Cloud Application Development Report

Vorgelegt von:

Timo Haas [redacted]
Oskar Borkenhagen [redacted]
Lars Bürger [redacted]

Gruppe: Porsche

Repo: https://github.com/cloud-porsche/cloud-porsche
App: [redacted]
Landing: [redacted]

an der

HTWG Konstanz

Master Informatik

Konstanz, 2024

Contents

1 Requirements

Figure 1: Techstack

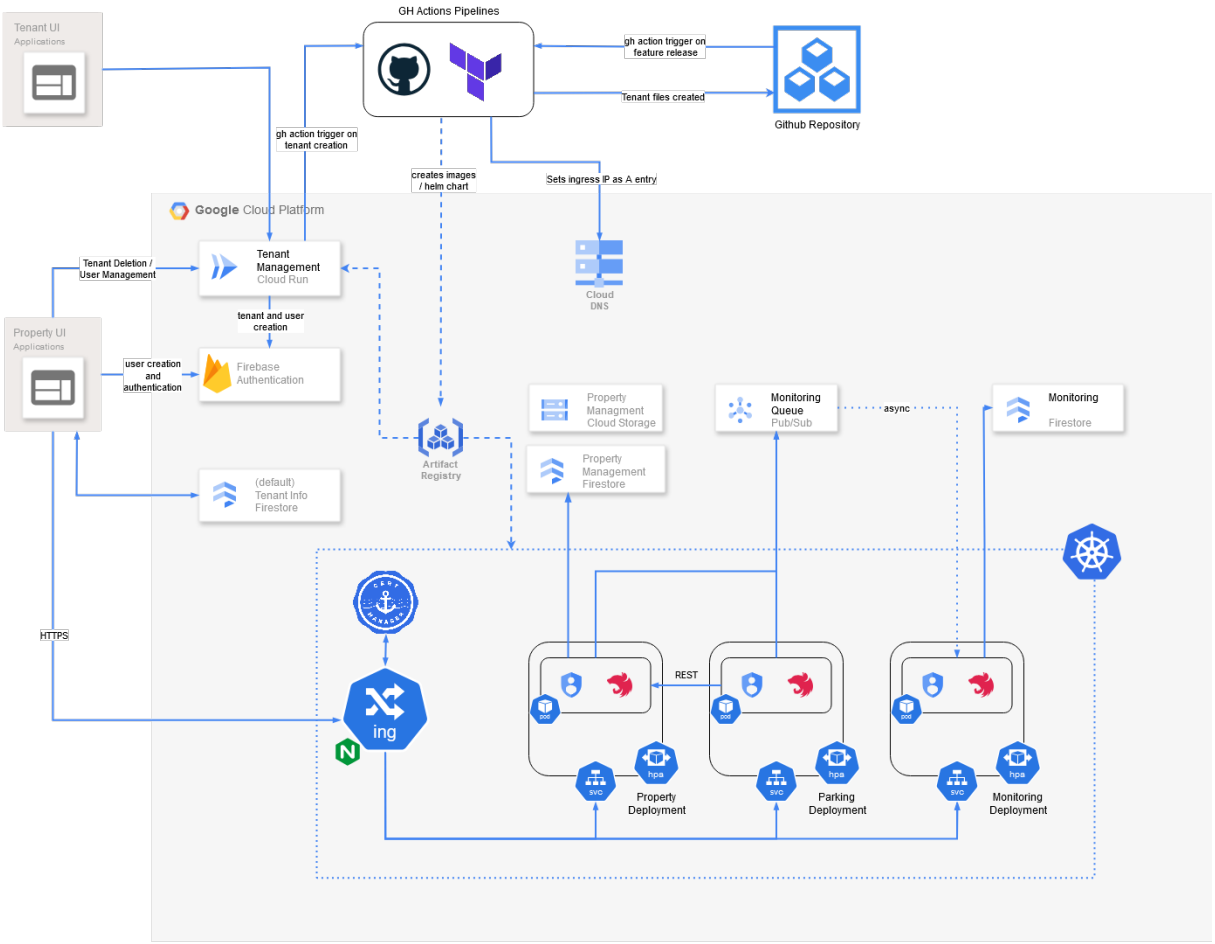

Figure 2: System Overview (1 Enterprise Tenant, bottom k8s block)

The project *Cloud Native Parking Management Application* was developed as part of the *Cloud Application Development* lecture. It aims to deliver a scalable and versatile solution for managing parking facilities. The application is specifically designed to meet the requirements of modern cloud architectures while optimizing the operation and administration of parking infrastructures.

The developed application covers the following core areas:

- **Property Management:** Management of parking facilities and associated assets, including a module for recording and handling defects.
- **Parking Management:** Monitoring and operation of parking spaces, including entry and exit control as well as real-time occupancy monitoring.
- **Reporting and Analytics:** Financial and operational reporting along with analytics to support data-driven decision-making.

Additionally, an API for external devices and applications has been implemented, enabling seamless integration with payment systems, entry and exit barriers, and occupancy displays.

To support the SaaS approach, the application features a multi-tenant architecture with three distinct pricing tiers (*Free*, *Pro*, *Enterprise*). These models allow for flexible customization to user needs and provide a clear differentiation of offered services and service levels.

The system is fully cloud-based and utilizes a microservice architecture. This not only satisfies the requirements of a modern cloud-native application but also ensures high efficiency and reliability through automated provisioning and scalability.

1.1 System Context Diagram

Figure 3: System Context Diagramm of the application

1.2 Use Case Overview

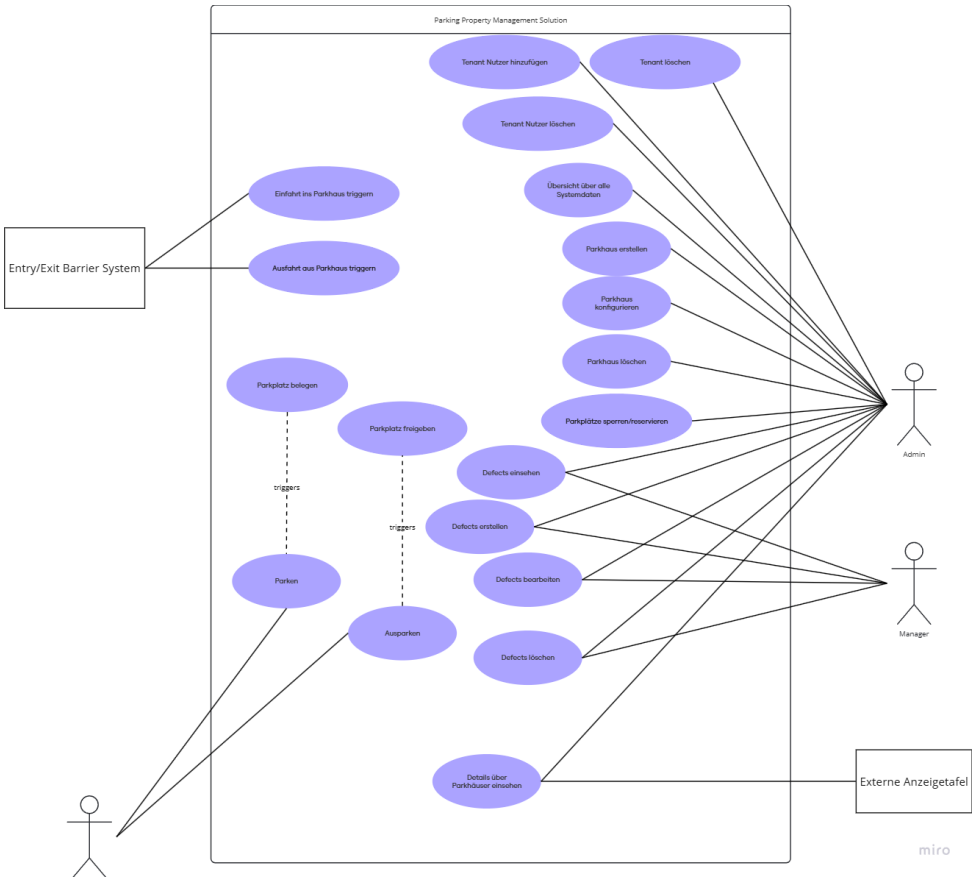

Figure 4: Use Case Diagram

2 Development View

2.1 Source Code Repositories and Organization

The source code for the application is hosted on GitHub, organized using a monorepo structure. Each service is placed in its own dedicated folder. The included services and components are:

- **Services:**
  - ParkingService
  - PropertyService
  - MonitoringService
  - TenantService
- **Frontend Applications:**
  - Tenant UI
  - Property UI
- **Shared Types:** A separate folder for shared types ensures consistency and seamless integration across all services.

