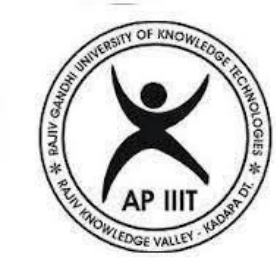
TRUEPUSH

A project report submitted in partial fulfilment of the requirement for degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING



Rajiv Gandhi University of Knowledge Technologies

R.K.VALLEY

Submitted by

G Mogileeswar Reddy– R170177

Under the guidance of

Internal Guide

E Susmitha

Assistant Professor

RGUKT RK Valley.

External Guide

Surya Narayana Raju G

Principal Lead Software Engineer

Way2Online Interactive India Private Limited

RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES

(**A.P.Government Act 18 of 2008)**

RGUKT-RK Valley

Vempalli, Kadapa, Andhrapradesh-516330.

CERTIFICATE OF PROJECT COMPLETION

This is to certify that the report entitled "**TRUEPUSH**" submitted by **Mogileeswar Reddy Gundluri(R170177)** partial fulfilment of the requirements for the award of Bachelor of Technology in Computer Science and Engineering is a bonafide work carried out by them under my supervision and guidance.

The report has not been submitted previously in part or in full to this or any other University or Institution for the award of any degree or diploma.

Project Guide

Ms. E. Susmitha,

Asst.Prof. in Dept of CSE

RGUKT-RK Valley.

Head of the Department

Mr. N. Satyanandaram,

Lecturer in Dept of CSE,

RGUKT-RK Valley.

RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES

(A.P.Government Act 18 of 2008)

RGUKT-RK Valley

Vempalli, Kadapa, Andhrapradesh-516330.

DECLARATION

We are certifying that, I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Signature of the student

G Mogileeswar Reddy

R170177

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of the people who made it possible and whose constant guidance and encouragement crown all the efforts success.

I am extremely grateful to our respected Director, **Prof. K. SANDHYA RANI** for fostering an excellent academic climate in our institution.

I also express my sincere gratitude to our respected Head of the Department **Mr. N. Satyanandaram** for his encouragement, overall guidance in viewing this project a good asset and effort in bringing out this project.

I would like to convey thanks to our guide at college **Ms. E. SUSMITHA** for her guidance, encouragement, co-operation and kindness during the entire duration of the course and academics.

My sincere thanks to all the members who helped me directly and indirectly in the completion of project work. I express my profound gratitude to all our friends and family members for their encouragement.

TABLE OF CONTENTS

ABSTRACT

The aim of this project is to develop a Web Push Notification SaaS tool that enables website owners to easily and effectively communicate with their users, increase engagement, and drive traffic to their websites.
The tool will have the following features:

1. Customizable Opt-In: The tool will provide a customizable opt-in prompt that will enable website visitors to subscribe to push notifications. The opt-in prompt will be designed to be unobtrusive, and website visitors will have the option to opt-out at any time.

2. Segmentation and Targeting: The tool will provide website owners with the ability to segment and target their push notifications based on user behavior, demographics, and interests. This will enable website owners to deliver more personalized and relevant notifications to their users, resulting in increased engagement.

3. Automated Campaigns: The tool will allow website owners to create automated push notification campaigns based on specific triggers such as user behavior, time of day, or location. This will enable website owners to send timely and relevant notifications to their users without the need for manual intervention.

4. Reporting and Analytics: The tool will provide website owners with detailed reporting and analytics on the performance of their push notification campaigns. This will enable website owners to make data-driven decisions to optimize their campaigns and improve their results.

5. Integration: The tool will integrate seamlessly with popular content management systems (CMS) such as WordPress, Drupal, and Joomla, as well as popular e-commerce platforms such as Shopify and Magento.

The Web Push Notification SaaS tool will be built using modern web development technologies such as React, Node.js, and MongoDB. The tool will be deployed on a scalabl cloud infrastructure to ensure high availability and performance.

Overall, the Web Push Notification SaaS tool will provide website owners with an easy-to-use and effective solution to engage with their users, increase traffic, and drive business results.

INTRODUCTION:

Truepush is a free and monetization Push Notifications tool for both web and mobile. Started off with a team of five, Truepush was launched in January 2019. From trending on the ProductHunt as the “Product of the day” to acquiring 150% of MoM growth, Truepush has received tremendous applause for its remarkable success. Truepush is equipped with rich features like RSS-to-push, Audience segmentation, Triggers, Project Duplication, etc. that are generally paid on other competitive platforms, thus helping brands re-engage with their users for free.

