UKRAINIAN CATHOLIC UNIVERSITY

BACHELOR THESIS

Development of web application for uptime status monitoring

Author:
Taras MAZURKEVYCH

Supervisor:
Yaroslav SIVACHENKO

A thesis submitted in fulfillment of the requirements
for the degree of Bachelor of Science

in the

Department of Computer Sciences
Faculty of Applied Sciences

APPLIED
SCIENCES
FACULTY

Lviv 2020

Declaration of Authorship

I, Taras MAZURKEVYCH, declare that this thesis titled, "Development of web application for uptime status monitoring" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Bachelor of Science

Development of web application for uptime status monitoring

by Taras MAZURKEVYCH

Abstract

For a software company, it is crucial to provide a reliable service. If something fails, engineers need to know about it as soon as possible to be able to avoid unacceptable system downtime and customers disappointment.

The goal of this work was to design and develop an uptime monitoring system which would include both automated uptime monitoring and a status page. The result of this work became a part of uptime monitoring setup in a software company in which the author works.

We analysed multiple uptime monitoring services and created a specification according to the company needs. It includes HTTP, WebSocket, cron jobs monitoring and web applications monitoring by simulating users actions.

We implemented all the planned components. HTTP monitoring has a high level of false alarms and thus is not a fit for complex system monitoring. Passive checks have no false alarms and are a good choice for monitoring of regular jobs. We are about to start using WebSocket and web applications monitoring.

The result of this work is servicecheck.io.

Acknowledgements

I would like to thank Yaroslav Sivachenko and Oles Dobosevych for supervising my work.

Thank Faculty of Applied Sciences and Ukrainian Catholic University for all these support, knowledge and experience.

Contents

Declaration of Authorship i

Abstract ii

Acknowledgements iii

1 Introduction 1

2 Related Works 2

3 Solution 5
- 3.1 Checks page . . . 6
- 3.2 Incidents page . . . 8
- 3.3 Status page . . . 10
- 3.4 Settings page . . . 12
- 3.5 Future improvements . . . 13

4 Architecture 15
- 4.1 Tech stack . . . 15
- 4.2 Architecture . . . 16
- 4.3 Future improvements . . . 18

5 Conclusions 19

Bibliography 20

List of Figures

2.1 Uptime monitoring services comparison 3

3.1 Checks page 6
3.2 Creating basic active check 7
3.3 Editing passive check 7
3.4 Slack messages on checks fails 8
3.5 Creating an incident 9
3.6 Editing an incident 9
3.7 Resolved incident on a status page 10
3.8 Status page settings page 10
3.9 Status page. All systems operational 11
3.10 Status page. Partial outage 12
3.11 Settings page 13

4.1 Cloud Pub/Sub. *Pub/Sub message flow* 16
4.2 Project architecture 18

List of Abbreviations

GCP	**G**oogle-**C**loud **P**latform
SaaS	**S**oftware-**a**s-**a** **S**ervice

Chapter 1

Introduction

For a software company, it is crucial to provide a reliable service. If something fails, engineers need to know about it as soon as possible to be able to avoid unacceptable system downtime and customers disappointment.

System uptime is a total time when the system is available and operational minus the time when it was unavailable or not operational, measured in percentage. A mathematician would assume that perfect uptime is 100 percent. That assumption would lead to slower development and less improvement. [Betsy Beyer and Murphy, 2016] There should be a space where it is safe to take risks and do experiments. For the system to be updated and be able able to progress, errors are unavoidable. So we need to plan for errors and prepare for them.

Uptime monitoring is a common practice that checks systems availability regularly in an automated way. Uptime status page is usually a publicly available page which displays both automated uptime monitoring results and manually added messages about system downtimes or planned maintenance.

The goal of this work is to design and develop an uptime monitoring system which will include both automated uptime monitoring and a status page. The author started this project in the company in which he works. Then the work on the project continues in this thesis. The result of this work will become a part of uptime monitoring setup in a software company in which the author works.

Chapter 2

Related Works

It all started with an idea to build a software as a service (SaaS) application that will include both automated uptime monitoring and status page.

Previously the company used multiple tools for monitoring, and none of them had a status page feature. The idea was to combine all these tools into a single one and to have full control of it to get new functionality when there is a need for it.

There were no specific requirements for the system. The idea was for to build something similar to statuspage.io, healthchecks.io and uptimerobot.com.

We need to do some definitions for a more precise understanding of uptime monitoring features.

Check is a unit of uptime monitoring. For example, a ping to a single specific URL with specific interval will be called a check.

Monitoring service is a service that provides uptime monitoring for other services.

Monitored service is a service that is monitored by monitoring service.

Active check is a check in which monitoring service sends requests to monitored service to check if it is available and operational.

Passive check is a check in which monitored service sends requests to monitoring service to confirm that the first one is operational.

Implicit fail for passive check is an option for monitored service to notify monitoring service about fail by sending a request to specific fail URL.

Time measuring for passive check is an option for monitored service to notify monitoring service about job execution start and finish.

Alarm is an event which happens on check fail.

Cron is a standard Unix utility that allows to schedule commands for a specific time.

Cron syntax is a special syntax to define the time for cron. *The quick and simple editor for cron schedule expressions by Cronitor*

We started by analysing these services. We divided the analysis into two categories: uptime monitoring and status page.

Core uptime monitoring features were:

- Active checks
- Passive checks
- Alarms on checks fails including email and Slack notifications
- Custom intervals including cron syntax support for intervals
- Implicit fail option for passive checks
- Measuring how much time task took for passive checks

Core status page features were:

- Ability to manually add messages to the status page
- Ability to show automated checks results on the status page
- Ability to customise the status page styles
- Private status pages. For example, password protected
- Showing response time
- Ability for users to subscribe for status page notifications

We also did a broader analysis of competitor services such as Postman, status.io, AdminLabs, Updown, Cachet, Sorry, Pingdom, Statusy, LambStatus, Site24x7, Nodeping, Uptimechecker, Simon App to see what else can we improve in current work.

There were two things we found in this analysis that we want to implement in current work: complex active checks and WebSockets active checks.

Complex active check is a check that consists of multiple actions opposite to basic active check which consists of a single HTTP request.

WebSockets active check is a check that uses WebSockets protocol opposite to basic active check that uses HTTP.

