An-Najah National University


Faculty of Engineering and Information Technology

Computer Engineering Department

Graduation Project

# Server Monitor

**Students:**

Rami Theeb
Ibramin Masri

**Advisers:**

Same Arandi, Ph.D.

Presented in partial fulfillment of the requirements for Bachelor degree in Computer Engineering.

May 26, 2021

# Abstract

Cloud computing is one of the most trending technologies these days and almost every developer needs to use it at some point, and to get the most of cloud services and reduce costs, strong monitoring and maintenance system is needed, where the system would provide a better understanding of server usage, anomalies, and other important aspects. In this paper a server monitoring system is presented, this system is meant to provide a powerful and simple monitoring suite that can be used by developers experienced in cloud computing or new to the subject.

# Acknowledgment

We would like to thank our families for their support and encouragement to always aim high and go beyond expectations

We then would like to show gratitude to Dr.Samer Arandi for his supervision throughout the project.

And finally a thank you to all the professors and lecturers in the Computer Engineering department who we owe for the knowledge we posses today.

# Disclaimer

---

This report was written by students at the Computer Engineering Department, Faculty of Engineering, An-Najah National University. It has not been altered or corrected, other than editorial corrections, as a result of assessment and it may contain language as well as content errors. The views expressed in it together with any outcomes and recommendations are solely those of the students. An-Najah National University accepts no responsibility or liability for the consequences of this report being used for a purpose other than the purpose for which it was commissioned.

# Table of Contents

# Lists of Figures and Listings

# List of Tables

# Chapter 1 Introduction

Cloud Computing (in a broad definition) is the utilization of on-demand remote computing resources, allowing users to access these resources according to their need.

Cloud computing offers its services in different flavours, with each offering a different level of flexibility and abstraction to the server admins, allowing cloud computing users to shape their server plans even more according to their needs.

Server mentoring is the process of tracking and recording various performance-related metrics, which allows server administrators to better understand how their server utilizes its resources and make them more aware of anomalies in the usage of these resources and determine when exactly these anomalies happen.

And although Cloud Computing takes a lot of system burdens of the cloud user, the need to monitor the Cloud is present and important to both the providers and consumers [1]. From the consumer side, server monitoring can be utilized to serve many administrative objects. Ranging from capacity and resource planning, where the administrator can analyze the recorded metrics and determine their exact resource usage to then change their plan accordingly, to troubleshooting application-related problems, where dramatic changes in these metrics may indicate that a problem has occurred [2], and monitoring the server will help determine exactly when it happened.

Another administrative objective that monitoring can serve, is performance measuring, due to some nodes on the cloud having varying performance than other nodes [3], and with services such as scientific applications, performance variability and availability is a very important concern [1].

Some of the popular Server Monitoring software out there are SolarWinds, Nagios, Zabbix and Promethues, they offer the main functionality of tracking performance-related data with differing extra features between them. Some of them have visualization

components while other focus on collecting and monitoring the data while delegating the visualization functionality to other applications.

The proposed Server Monitoring System named Server Monitor (SM) is discussed in this paper. SM is an open-source decentralized Server Monitoring and Control System, targeted towards small to medium sized cloud application. SM implements the main functionality of collecting and tracking system data in addition to introducing other services to complement that and create a full monitoring suite for its users. Enabling them to create alerts, analyze their web server's traffic logs, model recorded data to detect anomalies, execute command chains and many other services discussed later in the Modules chapter.

SM allows users to access its services through GraphQL, which is a query language for APIs that allows the user to ask for exactly what they need and nothing more [4] through one single endpoint.

SM also implements its own React-based front-end that visualizes the recorded metrics and allows access to the different provided services. However, SM was built to also allow users to create their own client-side applications by using an Apollo GraphQL Client.

This paper looks at SM, its properties, its features and its downsides. Chapter 2 talks about the constraints to the SM system. Chapter 3 reviews some related applications. Chapter 4 looks at the development aspects of the system. Chapter 5 then looks at the different modules that SM consists of and how they work. Chapter 6 dives into the security details of SM. Chapter 7 talks about the customizability of SM and how it's achieved. Chapter 8 contains the conclusion of the paper and future work for the system.

# Chapter 2 Constraints and Earlier course work

## 2.1 Constraints

- **Capabilities and Reliability:** SM uses web technologies to implement its application's client. Which in general is considered to be slower and have lesser reliability than native apps.

- **Realistic Testing:** One important step of building any successful system is testing the system in real use cases. Due to the difficulty of finding a real server that hosts a web application and has realistic traffic, testing the system's performance wasn't very accurate. Instead it was tested on an AWS instance with some generated traffic .

- **Computational Power:** The main target group of our SM is the developers who deploy their server to host personal projects or small application, who usually have a limited amount of Memory and CPU capacity, forcing the monitoring system to be light weight and have low CPU consumption, limiting the nature of our application features.

- **Secure Transport Layer**: Although SM implements an authentication system, it assumes that the data being transferred is confidential and its integrity can be verified, thereby, the use of SM is limited to systems using secure protocols like TLS and SSL.

## 2.2 Earlier course work

- **Software Engineering:** This course helped with building the right architecture of the application, and organizing the work progress of the project.

- **Web Development:** This course provided a good introduction to the basis of web development technologies used in building this project.

- **Database Systems:** Database Systems course demonstrated dealing with SQL databases and writing different SQL queries, which was used in the project to deal with the SQLITE database.

# Chapter 3 Literature Review

Since the rise of cloud computing, the demand for the non cloud-specialist to use cloud computing has increased significantly, increasing the need for simple tools that help these developers to monitor, maintain and utilize the full power of the machine. This chapter discusses some monitoring tools used by developers.

## 3.1 Prometheus

Prometheus is an open source monitoring system made by SoundCloud, it has a special query language called PromQL and uses the http pull model to retrieve data and update its dashboards. Its support is limited to the server side of monitoring and requires users to have another client tool used for graphing the output such as Graphana. It's more specialized with monitoring metrics like CPU, Memory, and Disk.

## 3.2 Linux Dash

Linux Dash is a popular web dashboard for Linux with almost 10k stars on GitHub. It's very simple and is used to monitor simple servers on real time without having the ability to view the history of the sampled data.

## 3.3 PM2

PM2 is a process manager for nodejs, used as a monitoring and alerting tool to detect web server metrics and failures. It has no access for system metrics. On the other hand, users can access web metrics, using a centralized dashboard under their domain.

## 3.4 Zabbix

Zabbix is an open-source system that has a lot of services like network, server, cloud and application monitoring with easy interaction with all statistics. It does not have a data store, instead, it could be connected with external Databases. Zabbix's is 23 years old and its back-end is written in C and the client is written using PHP.

# Chapter 4 Application Development

---

This chapter discusses the development details of this application including architecture, tech stack, and implementation .

## 4.1 Tech Stack

This section presents SM's tech stack and what were the motivations to use these tools specifically.

### 4.1.1 React

React is a front-end library used to build beautiful, high-performance and highly-reusable SPAs (Single Page Applications) in a declarative, efficient and flexible way. It comes with a great community and ecosystem that provides developers with many resources enhancing their learning experience.

1. Virtual DOM: React has the first and the most powerful implementation of Virtual DOM. It helps the UI to be updated efficiently and smoothly, which is a great fit for real-time apps.

2. One-way data binding: The data in react always flows from up to down (parent to child) which makes it easier to debug and more predictable.

3. High reusability: The nature of SM contains a lot of reusability of components,and the components-based approach of react highly supports this.

4. Readability and testability : It's easy to write readable and testable code using react and it's functional paradigm.

### 4.1.2 NodeJS

It's an asynchronous event-driven JavaScript out-of-browser runtime environment which can handle many connections concurrently using google v8 engine.

1. Easy syncing with the client since both use typescript, which may even make it possible to share logic between them and

enable moving the react app to the server to use SSR(Server Side Rendering) if needed.

2. A huge set of tools and libraries .

### 4.1.3 Typescript

Typescript is a super set of JS(javascript), it powers SM's frontend and backend NodeJS code with static type definitions and provide a way to describe the shape of objects.

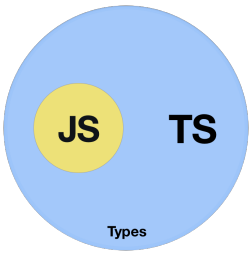



Figure 4-1: Typescript

1. Reduce bugs: Using TS(Typescript) reduces the number of bugs passed into production by 30% as proven by 1st tier companies like Airbnb,

2. Easy development : Gives a great developer experience by providing things like Mouse Hover Support, Code auto-completion, Real-time type checking and easier code refactor,

3. Clear documentation : all functions and components are now supported with a strong type system which will help with auto generating documentation for this functions and components.

### 4.1.4 GraphQL

GraphQL is a query language for both client and server, it holds descriptive information about the data in the server and powers the client with the ability to request exactly what they need.

1. Strongly typed : GraphQL comes with a custom type definition protocol.

2. Types sync: since SM uses typescript, it needs to define types for data. GraphQL and some other external tools help with keeping the types synced between client and server without rewriting them manually.

3. Minimize network latency: since GraphQL supports the ability to request just the needed fields from specific objects, this reduces the unneeded data transferred between client and server .e.x. if the name of the user is needed in the name of the user section in the home page, and the name and the image in the settings page, the same endpoint can be used to request the data in both places the client will be getting exactly what it needs in both places. Which solves a problem known as over-fetching.

4. Eliminate waterfalls in request : since GraphQL comes with the ability of hitting more than one single source in a single request, this eliminates a lot of the network latency, and also eliminates the waterfalls caused by the requests depends in each others .

### 4.1.5 Redis

Redis is an in-memory data structure widely used in cloud computing and system management. Redis provides many capabilities for the user and supports third party modules which add extra functionalities to it.