This exceptional growth is achieved due to the product’s capability and rockstar customer support. With more advanced features and plugins coming in the future, the Truepush team is now on the mission to build the most preferred Push Notification tool in the world with the capability of handling more than 1 Billion notifications a day.

HARDWARE REQUIREMENTS

- Processor : Intel Core i5
- HardDisk : 500 GB
- RAM : 16 GB
- Monitor: 15’’ LED

SOFTWARE REQUIREMENTS

- Operating System : Windows 10
- Tool : Visual Studio Code
- Deployment : Google Cloud Platform(GCP)
- Database : MongoDB
- Coding Language : Node.js, Express.js, Angular.js, HTML, CSS,JavaScript, Bootstrap

TECH STACK:

- **NODEJS**
- **ANGULAR**
- **REDIS**
- **RABBIT MQ**
- **MONGODB**

Node JS:

Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser. Node.js lets developers use JavaScript to write command line tools and for server-side scripting running scripts server-side to produce dynamic web page content before the page is sent to the user's web browser. Consequently, Node.js represents a "JavaScript everywhere" paradigm unifying web-application development around a single programming language, rather than different languages for server-side and client-side scripts.

MongoDB:

MongoDB is a document-oriented NoSQL database used for high volume data storage. Instead of using tables and rows as in the traditional relational databases, MongoDB makes use of collections and documents. Documents consist of key-value pairs which are the basic unit of data in MongoDB. Collections contain sets of documents and function which is the equivalent of relational database tables.

Collections → Table

Documents → Rows

Angular:

Angular is a development platform, built on TypeScript. As a platform, Angular includes A component-based framework for building scalable web applications. A collection of well-integrated libraries that cover a wide variety of features, including routing, forms management, client-server communication, and more. A suite of developer tools to help you develop, build, test, and update your code

With Angular, you're taking advantage of a platform that can scale from single-developer projects to enterprise-level applications. Angular is designed to make updating as straightforward as possible, so take advantage of the latest developments with minimal effort.

RabbitMQ:

RabbitMQ is the most widely deployed open source message broker. With tens of thousands of users, RabbitMQ is one of the most popular open source message brokers. From T-Mobile to Runtastic, RabbitMQ is used worldwide at small startups and large enterprises. RabbitMQ is lightweight and easy to deploy on premises and in the cloud. It supports multiple messaging protocols. RabbitMQ can be deployed in distributed and federated configurations to meet high-scale, high-availability requirements. RabbitMQ runs on many operating systems and cloud environments, and provides a wide range of developer tools for most popular languages.

Redis:

Redis is an open source (BSD licensed), in-memory data structure store used as a database, cache, message broker, and streaming engine. Redis provides data structuressuch as strings, hashes, lists, sets, **sorted sets** with range queries, bitmaps, hyperloglogs, geospatial indexes, and streams. Redis has built-in replication, Lua scripting, LRU eviction, transactions, and different levels of on-disk persistence, and provides high availability via Redis sentinels and automatic partitioning with Redis Cluster.

CONTRIBUTION TO TRUEPUSH:

- Introduced Clustering and Asynchronization in Backend for sending the notifications(changes)
- Servers Health Monitoring

Backend for sending the notifications :

- The TruePush tool need to push 1 Billion notification It is very important for the Backend to be more scalable and available and efficiently.
- Scalability is managed manually which is causing massive overhead and decreasing the performance eventually increasing the number of servers to handle the work.
- Developed a efficient process that scales easily and handle mass notification sending at one centrally managed process.
- A **cluster** of processes is created and managed by the Central process called Campaigner.

Asynchronization :

When you execute something asynchronously, you can move on to another task before it finishes.

Asynchronous processes, on the other hand, do not have their start and endpoints

synchronized:

```
ASYNCHRONOUS


 |--------A--------|


      |--------B--------|
```

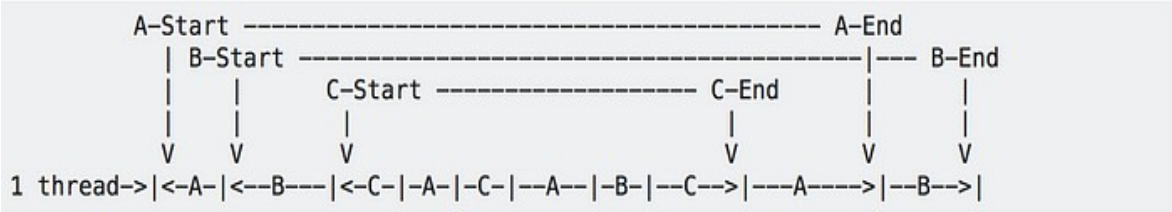
Where Process A overlaps Process B, they're running concurrently or synchronously (dictionary definition).