	A	B	C	D	E	F	G
1	Service Name	Website	Supports basic active checks	Supports passive checks	Supports WebSockets checks	Supports complex checks	Ability to create public status page
2	Postman	https://www.postman.com/	✅	❌	❌	✅	✅
3	Statuspage	https://www.statuspage.io/	✅	✅	❌	❌	✅
4	Healthchecks	https://healthchecks.io/	❌	✅	❌	❌	❌
5	UptimeRobot	https://uptimerobot.com/	✅	❌	❌	❌	✅
6	status.io	https://status.io/	❌	❌	❌	❌	✅
7	AdminLabs	https://www.adminlabs.com/	✅	❌	❌	❌	✅
8	Updown	https://updown.io/	✅	❌	❌	❌	✅
9	Cachet	https://cachethq.io/	✅	❌	❌	❌	✅
10	Sorry	https://www.sorryapp.com/	✅	❌	❌	✅	✅
11	Pingdom	https://www.pingdom.com/	✅	❌	❌	✅	✅
12	Statusy	statusy.co	❌	❌	❌	❌	✅
13	LambStatus	https://lambstatus.github.io/	❌	❌	❌	❌	✅
14	Site24x7	https://www.site24x7.com/	✅	✅	✅	❌	✅
15	Nodeping	https://nodeping.com/	✅	❌	✅	❌	✅
16	Uptimechecker	https://www.uptimechecker.io/	✅	❌	✅	❌	❌
17	Simon app	https://www.dejal.com/simon/	✅	❌	❌	❌	❌

FIGURE 2.1: Uptime monitoring services comparison

Services above are using different ways to implement complex active and WebSockets active checks.

For complex active checks, we found these implementations:

Postman allows creating a list of API requests that will run in sequence and writing tests in JavaScript to check the values received.

Pingdom allows creating a list of actions that simulate users behaviour on the website and check the values on the website.

We decided to stick with the second approach because it is closer to the user perspective.

For WebSockets active check, we found these implementations:
Site24x7 checks if it is possible to connect to the specific URL.
Nodeping and Uptimechecker connect to the specific URL, send a predefined message and check if the message received is equal to the predefined message from monitored service.
We decided to implement the first approach and then see if it is enough for company needs.
Then we prioritised and created a list of features to implement:

- Basic active checks
- Passive checks
- Alarms on checks fails including email and Slack notifications
- Custom intervals including cron syntax support for intervals
- Ability to manually add messages to the status page
- Ability to show automated checks results on the status page
- Ability to customise the status page styles

The second priority list is:

- Complex active checks
- WebSockets active checks

Chapter 3

Solution

In this chapter, we will describe the solution and provide an example of the configured system. We will describe the architecture in the next chapter.

Let us start with the user flow. First, the user sees the landing where they can read about the service.

After clicking "Sign up", a user needs to create an organisation. It includes choosing an organisation name, subdomain at which organisation status page will be available, buying a subscription using a credit card.

After the sign up, they get access to the application and can invite other users to the organisation. Sign in and Sign up are using the "Sign in with Google" feature and require the users to have Google account.

A user can only belong to a single organisation.

Below we provide definitions that are crucial for understanding what the solution includes:

Basic active check is a check in which monitoring service sends HTTP requests to monitored service to check if it is available and operational. The check is successful if the monitored service responds within the timeout with response status code 200 (OK). This status code indicates that the request was successful.

Passive check is a check in which monitored service sends requests to monitoring service to confirm that the first one is operational. This type of check has a parameter called safe interval, which determines the maximum interval between requests for check to be successful.

Complex active check is a check that consists of multiple steps of actions. The user can define a list of actions which will simulate the user's behaviour. Examples of actions are go to link, click the element on the page, fill in the field with a value, check the value on the page is equal to the value predefined in check parameters. The check is successful if all value checks match.

WebSockets active check is a check that uses WebSockets protocol. The check is successful if it the connection to monitored service was successful.

Status page is a publicly availably page. Organisations can own multiple status pages.

Alarm is an event which happens on check fail. If the check if linked to status page it will be displayed on the status page. Alarm is saved to the database, and if notifications are turned on for the check, the alarm with details is sent to users' emails and Slack provided.

Incident is an event that can be manually created by the user and attached to a status page.

There are four main pages: checks, incidents, status pages and settings.

3.1 Checks page

The first thing users see on in the app is checks page.

The app highlights the core info in the list:

- Green or red indicator to show if check is on alarm
- Check name
- Last 20 runs and their results. Hovering it will show the exact time of the execution
- The time of last check execution

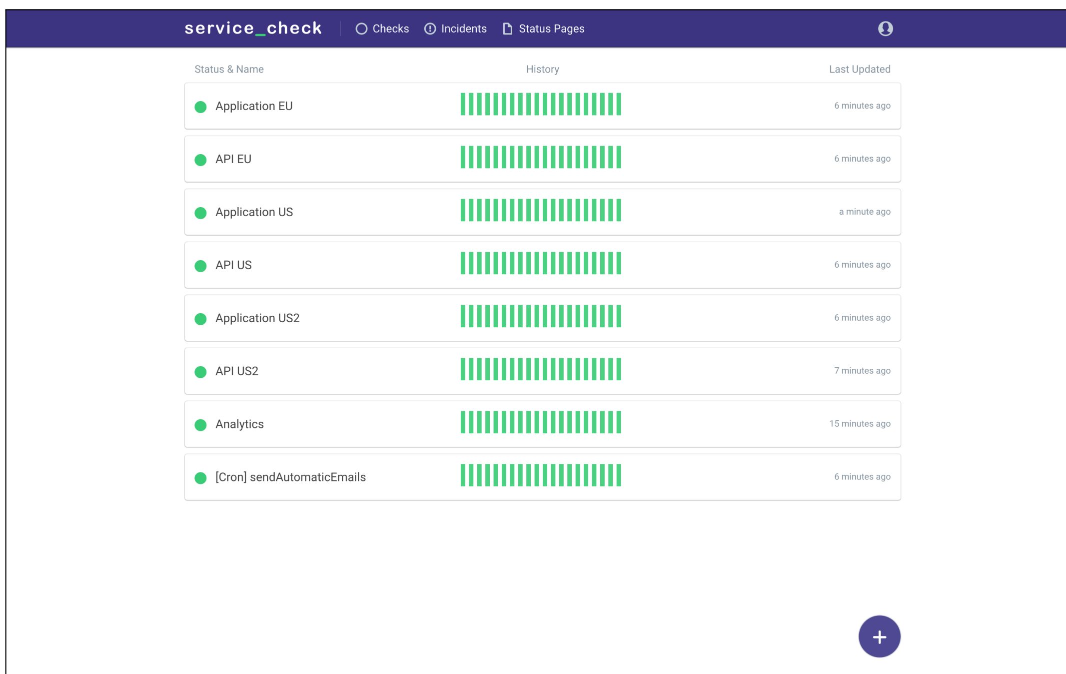

FIGURE 3.1: Checks page

Each type of check require specific params:

Basic active check requires URL and interval defined in cron syntax.

