Uptimatum

The Ultimate Self-Hosted Status Page Platform
Complete Documentation

Uptimatum Project

November 16, 2025

Abstract

Uptimatum is a complete Kubernetes-based microservices application for monitoring endpoint uptime with real-time status pages, embeddable badges, and beautiful dashboards. This document provides comprehensive documentation covering architecture, deployment, development, and usage of the platform.

Contents

List of Figures

1 Introduction

Uptimatum is a self-hosted status page platform designed to monitor endpoint uptime with real-time status updates. The system is built as a microservices application running on Kubernetes, featuring a modern web frontend, RESTful API backend, and a highly available PostgreSQL database cluster.

1.1 Key Features

- Multi-endpoint monitoring with real-time status updates
- 24-hour uptime percentage calculation
- Response time metrics tracking
- Status timeline visualization
- Incident management for service outages
- Beautiful, customizable status pages
- Public-facing status pages
- Embeddable widgets and SVG badges
- Dark mode support
- Mobile responsive design
- Optimized database writes (only inserts when status changes)

1.2 Technology Stack

- **Frontend**: SolidJS, TypeScript, TailwindCSS
- **Backend**: Hono, Bun, Drizzle ORM
- **Database**: PostgreSQL HA (Bitnami Helm chart with StatefulSets)
- **Infrastructure**: Google Kubernetes Engine (GKE)
- **Container Registry**: Google Artifact Registry

2 Architecture

2.1 System Architecture

The Uptimatum system follows a microservices architecture deployed on Google Kubernetes Engine. The architecture consists of multiple layers:

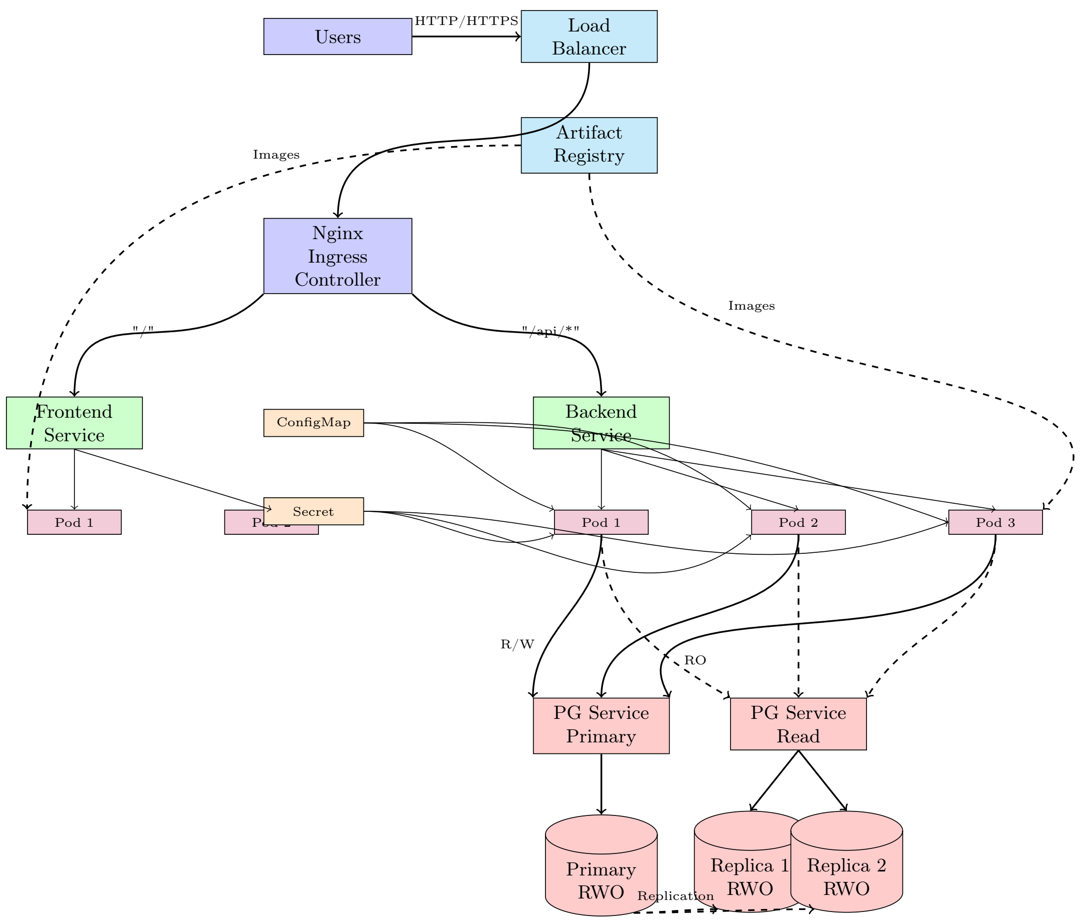


Figure 1: System Architecture Overview

2.2 Backend Project Technologies

Figure 2: Backend Project Technology Stack

The backend is built using:

- **Hono**: Fast web framework with OpenAPI support

- **Bun**: High-performance JavaScript runtime
- **Drizzle ORM**: Type-safe SQL ORM
- **Zod**: Schema validation
- **Pino**: Structured logging
- **node-cron**: Scheduled health checks

2.3 Frontend Project Technologies

Figure 3: Frontend Project Technology Stack

The frontend is built using:

- **SolidJS**: Reactive UI framework
- **TypeScript**: Type-safe development
- **TailwindCSS**: Utility-first CSS framework
- **Solid Router**: Client-side routing
- **Vite**: Build tool and dev server

