The Report committee for Moshiul Arefin

Certifies that this is the approved version of the following report:

KRISEM: A Cloud Solution for Service Watchdog

APPROVED BY

SUPERVISING COMMITTEE:

Supervisor:

Adnan Aziz

Kathleen Suzanne Barber

KRISEM: A Cloud Solution for Service Watchdog

by

Moshiul Arefin, B.S.E.E.

REPORT

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ENGINEERING

The University of Texas at Austin

May 2015

Dedication

I would like to dedicate this report to my family.

My parents, Abdul Halim and Hamida Begum, for all their love, encouragement, and guidance throughout the years.

My wife, Nazia, for her love, support, and incredible patience as I went through the Masters program while working full-time.

My son, Rayyan, so he would know, although I was so busy for so long, that there was a good reason; it was for him as well, not just for me.

Acknowledgments

I would like to acknowledge my supervisor, Dr. Adnan Aziz, for the support and guidance that he provided to make this project possible. I would also like to thank Dr. Suzanne Barber for reading and providing feedback on this report.

I would like to give my gratitude to Brian Morrow, who has given me his precious critique and proofreading.

I would also like to thank Youngsang Kim for reading and providing feedback on this report.

Abstract

KRISEM: A Cloud Solution for Service Watchdog

Moshiul Arefin, M.S.E.

The University of Texas at Austin, 2015

Supervisor: Adnan Aziz

As companies continue to grow and expand, Information Technology professionals must constantly ensure the integrity and availability of vital automated services (e.g., HTTP, air traffic control systems etc.). To ensure availability, key automated services needed to keep the organization operational must be monitored continuously. Factors such as program failure and operating system maintenance can cause these services to crash. Services need to be restarted immediately to keep the business operational. There are a variety of automated service monitoring solutions available, but those with universal applicability are of particular interest. The drawbacks of these systems have been their resource limitations as well as the need for complex configuration to make their outage data available from any location. A solution based in the cloud could remove many resource constraints and more easily achieve widespread applicability.

This report presents KRISEM, a cloud-based service management approach and application which provides monitoring for services running on remote servers while bypassing many limitations and otherwise necessary configurations to achieve widespread ease of deployment in its pursuit of ensuring higher services availability. User was able to monitor machine to machine communication gateway for Gardner Airport air traffic control system located at Springfield, Missouri with less than 30 minutes configuration time and it recovered from a crash in 2.5 seconds. The application provides all the necessary functionality such as: monitoring, alert notification, data visualization and archiving. The report presents the concept, design, implementation and envisioned future extensions to KRISEM.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Information technology service downtime is a major concern for any organization; small, medium or large. In today's technology-driven world, service unavailability could easily translate into huge economic losses (e.g., in e-commerce, such as Amazon) or even human tragedy (e.g., air traffic control systems). In order to minimize or prevent downtime, organizations have devised different processes and solutions. Continuous monitoring of operational services for availability is among those solutions.

Many tools [21,23,25,26] have been created to monitor information technology based services. These monitoring tools ensure services and processes are functioning properly. In the event of a failure, such tools can alert technical staff of the problem, allowing them to begin remediation procedures before outages affect business processes, end-users, or customers. Such tools also provide statistical information about the overall health of any service, as well as the entire system deployment, including for future planning. Successful use of such tools increases end-user confidence about the organization as well as its systems.

Most of the monitoring tools currently available require a secure access to the system being monitored, since any such tool is always polling the system being monitored for status. The deployment and maintenance of such tools can be very complex requiring highly experienced technical staffs. This requirement may make the overall solution very expensive by increasing operational cost and not ideal for many less technologically sophisticated organizations.

It is also very important to have an easy to deploy and broadly accessible monitoring tool. Some of these monitoring tools have limitations as to which deployment styles of systems they can monitor and some are only accessible using a private network, requiring staff to be on location or to have a pre-configured pre-configured system to allow them to access it remotely. For example, Nagios is very difficult to configure and lacks interactive user interface [34]. Such limitations can make it very inconvenient for the staff or expensive for the organization to allocate after hours support. This report presents the concept, design, implementation and analysis of KRISEM, a cloud-based monitoring tool that attempts to address these issues while providing a seamless user experience. The The process of creating KRISEM from design to fully functional implementation is explored in subsequent chapters.