## 4.2 Application Architecture

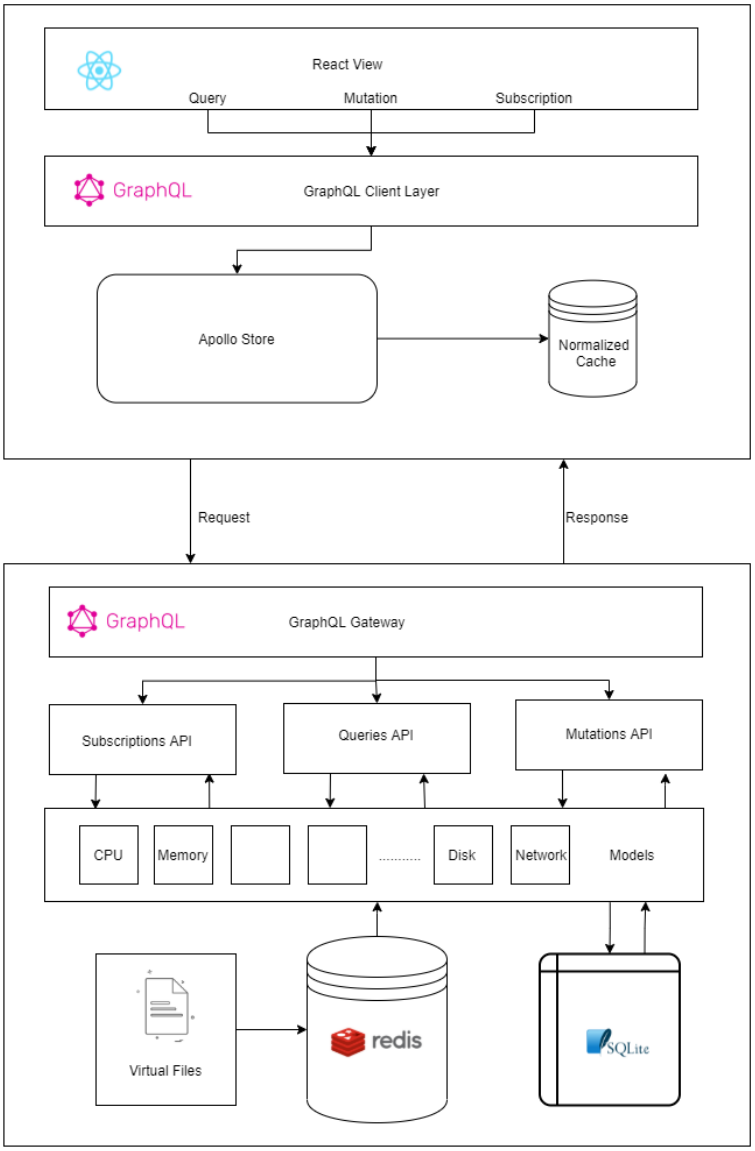



Figure 4-2: Application Architecture

SM is built to be decentralized, every server owner who wants to use this application has to setup the application on their server to operate separately from any other user, this will increase the performance significantly by minimizing the amount of data that have to be transferred out of the server, this also serves the purpose of the application of being a scalable open source in the future.

SM consists of a client and a server components, both of which work independently, enabling the user to choose using the backend

alone and access it using GraphQL playground without any problem or use the provided React client.

### 4.2.1 Frontend

The frontend architecture consists mainly of 4 layers

1. React View: Which is a set of separate reusable react components built using Material-UI design to be user friendly and easy to use. These components give the user the ability to present and visualize the data efficiently in both real-time and historical forms, the view also gives the user the ability to build their own dashboard that contains their collection of important components and data in one place.

2. GraphQL Client Layer: Which is a comprehensive state management layer that provides the ability to manage data flow in the application of both local and remote data. It's used to fetch, cache and modify data in the application while it automatically updates the cache and the UI. It provides an excellent developer experience with a declarative data fetching.

3. Apollo Store: This is used as a local state data storage for the frontend and and Apollo client in it's core. It is a state management library that uses GraphQL.

4. Normalized Cache: This layer is used to store the results of the queries in a normalized, in-memory cache to respond to the duplicate queries without refetching the data based on a specified TTL(Time To Live). This layer also contains a garbage collection strategy to keep the memory clean and avoid storing any excess data .

### 4.2.2 Backend

The backend architecture consists mainly of 4 layers

1. GraphQL Gateway: GraphQL server layer which provides a single endpoint to query, mutate, and subscribe data from the data sources. It also works as a communication layer with the frontend.

2. Query API: this layers handles all queries coming from the client in a simple, predictable way. It takes a schema that follows the GraphQL Query protocol in addition to some optional variables, it then returns the data in the same requested schema.

3. Mutation API: this layers handles all mutations (sending data to the server), or any request that may cause a side-effect in the server. It follows the same query syntax.

4. Subscriptions API: A layer that uses PubSub design pattern to handle all the real-time subscriptions coming from the client to a specific data source. A subscription is a long-lasting read operations which sends new data to the client every time a specific server event happens. It uses the WebSocket protocol and operates in a separate URL (wss://app/subscriptions).

## 4.3 SM Progressive Web App

PWAs (Progressive Web Apps) are web apps supported from with all new browsers and designed with a specific standards to be a capable, reliable, and installable. When integrated with technologies like push notifications, it provides the feel of a native mobile app using native web languages like HTML, CSS and JS.

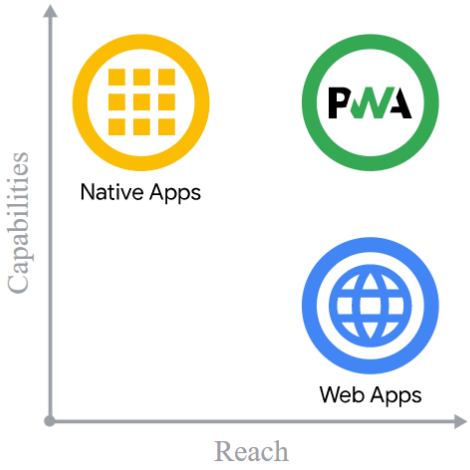



Figure 4-3: PWAs

To be able to achieve this "native app" feeling, the following measures where taken:-

1. Created an efficient caching strategy to cache all static assets to reduce first load time of the application .

2. Designed all pages to be responsive and mobile friendly.

3. Integrated the application with a web notifications service called Pusher to get the feel of native mobile push notifications.

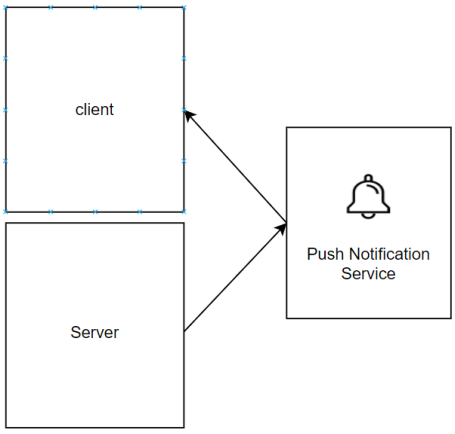



Figure 4-4: Push Notifications

4. Added an offline fallback page to make the website work offline and give the user a hint that they are not connected .

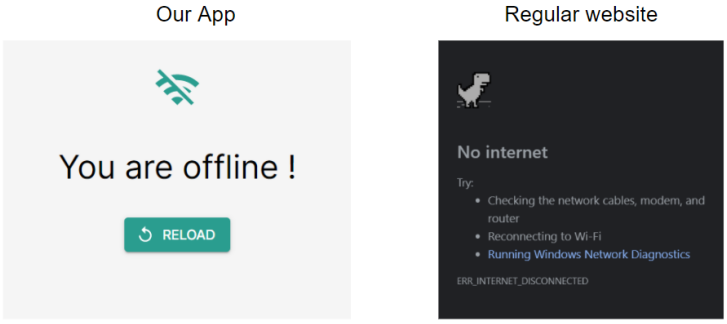



Figure 4-5: Offline Fallback

## 4.4 Development Tools

1. VS Code : Used as a main code editor for both client and server, contains a lot of plugins that help in increasing productivity and support clean coding like prettier for code styling, git blame and git lens for git integration.

2. GitHub : Used as a git cloud to organize project collaboration. Git system of PRs(pull request) was used to review different collaborator's codes to share knowledge and increase code quality .

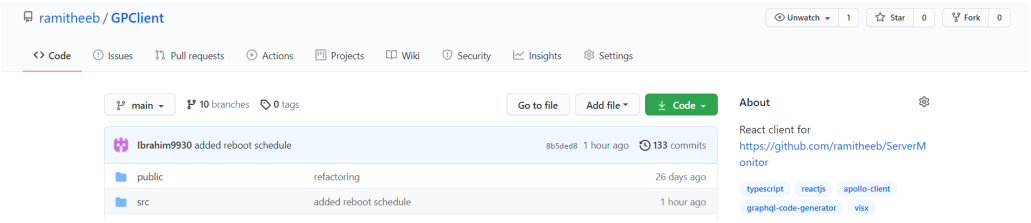



Figure 4-6: GitHub

3. Notion: This tool was used to organize the development process, create dashboards, track project progress, share knowledge and share resources.

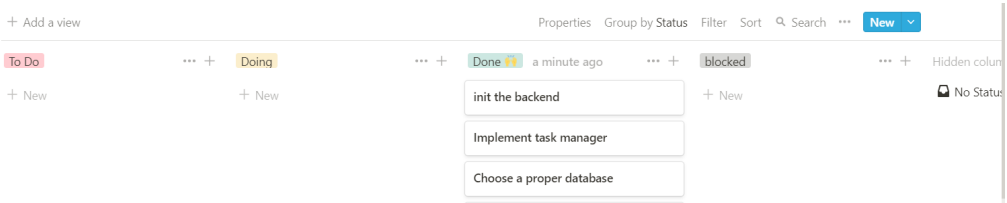



Figure 4-7: Notion

4. AWS: Used as a cloud provider to run and test SM on a real production environment.

5. Chrome DevTools : Used Chrome DevTools in some debugging tasks which helped us with solving bugs and improving rendering and network performance.

6. TSimulus : A tool used for generating random but realistic data for the time series databases, we used this tool to generate some historical data for our metrics.

7. Netlify : client code hosting system that has some CI/CD tools, Netlify helps with syncing the production website with the GitHub repository by deploying the client on each push to the main branch.

## 4.5 Documentation

This section explains in details the most important data that is being monitored in different modules.