2.4 Database Schema

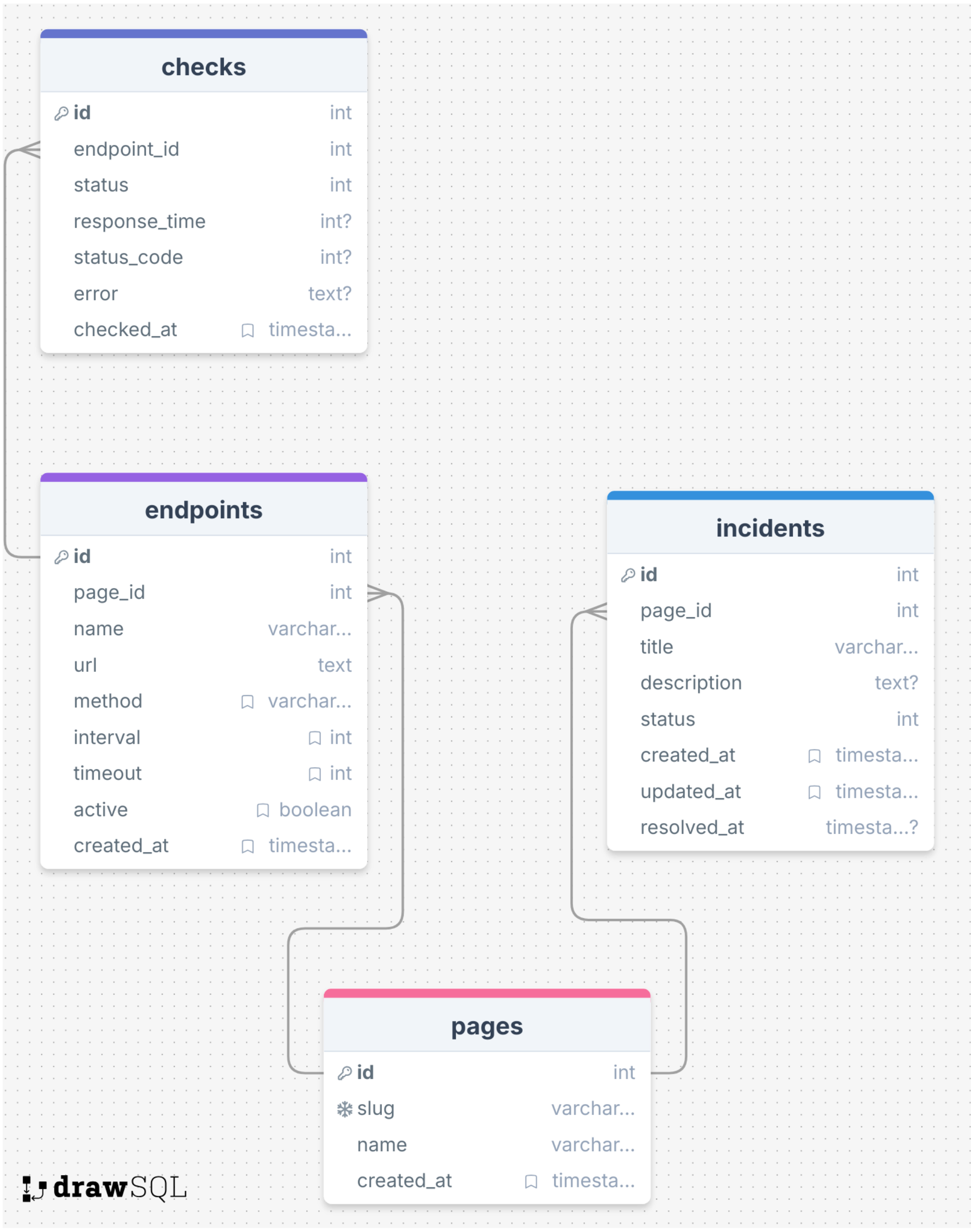


Figure 4: Database Entity Relationship Diagram

The database consists of four main tables:

pages Stores status page definitions with unique slugs

endpoints Monitored endpoints associated with pages

checks Health check results with status, response time, and timestamps

incidents Service incidents and outage tracking

2.5 Data Flow

The system follows a request-response pattern with background health checking:

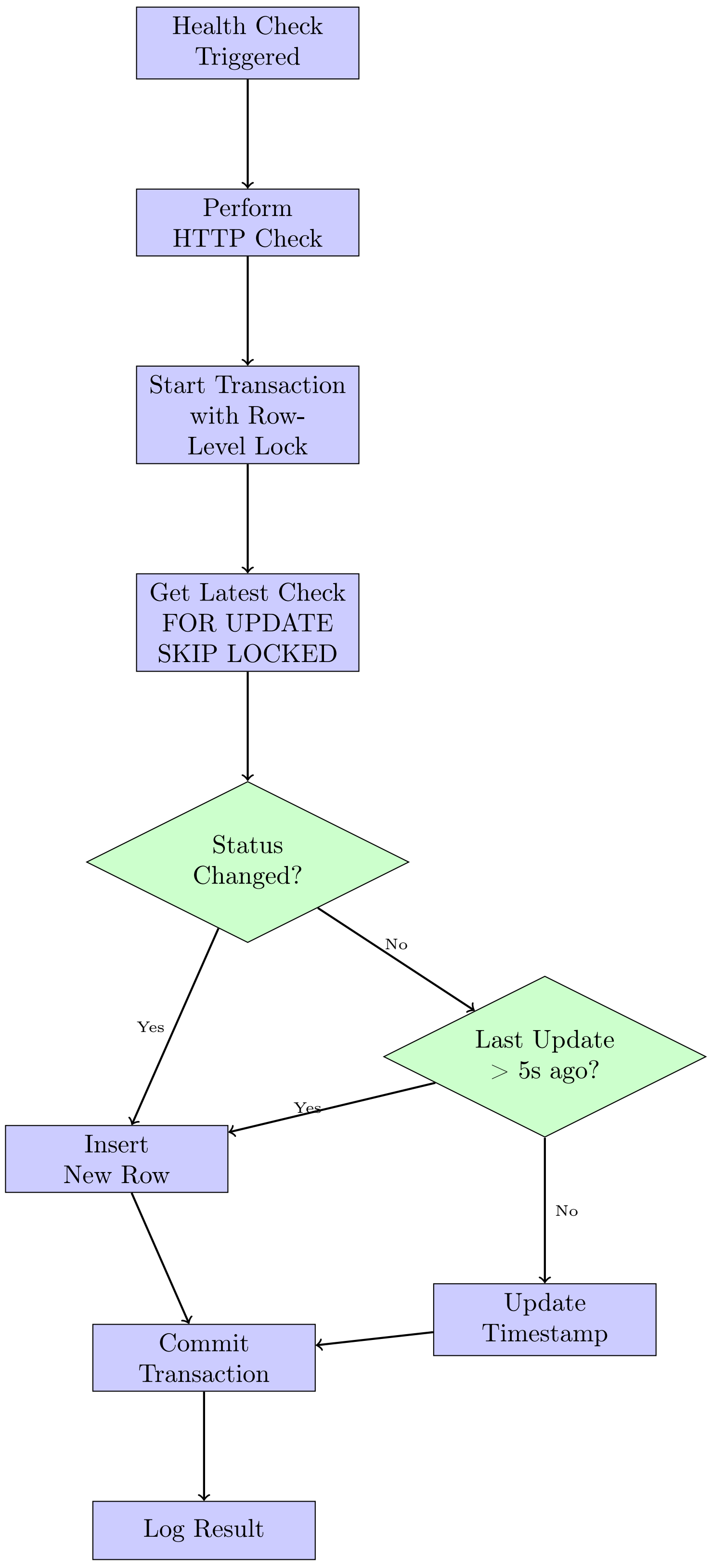


Figure 5: Health Check Database Write Optimization Flow

3 Deployment

3.1 Prerequisites

Before deploying Uptimatum, ensure you have:

1. Google Cloud Account with billing enabled
2. `gcloud` CLI installed and configured
3. `kubectl` installed
4. `helm` installed
5. `bun` installed (or use nvm + bun)

3.2 Quick Start (Automated)

The fastest way to deploy Uptimatum is using the master setup script:

```
./scripts/setup.sh
```

This script performs the following steps:

1. Enables required GCP APIs
2. Creates Artifact Registry
3. Creates GKE cluster with autoscaling
4. Installs Nginx Ingress Controller
5. Sets up PostgreSQL HA cluster (3 nodes)
6. Builds and deploys application

Time: Approximately 15-20 minutes

3.3 Step-by-Step Manual Setup

3.3.1 Step 1: Configure GCP Project

```
export PROJECT_ID=your-project-id
gcloud config set project $PROJECT_ID
gcloud config get-value project
```

3.3.2 Step 2: Enable Required APIs

```
gcloud services enable container.googleapis.com
gcloud services enable artifactregistry.googleapis.com
```

3.3.3 Step 3: Create Artifact Registry

```
export REGION=europe-west1
export REPO=uptimatum

gcloud artifacts repositories create $REPO \
  --repository-format=docker \
  --location=$REGION \
  --description="Uptimatum␣Docker␣images"

gcloud auth configure-docker ${REGION}-docker.pkg.dev
```

3.3.4 Step 4: Create GKE Cluster

```
export ZONE=${REGION}-b

gcloud container clusters create uptimatum-cluster \
  --zone=$ZONE \
  --num-nodes=3 \
  --machine-type=e2-standard-2 \
  --enable-autoscaling \
  --min-nodes=3 \
  --max-nodes=6

gcloud container clusters get-credentials uptimatum-cluster --zone=$ZONE
```

Note: Cluster creation takes 5-10 minutes.

3.3.5 Step 5: Install Nginx Ingress Controller

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-
    nginx/controller-v1.14.0/deploy/static/provider/cloud/deploy.yaml

kubectl wait --namespace ingress-nginx \
  --for=condition=ready pod \
  --selector=app.kubernetes.io/component=controller \
  --timeout=300s
```

3.3.6 Step 6: Setup PostgreSQL Database

```
./scripts/setup-db.sh
```

This script:

- Adds the Bitnami Helm repository
- Installs/upgrades PostgreSQL using Bitnami Helm chart
- Creates the `uptimatum` namespace
- Deploys PostgreSQL cluster with 1 primary + 2 read replicas using StatefulSets
- Waits for all pods to become ready

Primary-Replica Architecture:
Bitnami PostgreSQL Helm chart sets up a master-replica architecture using StatefulSets:

- **1 Primary (Master) StatefulSet**: Handles all read-write operations with RWO storage
- **2 Read Replicas StatefulSet**: Read-only nodes that sync from primary via streaming replication with RWO storage
- **Services**:
  - `uptimatum-db-postgresql-primary`: Points to PRIMARY (read-write) - used by backend
  - `uptimatum-db-postgresql-read`: Points to READ REPLICAS (read-only) - for read scaling

Each StatefulSet pod has its own PersistentVolumeClaim for data persistence.

3.3.7 Step 7: Build and Deploy Application

```
./scripts/deploy.sh
```

This will:

- Build backend Docker image
- Build frontend Docker image
- Push images to Artifact Registry
- Deploy to Kubernetes

3.4 Verifying Deployment

3.4.1 Check Pods

```
kubectl get pods -n uptimatum
```

Expected output:

```
NAME                                   READY   STATUS    RESTARTS   AGE
backend-xxx                             1/1     Running   0          2m
frontend-xxx                            1/1     Running   0          2m
uptimatum-db-postgresql-primary-0       1/1     Running   0          5m
uptimatum-db-postgresql-read-0          1/1     Running   0          5m
uptimatum-db-postgresql-read-1          1/1     Running   0          5m
```

3.4.2 Check Services

```
kubectl get svc -n uptimatum
```

Expected services for database:

- `uptimatum-db-postgresql-primary`: Read-write service (points to primary StatefulSet)
- `uptimatum-db-postgresql-read`: Read-only service (points to read replica StatefulSet)

3.4.3 Verify Primary-Replica Setup

```
# Check StatefulSets
kubectl get statefulset -n uptimatum

# Check which pods belong to primary vs read replicas
kubectl get pods -n uptimatum -l app.kubernetes.io/name=postgresql -o
    wide

# Check PersistentVolumeClaims (one per StatefulSet pod)
kubectl get pvc -n uptimatum
```

You should see:

- 1 StatefulSet for primary (1 pod)
- 1 StatefulSet for read replicas (2 pods)
- 3 PersistentVolumeClaims (one for each pod)

3.4.4 Access Application

Get the external IP:

```
kubectl get ingress uptimatum-ingress -n uptimatum -o jsonpath='{.status
    .loadBalancer.ingress[0].ip}'
```

Once you have the external IP:

- **Dashboard**: `http://<EXTERNAL_IP>`
- **Status Page**: `http://<EXTERNAL_IP>/status/demo`
- **API Docs**: `http://<EXTERNAL_IP>/doc`
- **Interactive Docs**: `http://<EXTERNAL_IP>/reference`

Note: It may take 2-5 minutes for the IP to be assigned.

4 Backend Documentation

4.1 Overview

The Uptimatum backend is a RESTful API built with Hono framework, providing type-safe endpoints for managing status pages, endpoints, checks, and incidents.

4.2 Features

- Type-safe API with OpenAPI documentation
- Interactive API docs with Scalar
- Structured logging with Pino
- Type-safe environment variables with Zod
- Drizzle ORM with migrations
- PostgreSQL database

- Health checker worker with cron scheduling
- Incident management API
- Timeline API for status history visualization
- Optimized database writes (only inserts when status changes, updates timestamps otherwise)

4.3 Setup

4.3.1 Prerequisites

- Node.js (use nvm with `.nvmrc`)
- Bun (or npm/pnpm/yarn)
- PostgreSQL (local or remote)

4.3.2 Installation

1. Install dependencies:

```
source ~/.nvm/nvm.sh && nvm use
bun install
```

2. Create `.env` file:

```
cp .env.example .env
```

   Edit `.env` with your database credentials:

```
DB_HOST=localhost
DB_PORT=5432
DB_NAME=uptimatum
DB_USER=uptimatum
DB_PASSWORD=your_password
PORT=3000
CHECK_INTERVAL=30
CHECK_TIMEOUT=10
CHECK_RETENTION_DAYS=30
NODE_ENV=development
```

3. Generate and run migrations:

```
bun run db:generate
bun run db:push
```

4. Start the server:

```
bun run dev
```

The API will be available at `http://localhost:3000`

4.4 API Documentation

- **OpenAPI Spec**: `http://localhost:3000/doc`
- **Interactive Docs**: `http://localhost:3000/reference` (Scalar)

4.5 Database Commands

- `bun run db:generate` - Generate migration files from schema
- `bun run db:push` - Push schema changes directly to database
- `bun run db:migrate` - Run pending migrations
- `bun run db:studio` - Open Drizzle Studio (database GUI)

4.6 API Endpoints

4.6.1 Health Check

```
GET /health
```

4.6.2 Pages

```
GET /api/pages                      # List all status pages
GET /api/pages/:slug                # Get page with endpoints and stats
GET /api/pages/:slug/timeline       # Get page timeline data
POST /api/pages                     # Create new status page
PATCH /api/pages/:slug              # Update status page
```

4.6.3 Endpoints

```
GET /api/endpoints/:id/history      # Get check history
POST /api/endpoints                 # Create new endpoint
DELETE /api/endpoints/:id           # Delete endpoint
```

4.6.4 Incidents

```
GET /api/incidents?page_id=:id      # List incidents for a page
POST /api/incidents                  # Create new incident
PATCH /api/incidents/:id            # Update incident
DELETE /api/incidents/:id           # Delete incident
```

4.6.5 Badges

```
GET /badge/:slug                    # Get SVG status badge
```

4.7 Environment Variables

Variable	Description	Default
DB_HOST	PostgreSQL host	-
DB_PORT	PostgreSQL port	5432
DB_NAME	Database name	-
DB_USER	Database user	-
DB_PASSWORD	Database password	-
PORT	Server port	3000

Variable	Description	Default
`CHECK_INTERVAL`	Health check interval (seconds)	`30`
`CHECK_TIMEOUT`	Request timeout (seconds)	`10`
`CHECK_RETENTION_DAYS`	Days to keep check records	`30`
`NODE_ENV`	Environment	`development`

4.8 Health Checker

The health checker worker runs every 30 seconds (configurable via `CHECK_INTERVAL`) and:

1. Fetches all active endpoints
2. Performs HTTP checks
3. Records results in the database using optimized writes:
   - **Inserts new row** only when status changes
   - **Updates timestamp** when status is the same (with 5-second threshold to prevent too frequent updates)
   - Uses **row-level locking** to prevent race conditions from parallel instances
4. Logs status using Pino logger

4.8.1 Database Write Optimization

The health checker implements intelligent write optimization to reduce database load:

- **Reduces database writes** by approximately 95% when status is stable
- **Prevents race conditions** using PostgreSQL row-level locking
- **Maintains accuracy** by updating response times and metadata
- **Handles concurrent instances** safely with `SKIP LOCKED`

Additionally, a cleanup job runs daily at 2 AM to remove check records older than `CHECK_RETENTION_DAYS` (default: 30 days) to prevent unbounded database growth.

5 Frontend Documentation

5.1 Overview

The Uptimatum frontend is a modern single-page application built with SolidJS, providing a responsive interface for viewing and managing status pages.

5.2 Features

- SolidJS - Reactive UI framework
- TypeScript - Type-safe development
- TailwindCSS - Utility-first CSS framework
- Solid Router - Client-side routing
- Real-time Updates - Auto-refresh every 30 seconds
- Responsive Design - Works on all devices
- Dark Mode - Automatic theme switching

5.3 Setup

5.3.1 Prerequisites

- Node.js (use nvm with `.nvmrc`)
- Bun (or npm/pnpm/yarn)

5.3.2 Installation

1. Install dependencies:

```
source ~/.nvm/nvm.sh && nvm use
bun install
```

2. Create `.env` file (optional, defaults work for local development):

```
cp .env.example .env
```

3. Start development server:

```
bun run dev
```

The frontend will be available at `http://localhost:5173`

5.4 Development Scripts

- `bun run dev` - Start development server with hot reload
- `bun run build` - Build for production
- `bun run serve` - Preview production build

5.5 API Proxy

In development, Vite automatically proxies API requests to the backend:

- `/api/*` → Backend API
- `/badge/*` → Badge endpoints
- `/doc` → OpenAPI documentation
- `/reference` → Scalar API reference

The proxy target can be configured via `VITE_API_URL` in `.env`.

6 User Interface Showcase

6.1 Dashboard

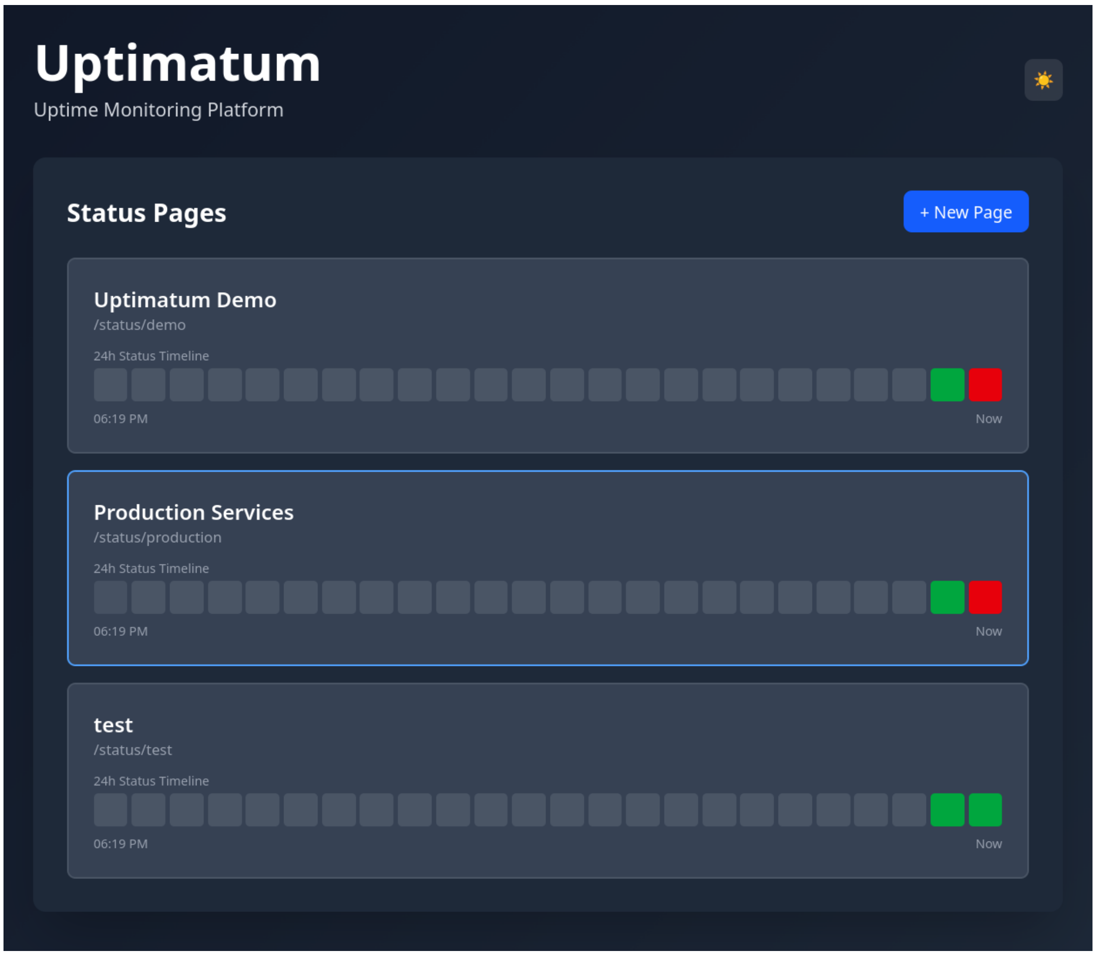
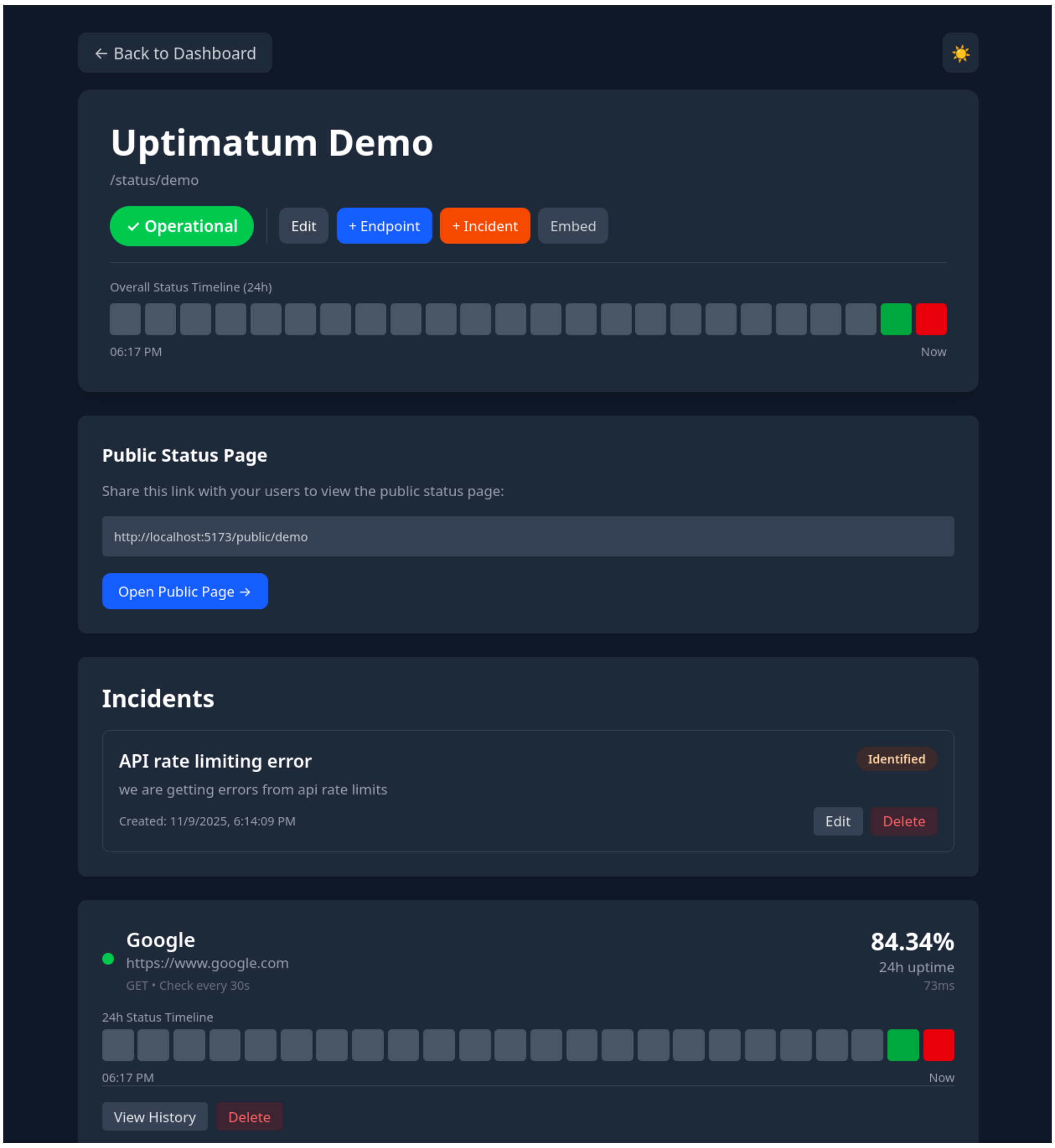


Figure 6: Uptimatum Dashboard

The main dashboard provides an overview of all status pages and quick navigation.

6.2 Status Page Management



Uptimatum Demo

/status/demo

Overall Status Timeline (24h)

06:17 PM — Now

Public Status Page

Share this link with your users to view the public status page:

http://localhost:5173/public/demo

Incidents

API rate limiting error