2.2 Software Components and Technologies

2.2.1 Backend Services

The backend services include:

- ParkingService
- PropertyService
- MonitoringService
- TenantService

All backend services are implemented in TypeScript and use the Nest.js framework to ensure scalability and maintainability.

2.2.2 Frontend Applications

- **Property UI:** Developed using Vue.js and Vuetify, based on Material Design principles.
- **Tenant UI:** Built with Nuxt.js, an advanced framework for Vue.js applications.

2.3 Important Libraries and Dependencies

The following libraries play a critical role in the development:

- **nestjs:** Base framework of every microservice
- **typescript:** Used everywhere
- **google-cloud/pubsub:** Monitoring service async data collection (message queue)
- **firebase-admin:** Enables integration with Firebase services, including:
  - Authentication
  - CRUD operations with Firestore databases
  - Google Cloud services such as Pub/Sub queues and object storage
- **fireorm:** Simplifies object-relational mapping and integrates seamlessly with Firestore.
- **highcharts:** Used extensively in admin dashboards for interactive visualizations.
- **socket.io:** WebSocket communication
- **octokit:** GitHub API used in Tenant Service (Tenant creation/GH Actions trigger)
- **nuxt + nuxt/ui:** Tenant/Landing Page
- **vue + vuetify + vite + VitePWA:** Main App UI

2.4 External Interfaces

The **Property API** offers an external interface that enables third-party providers to:

- Access real-time property capacity data.

This interface is crucial for integrating external systems and ensuring smooth data exchange.

3 Data Model

In our solution, we utilize Firebase from Google Cloud to handle persistent storage across various services. The architecture is structured in a way that different tenants have distinct databases, with tiered access to shared or individual databases based on their subscription level. Free-Tier and Pro-Tier have each one shared cluster and also shared databases. The difference between them is that the Pro-Tier cluster has Pod auto scaler and thus a better performance on heavier load than the Free-Tier. Each enterprise-tenant has their own cluster with isolated databases and storage. (**WOW-Factor**: deployment has a mix of isolated, shared and pooled data storage).

The core databases include:

- **(default) Database:** Contains the basic tenant information accessible to all users, depending on their tier. (document access given only to authorized users)
- **Property Service Database:** Stores detailed property and defect-related information specific to each tenant.
- **Monitoring Service Database:** Holds telemetry data across three core collections, focused on API calls, defect actions, and parking actions, which are essential for tracking usage, managing billing, and visualizing activity.

3.1 (default) Database

The Property Service Database is dedicated to storing property-specific data for each tenant. This database manages information related to properties, including details about individual properties and their associated defects. Each property document links to one or more defects via a defectId field, which allows for efficient tracking and management of defects within the system. This structure enables tenants to maintain a comprehensive record of their properties and any issues that require attention. The database schema supports easy retrieval and updating of property-related data, ensuring smooth operation of property management services.

3.2 Property Database

The Property Service Database is dedicated to storing property-specific data for each tenant. This database manages information related to properties, including details about individual properties and their associated defects. Each property document links to one or more defects

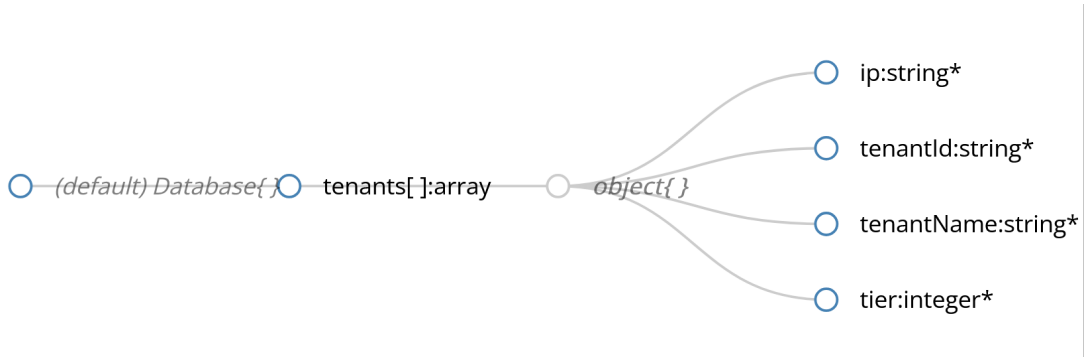

Figure 5: Tenant Collection JSON Schema

via a defectId field, which allows for efficient tracking and management of defects within the system. This structure enables tenants to maintain a comprehensive record of their properties and any issues that require attention. The database schema supports easy retrieval and updating of property-related data, ensuring smooth operation of property management services.

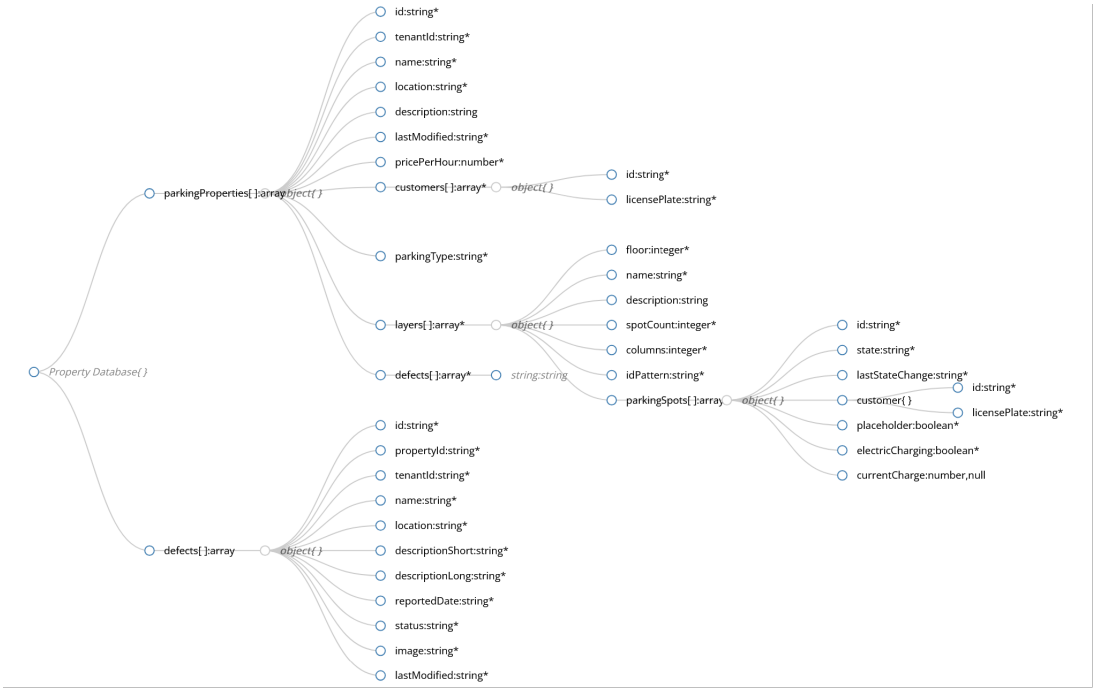

Figure 6: Property Collection JSON Schema

3.3 Monitoring Database

The Monitoring Service Database consists of three important collections for tracking and analyzing telemetry data related to system usage:

- **ApiCalls Collection:** Records every API call made from either the Property Service or the Parking Service.

  - For Free tier tenants, this data is used to monitor the number of remaining API calls within the free quota before restrictions are imposed.
  - For Pro and Enterprise tenants, the data is utilized to calculate additional charges based on the number of API calls exceeding the allocated limit.

- **DefectsActions Collection:** Tracks all CRUD (Create, Read, Update, Delete) actions performed on defects. This telemetry data is essential for generating insights and displaying action history on the admin dashboard. It provides valuable insights into user interactions with defects, monitoring system usage trends.

- **ParkingActions Collection:** Captures data related to actions performed on parking properties. This is the largest collection in the monitoring database and is critical for tracking parking-related activity. It supports calculations of a tenant's income within specific timeframes based on parking usage, helping tenants monitor financial performance and usage trends.