When a task is executed asynchronously, you can directly switch to another task before the previous has been completed. One task does not depend on the other.

In programming, tasks are executed on a "*thread*". It is necessary to understand that several threads can be created for tasks in order to avoid stacking in a single queue.

That being said, the concept of synchronous and asynchronous do not have much to do with threads, the reason I mention it is that it is rare to see tasks running asynchronously on a single thread (although technically possible).

Asynchronous (one thread):



Asynchronous (multi-Threaded):

Start and end points of tasks A, B, C represented by `<`, `>` characters.

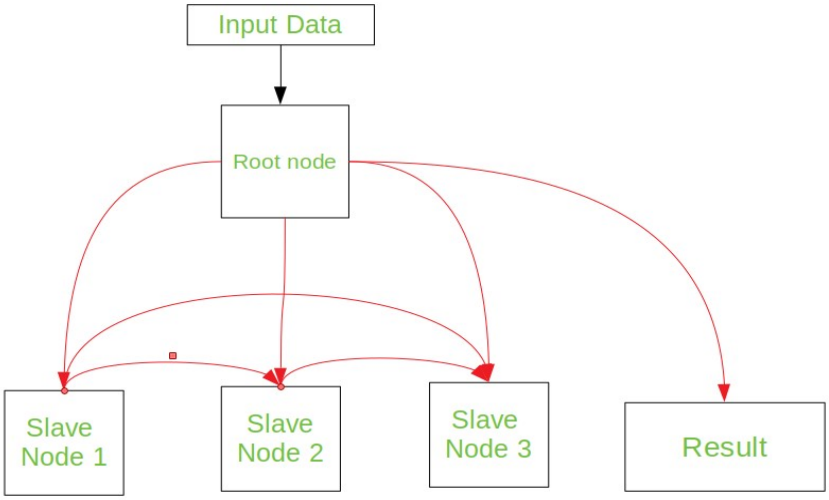
CPU time slices represented by vertical bars `|`

Clustering:

A cluster is a set of loosely or tightly connected computers working together as a unified computing resource that can create the illusion of being one machine. Computer clusters have

each node set to perform the same task, controlled and produced by the software.
Clustered Operating Systems work similarly to Parallel Operating Systems as they have many CPUs. Cluster systems are created when two or more computer systems are merged. Basically, they have an independent computer but have common storage and the systems work together.