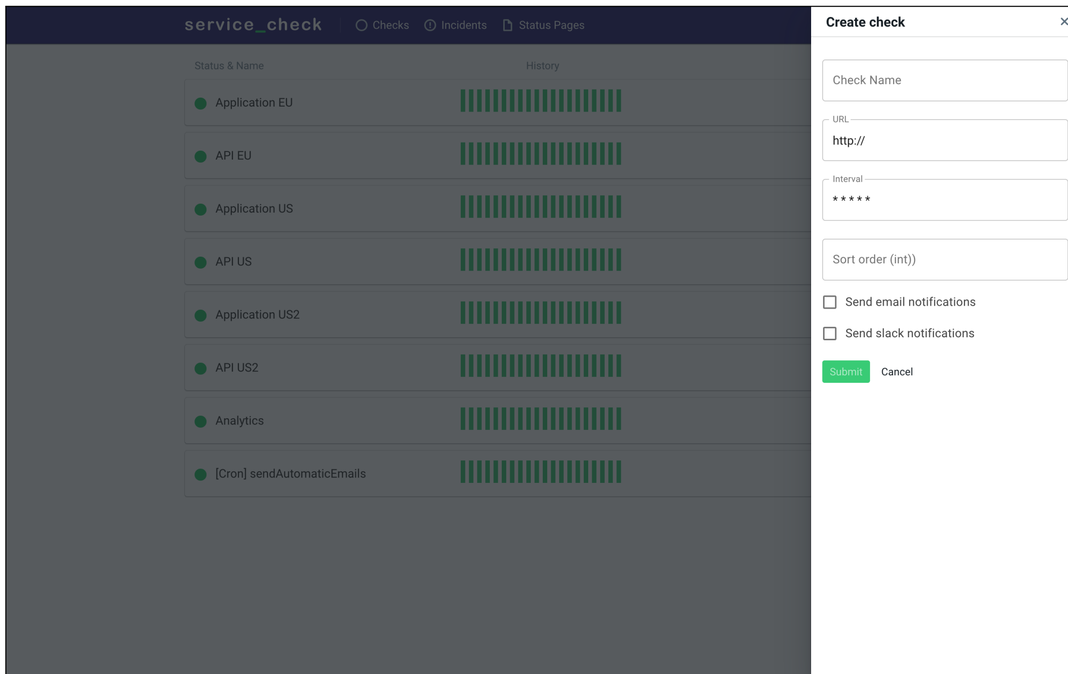

FIGURE 3.2: Creating basic active check

Passive check requires a safe interval number in minutes. After creating it will provide the unique URL of this check to send requests to.

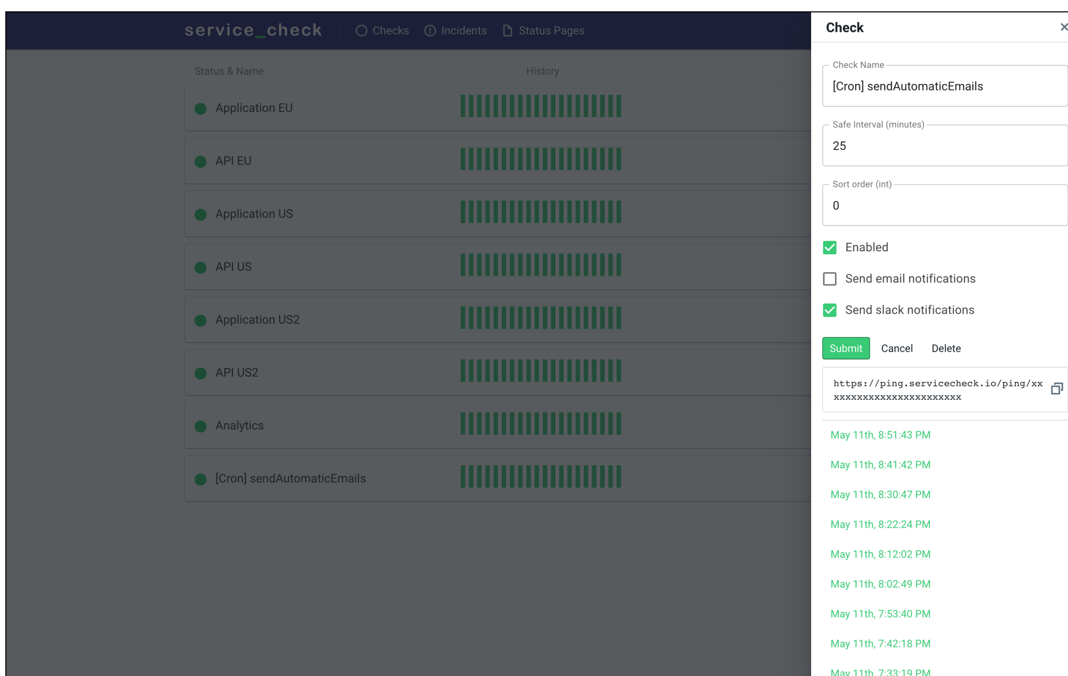

FIGURE 3.3: Editing passive check

Complex active check requires interval defined in cron syntax and a list of commands to run.

WebSockets active check requires URL.

Each check requires a name. Users can sort checks on the page by assigning a sort order to them.

It is important to know about downtimes quickly when they happen. That is why we built support of Slack notifications and email notifications. There is also an option in each check to allow notifications.

Each message includes check name, check type, date, time, URL, fail reason and details.