### 4.5.1 CPU

| Field | Description |
|---|---|
| manufacturer | - |
| brand | - |
| speed | - |
| speed Min | - |
| speed Max | - |
| cores | number of CPU cores |
| physical Cores | number of CPU physical cores |
| cache | cache in bytes |
| virtualization | virtualization supported |

Table 4-1: CPU Fields

### 4.5.2 Memory

| Field | Description |
|---|---|
| total | total memory in bytes |
| free | not used in bytes |
| cached | used by cache |
| swaptotal | - |
| swapused | - |
| swapfree | - |

Table 4-2: Memory Fields

### 4.5.3 Disk I/O

| Field | Description |
|---|---|
| rIO | read IOs on all mounted drives |
| wIO | write IOs on all mounted drives |

Table 4-3: Disk I/O Fields

### 4.5.4 System

| Field | Description |
|---|---|
| server local current time | - |
| uptime | server up time |
| timezone | server timezone |
| manufacturer | e.g. ’MSI’ |
| Platform | - |
| Distribution | - |

Table 4-4: System Fields

### 4.5.5 Network

| Field | Description |
|---|---|
| iface | interface |
| operstate | up / down |
| rx bytes | received bytes overall |
| tx bytes | transferred bytes overall |
| rx sec | received bytes per second- |
| tx sec | transferred bytes per second |

Table 4-5: Network Fields

### 4.5.6 Docker images

| Field | Description |
|---|---|
| container | container ID |
| os | - |
| architecture | - |
| parent | - |
| size | image size |
| created | created date / time |

Table 4-6: Docker Images Fields

### 4.5.7 Docker Containers

#### 4.5.7.1 Static

| Field | Description |
|---|---|
| name | - |
| image | - |
| created | - |
| started | - |
| finished | image size |
| createdAt | creation date time string |
| state | created, running, exited |

Table 4-7: Docker Containers Static Fields

#### 4.5.7.2 Real Time

| Field | Description |
|---|---|
| memUsage | memory usage in bytes |
| memLimit | memory limit (max mem) in bytes |
| memPercent | memory usage in percent |
| cpuPercent | cpu usage in percent |
| netIO.rx | received bytes via network |
| netIO.wx | sent bytes via network |
| memoryStats | detailed memory stats |
| cpuStats | detailed cpu stats |

Table 4-8: Docker Containers Real Time Fields

# Chapter 5 Modules

SM's services implementation is organized into modules, each module is responsible for one aspect of the system. It communicates with the GraphQL server and sometimes other components outside the server's environment to collect and store data to implement its functionality. This chapter looks at those modules, the features they implement and dives more into how they work.

## 5.1 Sampler

The Sampler module is concerned with collecting, storing and publishing data collected from the system.

The Sampler performs a sampling operating periodically, which collects system data such as CPU current load and the amount of used memory. After it collects this data, it stores them in the Redis Time Series storage. Each metric has four associated time series units, each with a specific precision and retention period (which defines how long each sample stays in the time series), and samples propagate through these time series units, meaning that if the sample is added the first unit (named runtime series), it gets automatically added to the other units.

Figure 5-1 illustrates this behaviour, where the four time series units (namely runtime, short, medium and long) are shown. New samples are added to the runtime series, when enough samples have been gathered at the runtime series to create a sample at the short series, the average of their value gets inserted into the short series, and the same thing happens between the rest of the units.

The sampler module also allows subscription to data, where users can view the data at runtime as its being sampled. After the sampler issues the store commands, it publishes that data to all listening subscribers through the Redis PubSub module and the Apollo GraphQL Subscription resolvers.

Figure 5-2 shows how the front-end application visualizes the runtime retrieved data.

The rate at which the module samples data is configurable and can be changed by using the configuration file (explained later in chapter

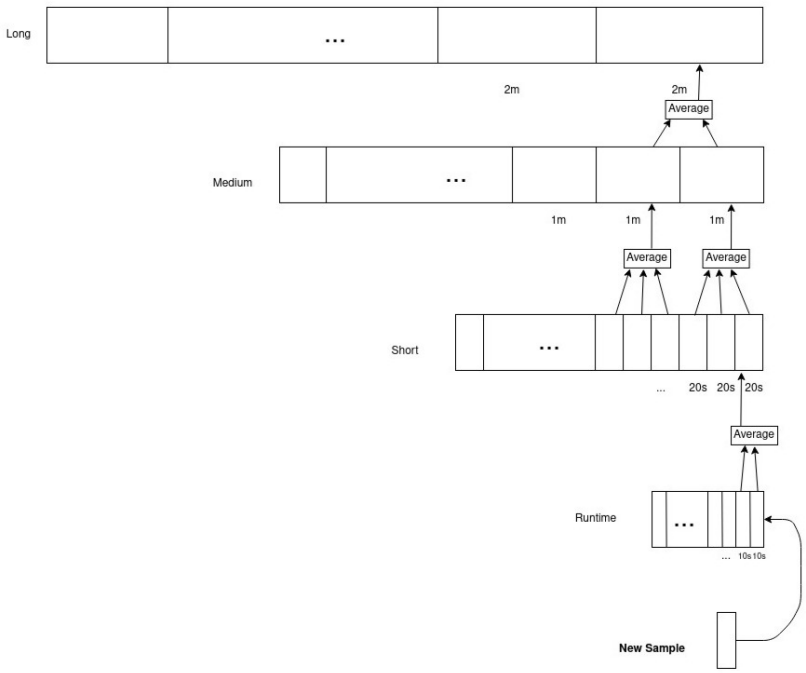



Figure 5-1: Redis Time Series Units

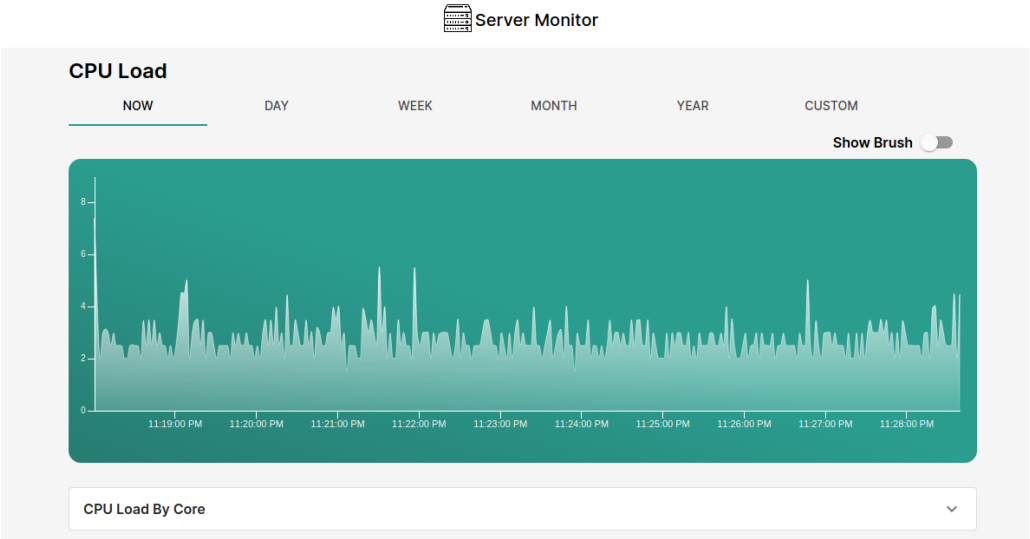



Figure 5-2: CPU load chart at runtime

5), there are two sampling rates, one of them controls the sampling rate which the sampler uses when no active subscriptions are open, the other one controls the rate when there exists a subscriber to any metric in the server, and because data is only added to the runtime series, increasing the sample rate doesn't increase the number of samples in the short, medium and long series, because data is inserted

| Option | Range |
|---|---|
| Day | last 7 days |
| Week | Last 4 weeks |
| Month | Last 12 months |
| Year | Last 5 years |
| Custom | Implicit time range |

Table 5-1: The options for the get<metric>History query

into these series by aggregating it over fixed periods, not fixed sample numbers.

To retrieve stored data, the user sends a get<metric>History query, which has an option parameter that allows the user to retrieve data in predefined ranges, or specify the timestamp range of retrieved data. listed in table 5-1 are the options of the get<metric>History query.

## 5.2 Alerts

The alerts module takes on the tasks of setting up and controlling user-defined alerts, periodically checking system metrics to determine if an alert should be fired and modelling the system metrics to check for anomalies.

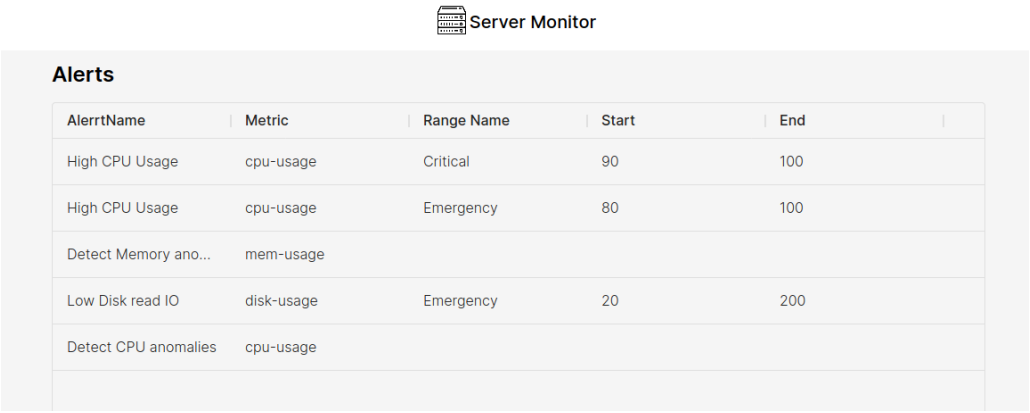



Figure 5-3: Alerts front-end view

### 5.2.1 Alert types

SM implements two types of alerts, static and dynamic, the following subsections talks more about each type.

#### 5.2.1.1 Static alerts

Static alerts allow the user to specify a range of values and the system metric to watch, whenever this metric value enters or exits the range, the module fires an alert.