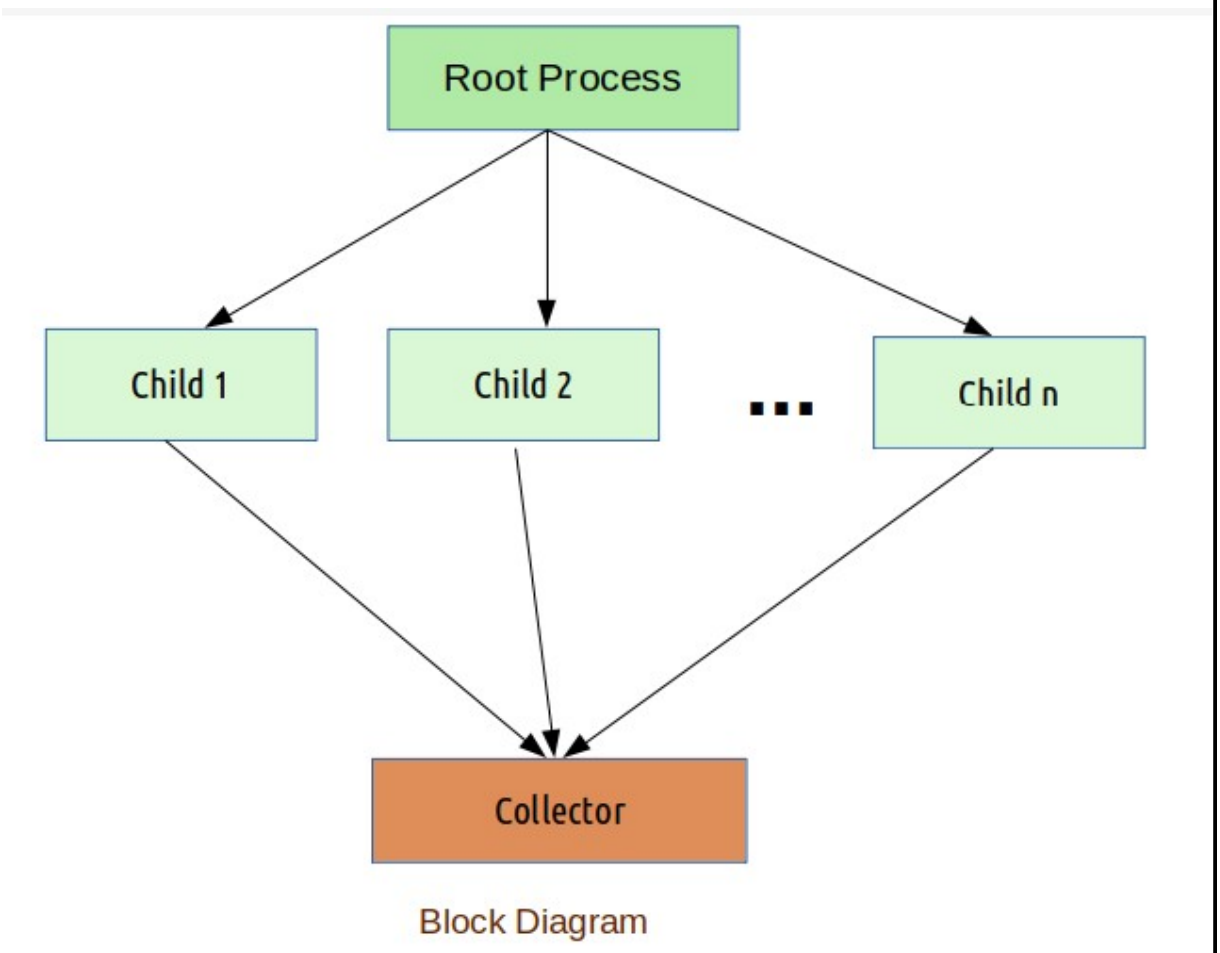
The *components of* clusters are usually connected using fast area networks, with each node running its own instance of an operating system. In most circumstances, all the nodes use the same hardware and the same operating system, although in some setups different hardware or different operating systems can be used in some setups.



A cluster requires an effective capability for balancing the load among available computers. In this, cluster nodes share a computational workload to enhance the overall performance. For example- a high-performance cluster used for scientific calculation would balance the load from different algorithms from the web-server cluster, which may just use a round-robin method by assigning each new request to a different node. This type of cluster is used on farms of Web servers (web farm).

Clusters are usually set up to improve performance and availability over single computers, while typically being much more cost-effective than single computers of comparable speed or availability.

How Clustering integrated in TRUEPUSH:



Block Diagram

Root Node :

- Responsible for getting a chunk of data and assign chunk of subscribers data to a child process.
- Scaling for number of children and maximum children capacity is controlled here.
- Responsible to control the cluster of processes.

Childs:

- Responsible for doing the actual work and generate the intermediary results for its given chunk of input.

- Child process send a given set of subscribers data and send a push notification for its set of subscribers.

Collector:

- Reduce all the Intermediate results into a single Stream of output that handles the integrated output
- Communication between children and collector is handled with IPC (Inter Process Communication).

IPC (Inter Process Communication):

In general, Inter Process Communication is a type of mechanism usually provided by the operating system (or OS). The main aim or goal of this mechanism is to provide communications in between several processes. In short, the intercommunication allows a process letting another process know that some event has occurred.

Sockets:

Just as pipes come in two flavors (named and unnamed), so do sockets. IPC sockets (aka Unix domain sockets) enable channel-based communication for processes on the same physical device (*host*), whereas network sockets enable this kind of IPC for processes that can run on different hosts, thereby bringing networking into play. Network sockets need support from an underlying protocol such as TCP (Transmission Control Protocol) or the lower-level UDP (User Datagram Protocol).
By contrast, IPC sockets rely upon the local system kernel to support communication; in particular, IPC sockets communicate using a local file as a socket address. Despite these implementation differences, the IPC socket and network socket APIs are the same in the essentials. The forthcoming example covers network sockets, but the sample server and client programs can run on the same machine because the server uses network address l*ocalhost* (127.0.0.1), the address for the local machine on the local machine.
Sockets configured as streams (discussed below) are bidirectional, and control follows a client/server pattern: the client initiates the conversation by trying to connect to a server, which tries to accept the connection. If everything works, requests from the client and responses from the server then can flow through the channel until this is closed on either end, thereby breaking the connection.