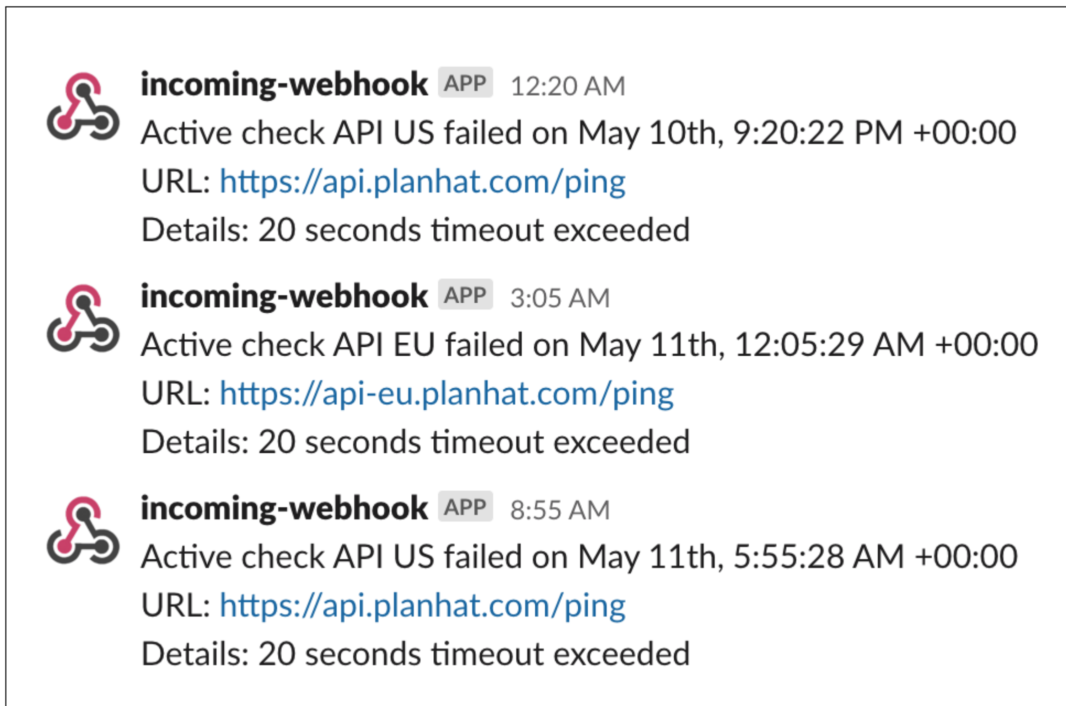

FIGURE 3.4: Slack messages on checks fails

3.2 Incidents page

To put a message on the status page, the users need to create an incident.

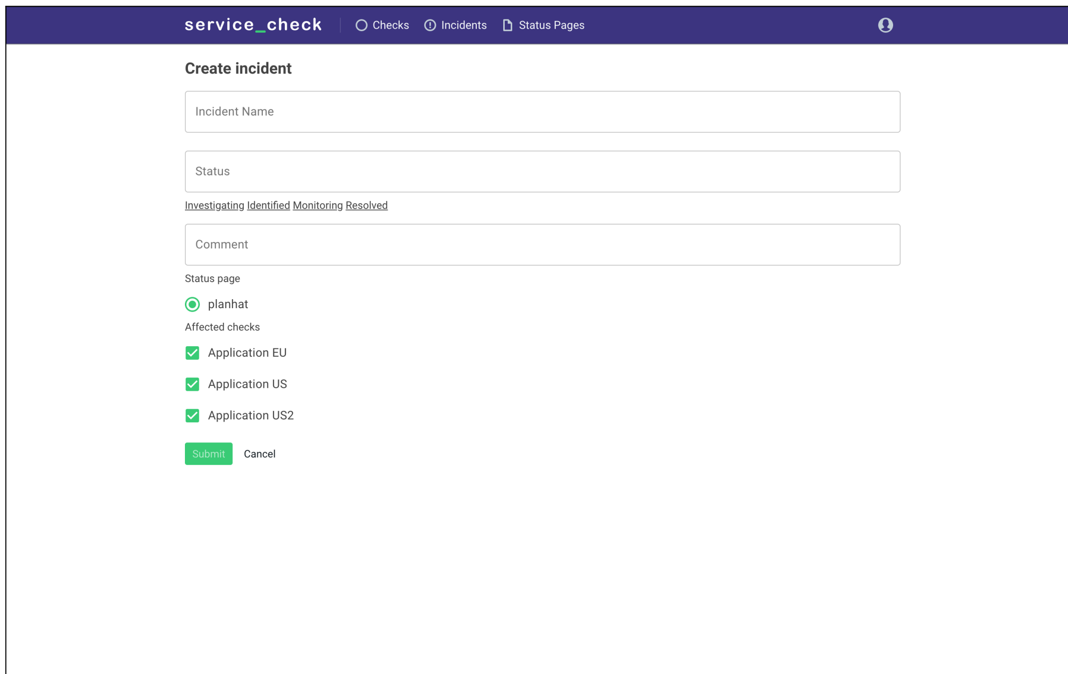

FIGURE 3.5: Creating an incident

Incident has a name, comment, status and list of checks this incident affects. After creating an incident the user can update it with new status and comment.

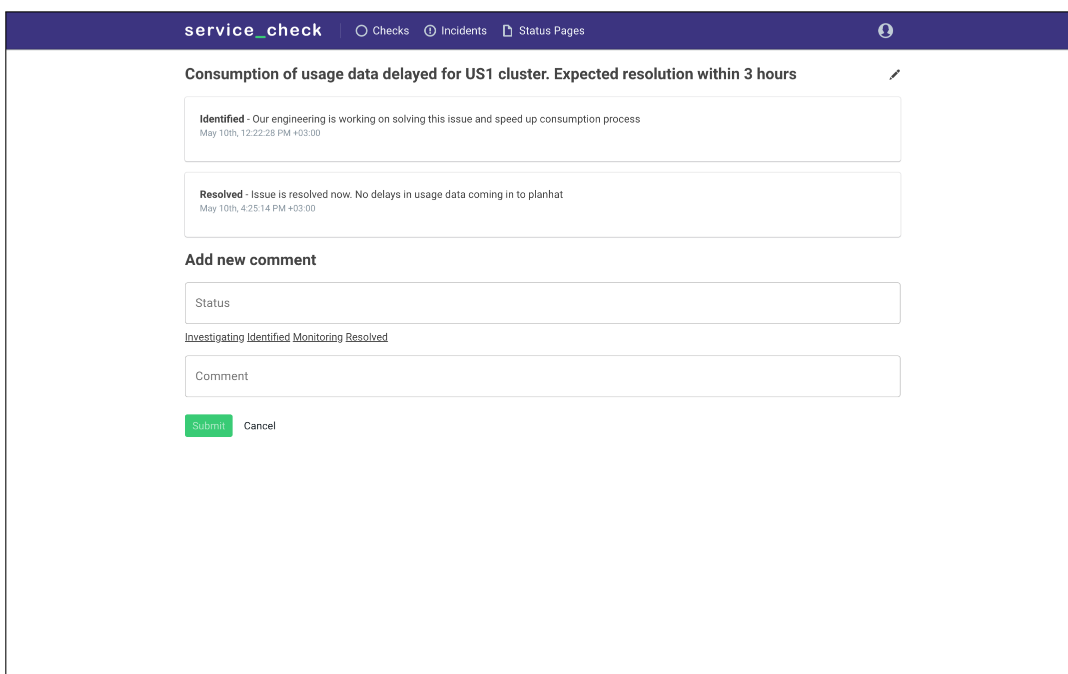

FIGURE 3.6: Editing an incident

Incident is active until its status becomes "Resolved". If incident is active it will be pinned on a status page. After resolving it will stay in history.