1.1 KRISEM

KRISEM is a cloud-based service monitoring tool that tries to address many many limitations of existing service monitoring tools. Figure 1 shows a typical service monitoring configuration to demonstrate how these limitations can be inconvenient. In this scenario, the monitoring tool (e.g., Nagios [1]) has to go through the firewall to retrieve or monitor service status, since it needs to poll status information. The firewall has to be reconfigured almost every time the user decides to add a service into the monitoring tool. The deployment and maintenance of the tool is fairly complicated for any organization with limited resources. There is also a potential security risk which arises from allowing an external system to access a protected system. This configuration also depends on each remote service to respond if it is running. This may trigger a false alert, if the service fails to respond in timely manner due to network congestion. It will also log the false failure event and introduce error in the statistical analysis information.

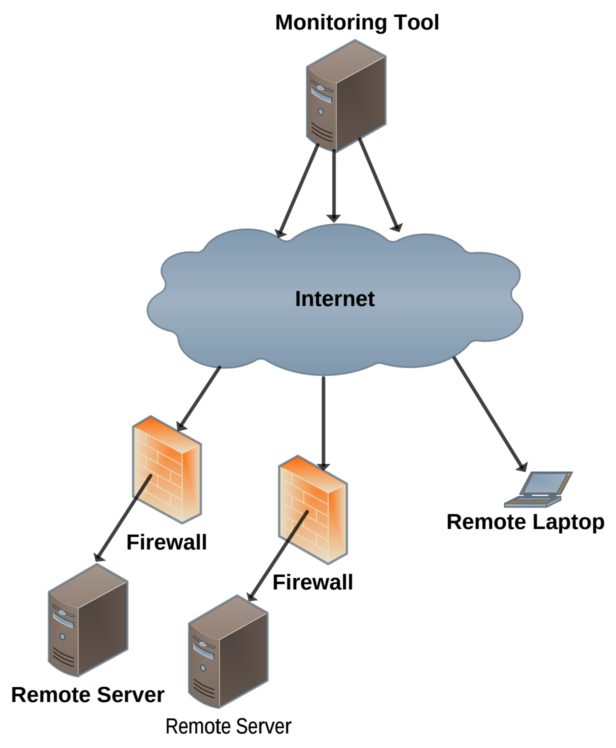


Figure 1: Typical monitoring tool configuration

A cloud-based solution can eliminate many issues in a typical configuration like the one in Figure 1. Since this type of system relies on flexible shared resources such as hardware, web server and database server to maximize its effectiveness [2], it would be able to receive, process, and store data from a large number of systems. Having all the relevant data in a single location means that the user can be presented with a simpler interface where any issues can be immediately recognized and from where all other data can be easily accessed. Furthermore, by having the systems push their data to a single server, there's no need to access multiple servers remotely. This virtually eliminates any need for special network configurations, as all that is required is a connection to the Internet. Figure 2 shows an example of a deployment using KRISEM.

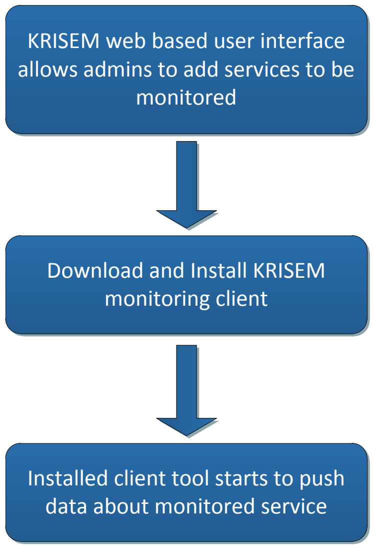

KRISEM web based user interface allows admins to add services to be monitored
Download and Install KRISEM monitoring client
Installed client tool starts to push data about monitored service

Figure 2: KRISEM configuration

The fundamental difference between both scenarios presented above is the way in which data is obtained from different systems. Unlike a typical application where services are polled directly, a KRISEM client will be installed in the remote system to monitor and push event data to KRISEM's cloud-based server. There will be no need for special network configuration to be able to monitor services remotely. Users only need an Internet connection and a compatible Web browser to retrieve service status information. The challenge with push is that the server needs to be always available to accept data and be able to grow on demand. KRISEM cloud-based solution is configured to ensure high availability and auto scalability [14,15]. By making such data highly available and easily accessible in the cloud, KRISEM could make service monitoring more attractive to non-technical users or even users who do not have permission to access these special network configurations.

In addition to the fundamental difference in benefits described above, KRISEM has set of features that make it very attractive for all users. The web interface let's user enter email or phone number to receive notification about failure or warning events via email or SMS text messages. The notification can be customized to type, frequency or the severity of the event. User can also use the web interface to configure the tool to make multiple attempts to restart services automatically in case of a crash. KRISEM is designed to continuously log events, analyze the events and plot them to identify trends or view a particular time range. It is capable of presenting service health as well as overall system health by aggregating multiple services and servers. KRISEM also tries to predict root cause and possible resolution of a failure, by analyzing details about the failures to make the troubleshooting more efficient.