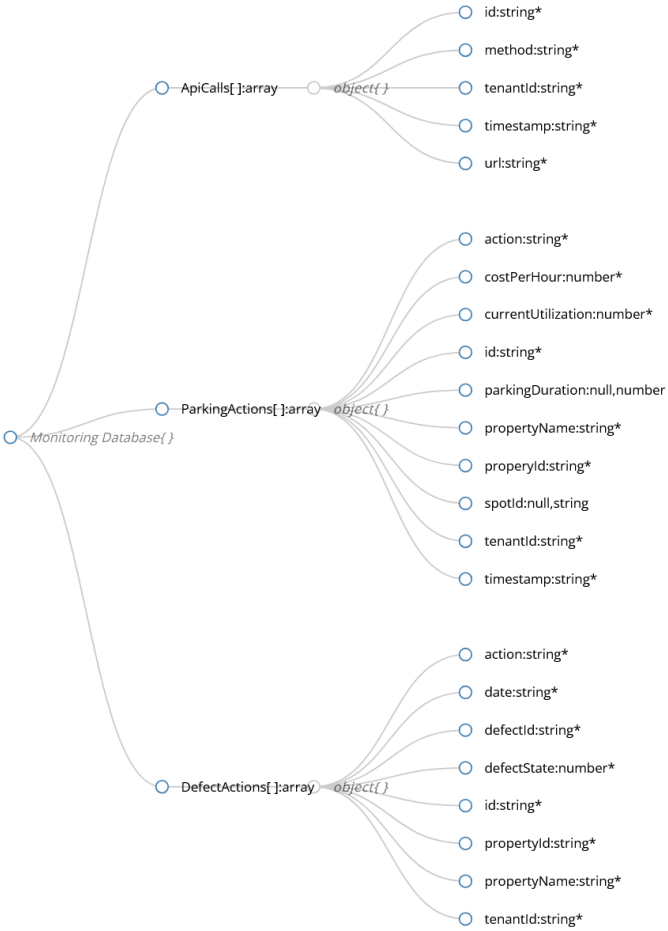

Figure 7: Monitoring Collection JSON Schema

3.4 Google Cloud Image Storage

The application uses gcloud storage to store and retrieve images associated with defects. Free and Pro-Tenants are each sharing one storage bucket while every enterprise tenant has its own resource. Images are displayed in the user profile and the defect list of every property like seen in Figure 8. **(Hygiene-Factor: shared and isolated storage resources)**

Figure 8: Example defect with associated image

4 Runtime View

4.1 Tenant Identification Approach and Authentication mechanisms

4.1.1 Frontend

In the frontend, we identify each tenant by adding their `tenantId` to the URL path. For example, the URL changes from `cloud-porsche.com/#/` to `cloud-porsche.com/#/tenant1/`. If no `tenantId` is specified, the user is automatically redirected to `/free-tier/`.

When a user logs in, there are two scenarios:

- **Free-Tier Login**: If the tenantId equals `free-tier`, the user logs in without a specific tenant, which means they are limited to only one user per account and cannot add new users.
- **Pro/Enterprise-Tier Login**: The `tenantId` is included in the header.

To ensure security, if a user changes the `tenantId` in the URL path and it doesn't match their logged-in tenant, they will be immediately logged out. If the user's tenantId changes during authentication, the app will attempt to update the user, and the process fails if the tenantId is no longer valid.

Sort-of-WOW-Factor: PWA

The main UI is a PWA, supporting chrome/safari installation.

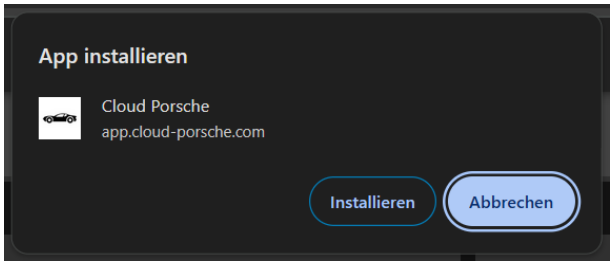

Figure 9: PWA install dialog

4.1.2 Backend

For each request from the frontend, the user's authentication token is included, along with an optional `tenantId`:

- **Free-Tier Users**: The request header doesn't contain a `tenantId`

- **Pro/Enterprise-Tier Users**: The `tenantId` is included in the header.

Each backend service has an authentication middleware that checks whether the user is authorized. If a Pro or Enterprise tenant is logged in, the service ensures the request is tied to that tenant. If the user is a free-tier user, the backend assumes they are not associated with a tenant. Once a free-tier user is authenticated, their UID is set as the `tenant-id` in the request header. This approach ensures the application can differentiate between free-tier and Pro/Enterprise tenants.

4.1.3 Database Security

Each document in the database contains a `tenantId` field. For free-tier users, the `tenantId` is their UID because they don't belong to a specific tenant. For Pro and Enterprise tenants, the `tenantId` matches the tenant's ID. This setup ensures that:

- Free-Tier users access only their own properties (using their `UID` as `tenantId`)
- Pro/Enterprise tenants only access properties linked to their specific organization.

This way, the system ensures that users only see the properties they are authorized to view.

4.2 Microservices

4.2.1 Security Setup

Every Microservice contains an authentication middleware that authenticates the request with a user token

Token is generated by the user and depending on a free tier or higher tier tenant the middleware authenticates either to or to no tenant. Next to the authentication, specific requests need a authorization by using role-guards at the said requests. **(Hygiene-Factor: Multi-User Service with authentication and authorization**)

Theres three roles a user can have:

- **user**: A normal user is only able to view everything. he cant create, update or delete parking properties or defects. He also cant add, update or remove users of the tenant hes in.
- **manager**: The manager role has the same rights as the user but he's additionally able to create, update and delete the defects of the parking properties of the tenant he's in.
- **admin**: The most powerful role is the admin-role. The admin is able to execute every CRUD operation on the properties, the defects and the users of the tenant like seen in Figure 10. **(Hygiene-Factor: tenant-management capabilities for the provider of the solution)**

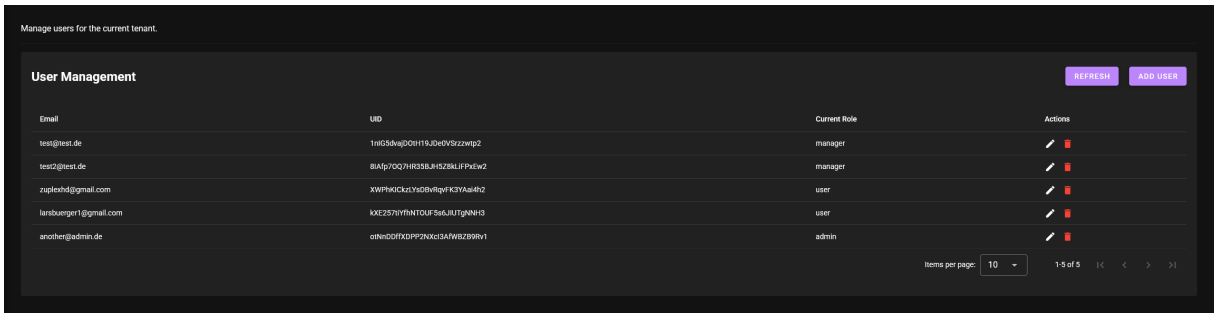

Figure 10: User Management for users with the admin role

4.2.2 PropertyManagement Service

This service handles all CRUD operations related to user properties and property defects. It includes a storage service for uploading and retrieving images, integrated with Google Cloud Storage.

A WebSocket (WS) gateway is included, allowing UI clients to connect and receive live updates on property changes. The WebSocket listens to Firestore database snapshots and broadcasts updates to all connected clients in real-time.

The service also incorporates a Pub/Sub system that connects to an external Google Cloud Pub/Sub message queue. It processes telemetry data by queuing relevant actions asynchronously.

Additionally, it connects to an external Firebase database for storing property and defect data, and to Google Cloud Storage for saving and retrieving images associated with defects.