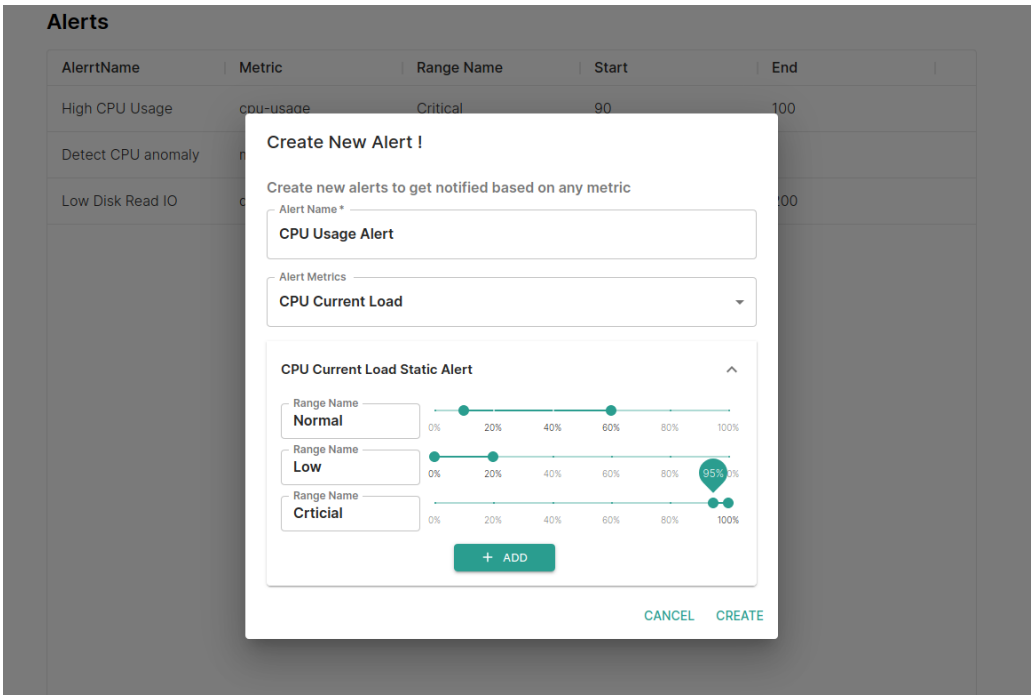



Figure 5-4: Creating a static alert

Figure 5-4 shows how the user creates a static alert, they first specify the name of the alert, followed by which system metric to watch for, in the figure, the CPU's current load is watched. After that, the user starts creating ranges, each with a name and a numerical range, whenever the metric enters or leaves one of these ranges, an alert specifying the alert name and range is fired.

#### 5.2.1.2 Dynamic alerts

Although static alerts provide users with the ability to keep track of their system and be notified of when certain events can occur, they might not be suitable for all use cases, where the user wants a more customized alerting system that takes into account previous history of the metrics, and models the system to then be able to tell if a certain value, at that certain point of time and with that certain context, is an anomaly and hence worth reporting to the user.

Dynamic alerts are implemented for that specific purpose, where the value of the system's performance metric is compared against dynamically-generated thresholds extracted from the built model.

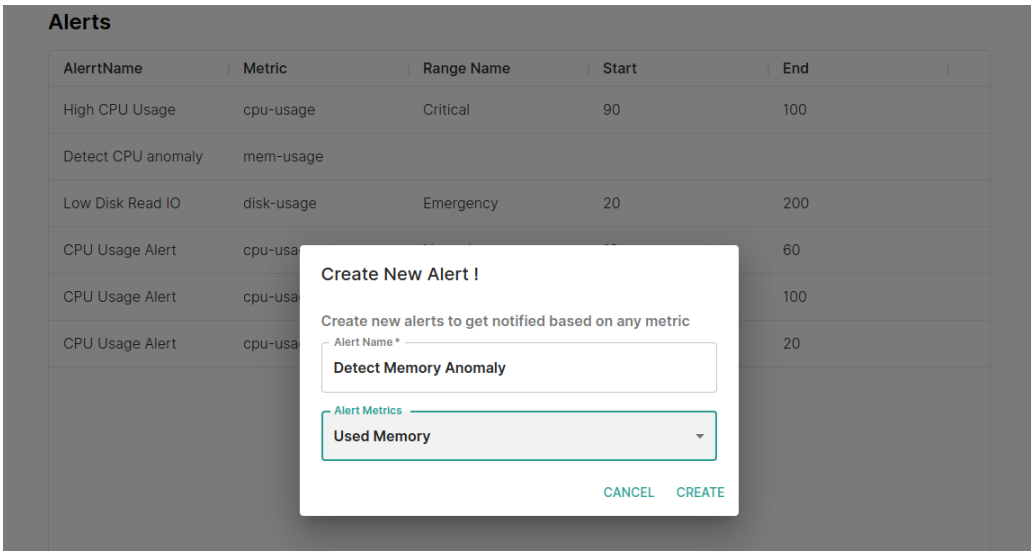



Figure 5-5: Creating a dynamic alert

Figure 5-5 shows how the user creates a dynamic alert through the web application. They only have to specify a name for the alert, and which metric to watch. The system is then responsible for building an adaptive model of the system, and dynamically calculate the anomaly-defining thresholds.

### 5.2.2 System Modeling

The alerts module is responsible for building models that represent the "normal" operation values of different metrics. The module uses adaptive statistical filtering and moving averages to create these models.

Statistical filtering is to "compare a measured value with thresholds that are computed through a statistical analysis of a set of previously measured values" [5]. Basically, the previous measured values are used to calculate the average and standard deviation values of the data. Assuming that the data has normal distribution under normal operation circumstances, 99.7% of the data should fall between three standard deviations around the average, which means that any value outside that range has a low probability of being a "normal" value. Furthermore, the probability of the value occurring four times in a row is $8.1^{-11}$% , which indicates that the system is probably not operating

under normal circumstances [6], and hence, the value indicates an anomaly in the system.

However, assuming that a single normal distribution applies to the system at all times is not very accurate, system loads typically vary from hour to hour [5], for instance, the amount of bandwidth used during noon hours where traffic to the server maybe high, could vary greatly from that at 4 am in the early morning.

To solve this problem, a more accurate model called, Adaptive Statistical Filtering is used. The main idea is using different reference sets for different hours of the day and days of the week to calculate the dynamic thresholds (calculate different average and standard deviation values for different hours of the day and different day of the week). Resulting in 168 reference sets, each one modeling a certain of hour of a certain day in the week (24 hours x 7 days = 168 sets).

Moreover, to further model the server's trends, instead of using normal averages and standard deviations, where all the values in the reference sets are used in the calculations, moving averages and standard deviations are used, where the average changes over time, taking into account newer values and dropping older ones. This is meant to make the model keep up with changes of the system's performance that may occur overtime, where for example if the system hosted a website, the website's traffic may increase over time or decrease.

The algorithm that finds the statistical characteristics of the sets (average and sigma values) runs periodically. The time at which the algorithm runs is decided by the Scheduler module. As mentioned before, the built model uses moving averages, where the amount of samples used to calculate the average and sigma is fixed, whenever a new value appears, it gets added to the average and sigma calculation while the oldest value in the calculation gets discarded. If there was, however, a number of samples less than the required amount, new values are added without discarding old values until the number of samples is equal to the required amount. The amount of samples that SM uses is 12 samples.

### 5.2.3 Push notifications

After one of the alerts fires, the user should be notified about the alert going off. SM uses push notifications to inform the user about fired alerts.

## 5.3 Web Server Log Parser

Many cloud-based servers are used to host websites, where a web server is usually used to manage the traffic to that website and allow for things like node balancing and reverse proxy.

Many of these web servers, including Nginx and Apache, log details about the HTTP requests sent to them in access logs, these details include the IP address of the client, the destination URI, the time of the message sent and many other fields.

The Web Server Log Parser module reads these access logs, parses them to extract the request's details, and then uses the parsed information to analyze the traffic to that web server.

The module first uses the logs to monitor the amount of traffic to the server, and implements history queries similar to those of the Sampler module to access the monitored data. Figure 5-6 illustrates how the client visualizes that data.

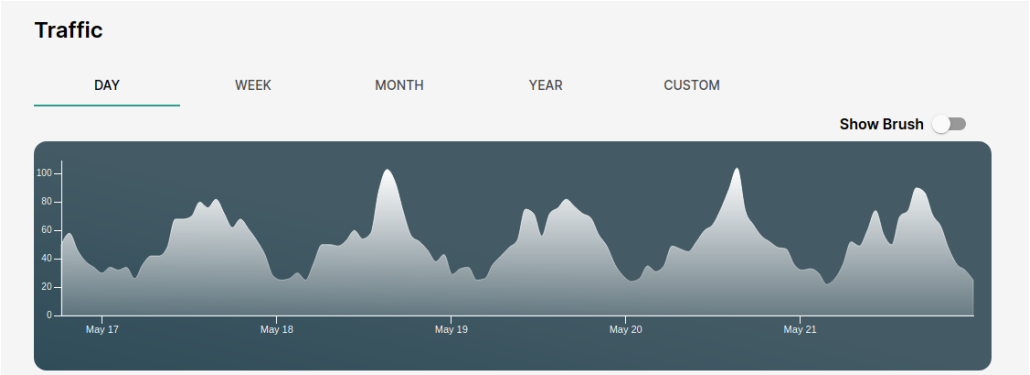


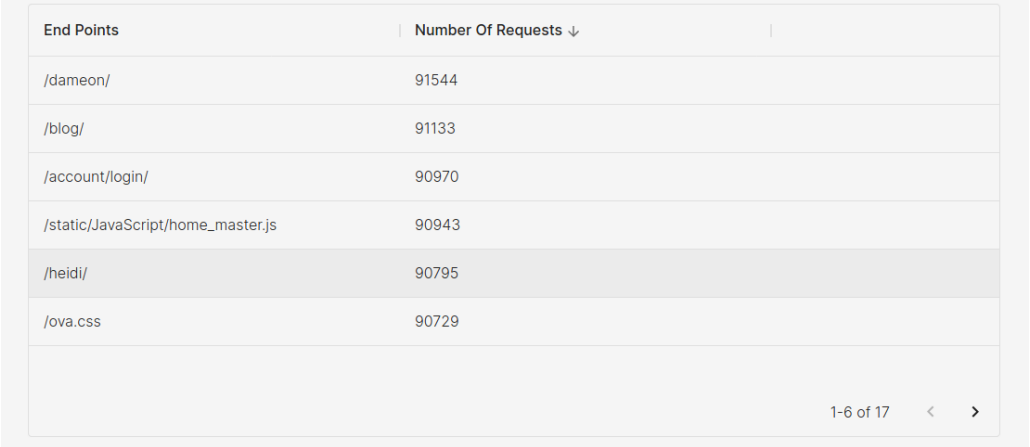

Figure 5-6: Traffic graphs

The module also uses the logs to find the number of requests directed at each endpoint, allowing the user to better understand how the traffic at their website is distributed.