An *iterative* server, which is suited for development only, handles connected clients one at a time to completion: the first client is handled from start to finish, then the second, and so on. The downside is that the handling of a particular client may hang, which then starves all the clients waiting behind. A production-grade server would be *concurrent*, typically using some mix of multi-processing and multi-threading. For example, the Nginx web server on my desktop machine has a pool of four worker processes that can handle client requests concurrently. The following code example keeps the clutter to a minimum by using an iterative server; the focus thus remains on the basic API, not on concurrency.
Finally, the socket API has evolved significantly over time as various POSIX refinements have emerged. The current sample code for server and client is deliberately simple but

underscores the bidirectional aspect of a stream-based socket connection. Here's a summary of the flow of control, with the server started in a terminal then the client started in a separate terminal:

- The server awaits client connections and, given a successful connection, reads the bytes from the client.
- To underscore the two-way conversation, the server echoes back to the client the bytes received from the client. These bytes are ASCII character codes, which make up book titles.
- The client writes book titles to the server process and then reads the same titles echoed from the server. Both the server and the client print the titles to the screen. Here is the server's output, essentially the same as the client's:

Sample Socket:

```
Listening on port 9876 for clients...
```

Higher abstraction code for socketing:

- The socket server
- The socket client

The socket server:

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/tcp.h>
#include <arpa/inet.h>
#include "sock.h"
void report(const char* msg, int terminate) {
  perror(msg);
  if (terminate) exit(-1); /* failure */
}
int main() {
  int fd = socket(AF_INET,     /* network versus AF_LOCAL */
                  SOCK_STREAM, /* reliable, bidirectional,
arbitrary payload size */
```

```
                    0);           /* system picks underlying
protocol (TCP) */
  if (fd < 0) report("socket", 1); /* terminate */
  /* bind the server's local address in memory */
  struct sockaddr_in saddr;
  memset(&saddr, 0, sizeof(saddr));             /* clear the
bytes */
   saddr.sin_family = AF_INET;                  /* versus
AF_LOCAL */
    saddr.sin_addr.s_addr = htonl(INADDR_ANY); /*    host-to-
network endian */
  saddr.sin_port = htons(PortNumber);         /* for listening
*/
  if (bind(fd, (struct sockaddr *) &saddr, sizeof(saddr)) < 0
)
    report("bind", 1); /* terminate */
  /* listen to the socket */
  if (listen(fd, MaxConnects) < 0) /* listen for clients, up
to MaxConnects */
    report("listen", 1); /* terminate */
  fprintf(stderr, "Listening on port %i for clients...\
n", PortNumber);
  /* a server traditionally listens indefinitely */
  while (1) {
    struct sockaddr_in caddr; /* client address */
    int len = sizeof(caddr);  /* address length could change
*/

  int client_fd = accept(fd, (struct sockaddr*) &caddr, &len)
;  /* accept blocks */
    if (client_fd < 0) {
```