4.2.3 ParkingManagement Service

The ParkingManagement Service handles all parking related operation like listed below:

- **enter**: Customer enters a parking property
- **occupy**: Customer occupies a specific spot
- **free**: Customer frees a specific spot
- **leave**: Customer leaves the parking property

This service additionally includes a simulation that can simulate workload by letting customers enter, park, free and leave the specified parking property. The simulation contains multiple speeds to also simulate a higher load on the microservices. (**WOW-Factor**: based on this simulated workload it also shows that the services in the Pro-Tier automatically scale on higher load like seen in Figure 11)

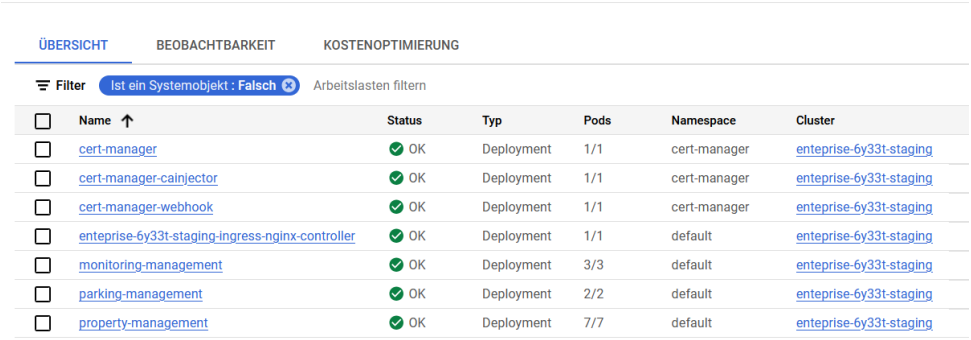

Figure 11: Pod-Autoscaling while running 4 simulations

4.2.4 Monitoring Service

The monitoring service is designed to collect data generated by the parking and monitoring systems and store it in the monitoring database. This process is handled asynchronously using a pub/sub task queue, ensuring that no data is lost, even if the monitoring service is temporarily unavailable due to shutdowns or crashes. **(Hygiene-Factor: Analytics workload is handled asynchronous)**

In addition to data collection, the service provides telemetry data for the frontend, enabling users to monitor and visualize their application's performance. (**WOW-Factor**: More complex customization capability as extension point for custom data)

- **Free-tier users** receive a limited subset of telemetry data, ensuring basic insights. See Figure 12

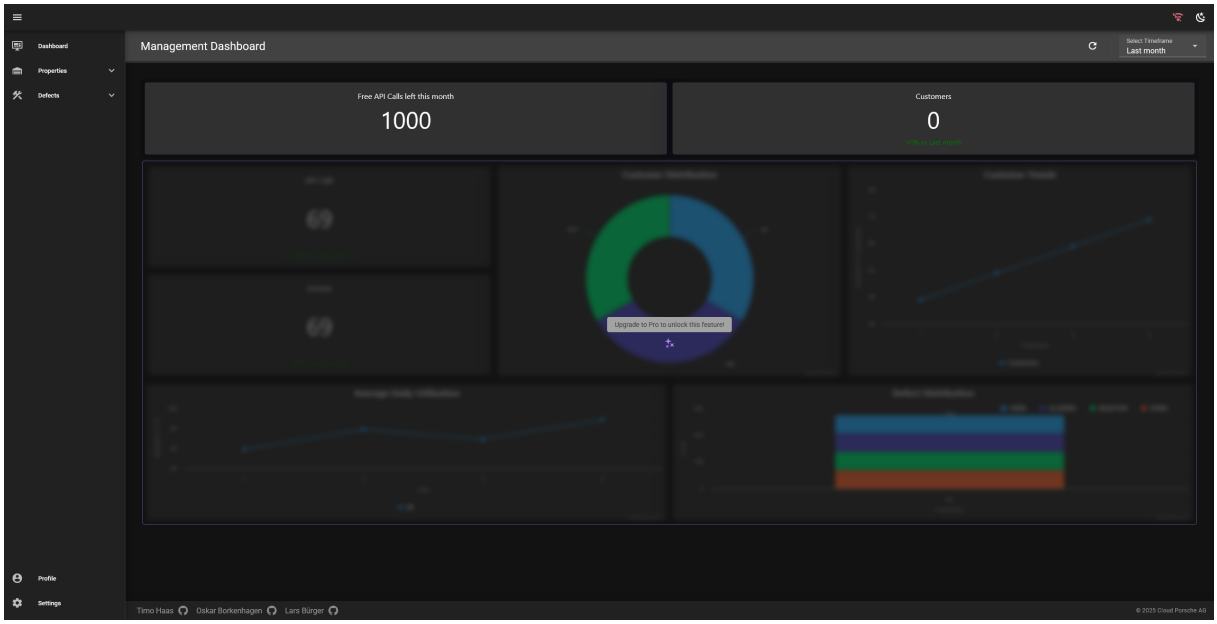

Figure 12: Dashboard with very limited telemetry data

- **Pro and Enterprise tenants** gain access to comprehensive and detailed information about their applications, tailored to their advanced needs. See Figure 13

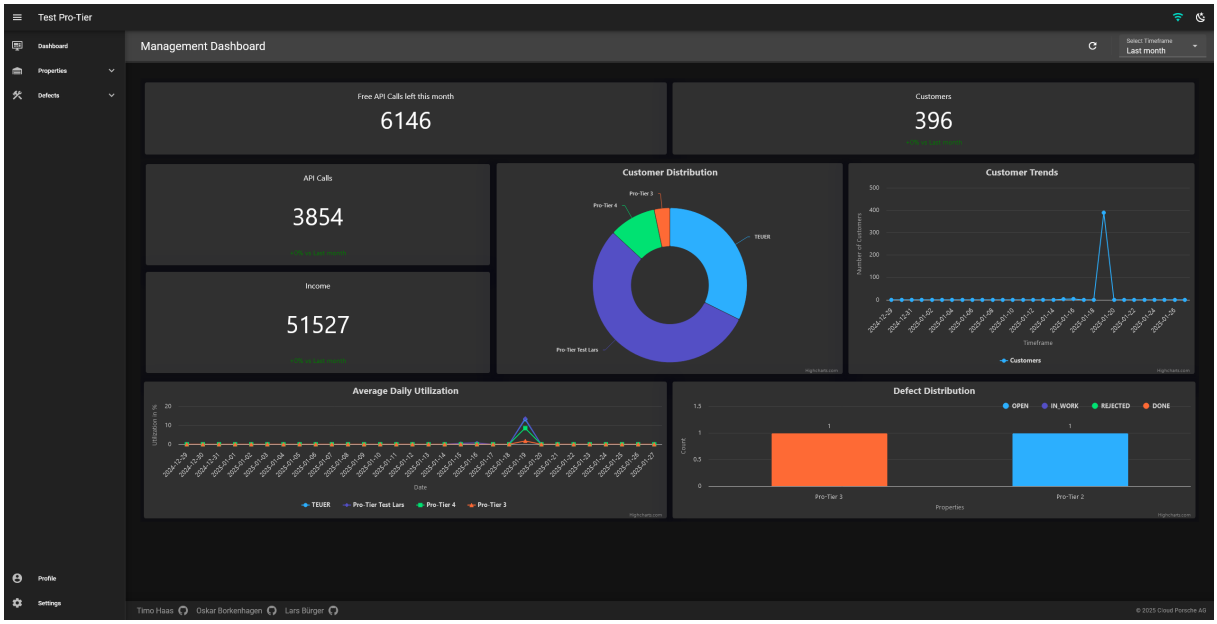

Figure 13: Extensive statistics for Pro and Enterprise Users

4.2.5 Tenant Service and Tenant UI

See at 5.4

4.2.6 PropertyManagement UI

The Frontend Microservice provides a user-friendly interface for managing parking properties, general functions like settings, account information, and the monitoring dashboard. Users can create and delete parking garages directly through the interface, while Pro and Enterprise customers enjoy advanced customization options, including multi-floor layouts and grid-based parking space design to mirror real-world facilities 14. They can also toggle between 2D and exclusive 3D representations for each parking garage 15 and simulate workloads to monitor car entries and exits 16. **WOW-Factor**: Complex customization capabilities through extension points for custom functionality