Figure 5-7 shows the table in the client that shows the different endpoints of the website, and the corresponding amount of traffic directed at that endpoint.

Another analysis that the module does with the parsed data is demographic analysis, where it uses GeoIP Databases to find out the geographical location of the different IP addresses. Enabling the user to have a better understanding of where most of his traffic is coming from.

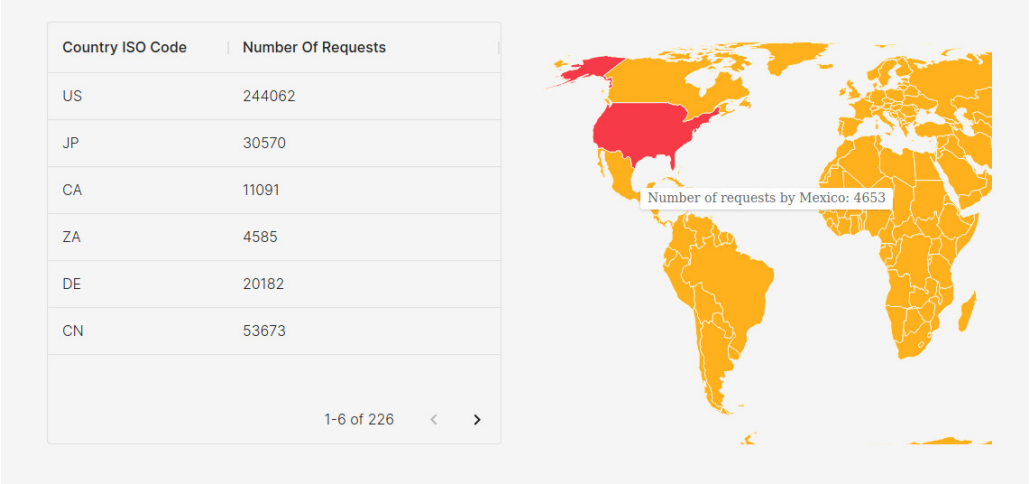
Figure 5-8 shows how the client visualizes the demographic analysis, the table on the left contains all the countries that the



| End Points | Number Of Requests ↓ |
|---|---|
| /dameon/ | 91544 |
| /blog/ | 91133 |
| /account/login/ | 90970 |
| /static/JavaScript/home_master.js | 90943 |
| /heidi/ | 90795 |
| /ova.css | 90729 |

1-6 of 17

Figure 5-7: Traffic graphs



| Country ISO Code | Number Of Requests |
|---|---|
| US | 244062 |
| JP | 30570 |
| CA | 11091 |
| ZA | 4585 |
| DE | 20182 |
| CN | 53673 |

1-6 of 226



Figure 5-8: Demographic analysis

requests came from, and the number of requests coming from each one. The map is a heat map where colour intensity reflects the number of requests of each country, hovering over any country shows the exact number of visits from that particular country.

For the user to use the parser module, they have to first enable it in the configuration file. Then, they specify the path to the site's configuration file, where SM would create a new log file with the required format and store it in a proper location so it can find it later and use it.

## 5.4 Command Chains

The Command Chains module implements the ability to execute a chain of commands on the server's terminal, adding a slight aspect of control on top of the monitoring functionality of SM.

The idea behind Command Chains is to try to make it easier for system administrators to execute a sequence of commands that they use repeatedly. It provides them with the ability to store all these sequences in a single place and be able to execute them remotely and securely.

The Command Chains module takes on the tasks of managing the chains, executing them, giving them sudo privilege and protecting and restricting their execution.

### 5.4.1 CRUD

The module implements and controls the processes of creating, reading, updating and deleting different command chains. The actual sequence of commands is stored on the server as shell scripts, where the module adds the necessary headers to the provided sequence to create these shell scripts.

The module also allows for the user to create arguments that are passed to these scripts, where they can pass different arguments every time they execute a script, creating a more dynamic behaviour for these sequences. The arguments are accessed in the script by using $i, where *i* is the number of that argument (first, second, etc...).

Figure 5-9 shows the client-side page for command chains (called "Command Center" in the client). Each card in the page represents one of the chains, where it shows the name of the chain and the relative location of the shell file it's stored in at the top. Next, there are the arguments that the user passes to the chain. The user can edit these arguments before executing the command chain to have more control over the behaviour of the chain and create a more dynamic nature for them. The icon on the bottom left executes the command. After the command has been executed successfully, its output (if there exists any) would be displayed on the black box above the execute button. Finally, the checkbox next to the execute button allows for executing the command with the SUDO privilege, checking this option would prompt a further authentication dialogue discussed in the protection subsection.

Figure 5-10 shows the interface for adding and editing chains.

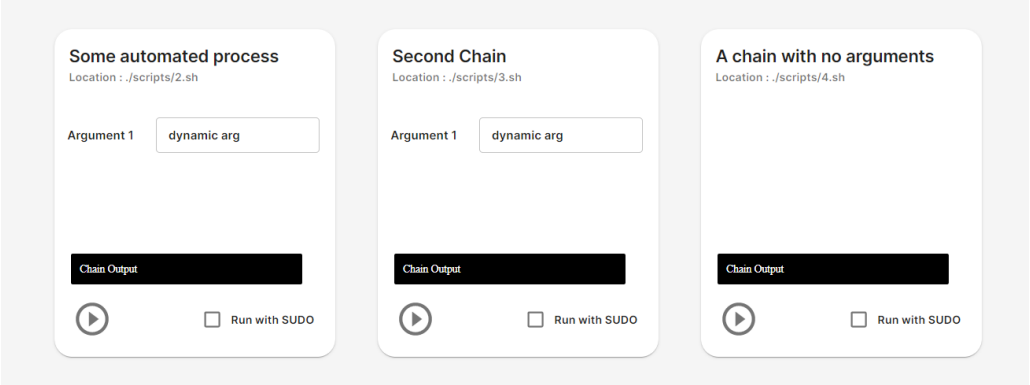



Figure 5-9: Command Chains view

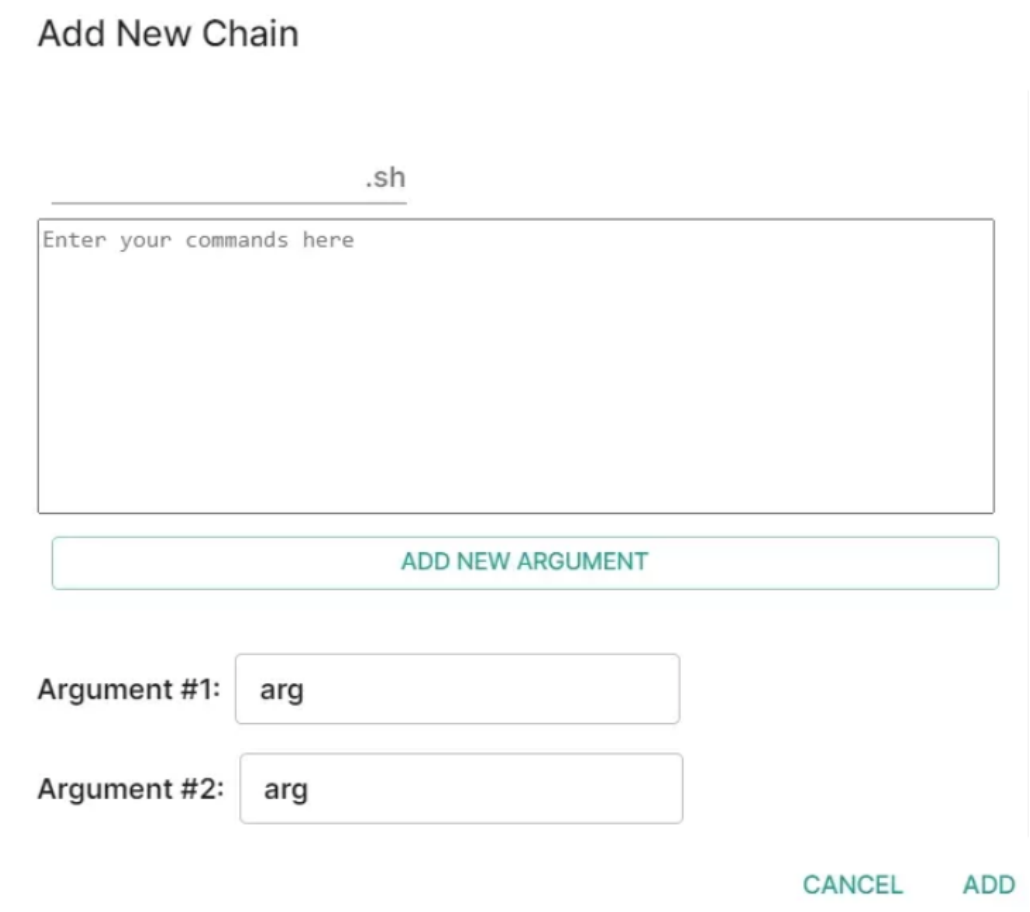



Figure 5-10: Adding and editing a chain

### 5.4.2 Execution Protection

The Command Chains module adds an extra security level to command execution to guard them against exploitation and unauthorized access.

Each chain has a password protected field, if this field is set to true, execution of the chain would require the user to enter a one-time password sent to their mobile phones. Furthermore, this level of protection is also added to chains that the user wants to execute with the SUDO privilege, where the user is also asked to enter a one-time password if the "run with SUDO" option was checked.

Figure 5-11 shows the dialogue that appears to the user in the client whenever they try to execute a password protected command, or a command with the SUDO privilege.