```
      report("accept", 0); /* don't terminate, though there's
a problem */
      continue;
    }
    /* read from client */
    int i;
    for (i = 0; i < ConversationLen; i++) {
      char buffer[BuffSize + 1];
      memset(buffer, '\0', sizeof(buffer));
      int count = read(client_fd, buffer, sizeof(buffer));
      if (count > 0) {
        puts(buffer);
        write(client_fd, buffer, sizeof(buffer)); /* echo as
confirmation */
      }
    }
    close(client_fd); /* break connection */
  }  /* while(1) */
  return 0;
}
```

The socket client:

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
```

```c
#include <arpa/inet.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <netdb.h>
#include "sock.h"
const char* books[] = {"War and Peace",
                       "Pride and Prejudice",
                       "The Sound and the Fury"};
void report(const char* msg, int terminate) {
  perror(msg);
  if (terminate) exit(-1); /* failure */
}
int main() {
  /* fd for the socket */
  int sockfd = socket(AF_INET,      /* versus AF_LOCAL */
                      SOCK_STREAM,  /*  reliable,
bidirectional */
                      0);           /* system picks protocol
(TCP) */
  if (sockfd < 0) report("socket", 1); /* terminate */
  /* get the address of the host */
  struct hostent* hptr = gethostbyname(Host); /*    localhost:
127.0.0.1 */
  if (!hptr) report("gethostbyname", 1); /* is hptr NULL? */
  if (hptr->h_addrtype != AF_INET)         /*  versus  AF_LOCAL
*/
    report("bad address family", 1);
  /* connect to the server: configure server's address 1st */
  struct sockaddr_in saddr;
  memset(&saddr, 0, sizeof(saddr));
  saddr.sin_family = AF_INET;
  saddr.sin_addr.s_addr =
```

```
    ((struct in_addr*) hptr->h_addr_list[0])->s_addr;
  saddr.sin_port = htons(PortNumber); /* port number in big-
endian */
  if (connect(sockfd, (struct sockaddr*) &saddr, sizeof(saddr
)) < 0)
    report("connect", 1);
  /* Write some stuff and read the echoes. */
  puts("Connect to server, about to write some stuff...");
  int i;
  for (i = 0; i < ConversationLen; i++) {
    if (write(sockfd, books[i], strlen(books[i])) > 0) {
      /* get confirmation echoed from server and print */
      char buffer[BuffSize + 1];
      memset(buffer, '\0', sizeof(buffer));
      if (read(sockfd, buffer, sizeof(buffer)) > 0)
        puts(buffer);
    }
  }
  puts("Client done, about to exit...");
  close(sockfd); /* close the connection */
  return 0;
}
```

Servers Health Monitoring :

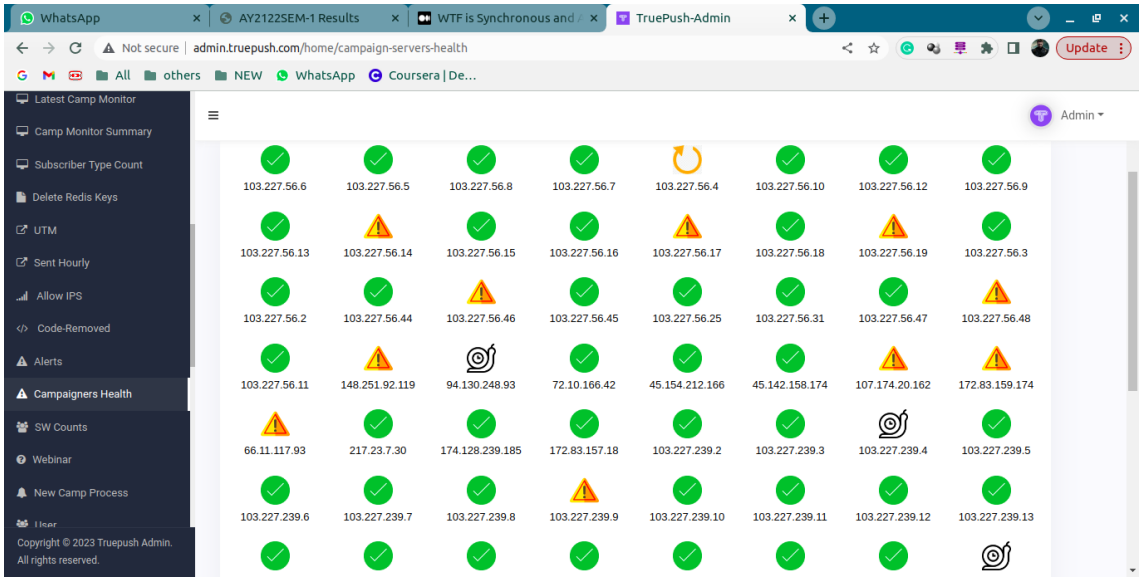
A dashboard for monitoring all the office servers and implementing an alert system for server blackouts. This work is an essential part of ensuring the health and reliability of the IT infrastructure of your organization.