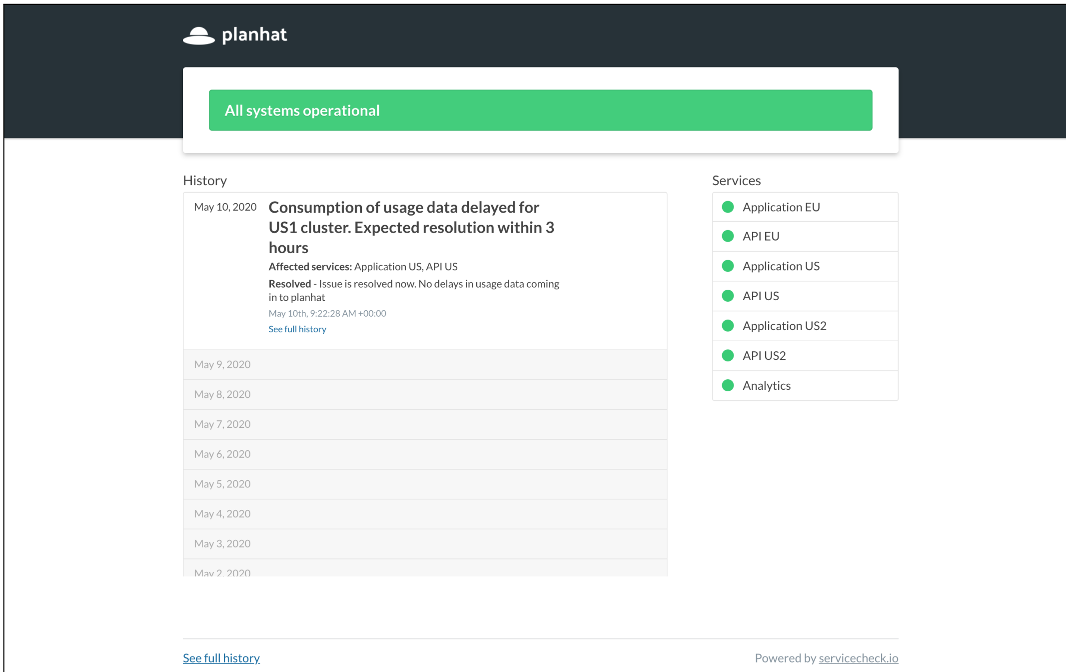

FIGURE 3.7: Resolved incident on a status page

3.3 Status page

Organisation can own multiple status pages. Each of them can be customised. Users can customise header background image and colour, page logo, checks to show on the page, page title and labels. There can be multiple status pages. In such a case, the user can choose which of them will be the main one. The main one will be available at `[organization_subdomain].servicecheck.io`. The other ones will be available at `[organization_subdomain_name].servicecheck.io/[statuspage_name]`.

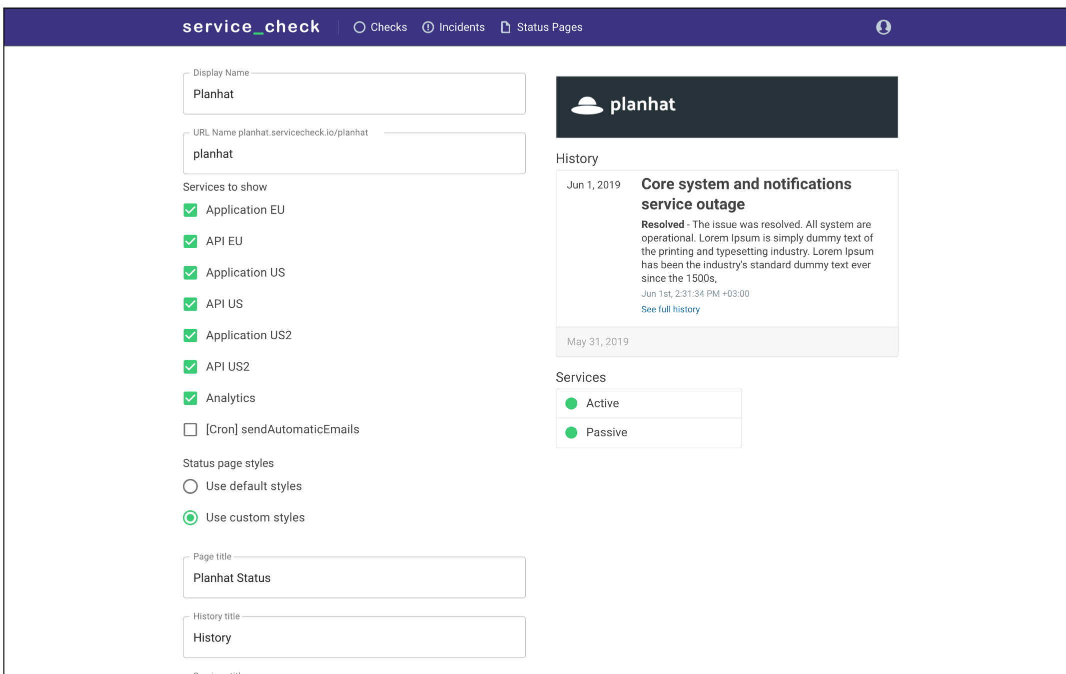

FIGURE 3.8: Status page settings page

The status page consists of summary, history and the list attached of checks. Incidents attached to the status page will appear in history. We recommend attaching up to 20 checks to a status page. By default, the summary says "All systems operational" and its colour is green. If some of the attached checks fail the status will become "Partial outage" and the colour will become orange. In such a case, the failed check will also have an orange status indicator. If there is an active incident it will be pinned on the page.