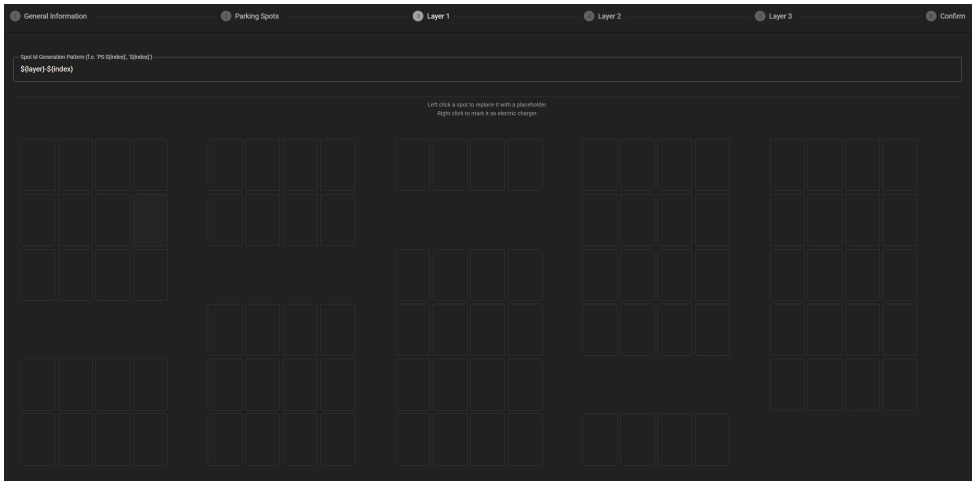

Figure 14: Extended customization for pro and enterprise tenants

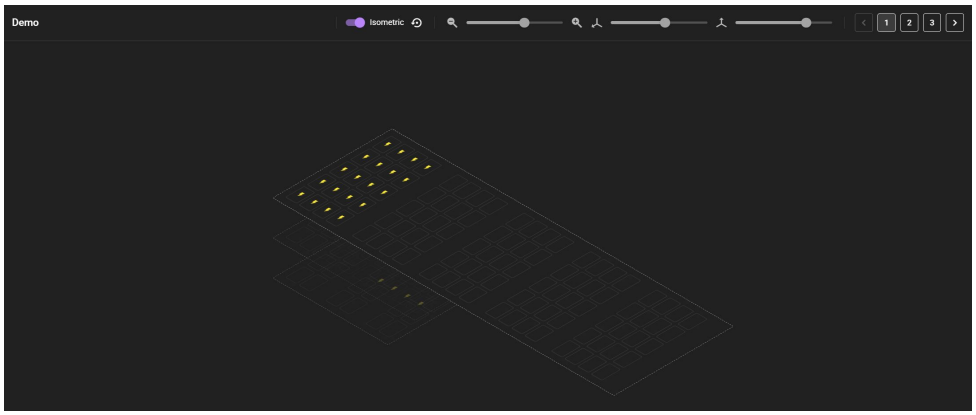

Figure 15: 3-D Model of the properties

Additionally the user is able to mark specific places as electronic charger to also be able to integrate its e-charging infrastructure as the application is also able to track how much a customer charged up to calculate an appropriate pricing. (**WOW-Factor**: E-Charging Infrastructure)

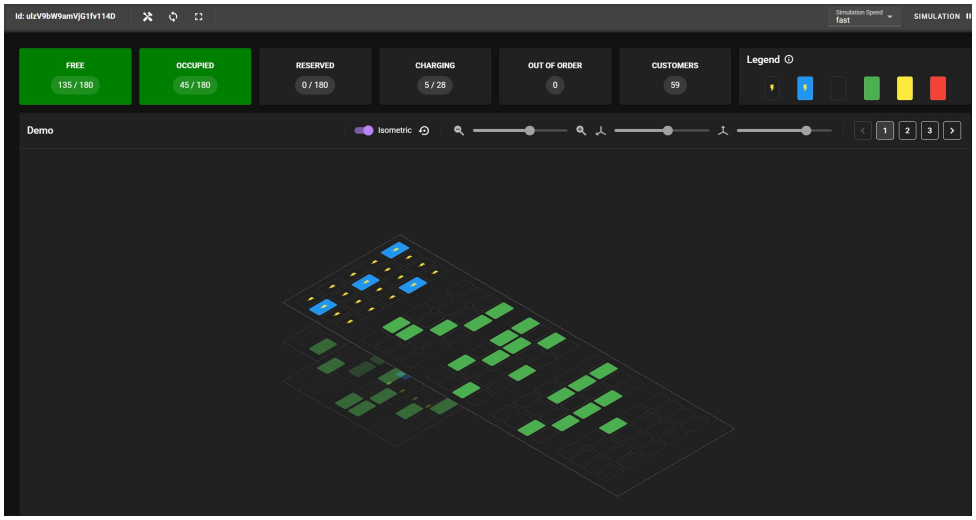

Figure 16: 3-D Model with running simulation

Sort of WOW: The application UI was designed for maximum user-friendliness and responsiveness. Significant effort was dedicated to the frontend to ensure a seamless and comfortable experience for every tenant.

5 DevOps

5.1 Environments

We have a complete separation for staging and production: **(Hygiene-Factor: A clear separation between staging and production environments ensures stability and reliability)**

- UI:
  Deploy to sub-folder (develop → staging, master → prod);
  results: app.cloud-porsche/prod/#/... and app.cloud-porsche/staging/#/...
- Terraform:
  2 sub-folder containing tenant files + shared modules for module based resource creation (so we have seperate tenants for staging/prod, since actual tenants are always prod and testing tenants can be created for staging (free/pro tier are duplicated))

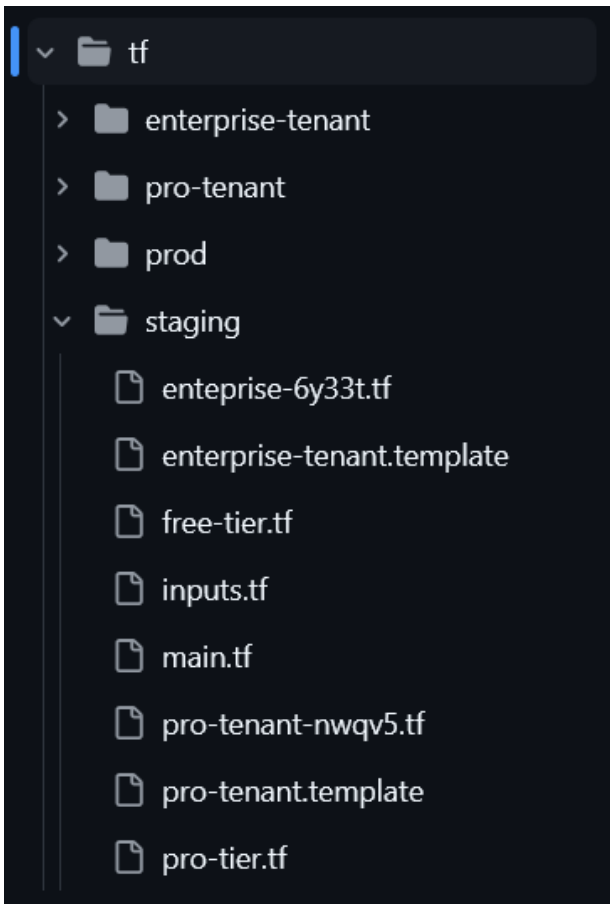

Figure 17: GitHub Terraform folder structure

5.1.1 Certificates / K8s cert manager

Hygiene-Factor: HTTPS / secure networking

Both environments (staging/prod) are secured with let's encrypt and cert-manager, installed in the corresponding clusters (ClusterIssuer). For each cluster a variable can be set to switch the certificate api endpoint. In production we receive a production ready let's encrypt certificate while for staging we use the staging, non-root certificate which can be refreshed much more often (Let's Encrypt - Staging). This is needed since in development the 5 production certificates refresh limit was used up by removing/creating clusters too often. Also see 2 for a k8s cluster overview.

5.2 Roles and Role Mapping

1 service account per enterprise tenant (and free-tier and pro-tier as they sort of count like an enterprise tenant):

- roles/datastore.owner // to access firestore db's
- roles/storage.admin // to access buckets
- roles/pubsub.admin // to use/create subscriptions to message queues
- roles/dns.admin // to create an initial A record in cloud dns

1 extra service account for local development, and 1 extra service account for tenant-management service **(Hygiene-Factor: identity and access management)**
Additionally, firestore access is secured with firebase security rules, allowing access to data only when:

- authenticated
- accessing data corresponding to the current tenant id, or if in free tier, the user id instead

5.3 Pipelines and Release of New Features

(**WOW-Factor**: Fully automated deployment of new software versions, enabling the seamless delivery of features or bug fixes)
A push to the `develop` branch triggers a build in the `staging/latest` environment.
Similarly, a push to the `master` branch initiates a build in the `prod/stable` environment.
Terraform can be executed via dispatch in either the `staging` or `production` environments, enabling a completely automated deployment process for new features. **(Hygiene-Factor: maintaining a continuous, fully automated delivery pipeline.)**

Website Build static files & Deploy to GitHub Pages 18

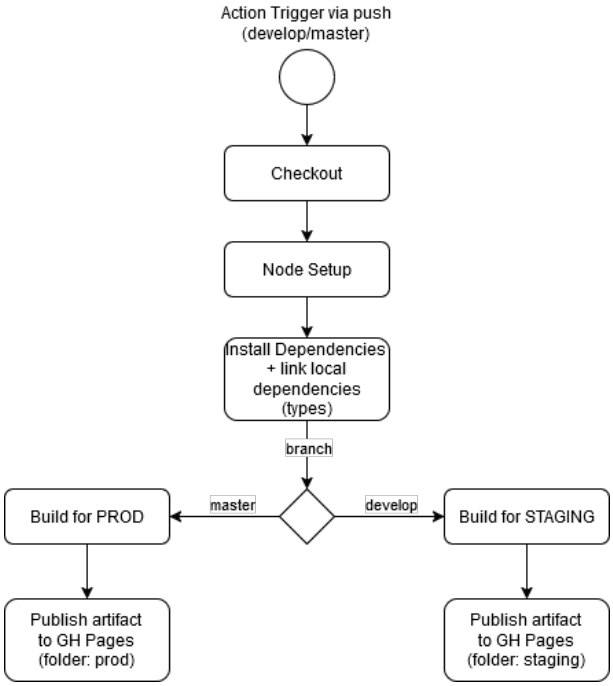

Figure 18: Web App Build/Publish - Workflow

Build Docker images and helm tarball and publish them to GCloud Artifact Registry 19