1.2 User Stories

Here are some of the user stories from information technology and telecommunications. These user stories introduce two fictional characters Robert and Tom to create some scenarios to understand how KRISEM can be used in various real life situations. Robert works for an information technology company while Tom works for small value-added service provider to telecommunication companies.

1.2.1 Information technology

Robert works for a small company and is responsible for developing and managing software to support overall operations of the company. It is very critical that all the services responsible for M2M (machine to machine) communication (e.g., SMS, TCP/IP) running on the server must restart successfully after a crash, server reboot or any other failure for unknown reasons. He configured all the services to restart after a failure using Windows' service management console. He noticed almost half of the times when services fail, they do not restart automatically and they do not notify him that they have failed and have not restarted, so he does not know until customers complain (often on weekends and after hours).

In order to efficiently manage his services, he started using KRISEM. He uses a secure connection to register his Windows server and the M2M communication services he wants to manage with KRISEM. He can now access the service status from anywhere he has an Internet connection. He configured KRISEM to automatically restart in case of a failure, so he doesn't need to be on a secure network to restart remote services. He also configured KRISEM to allow anonymous access to the status page, so other users can notify him about any unusual events as well. Now, he does not have to be on call after hours or on the weekends like before.

1.2.2 Telecommunication

Tom works for a small company which provides value-added services such as music and live chat to subscribers of telecommunication companies. For the company interest, it is necessary that the services should be running ‘round the clock. In case services go down for a long duration of time, it can cause a serious loss of revenue.

Tom is using software which alerts him via email and he checks the status of services by logging onto the company network using a VPN. It is very inconvenient for Tom to constantly check email especially, or sometimes he has VPN connectivity issues.

Tom registers with KRISEM, adds the services and installs client tool to efficiently manage his services. Once any service goes down, KRISEM alerts him, logs the event for history and restarts the service immediately, thereby avoiding long downtimes. He can also check the status of services by using KRISEM’s Web interface, thereby avoiding any VPN connectivity issues.

1.3 Contributions

This report presents details of KRISEM as it is developed from a small idea to a full-fledged application. The following areas will be covered in this report:

- Vision and Design: KRISEM is designed to reduce service downtime, thereby solving a major problem across all organizations. It solves the problem by providing a simple but cost-effective cloud-based solution for small, medium or large companies. KRISEM has been evolved and refined with features that will make it desirable for a wide range of possible users.

- Implementation: To realize the vision, KRISEM has been implemented by using the latest technology currently available. Two separate versions of KRISEM were implemented using .NET and HTML Web technologies. The application has proven to be extremely successful while in service over an extended period of time and performed as it was supposed to do.
- Analysis: Quantitative results were obtained by testing each implementation of KRISEM. The tool was analyzed in terms of its speed, required memory, power, network bandwidth and other metrics that are relevant to its performance.

KRISEM is different from other service management applications currently available in the sense that small companies having limited budgets can afford to have it configured into their server infrastructure. It is also different from the competition in that it doesn’t require highly trained and technical staff for its configuration and management, since it has a user friendly Web interface. Most of the operations including adding new machine or new service are only a two-step process. Furthermore, its interface can be accessed from any location. No additional hardware or software is required.

1.4 Report outline

The remainder of this report will explore the design and development of KRISEM:

- Chapter 2, KRISEM specifications and requirements, will detail the specific objectives and behavior of the system
- Chapter 3, Implementation, gives a top level view of how the system is put together and what each component does.
- Chapter 4, summary of results, presents the outcome of the project and analyzes the performance of KRISEM.
- Finally, Chapter 5 presents the summary and envisioned future enhancements.

Chapter 2: KRISEM Specifications and Requirements

This chapter provides details on what KRISEM should do and how it actually works. It specifies the functional and non-functional requirements that define the interaction between tool and users. A mockup subsection has been provided as a guide showing how a user interface will be implemented.