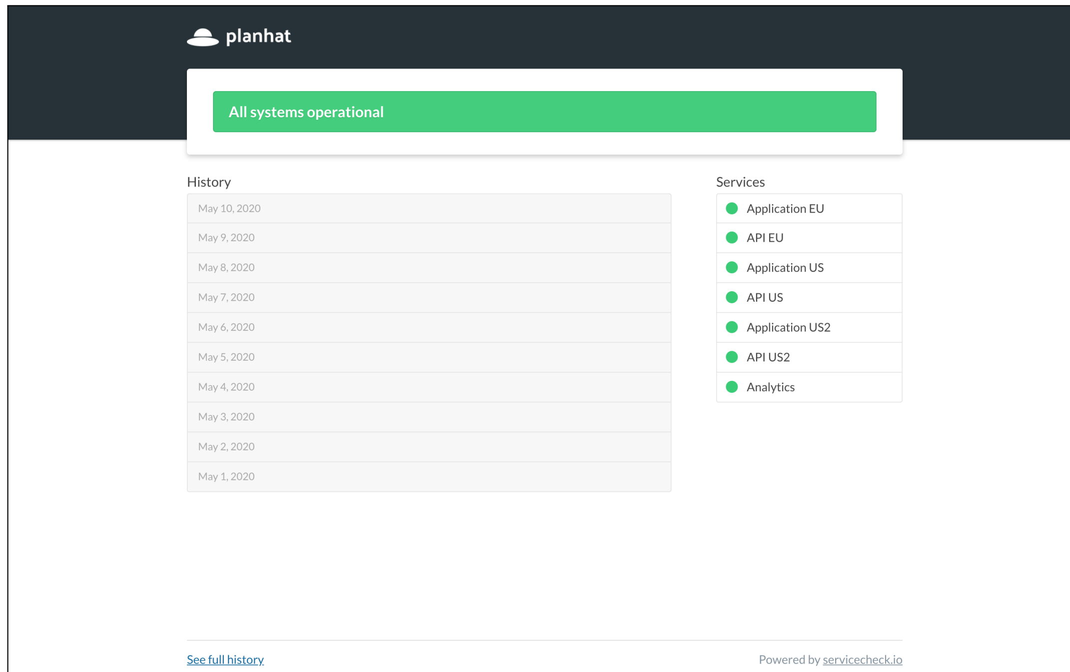

FIGURE 3.9: Status page. All systems operational

The page will refresh the data on the status page every minute.

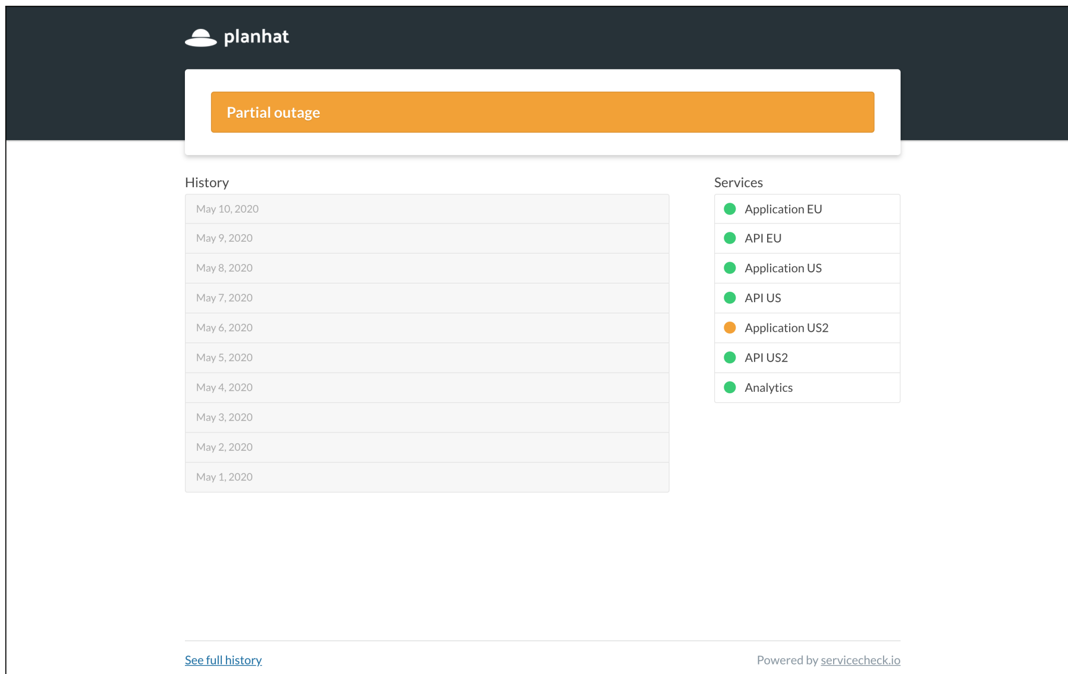

FIGURE 3.10: Status page. Partial outage

3.4 Settings page

On settings page users can:

- Choose the main status page
- Set Slack webhook to receive Slack notifications on alarms
- See organisation users, change their roles, remove them from the organisation
- Invite new users to the organisation
- Manage billing

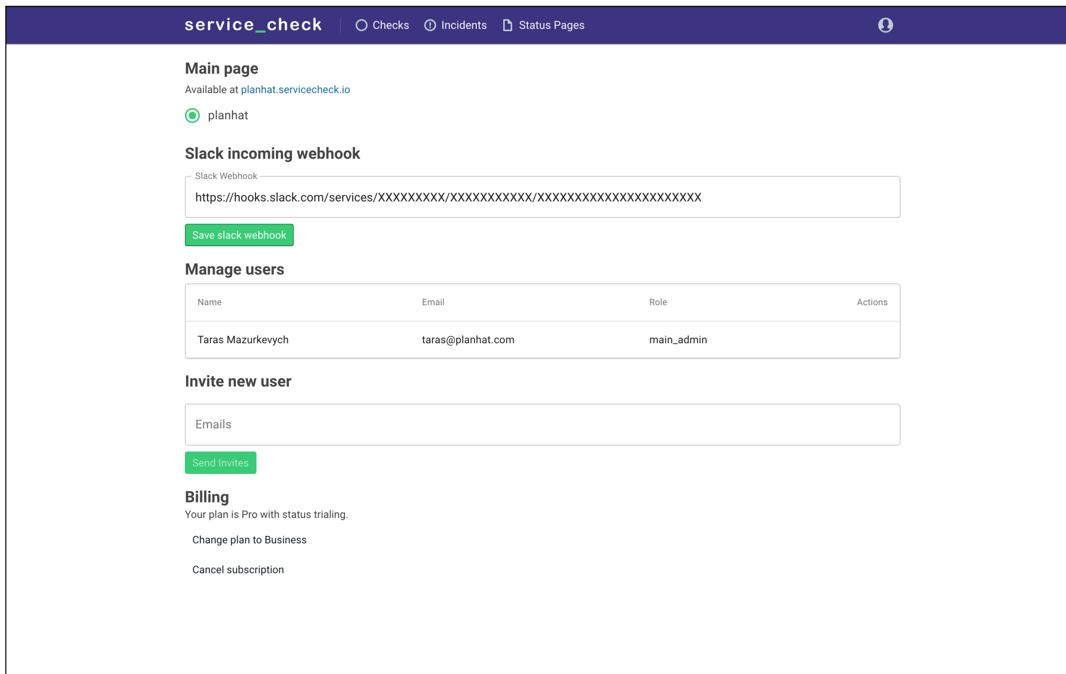

FIGURE 3.11: Settings page

To set up a Slack webhook, the user should go to their Slack workspace and open Incoming webhooks application. Then the user should choose "Add to Slack" option and select a target channel for the webhook. After the user has created the incoming webhook, they should save the incoming webhook url on the application settings page.