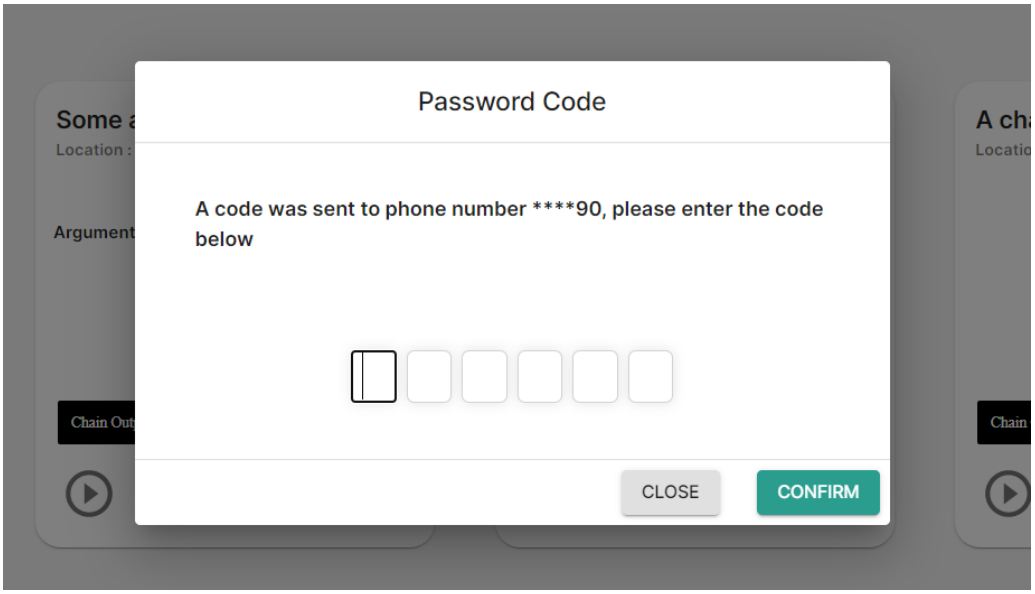



Figure 5-11: Execution protection

The module makes sure that if the "run with SUDO" option was not checked, the command would run as a normal user. This is done by explicitly changing the user running the script before executing that script. This is done to safeguard against implicit SUDO privileges given to scripts ran by a process with SUDO privileges.

## 5.5 Scheduler

Some tasks in SM, like metric modeling and log parsing, are required to be scheduled and run periodically. This scheduling is implemented in the Scheduler module, where it does some analysis to try to find an optimal time for these tasks to be run, so that they would not disturb other tasks running on the system at that time by taking over the CPU load. The module also implements the ability to schedule system reboots, where it also tries to find the optimal time for those reboots.

### 5.5.1 CPU load analysis

Based on the CPU load model built by the Alarms module, the Scheduler analyzes this model to find points in time at which the CPU is more likely to have a relatively low load, in an effort to predict the points in time that the server isn't doing much and the CPU can be used to run the periodic SM tasks, or the server can perform its reboots at that time.

### 5.5.2 Scheduling

The module uses the Cron Job tool to schedule different tasks a. These tasks include metric model building (done by Alerts module), web server log parsing (done by Web Server Log Parser module) and a task that finds the optimal points in time to run the previous tasks (and itself) and to issue a system reboot.

It's worth mentioning that the point in time that the system uses to schedule the previous tasks is different than the one used to determine the optimal reboot time. This is done in an effort to avoid having the system reboot while trying to execute these tasks.

### 5.5.3 Reboots

Reboots can be scheduled to run at any point in time in the future, the user can either issue the reboot at the recommended point of time (the one calculated in the scheduling process), or have it run at a time they prefer instead. Figure **??** shows the client side interface that schedules a reboot, the button is located at the "Command Center" page, where the user either selects the time they want the reboot to happen, or checks the "use calculated optimal downtime" box to use the calculated optimal downtime.

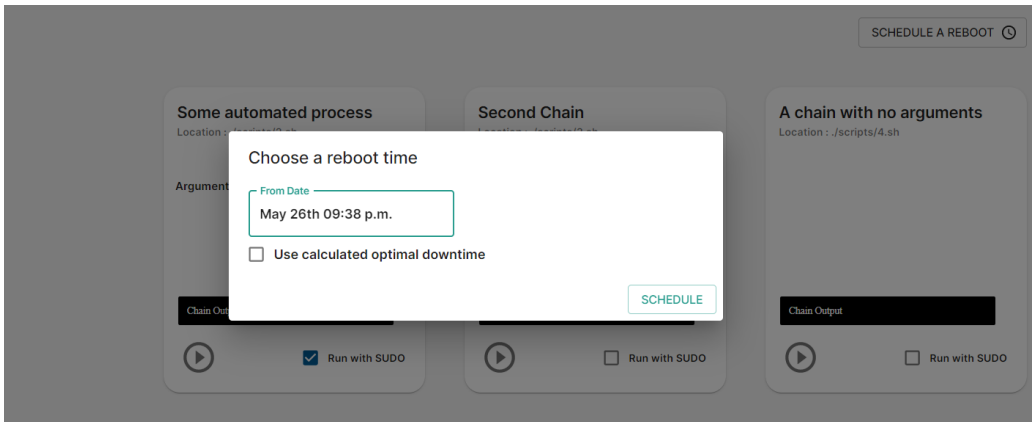



Figure 5-12: Scheduling a reboot

## 5.6 Docker

Docker is an open platform that helps developers with developing, shipping, and running applications. and it's used almost by every developer to deploy their applications to the cloud, so docker monitoring is an important part of our system.

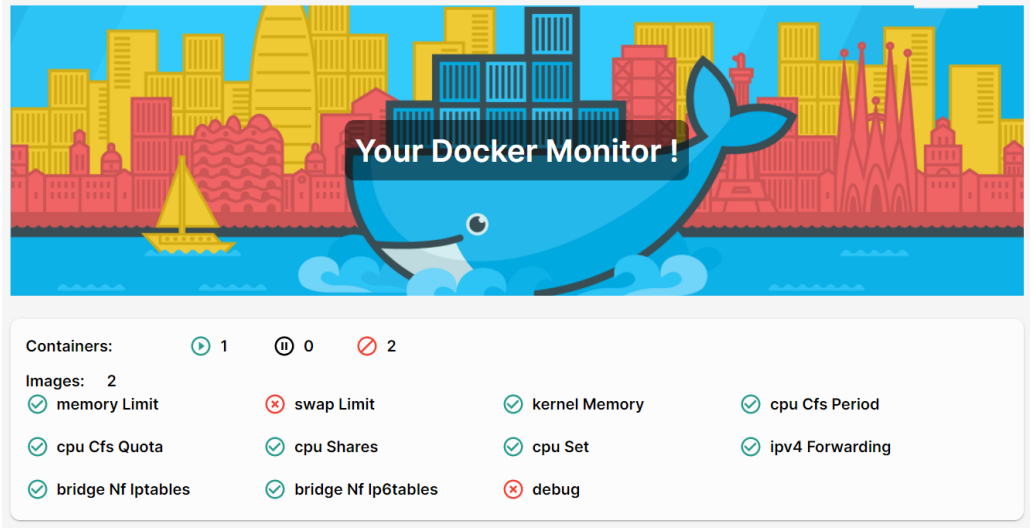



Figure 5-13: Docker

Through this page the user can see how many images you have, the number of running, paused, and stopped containers, and other docker flags like memory limit that indicates if you set a limit on the amount of memory that could be used by docker.

The user can also view all their downloaded images and general data about them like size, created by whom, etc..., they can also view more info about the currently running containers like restart count and starting time.

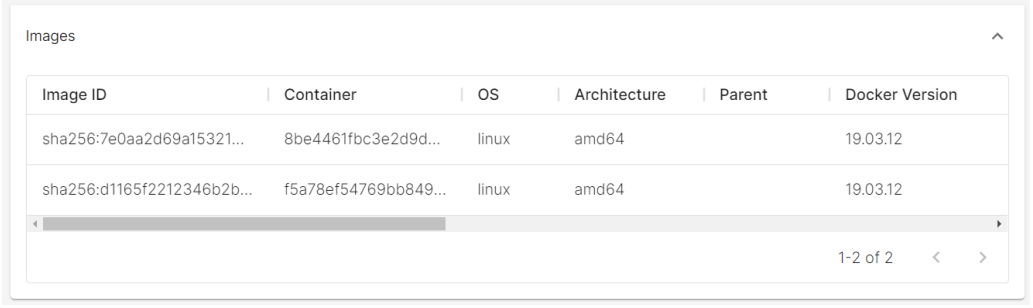



Figure 5-14: Docker Images

To improve the quality of monitoring SM provides the user with the ability to monitor important metrics like I/O, CPU, Restart Count, and Memory for a certain container in real-time.

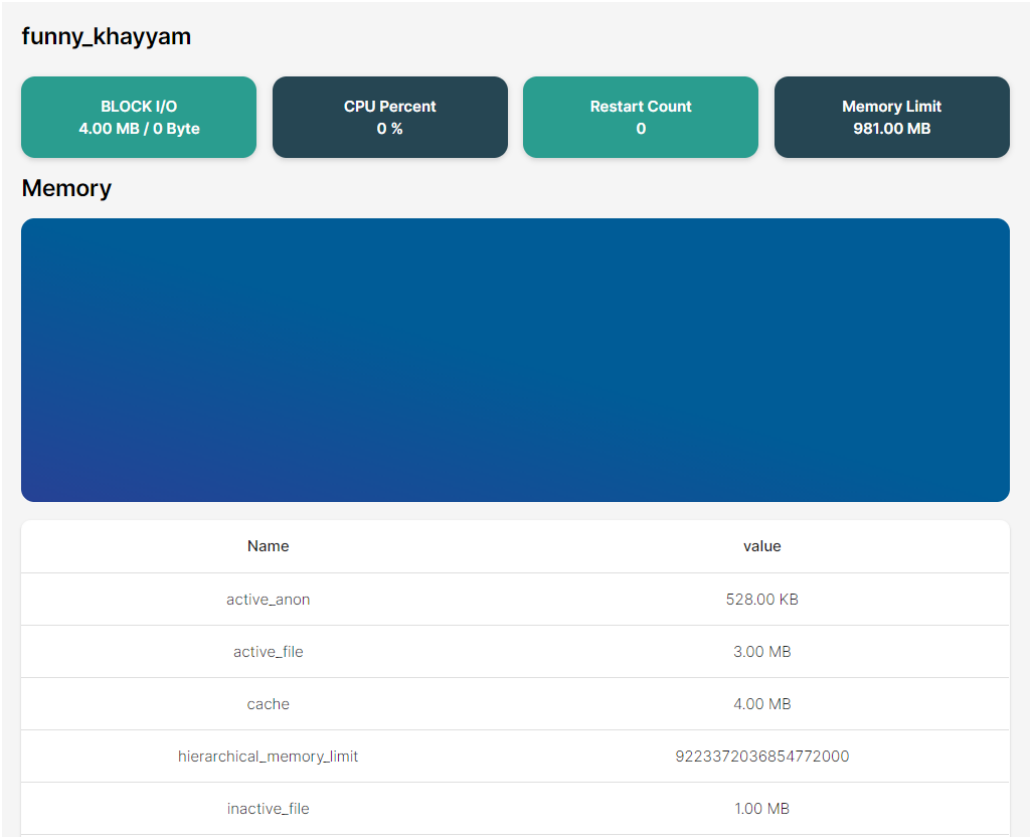



Figure 5-15: Docker Containers

# Chapter 6 Security

Due to SM's application nature and some of the features it implements, some security concerns may arise considering issues like unauthorized access to application and system's data and exploitation of the Command Chains module.