2.1 Functional Requirements

The following is a list of basic features that will be supported by KRISEM:

- Monitor Services: KRISEM must be able to monitor custom services (e.g., air traffic control systems) in addition to publicly available services (e.g., HTTP, FTP, SSH, SMTP etc.) using available protocol such as Service Controller [4] or Simple Network Management Protocol (SNMP) protocol [13].
- Event Data Upload: KRISEM must be reachable by service monitors with Internet access. It must also be capable of establishing a connection with any monitor using TCP/IP and of receiving data from it using standard HTTP requests. Service monitors must be capable of periodically sending this data in appropriate intervals. This means all important data events must be captured, but excessive data volumes must be avoided. Table 1 shows example of event data in xml format captured while monitoring KRISEM test service.

Table 1: Event data in xml format

```
<ServiceActivity
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
  <ServiceID="6">
    <ServiceStopped>2015-04-07 13:19:39.000</ServiceStopped>
    <ServiceStarted>2015-04-07 13:23:39.000</ServiceStarted>
    <ServiceMessage>
      Service cannot be started.
      System.IO.DirectoryNotFoundException:
      Could not find a part of the path 'C:\Logs\TestKrisem.txt'.
    </ServiceMessage>
    <ReportTime>2015-04-07 13:23:39.000</ReportTime>
  </ServiceID>
</ServiceActivity>
```

- Event Data Processing: KRISEM must be capable of processing the received data to identify: the service monitor sending it; information about the remote server; and, any other data that may be deemed valuable. Monitors send event data using LINQ (Language Integrated Query) to SQL (Structured Query Language) protocol.
- Persistent Data Storage: Event data received by KRISEM must be stored in persistent storage for later use. Data to be stored should include the latest service data as well as historic event data. This data should be easily and efficiently retrievable; and, it should be persistent until cleared by the application.
- Data presentation: Upon user request, KRISEM should be able to present data on a Web-based graphical user interface. This interface should be simple, easy to navigate and understandable by all users. Users should be able to access this interface through standard HTTP requests over TCP/IP and they must be able to explore current as well as past data.

- User control: Users should be required and able to login to KRISEM before they can view their data. Users' credentials must be handled and properly encrypted in order to control user access.
- Event Notification: Users must be able to configure alerts based on triggering events. KRISEM will notify users via email or SMS text messages about any triggering event. It will also notify users if the condition gets resolved. Users should also be able to receive periodic notification about overall system health.
- Service Management: KRISEM will act as a fail-safe application to reduce overall downtime. Administrative users should be allowed to set automatic restart attempts in case of a service crash or failure. The number of attempts will also need to be set by the administrative user.
- Monitor Availability: If a service monitor fails to push data to KRISEM for an extended period of time, that monitor must be marked as missing by the system. Any recipients configured for any notification on a missing monitor will be notified of the situation, so that communications can be restored.
- Event Analysis: KRISEM should continuously analyze events and plot them to identify trends for individual services as well as for the whole system. KRISEM should also try to predict possible future failures, as well as identify the root cause of any past failures and suggest a possible resolution.

2.2 Graphical User Interface Mockups

KRISEM depends on a Web front-end graphical interface to interact with the users. Following are the user interface mockups that will show what should be implemented in final design of this tool.

2.2.1 Login

Figure 3 shows the initial login screen of the application. It prompts the user to enter his/her user name and password to login to KRISEM.