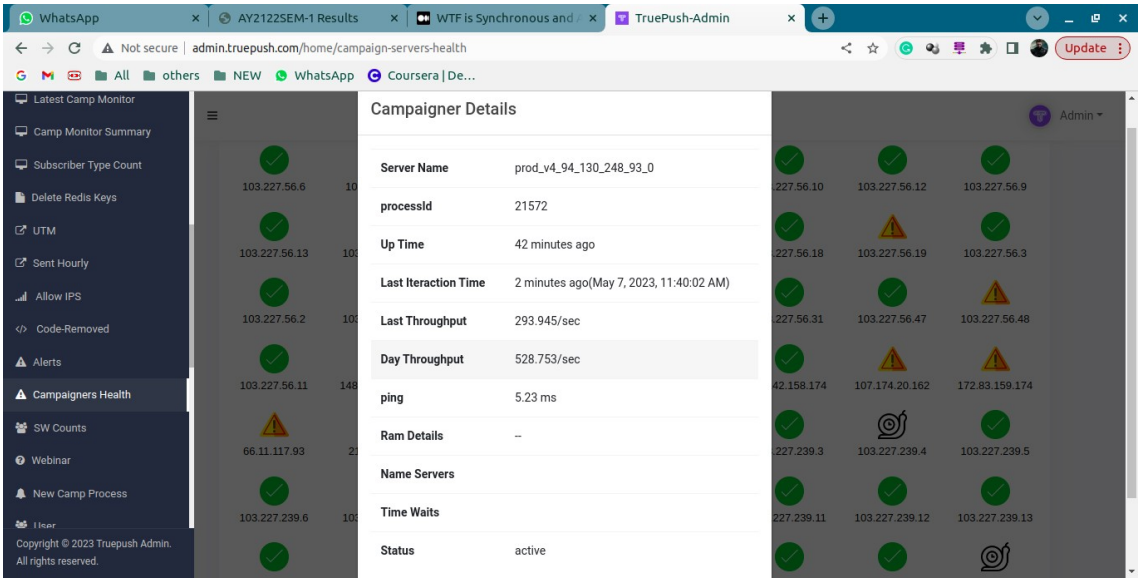
Dashboard is a critical tool for monitoring the health of the servers, and it provides real-time data that enables you to quickly identify and resolve any issues. The dashboard provides a comprehensive overview of the servers' status, including server uptime, CPU usage, memory usage, disk usage, and network traffic. You can also view historical data and trends, which

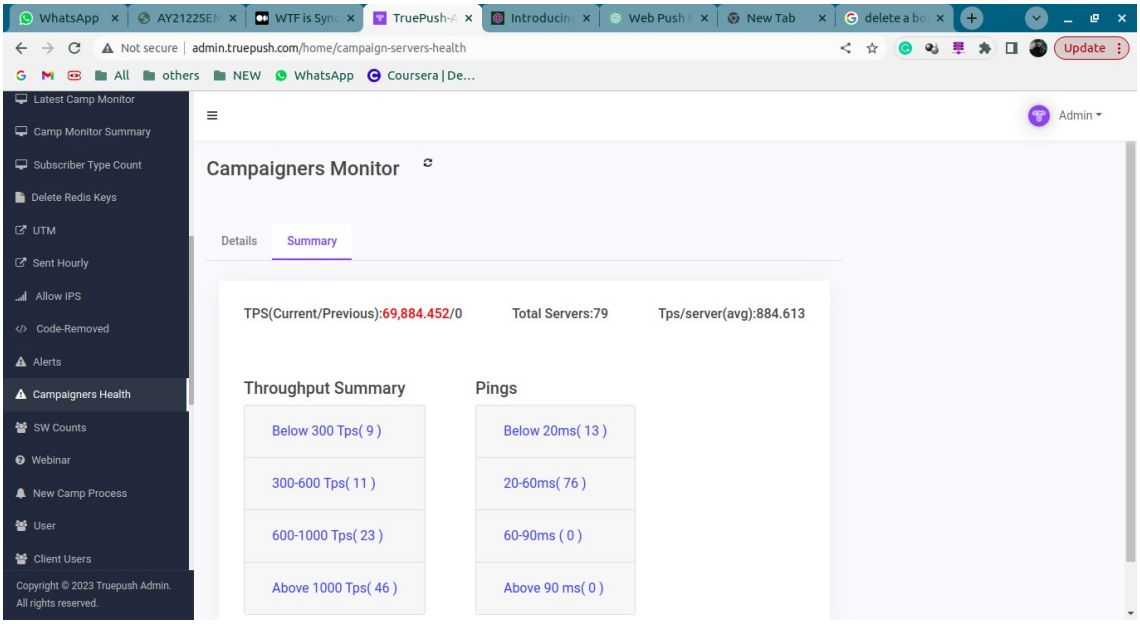
enables you to identify patterns and proactively address any potential issues.