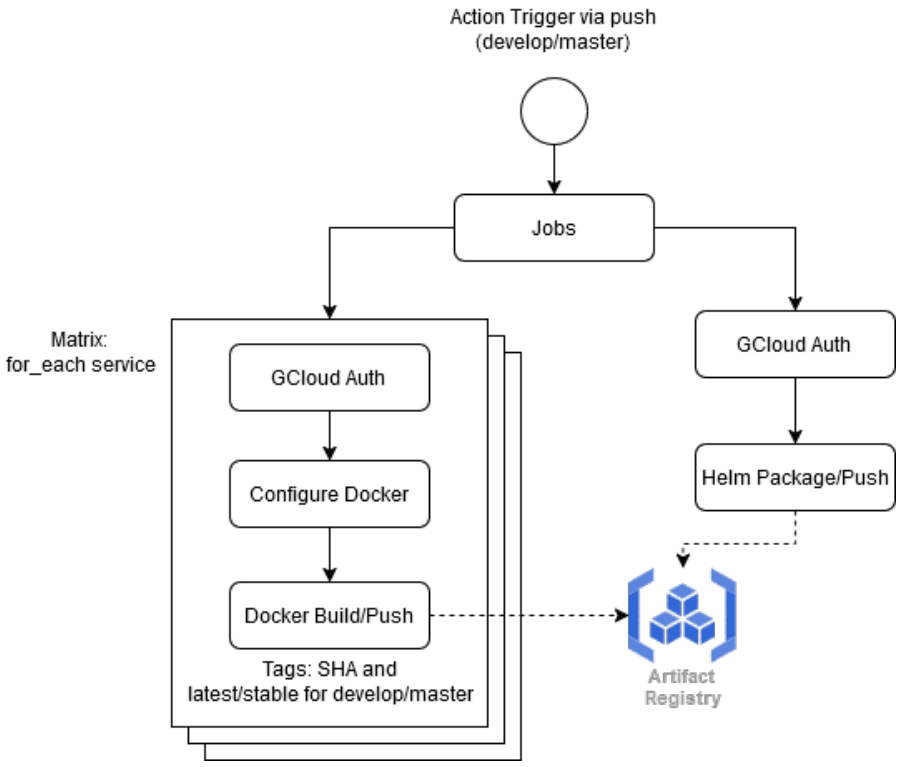

Figure 19: GCloud Microservice Docker Build/Push - Workflow

Tenant Management Service Docker build (including ui build and copy to static file folder of same service) and deploy to cloud run 20

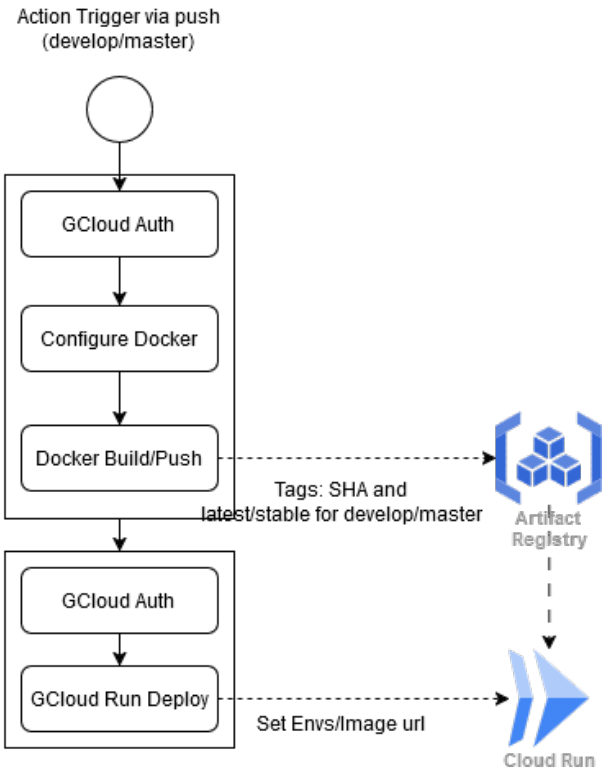

Figure 20: Tenant Management Service

Terraform - Includes 4 Exclusive Jobs with manual triggers with parameters; Apply/Destroy/Create Tenant/Delete Tenant 21

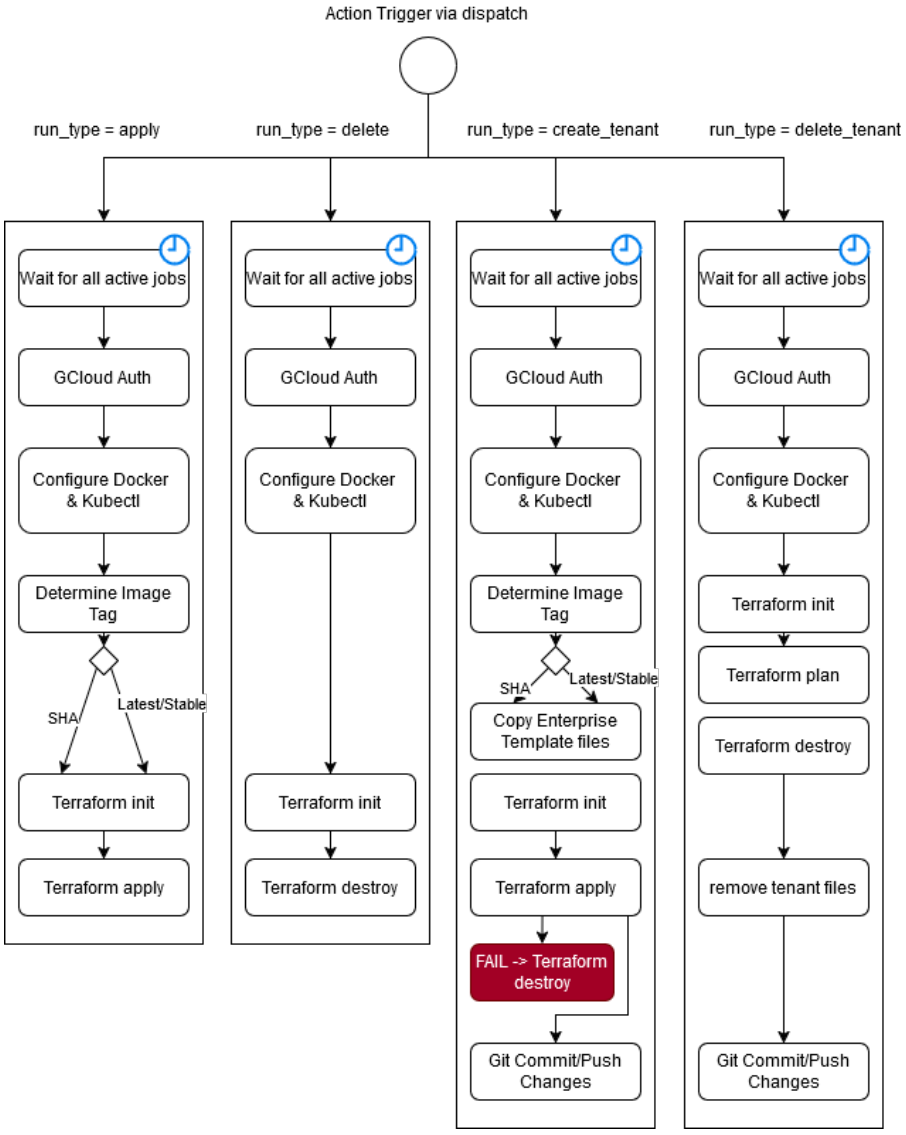

Figure 21: Terraform - Workflows

5.4 New Tenants

https://tenant-management.prod.cloud-porsche.com/
(**WOW-Factor**: Completely automated Tenant Creation)

1. Register/Buy Page - (**Sort-of-WOW-Factor**: Including Location Customization for Enterprise 22)
2. Service creates auth tenant + triggers gh action job
3. Creates terraform template file with necessary information
4. Applies new tenant resources to gcloud
   4.5 If Terraform fails, destroy again in case something was created
5. Commits new tenant to repo