we are getting errors from api rate limits

Created: 11/9/2025, 6:14:09 PM

Google

https://www.google.com

GET • Check every 30s

84.34%

24h uptime

73ms

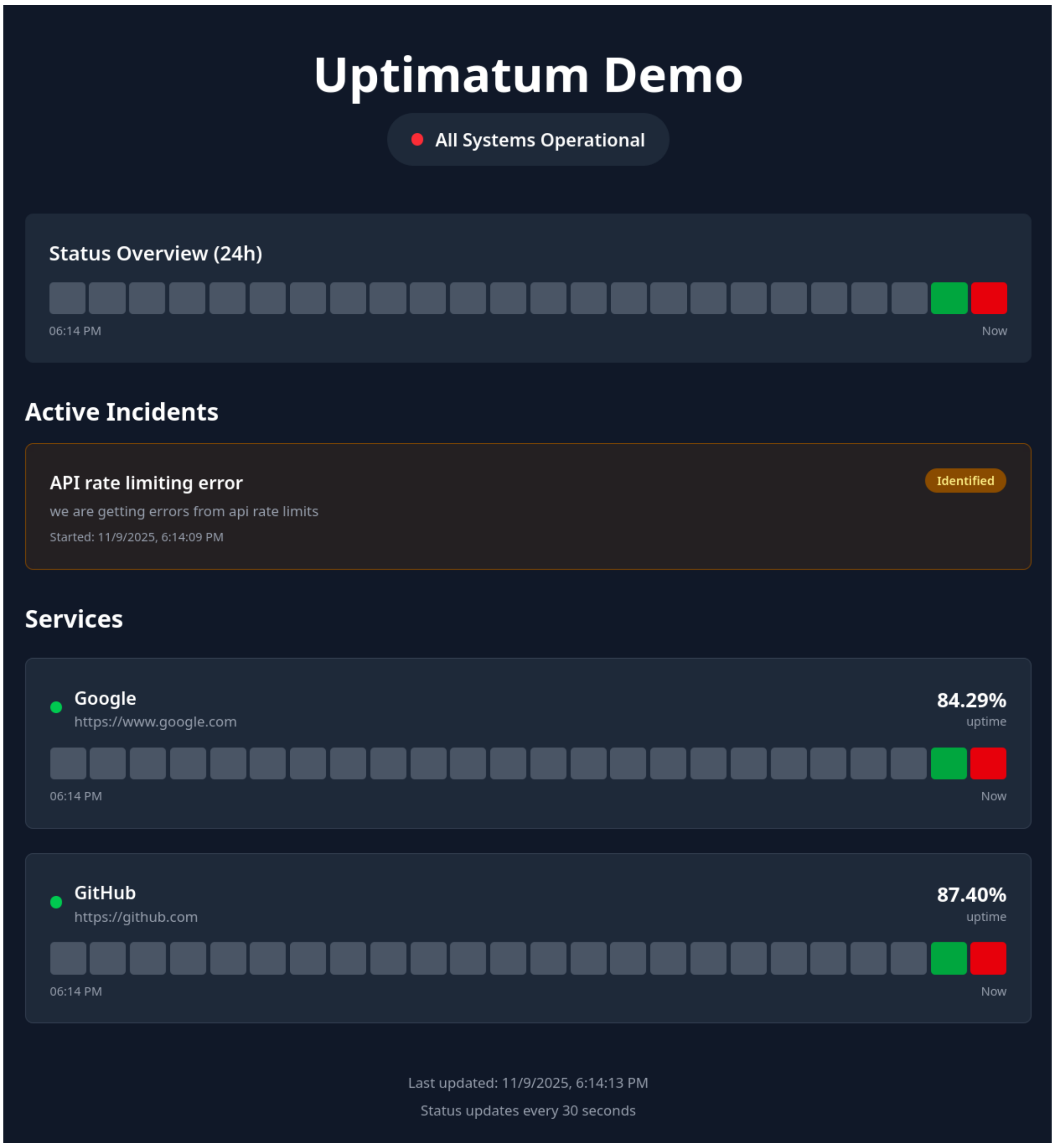
24h Status Timeline

06:17 PM — Now

Figure 7: Status Page Management Interface

The status page management interface allows you to create, edit, and manage status pages with endpoints and incidents.

6.3 Public Status Page


Uptimatum Demo

All Systems Operational

Status Overview (24h)

06:14 PM — Now

Active Incidents

API rate limiting error

Identified

we are getting errors from api rate limits

Started: 11/9/2025, 6:14:09 PM

Services

Google

https://www.google.com

84.29% uptime

06:14 PM — Now

GitHub

https://github.com

87.40% uptime

06:14 PM — Now

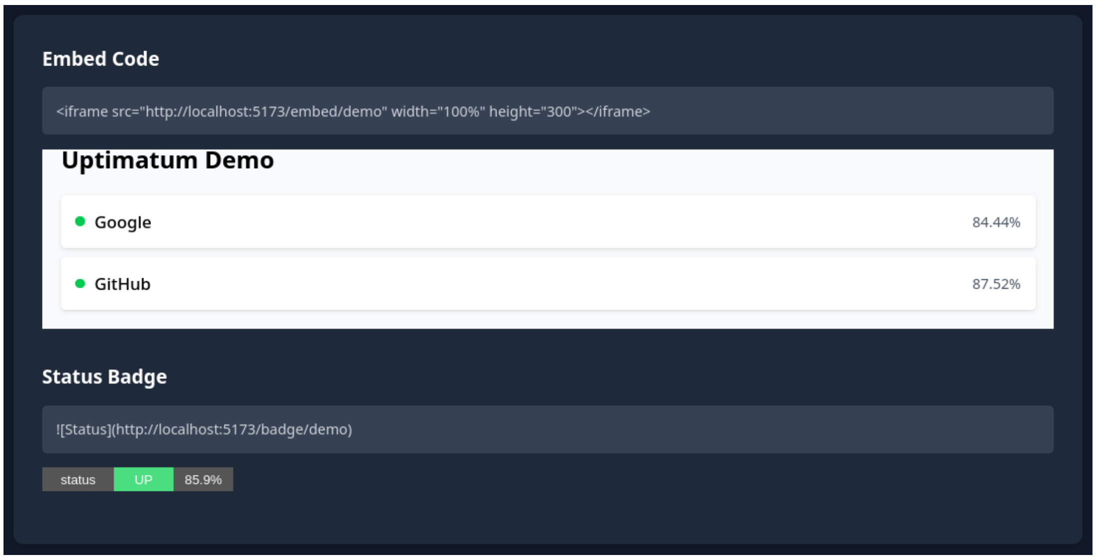
Last updated: 11/9/2025, 6:14:13 PM

Status updates every 30 seconds

Figure 8: Public-Facing Status Page

Public-facing status pages that can be shared with users to show service status.

6.4 Embed Widget


Embed Code