SM implements some countermeasures to try to eliminate, or at least reduce the possibility of these issues happening.

These countermeasures consist of an authentication system that allows for user-defined authentication methods, while natively supporting a key-pair authentication method. A one-time password system created to protect the Command Chains module. And the use of JSON Authentication Tokens and sessions.

## 6.1 Authentication

The idea behind SM's authentication system is to allow users to create their own authentication methods, while having a built-in key pair authentication mechanism they can use if they don't want to create their own mechanisms. This is done by implementing the Generic Message Exchange Authentication for the Secure Shell Protocol (SSH).

### 6.1.1 Generic Message Exchange Authentication for the Secure Shell Protocol (SSH)

The Generic Message Exchange Authentication for the Secure Shell Protocol (SSH) is a protocol designed to allow for a general purpose authentication method for SSH authentication [7]. This allows for users to have more control over the authentication of the system while providing them with a framework to use. Some modifications were made on the protocol to adapt it to the GraphQL mutation mechanisms and utilize NodeJS's provided features.

The exchange protocol starts with the client sending an Authentication Request mutation, where they inform the server of their username and the name of the authentication mechanism they want to use (called service name), the server then looks at the provided username and authentication method, if the service name sent is not recognized by the server, the server throws a service name not defined exception, otherwise, it responds with an authentication information

request, which includes instructions for the client to authenticate themselves, along with a list of values that the user may need to complete the authentication method. The request also contains a list of prompt strings for each field required to be filled by the client to complete that authentication step, a list of flags that informs the client whether it's okay to echo the user's responses and a field indicating the number of fields required to be filled by the user.

The server stores information about this authentication attempt in a server-side stored sessions to create a session over HTTP.

The client should then reply with an Authentication Information Response mutation, where they include the responses for the authentication fields in the Authentication Information Request. The response also includes the number of responses sent.

When the server receives the Authentication Information Response mutation, it firsts checks its session storage to extract data about this particular authentication attempt, if a session doesn't exist, the server replies with a failure message, otherwise, it continues on by passing the responses to the appropriate authentication handler for the specified service name and the number of authentication messages sent, the handler then either accepts the responses sent by the client and finish the authentication process by sending a success message, rejects the responses and replies with a failure message, or accepts the message and prompts the user for further authentication steps by sending them another Authentication Information Request. This allows for authentication mechanisms that require multiple steps like two-factor authentication.

If the server receives an Authentication Request with a username that doesn't exist, it doesn't send a failure message, but instead sends a bogus Authentication Information Request and discards the session's information, so that when the user sends its authentication response, it will be met with a failure message. This measure is taken to make it harder for anyone to find out the name of valid users by sending different authentication request with different usernames.

Figure 6-1 shows the message exchange operation for an authentication mechanism where the user has to enter two passwords to get authenticated.

For the user to implement their own authentication method, they can use the addAuthHandler method provided by the authentication module that takes on authentication tasks. The function takes in the service name and list of authentication handler functions as a parameter, each one of those functions is used to authenticate one of the steps in the process and should return either a boolean to indicate a

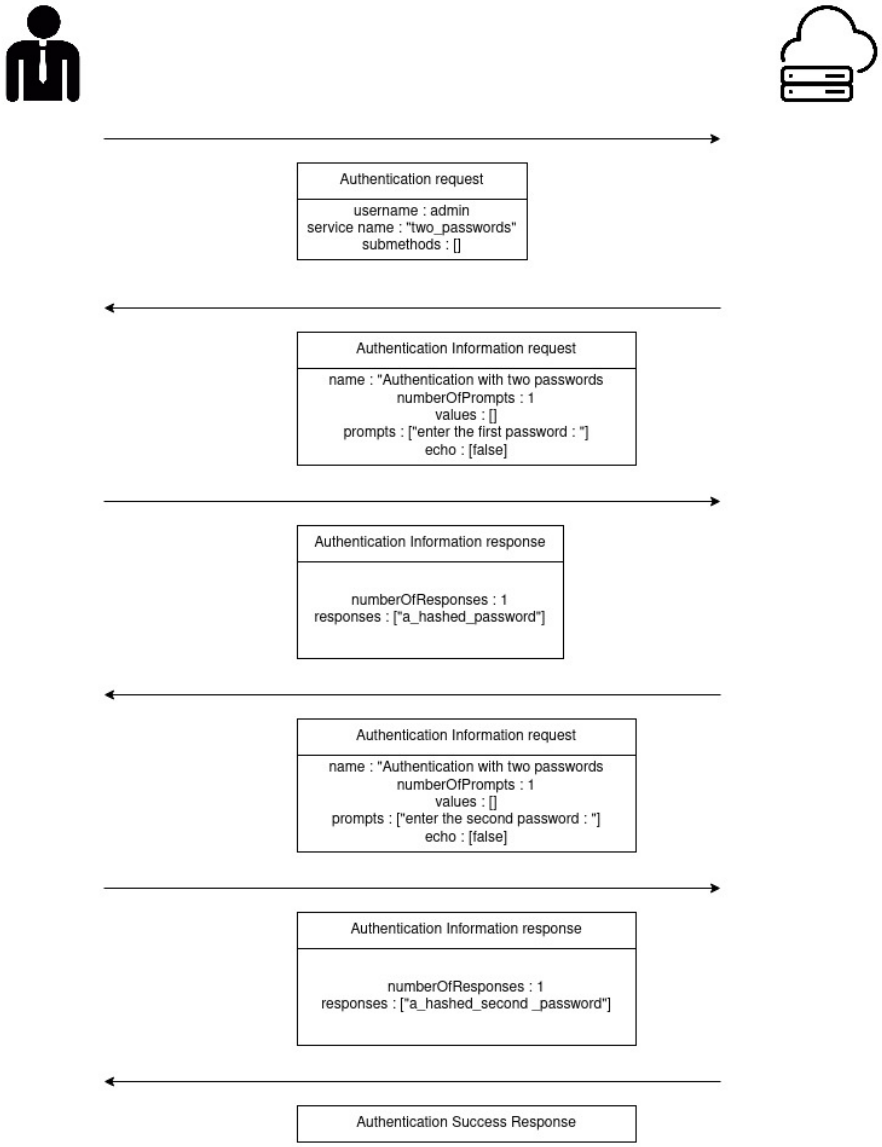



Figure 6-1: Authentication process for a two password mechanism

success or a failure message, or an Authentication Information Request object to carry on the authentication process for another step.

### 6.1.2 Key pair authentication

Using the authentication method defined in the previous section, SM implements an authentication mechanism based on key-pairs, where the user shares their public key with the server and use their private key to prove their identity.

The client starts by sending an authentication request with the service name "key_pair_RSA_SHA256" and their username, the server then checks their user name and pulls the user's public key if there exists one. It then generates a random 64 byte string that it sends to the client with the values field in the Authentication Information Request and stores in the session information.

The client then asks the user for their private key file, which is used to digitally sign the string and sends the signed string back in the Authentication Information Response. The server then uses its public key to verify that the sent signature is of the generated key, if that was the case, a success message is sent to the client.

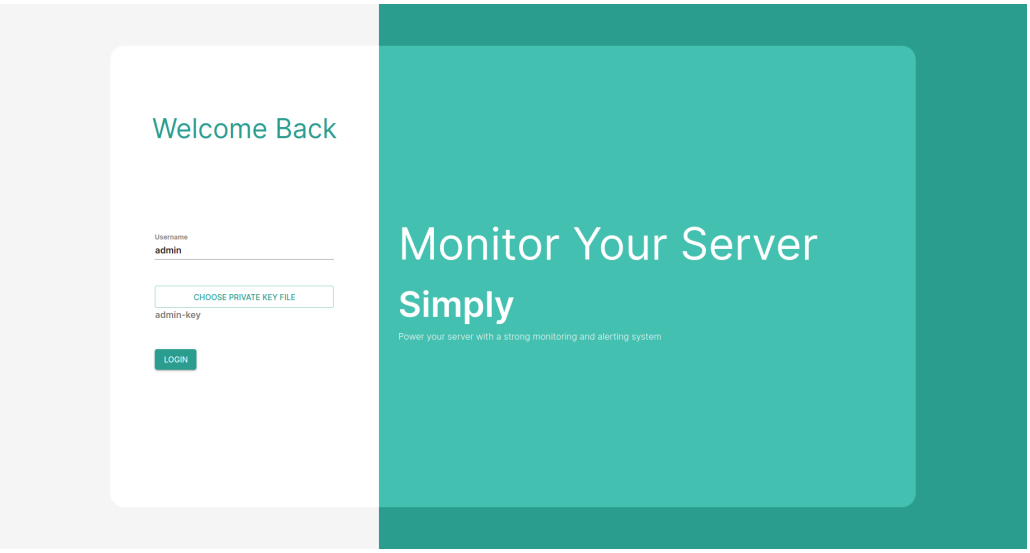



Figure 6-2: Login page using key pair authentication

Figure 6-2 shows how the user can authenticate themselves in the client, they specify their username and choose their private key file, the client then takes over from there and does the authentication process.