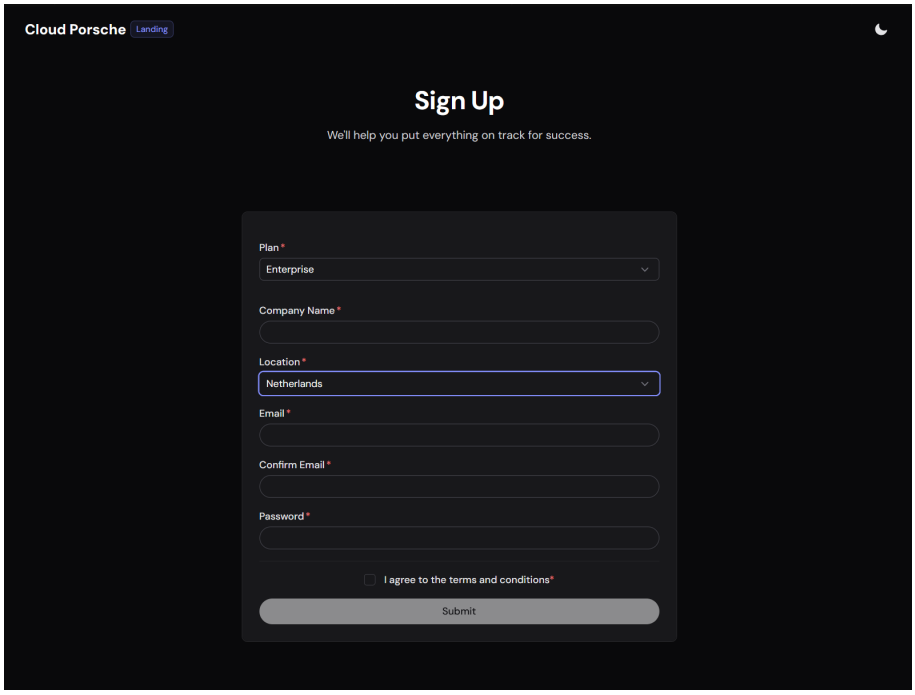

Figure 22: Tenant Signup Page

(**WOW-Factor**: Automated Tenant Deletion) 23
Admin only tenant deletion (in prod, data will be left to manual deletion for data safety)
(**WOW-Factor**: Tenant Data Migration) 24
Admin only tenant data migration (2 tenant admin accounts must exist)
This will use the (cluster external) tenant management service to access both firestore db's (old/new tenant or free/pro db) and copy the files over (or change tenant ids if in the same tier). This way migration from any tier is possible to all tiers except free-tier.

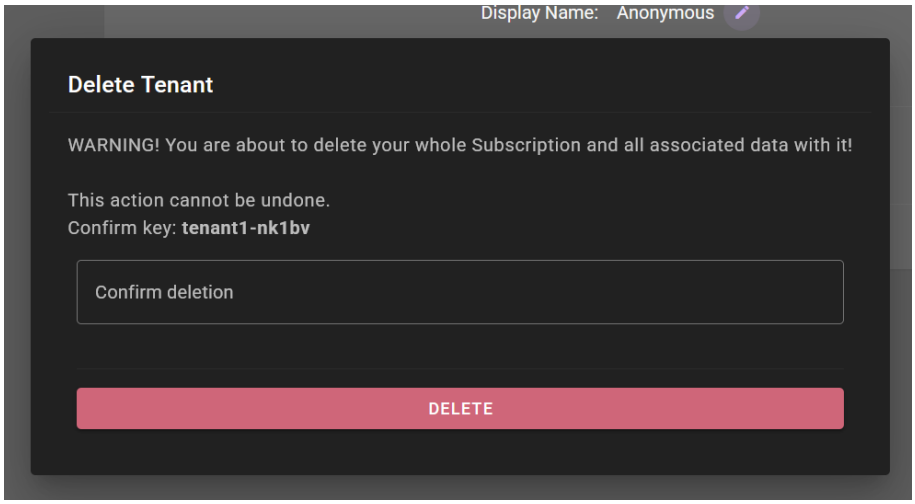

Figure 23: Tenant Deletion Dialog

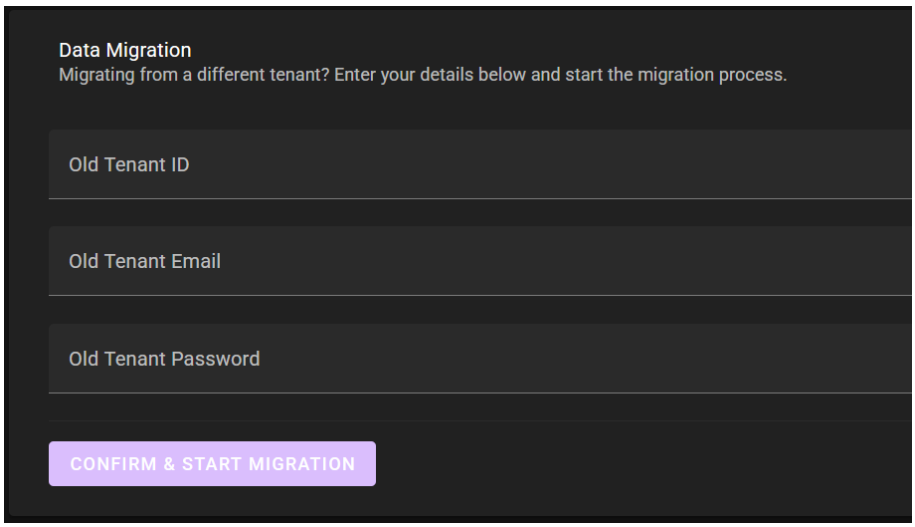

Figure 24: Tenant Migration Setting

5.5 Load Testing

Done via Simulation feature, see 4.2.3.

6 Commercial Model

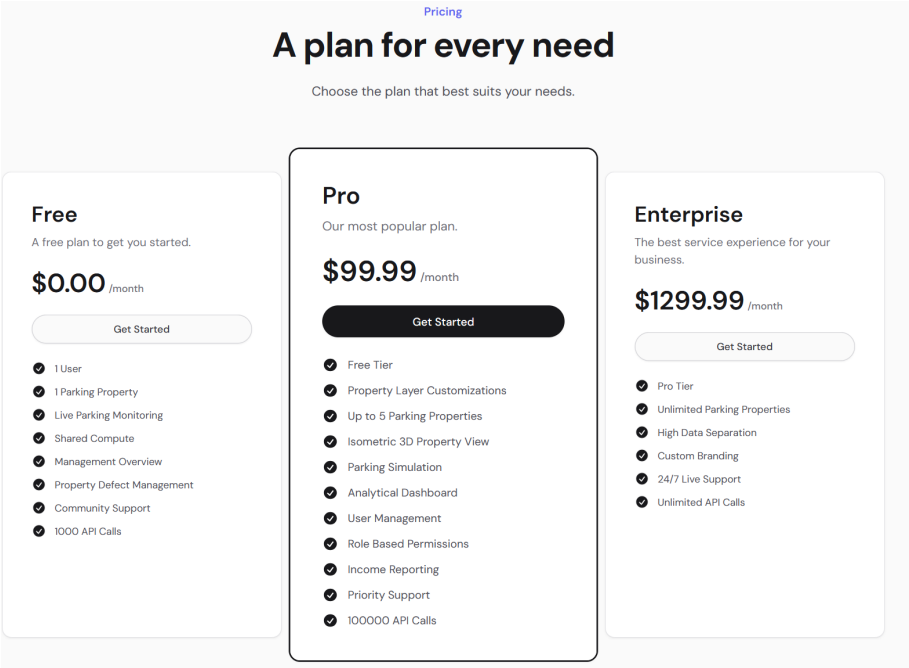

Figure 25: Tenant Pricing

6.1 Tenant Types

(**WOW-Factor**: The deployment has a mix of isolated, shared and pooled workloads)

6.1.1 Free-Tier

The Free Tier is designed for individuals or small-scale parking management with minimal complexity and shared resources.

- Management of only one parking garage
  - Low complexity (e.g., only see total number of parking spaces available/used, no levels or varying prices)
  - No E-charging support
  - No facility manager role (only one user allowed)
  - Best Effort support

- Shared databases and services
- Ultra Basic Analytics

6.1.2 Pro-Tier

The Standard Tier offers more features for small-to-medium businesses managing multiple parking garages with dedicated resources.