```
<iframe src="http://localhost:5173/embed/demo" width="100%" height="300"></iframe>
```

Uptimatum Demo

● Google	84.44%
● GitHub	87.52%

Status Badge

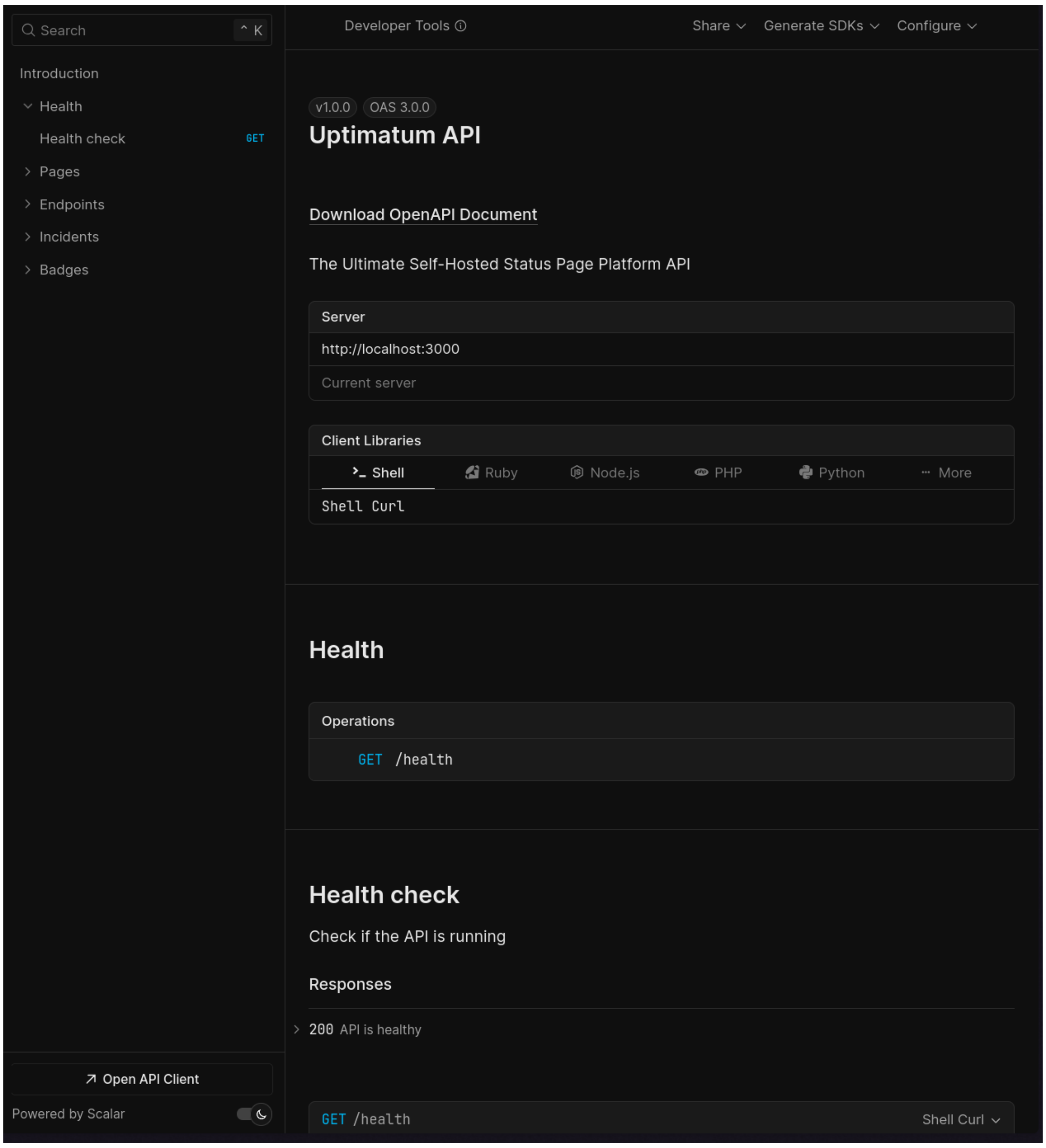
```
![Status](http://localhost:5173/badge/demo)
```

status UP 85.9%

Figure 9: Embeddable Status Widget

Lightweight embeddable widget that can be integrated into any website.

6.5 API Documentation


Search ⌃ K

Introduction

Health

Health check GET

Pages

Endpoints

Incidents

Badges

Open API Client

Powered by Scalar

Developer Tools ⓘ Share Generate SDKs Configure

v1.0.0 OAS 3.0.0

Uptimatum API

Download OpenAPI Document

The Ultimate Self-Hosted Status Page Platform API

Server

http://localhost:3000

Current server

Client Libraries

Shell Ruby Node.js PHP Python More

Shell Curl

Health

Operations

GET /health

Health check

Check if the API is running

Responses

200 API is healthy

GET /health Shell Curl

Figure 10: Interactive API Documentation

Interactive API documentation powered by Scalar, providing a comprehensive reference for all API endpoints.

7 Deployment Scripts

7.1 Master Scripts

7.1.1 setup.sh - Complete Setup

One command to deploy everything from scratch:

```
./scripts/setup.sh
```

Sets up:

- GCP APIs

- Artifact Registry
- GKE Cluster
- Nginx Ingress
- PostgreSQL Database (Bitnami Helm chart with StatefulSets)
- Application Deployment

Time: Approximately 15-20 minutes

7.1.2 cleanup.sh - Complete Cleanup

One command to remove everything:

```
./scripts/cleanup.sh
```

Removes:

- PostgreSQL Helm release
- Kubernetes resources
- GKE Cluster (optional)
- Artifact Registry (optional)

7.2 Individual Scripts

Script	**Purpose**
`setup.sh`	Master script - Complete setup from scratch
`setup-infra.sh`	Setup GCP infrastructure (APIs, Artifact Registry, GKE)
`setup-db.sh`	Deploy Bitnami PostgreSQL cluster with StatefulSets
`deploy.sh`	Build images and deploy to Kubernetes
`cleanup.sh`	Master cleanup - Remove all resources
`demo.sh`	Demo presentation script
`test-local.sh`	Local testing setup
`docker-registry.sh`	Docker registry management script

8 Kubernetes Requirements

The deployment meets the following Kubernetes examination requirements:

- **2 Deployments**: Frontend (2 replicas), Backend (3 replicas) with rolling updates
- **Rolling Updates**: Configured with `maxSurge` and `maxUnavailable`
- **Multiple Services**: `frontend-service`, `backend-service`
- **Ingress**: Nginx Ingress with path-based routing
- **ConfigMap**: Application configuration
- **Secret**: Database credentials

- **PostgreSQL StatefulSet**: PostgreSQL HA cluster (1 primary + 2 read-only replicas) using Bitnami Helm chart
- **StorageClass**: PersistentVolumeClaims for each StatefulSet pod (RWO for primary, RWO for each replica)
- **Unique**: Self-hosted status page platform with embeds

9 Configuration

9.1 Environment Variables

Backend environment variables are set via:

- **ConfigMap**: `k8s/configmap.yaml`
- **Secret**: `k8s/secret.yaml`

9.2 Database Configuration

PostgreSQL cluster is defined in:

- **Helm Values**: `k8s/postgresql-values.yaml`
- **Helm Chart**: Bitnami PostgreSQL (installed via `setup-db.sh`)

9.3 Kubernetes Manifests

All Kubernetes resources are in:

- **Directory**: `k8s/`

10 Troubleshooting

10.1 Port Conflicts

If you get port conflicts, you can change the ports:

- **Backend**: Set `PORT` in `backend/.env`
- **Frontend**: Change `server.port` in `frontend/vite.config.ts`

10.2 Database Connection Issues

1. Verify PostgreSQL is running: `psql -h localhost -U uptimatum -d uptimatum`
2. Check environment variables in `backend/.env`
3. Ensure database exists: `CREATE DATABASE uptimatum;`

10.3 Frontend Can't Connect to Backend

1. Ensure backend is running on the expected port
2. Check `VITE_API_URL` in `frontend/.env` (if set)
3. Verify Vite proxy configuration in `vite.config.ts`

10.4 Kubernetes Deployment Issues

1. Check pod logs: `kubectl logs -n uptimatum deployment/backend`
2. Verify ConfigMap and Secret: `kubectl get configmap,secret -n uptimatum`
3. Check ingress: `kubectl describe ingress -n uptimatum`

11 Upcoming Features

11.1 Infrastructure and Architecture

- **Kafka Message Queue**: Event-driven architecture for better scalability and reliability
  - Event-based database updates to prevent race conditions
  - Decouple services from direct database writes
  - Eliminate concurrent write conflicts (multiple backend instances updating the same row)
  - Simplify better-auth integration with event-driven user management
  - Enable real-time event streaming for status updates
  - Support for event replay and audit trails

Benefits:

- **Race Condition Prevention**: Kafka ensures ordered, sequential processing of database updates
- **Better Scalability**: Services can scale independently without database contention
- **Simplified Architecture**: Event-driven pattern makes authentication and data flow more straightforward
- **Reliability**: Message persistence and replay capabilities improve system resilience

11.2 Authentication and Authorization

- **Better-Auth Integration**: Secure authentication system
  - User registration and login
  - Role-based access control (Admin, Viewer)
  - Protected admin routes for status page management
  - Public access to status pages without authentication
  - Session management and secure token handling
  - Event-driven user management via Kafka (simplifies integration)

11.3 Additional Planned Features

- Email Notifications: Alert administrators when endpoints go down
- Webhook Support: Integrate with Slack, Discord, and other services
- Custom Status Page Themes: Customizable branding and styling
- Advanced Analytics: Detailed uptime reports and historical data
- API Rate Limiting: Protect API endpoints from abuse
- Multi-Language Support: Internationalization for status pages

12 Resource Overview

12.1 What Gets Created

GCP Resources:

- GKE Cluster (`uptimatum-cluster`)
- Artifact Registry (`uptimatum`)
- Load Balancer (via Ingress)

Kubernetes Resources:

- Namespace: `uptimatum`
- Deployments: `backend` (3 replicas), `frontend` (2 replicas)
- Services: `backend`, `frontend`
- Ingress: `uptimatum-ingress`
- PostgreSQL StatefulSets: `uptimatum-db-postgresql-primary` (1 primary) + `uptimatum-db-postgresql` (2 replicas)
- ConfigMap: `uptimatum-config`
- Secrets: `uptimatum-secret` (database credentials managed by Helm)

12.2 Estimated Costs

GKE Cluster:

- 3 nodes × e2-standard-2 = approximately $150-200/month
- Autoscaling: 3-6 nodes based on load

Artifact Registry:

- Storage: approximately $0.10/GB/month
- Network egress: varies

Load Balancer:

- Approximately $18/month (regional)

Total: Approximately $170-220/month (estimate)

13 License

This project is licensed under the MIT License.

14 Contributing

1. Fork the repository
2. Create a feature branch
3. Make your changes
4. Submit a pull request

15 Additional Resources

- GKE Documentation: `https://cloud.google.com/kubernetes-engine/docs`
- Artifact Registry Documentation: `https://cloud.google.com/artifact-registry/docs`
- Kubernetes Documentation: `https://kubernetes.io/docs/`
- Helm Documentation: `https://helm.sh/docs/`
- Drizzle ORM Documentation: `https://orm.drizzle.team/`
- Hono Documentation: `https://hono.dev/`
- SolidJS Documentation: `https://www.solidjs.com/`