This authentication system assumes that the information being transferred is confidential and encrypted, that is why the system should only be ran on TLS or SSL connections in production, otherwise, people would be able to intercept the communications and maybe use them in later authentication method to fake their identities.

## 6.2 One time passwords

The Command Chains module implements the ability to protect executing certain chains with a one-time password mechanism, where to be able to run the chain, the user has to enter a one-time password

sent to their mobile phones. These passwords (as the name implies) are unique and can only be used once, every time the user wants to run the protected chain, a different password is generated and sent to the user. This level of protection is also added to the scripts that are ran with SUDO privileges.

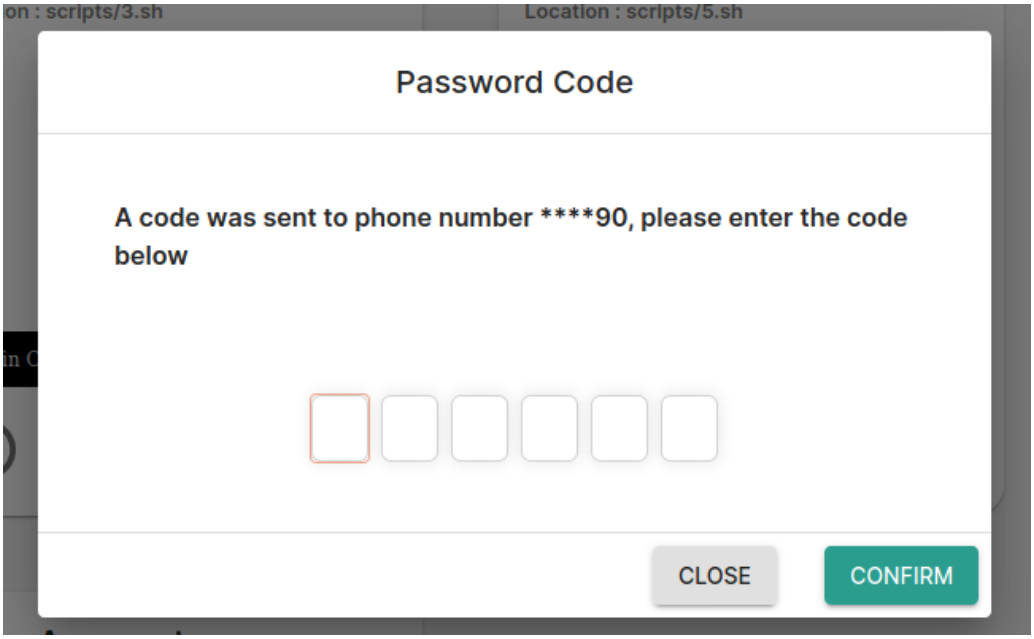



Figure 6-3: One-time password dialogue

Figure 6-3 shows the client's password dialogue that appears to the use when attempting to either run a protected chain, or a chain with SUDO privilege.

## 6.3 JSON Web Tokens

JSON Web Token (JWT) is an open standard which defines a way for transmitting data securely between two parties as a JSON object. This object is signed in the server using a secret-key with HMAC or RSA algorithm. SM uses this method in its authentication system, to ensure that a specific user is logged in by verifying that a user detailed object stored in the browser cookies is signed by SM's secret-key .

# Chapter 7 Configuration

One of SM's main features that it provides to its users is customizability, which is made possible by allowing them to implement their own front-end client. And giving them more control over what SM can and cannot do and some of its configuration parameter using an external configuration file they can access and edit.

The file named "SM_conf.toml" is a configuration file with the TOML format, which is a file format meant to be highly readable and have obvious semantics [8].

The configuration file looks like this: -

```
# This section configures which metrics to track, it allows you to enable and disable storing measurements, retrieving stored measurements,
# subscribing and retrieving component related data to these different metrics


[Tracking]

# A list of all the tracked metrics
# To disable the tracking of any of them just remove the metric from the list
# eg. to enable tracking of Disk and Memory while disabling it for anything else, the list would be:-
#   tracked-metrics = [
#         "Memory",
#         "Disk",
#   ]
tracked-metrics = [
    "CPU",
    "Memory",
    "Disk",
    "System",
    "Docker",
    "Network"
]

# The time interval between publishing metric's data
# Unit is ms
publish-subscription-data-every = 2000

# The time interval between taking samples of metric's data to then store it, decreasing the time between samples doesn't necessarily increase
# the number of samples stored due to the values being aggregated for fixed time buckets
# Unit is ms
sample-data-every = 10000
```

```
31
32
33 # This section allows you to control the alerting subsystem
34 [Alerts]
35
36 # Enables the alerting subsystem, if this flag is set to false, no alert reporting or
       checking would be performed by the system
37 enable-alerts = true
38 # Enables the use of dynamic alerts, which uses a model of the system to determine if
       a certain value is an anomaly, if this option is disabled and
39 # the scheduler is set to run at a certain time by setting the run-on parameters, the
       adaptive analysis task is disabled
40 # The enable-alerts value has to be set to true for this to take effect
41 enable-dynamic-alerts = true
42
43 # The notifcation API secret key
44 push-notification-API-key = ""
45
46 # Notification API instance ID
47 push-notification-instance-ID = ""
48
49 push-notification-domain = "localhost:3006"
50 # The time interval between checks performed by the system to check if the metric's
       data requires an alert triggering, the value picked is the average
51 # of the values in the last n milliseconds, where n is the value specified below
52 # Unit is in ms
53 # The enable-alerts value has to be set to true for this to take effect
54 check-alert-every = 60000
55
56 # This section allows you to control the command chain subsystem
57 [Command_Chains]
58
59 # Enables the command chain subsystem, if this flag is set to false, no chain would
       be stored or executed
60 enable-chains = true
61
62 # Enables the option of running certain scripts under SUDO privilege
63 # The enable-chains value has to be set to true for this to take effect
64 enable-chains-sudo-execution = true
65
66 # Where to store the scripts, if the system isn't running under SUDO, the specified
       directory should have read, write and execute
67 # permission for the woof user
68 scripts-dir = "./scripts"
69
70 # API key for the one time password used in running SUDO and protected chains
71 one-time-password-API-key = ""
72
73
```

```
# This sections is used to enable the tracking of sites implemented via Nginx, it also specifies some parameters used to
# analyze and find data related to the site
[Nginx]

# Allow the tracking of Nginx web-service specific data
track-nginx = true

# The configuration file of the tracked site
site-location = "/etc/nginx/sites-available/sombrero_project"

# The location of the GeoIP database used to analyze the demographic of the site's traffic, if the value was set to null,
# the demographic analysis would be disabled
GeoIP-database-location = "./src/Nginx/GeoLite2-Country_20210309/GeoLite2-Country.mmdb"


# The schedular runs tasks such as parsing Nginx logs and finding the optimal restart time and the optimal schedular task runtime
[Scheduler]

# The schedular runs the analysis tasks on the times that tend to have low CPU usage and schedules system reboots


# Allow the user to perform a system restart
allow-restart = true

# This section is to define Redis parameters
[Redis]

# Redis IP address
redis-IP = "127.0.0.1"

# Redis port number
redis-port = 6379
```

As shown above, the file is filled with comments describing what each field does and how to use it.

The configuration file allows for users to control many aspects of the system, such as turning on and off some features that they don't want like alerting and command chains, choosing what information they want the system to collect limiting its access to the server's data, controlling the time intervals for some operations like sampling and alert checks, and defining some parameters required for some of the system's modules to operate like the Nginx site configuration file location and some API keys.

# Chapter 8 Conclusion

Put simply, Server Monitor (SM), is a monitoring tool designed to help cloud users -both experienced and new- to easily monitor their machines, understand its performance, be alerted in case of any events deemed important by them, automate many of their administrative tasks and other features, creating a system that simplifies their cloud-computing experience and allows them to utilize it to the fullest.

SM uses different state-of-the-art technologies to create a light-weight and user-friendly application. These technologies include GraphQL, React and Redis.

The system was designed to make it easily extensible by providing a separated back and front-ends, allowing users to either use the provided front-end or implement their own GraphQL client.

SM provides the users with different functionalities, from collecting different data samples, storing them and allowing users to retrieve and subscribe to them, to providing a smart alerting system that notifies them for different events of the system. Other functionalities include parsing and analyzing web server's access logs to extract different information about the website's traffic if the server was used to host a website, a command chain sub-system that allows them to create chains of commands that they can access easily in one place.

On top of these features and functionalities, to make sure that the information sent by SM is accessed only by authorized personnel, SM implements an authentication system that allows users to create their own authentication methods while providing a built-in key-pair authentication method. Some other features of SM is an internal task scheduler and monitoring capabilities on the Docker level.

## 8.1 Future Work

Although SM attempts to create a full monitoring suite with its provided features, there are still some work that has to be done for it be a truly full and simple monitoring tool.

- Having the application run on the container-orchestration systems level with technologies like kubernetes.

- Extend the log parsing capabilities of the Web Server Parser module by extracting more useful data from the logs like request processing time and average body size per endpoint.
- Automate and simplify the installation process of the application to make easier to use for less experienced users.
- Implement some of the CPU-intensive tasks like data modelling and log parsing in languages that have better performance and are generally faster than Javascript like Go or Rust.

# References


[1] G. Aceto, A. Botta, W. De Donato, and A. Pescapè, “Cloud monitoring: A survey,” *Computer Networks*, vol. 57, no. 9, pp. 2093–2115, 2013.

[2] *IT system monitoring and troubleshooting*. [Online]. Available: `https://www.rapid7.com/fundamentals/system-monitoring-and-troubleshooting/`.

[3] A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, *et al.*, “Above the clouds: A berkeley view of cloud computing,” *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, vol. 28, no. 13, p. 2009, 2009.

[4] [Online]. Available: `https://graphql.org/`.

[5] J. P. Buzen and A. W. Shum, “Masf-multivariate adaptive statistical filtering,” in *Int. CMG Conference*, 1995, pp. 1–10.

[6] *Anomaly detection with the normal distribution*, Dec. 2018. [Online]. Available: `https://anomaly.io/anomaly-detection-normal-distribution/index.html`.

[7] F. Cusack and M. Forssen, “Generic message exchange authentication for the secure shell protocol (SSH),” RFC 4256, January, Tech. Rep., 2006.

[8] [Online]. Available: `https://toml.io/en/`.