- Management of up to five parking garages
- Defect management included
- Dedicated databases or storage buckets with shared services
- Better Performance (higher mem/cpu shared cluster)
- Unlock all hidden pro features (3D-View, Role Management, ...)
- Standard support
- Basic Analytics

6.1.3 Enterprise-Tier

The Enterprise Tier provides a comprehensive solution for large-scale parking management with advanced analytics, priority support, and high resource separation.

- Unlimited number of parking garages
- Dedicated services and high resource separation (e.g. own cluster/db's/...)
- Priority support
- Detailed Analytics with export functionality

6.2 Pricing Model

Sort of WOW: More dynamic pricing models are preferred over simple down payment models, although they may have their place. Dynamic pricing models should be built upon own telemetry data.

Base Price with dynamic cap (API Call limit)

Reaching the api call limit will result in being locked out of the application for the rest of the month/period.
This really only affects the free tier as pro/enterprise limits are only to prevent misuse 26.

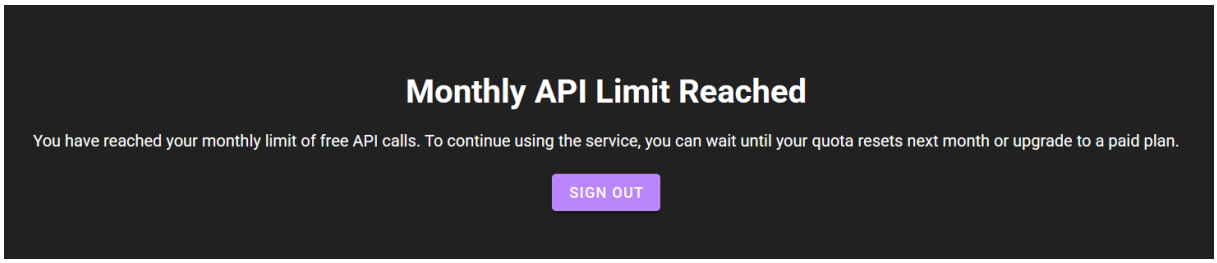

Figure 26: Monthly API limit reached Dialog

6.3 Cost Model

Our cost is mainly oriented at the worst-case enterprise tenant cost calculation, calculations are below.
Values were used after simulating a realistic workload for about half a day using our Simulation feature.
Pro Tier is estimated to have about 13 tenants share the same cluster/resources as 1 enterprise tenant.

WOW-Factor: Provide Detailed Cost Analysis based on usage parameters and a profitability calculation

Best Case Estimate

This estimate is a more optimistic approach as it assumes occassional spread out traffic resulting in almost no scaling, keeping resource usage to a minimum (1 Node, 2 vCPUs, 6GB RAM).

Total estimated cost

160,14 $ / mo

Download .csv

Duplicate estimate

Next steps

Request a custom quote

Connect with our sales team and learn about any eligible discounts.

Contact sales

Start your proof of concept

Sign up for Google Cloud, new customers get $300 in free credits.

Get started

Google Cloud

Cost Estimate Summary

As of 20.01.2025 • 18:16

Prices in USD

9,95 $

Artifact Registry	9,95 $
Service type Artifact Registry	
Storage 100 GiB	9,95 $

COMPUTE

131,59 $

GKE Best Case (Kubernetes Engine)	126,81 $
Service type GKE	
Number of Regional clusters 1	73,00 $
Boot disk type Balanced persistent disk	N/A
Boot disk size (GiB) 20 GiB	2,20 $
Node-time 730 Hours	N/A
Machine type custom, vCPUs: 2, RAM: 6 GB	51,61 $
Number of Nodes 1	N/A
Operating System / Software Free: Container-optimized	N/A
Kubernetes Edition GKE Standard Edition	N/A
Provisioning model Regular	N/A
Enable Confidential GKE Nodes false	N/A
Add GPUs false	N/A
Region Netherlands (europe-west4)	N/A
Committed use discount options None	N/A
Cloud Run	4,78 $

Service type Cloud Run	
CPU amount per instance 1	1,50 $
Memory amount per instance 0.5 GiB	0,08 $
Number of requests per month (million) 10 Million	3,20 $
Region europe-west4 (Netherlands) - Tier 1	N/A
Resource Type Service	N/A
Billing Charged only when processing requests. CPU is limited outside of requests.	N/A
Execution time per request (ms) 500 ms	N/A
Number of concurrent request per instance 80	N/A
Number of Gpus 1	N/A
Committed use discounts None	N/A

DATA ANALYTICS 5,86 $

Pub/Sub	5,86 $
Service type Pub/Sub	
Amount of published data daily 1 GiB	N/A
Message delivery type Basic	0,80 $
Number of subscriptions 3	3,17 $
Topic retention days 7	1,89 $

DATABASES 12,73 $

Firestore	12,73 $
Service type Firestore	
Document Reads (per day) 100000	0,46 $
Document Writes (per day) 200000	4,93 $

Document Deletes (per day) 10	0,00 $
Total Stored Data (per day) 50 GiB	7,35 $
Location type Region	N/A
Location Iowa (us-central1)	N/A

STORAGE 0,00 $

Cloud Storage	0,00 $
Service type Cloud Storage	
Data Transfer within Google Cloud 1 GiB	0,00 $
Total amount of storage 1 GiB	0,00 $
Location type Region	N/A
Location Iowa (us-central1)	N/A
Storage class Standard Storage	N/A
Source region North America	N/A
Destination region Europe	N/A

Total estimated cost 160,14 $ / mo

The estimated fees provided by Google Cloud Pricing Calculator are for discussion purposes only and are not binding on either you or Google. Your actual fees may be higher or lower than the estimate.

Worst Case Estimate

This estimate is very pessimistic, as it assumes 24/7 runtime of the full allocated cluster resources (4 Nodes, 12 vCPUs, 32GB RAM).

Total estimated cost

1.314,36 $ / mo

Download .csv

Duplicate estimate

Next steps

Request a custom quote

Connect with our sales team and learn about any eligible discounts.

Contact sales

Start your proof of concept

Sign up for Google Cloud, new customers get $300 in free credits.

Get started

Google Cloud

Cost Estimate Summary

As of 20.01.2025 • 18:12

Prices in USD

9,95 $

Artifact Registry	9,95 $
Service type Artifact Registry	
Storage 100 GiB	9,95 $

COMPUTE	**1.285,82 $**
GKE Worst Case (Kubernetes Engine)	1.281,04 $
Service type GKE	
Number of Regional clusters 1	73,00 $
Boot disk type Balanced persistent disk	N/A
Boot disk size (GiB) 20 GiB	8,80 $
Node-time 2920 Hours	N/A
Machine type custom, vCPUs: 12, RAM: 32 GB	1.199,24 $
Number of Nodes 4	N/A
Operating System / Software Free: Container-optimized	N/A
Kubernetes Edition GKE Standard Edition	N/A
Provisioning model Regular	N/A
Enable Confidential GKE Nodes false	N/A
Add GPUs false	N/A
Region Netherlands (europe-west4)	N/A
Committed use discount options None	N/A
Cloud Run	4,78 $

Service type Cloud Run	
CPU amount per instance 1	1,50 $
Memory amount per instance 0.5 GiB	0,08 $
Number of requests per month (million) 10 Million	3,20 $
Region europe-west4 (Netherlands) - Tier 1	N/A
Resource Type Service	N/A
Billing Charged only when processing requests. CPU is limited outside of requests.	N/A
Execution time per request (ms) 500 ms	N/A
Number of concurrent request per instance 80	N/A
Number of Gpus 1	N/A
Committed use discounts None	N/A

DATA ANALYTICS 5,86 $

Pub/Sub 5,86 $

Service type Pub/Sub	
Amount of published data daily 1 GiB	N/A
Message delivery type Basic	0,80 $
Number of subscriptions 3	3,17 $
Topic retention days 7	1,89 $

DATABASES 12,73 $

Firestore 12,73 $

Service type Firestore	
Document Reads (per day) 100000	0,46 $
Document Writes (per day) 200000	4,93 $

Document Deletes (per day) 10	0,00 $
Total Stored Data (per day) 50 GiB	7,35 $
Location type Region	N/A
Location Iowa (us-central1)	N/A

STORAGE 0,00 $

Cloud Storage	0,00 $
Service type Cloud Storage	
Data Transfer within Google Cloud 1 GiB	0,00 $
Total amount of storage 1 GiB	0,00 $
Location type Region	N/A
Location Iowa (us-central1)	N/A
Storage class Standard Storage	N/A
Source region North America	N/A
Destination region Europe	N/A

Total estimated cost 1.314,36 $ / mo

The estimated fees provided by Google Cloud Pricing Calculator are for discussion purposes only and are not binding on either you or Google. Your actual fees may be higher or lower than the estimate.