The alert system that you have implemented is also a critical feature of your dashboard. The alert system enables you to receive real-time notifications of any issues or anomalies, which allows you to quickly address them before they become critical problems. The alert system is configurable, and you can set up different alerts for different servers or specific metrics. Your work has greatly improved the IT infrastructure's reliability and has enabled the organization to avoid server blackouts that could have resulted in costly downtime and lost productivity. The dashboard provides a comprehensive view of the organization's server infrastructure and enables the IT team to make data-driven decisions to optimize the servers' performance.

Overall, my work has been instrumental in ensuring the health and reliability of the organization's IT infrastructure. The dashboard and alert system have greatly improved the IT team's ability to monitor and maintain the servers, and they are critical tools for ensuring that the servers remain up and running. My work is an excellent example of how technology can be used to improve organizational efficiency and productivity.




WhatsApp
AY2122SEM-1 Results
WTF is Synchronous and
TruePush-Admin
Not secure | admin.truepush.com/home/campaign-servers-health
Update
All
others
NEW
WhatsApp
Coursera | De...
Latest Camp Monitor
Camp Monitor Summary
Subscriber Type Count
Delete Redis Keys
UTM
Sent Hourly
Allow IPS
Code-Removed
Alerts
Campaigners Health
SW Counts
Webinar
New Camp Process
User
Copyright © 2023 Truepush Admin.
All rights reserved.
Campaigner Details
Server Name
prod_v4_94_130_248_93_0
processId
21572
Up Time
42 minutes ago
Last Iteraction Time
2 minutes ago(May 7, 2023, 11:40:02 AM)
Last Throughput
293.945/sec
Day Throughput
528.753/sec
ping
5.23 ms
Ram Details
--
Name Servers
Time Waits
Status
active
Admin


WhatsApp
TruePush-A
Introducing
Web Push
New Tab
delete a bo
Not secure | admin.truepush.com/home/campaign-servers-health
Update
All
others
NEW
WhatsApp
Coursera | De...
Latest Camp Monitor
Camp Monitor Summary
Subscriber Type Count
Delete Redis Keys
UTM
Sent Hourly
Allow IPS
Code-Removed
Alerts
Campaigners Health
SW Counts
Webinar
New Camp Process
User
Client Users
Copyright © 2023 Truepush Admin.
All rights reserved.
Admin
Campaigners Monitor
Details
Summary
TPS(Current/Previous):69,884.452/0
Total Servers:79
Tps/server(avg):884.613
Throughput Summary
Below 300 Tps(9)
300-600 Tps(11)
600-1000 Tps(23)
Above 1000 Tps(46)
Pings
Below 20ms(13)
20-60ms(76)
60-90ms (0)
Above 90 ms(0)

IMPACT :

- Changes made to call API of fcm.google.com from https to http decreasing the network bandwidth by **70 percentage.**
- With the integration of clustering in campaigner processes to send notification the performance and Throughput of each server increased by **3X times** compared with no clustering.
- Because of increased Throughput the number of Servers to handle the Campaigner is expecting to decrease from 70 to 40.
- Created dashboard for servers health monitoring helps in less blackout.

CONCLUSION:

Clustering ,Load Balancing and Scalability are the pillars of a system that handles high volume of data intensive operations. Clusters are primarily designed with performance in mind, but installations are based on many other factors. Fault tolerance allows for scalability , and in high-performance situations, low frequency of maintenance routines, resource consolidation, and centralized management. Advantages include enabling data recovery in the event of a disaster and providing parallel data processing and high processing capacity. In terms of scalability, clusters provide this in their ability to add nodes horizontally. This means that more computers may be added to the cluster, to improve its performance, redundancy and fault tolerance. This can be an inexpensive solution for a higher performing cluster compared to scaling up a single node in the cluster. This property of computer clusters can allow for larger computational loads to be executed by a larger number of lower performing computers.With the motto of serving websites to generate higher ROI, TruePush provide all advanced features for our clients with the Truepush free push service.Websites get access to features like audience segmentation, RSS to push, triggers, schedule notifications, opt-in styles.

References:

https://nodejs.dev/en/learn/

https://www.javatpoint.com/expressjs-tutorial

https://docs.angularjs.org/tutorial

https://www.w3schools.com/mongodb/

https://www.w3schools.com/js/