauf die Kacheln mit Jobs, soll das komplett von Atlas gerendert werden oder sollen die Daten nach außen geschickt werden und die Webseite selbst rendert das ganze dann?

#### Integration

Die Integration für 1.0 ist möglichst einfach gehalten.

Atlas benutzt [webpack](#) als Bundler und erzeugt damit eine einzelne [.js](#) Datei "atlas.js", sowie 2 [.css](#) Dateien, die von openlayers stammen.

[atlas.js](#) kann dann auf der bestehenden Seite geladen und verwendet werden.

Lediglich ein `<div id="xyz">` ist erforderlich.

```
<script src="atlas.js"></script>
...

<div id="map-container"></div>
```



Die Karte wird dann so initialisiert:

```
jobs = // create Job[] object
map = new atlas.Map("map-container")
map.setJobs(jobs)
```

Die [Job API](#) der Jobbörse gibt derzeit sowohl Orte als auch Jobs zurück.

Atlas führt ein neues [Format](#) ein, dass beides vereint. das wird aber nicht mehr gebraucht. Entweder die API kann geändert werden, oder die Daten werden Clientseitig umgeformt und durch `map.setJobs()` geladen.

## API Änderung

Entweder sollte es einen neuen Endpunkt geben, der den Score bereits hinzufügt und wie [hier](#) formatiert zurück gibt:

```
HTTP/2 200
content-type: application/json
{
  "jobs": [
    {
      ...
    },
    ...
  ]
}
```

## Frontend

Oder existierende Code kümmert sich um die Score Berechnung und erstellt ein Javascript Object, dass den [Job](#) Typ implementiert.

Charon