There are three roles in the application: `main_admin` , admin, user. The user who created the organisation has a role `main_admin` . They can invite others users and assign the role admin or user to them. The difference between admin and user is that user can not manage settings. The difference between `main_admin` and admin is that the admins can not remove a user with `main_admin` role from the organisation.

If a user invites the new user to the organisation, the second one gets the invite to their email. The application considers email a unique identifier. Currently, the user can be linked to only a single organisation. User should click the invite link from the email and then sign in with their Google account. At that moment, the account is activated, and the user gets access to the application.

The user can choose a billing plan while creating the organisation. If the user wants to change the billing plan or cancel the subscription they can do it on the settings page.

3.5 Future improvements

In this section, we will describe possible future improvements in order of priority.

- Implicit fail option for passive checks
- Measuring how much time task took for passive checks
- Ability to subscribe to public status page updates
- Showing historical response time on the status page

- Run Active Checker in multiple locations for more precise uptime monitoring

Chapter 4

Architecture

4.1 Tech stack

The tech stack includes NodeJS, PostgreSQL, ReactJS.

NodeJS is a JavaScript runtime built on Chrome's V8 JavaScript engine.

PostgreSQL is an object-relational database.

ReactJS is a JavaScript library for building user interfaces.

The project is running on Google Cloud Platform (GCP). It was a company requirement to use GCP. The goal was to use fully managed solutions, so we do not have to spend time on DevOps actions like setting up servers.

There are the GCP services the project uses:

- App Engine
- Cloud Pub/Sub
- Cloud Functions
- Cloud Storage
- Cloud SQL

App Engine is a fully managed serverless application platform. It allows deploying applications without any infrastructure management. It also provides applications scaling out of the box. There are two types of environment in App Engine: flexible and standard. Flexible environment applications run within Docker containers on virtual machines. Standard environment applications run in a sandbox.

Cloud Pub/Sub is a real-time messaging service. Cloud Pub/Sub has four key concepts:

- Topic is a place to which the messages are sent by publishers
- Subscription defines a stream that receives messages from specific topic
- Message is a unit of data sent
- Message attribute is key-value pair that publisher can define for message

The Pub/Sub flow actors are Publisher and Subscriber. After receiving a message to prevent from receiving it again Subscriber acknowledges that they received a message.

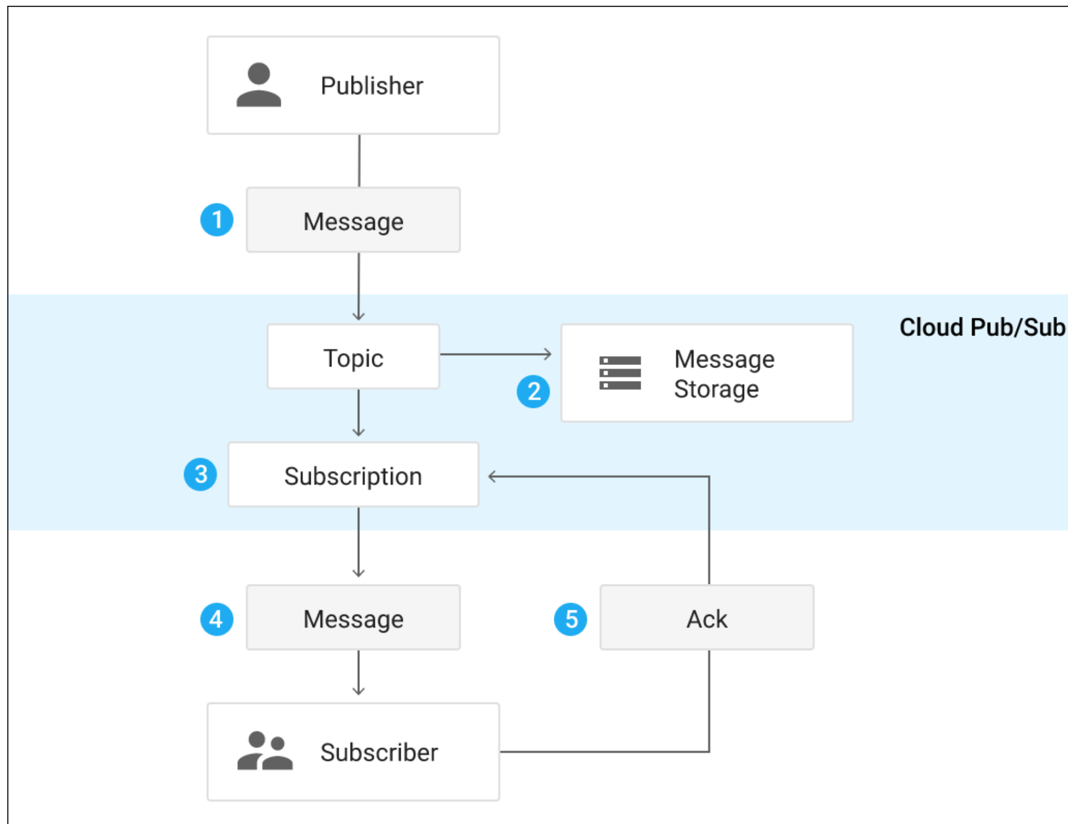

FIGURE 4.1: Cloud Pub/Sub. *Pub/Sub message flow*

Cloud Functions is an event-driven serverless compute platform. Cloud function is a single piece of code that has a single purpose. It can be executed multiple times and concurrently. It is possible for a function to subscribe to Cloud Pub/Sub topic and execute each time a message is published into the topic.

App Cloud Storage is online file storage.

App SQL is a fully managed database service.

4.2 Architecture

There are three NodeJS App Engine services running: Web App, Active Checker, Passive Checker. All the requests from frontend communicate only with Web App.

Web App also sends emails to users and messages to Slack. To send emails to users, the application uses Mailgun. Mailgun is a transactional email service available via API. To send slack messages, the app uses Slack Incoming webhooks application. Web App sends a POST method request to the provided on the settings page URL.

Web App is running on App Engine flexible environment. Active Checker and Passive Checker are running on App Engine standard environment. This setup helps to reduce costs and minimise deployment time by using standard environment and get more control over the service by using flexible environment.

Active Checker runs Cloud functions which run checks. This architecture allows running more checks than it would be possible to run on the App Engine service. Then it receives the results of the checks. If there are failed ones, it creates alarm in a database and sends alarm to Web App to process it.

To get a more detailed understanding of how Active Checker work, we created a list of steps:

1. Cron job runs every minute
2. Active Checker loads all the checks that will run during this minute
3. Active Checker pushes them to Cloud Pub/Sub topics for running active checks
4. Cloud functions trigger on new Cloud Pub/Sub messages and run the checks
5. Cloud functions push checks results to Cloud Pub/Sub active checks result topic
6. Active Checker receives new messages from Cloud Pub/Sub active checks result topic
7. Active Checker saves checks results to the database
8. If there are failed checks Active Checker creates alarms in the database and sends an alarm to Web App via Cloud Pub/Sub
9. If Web App receives an alarm and check notifications are enabled Web App sends emails and Slack messages to notify users about failed checks

Passive Checker receives HTTP requests from monitored services and updates the checks in the database. Passive Checker provides unique URLs for each passive check. Every minute the cron runs to find the checks which exceded the safe interval.

There are three cloud functions we built for Active Checker: basic active, complex active, websockets active.

Basic active check sends HTTP request to defined URL and expects to get 200 (OK) response status code in less than 20 seconds timeout.

Complex active check uses Headless Chrome *Getting Started with Headless Chrome* to open the website and simulate user actions defined in its params. There are also validations in check params. Check is successful if all validations on the website passed. Headless Chrome is a tool that allows running Chrome browser from the command line or in code. It was built for automated testing, server environments and use cases where the user does not need visible UI.

WebSockets active check use WebSocket protocol *How Do Websockets Work?* to connect to monitored service.

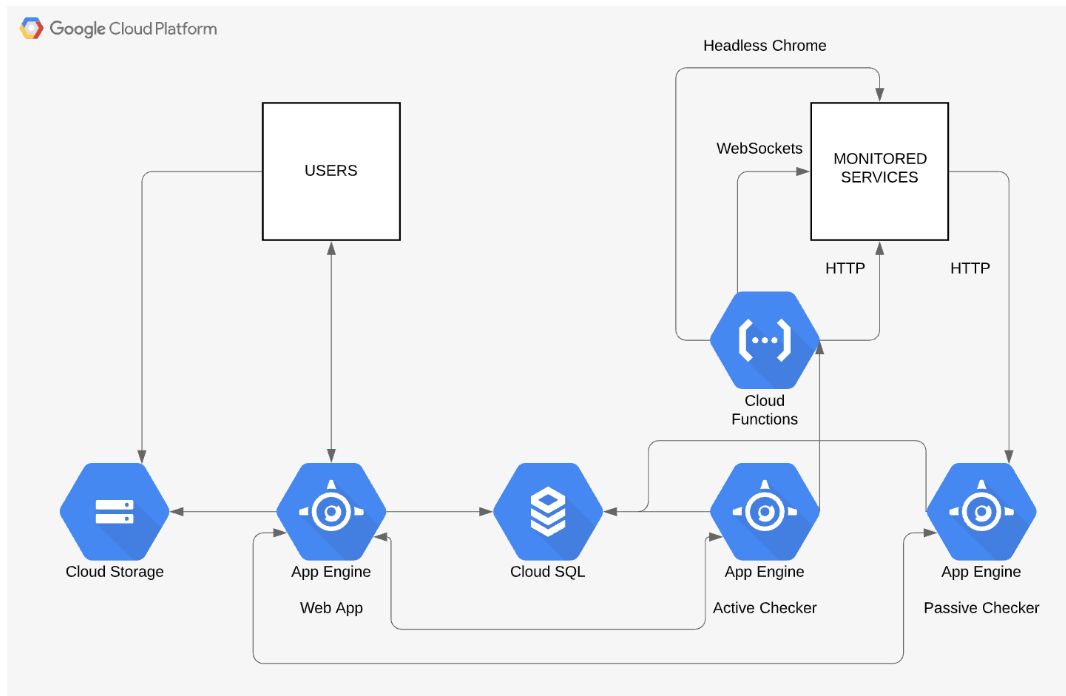

FIGURE 4.2: Project architecture

Web App, Active Checker, Passive Checker and Cloud Functions communicate via Cloud Pub/Sub.

The system stores the images for status pages customisation on Cloud Storage.

The system uses App Engine cron.

The system communicates with monitored services via the HTTP protocol, WebSocket protocol and Headless Chrome features.

We built payments using Stripe. Stripe is an online payment processing platform. When the users sign ups and enters the credit card Stripe saves the payment data and creates a subscription. The application only stores the subscription id on the organisation profile. Stripe integration is event-driven. When the event happens, for example, the monthly payment fails, Stripe sends the event to Web App.

4.3 Future improvements

In this section, we will describe possible future improvements in architecture.

Before doing any of the items below, it is required to run load tests to have a value for comparison. This will help to understand the impact of the changes better.

- Run active checks from multiple locations to make them more precise
- Run Web App in multiple locations
- Rebuild Active Checker and Passive Checker to handle high load amount of Pub/Sub messages

Chapter 5

Conclusions

The goal of this work was to design and develop an uptime monitoring system which would include both automated uptime monitoring and a status page. All these components were designed and implemented successfully.

We actively use status page to communicate incidents to our customers.

After using basic active checks for some time, we discovered that they have a high level of false alarms and thus are too simple to monitor a complex system.

Passive checks have no false alarms and are a good choice for monitoring of regular jobs.

At the moment of writing this text, we are about to start using complex active checks and WebSockets active checks so could not share any results of effectiveness yet.

Bibliography

Betsy Beyer Chris Jones, Jennifer Petoff and Niall Richard Murphy (2016). *Site Reliability Engineering*. URL: https://landing.google.com/sre/sre-book/chapters/embracing-risk/ (visited on 05/10/2020).

Bidelman, Eric. *Getting Started with Headless Chrome*. URL: https://developers.google.com/web/updates/2017/04/headless-chrome (visited on 05/10/2020).

Cronitor. *The quick and simple editor for cron schedule expressions by Cronitor*. URL: https://crontab.guru/ (visited on 05/10/2020).

Google. *Pub/Sub message flow*. URL: https://cloud.google.com/pubsub/docs/overview#concepts (visited on 05/11/2020).

Sookocheff, Kevin. *How Do Websockets Work?* URL: https://sookocheff.com/post/networking/how-do-websockets-work/ (visited on 05/10/2020).