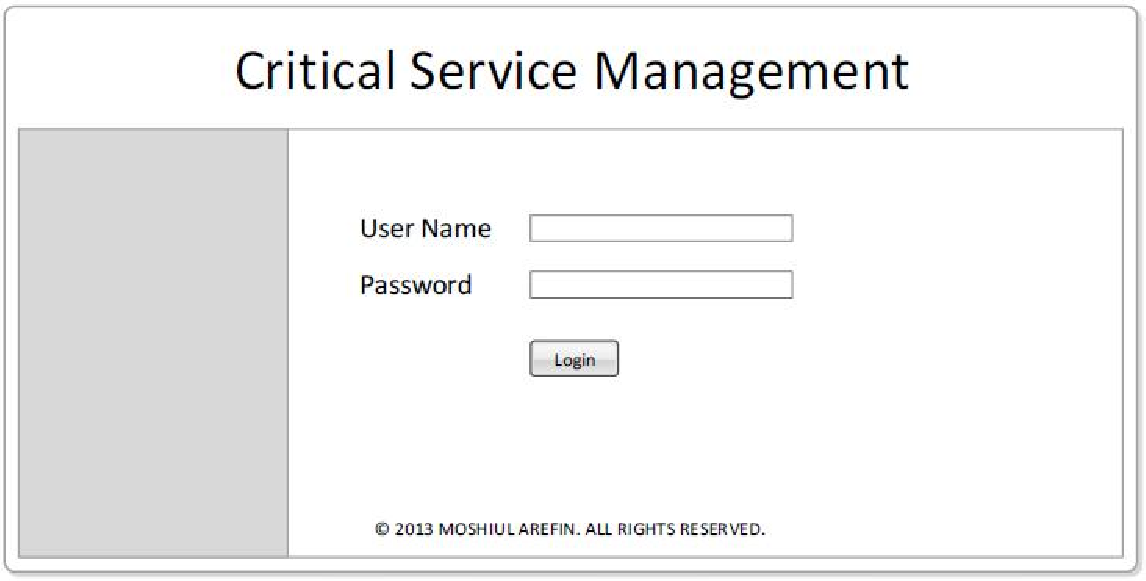
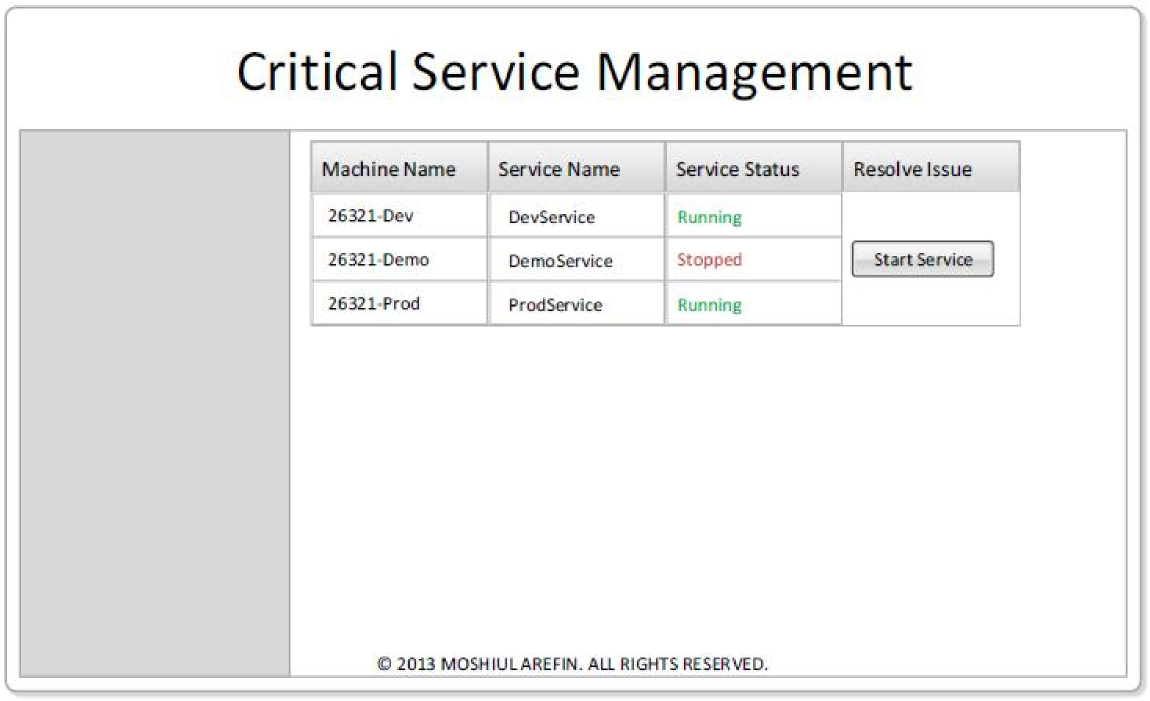

Critical Service Management
User Name
Password
Login
© 2013 MOSHIUL AREFIN. ALL RIGHTS RESERVED.

Figure 3: Login screen

2.2.2 Service Status

Figure 4 shows the default screen of the application. This page will list all of the services the user is currently monitoring with KRISEM. The list will show the name of the service, name of the server on which it's running, service status, as well as show a button triggering some ability to resolve issues. The services which are running are shown in green color while the services which are stopped are shown in red color to distinguish between them. Clicking on any item on the list would bring up the service view page for that item. This page acts as an overall summary and would allow users to see if there is any issue with the services being monitored.


Critical Service Management

Machine Name	Service Name	Service Status	Resolve Issue
26321-Dev	DevService	Running	
26321-Demo	DemoService	Stopped	Start Service
26321-Prod	ProdService	Running	

© 2013 MOSHIUL AREFIN. ALL RIGHTS RESERVED.

Figure 4: Service status screen

2.2.3 Add Server

Figure 5 shows how a new server can be added to KRISEM. The screen prompts the user to enter an identifiable server name, user name, password as well as a dropdown to choose login protocol for the remote server. A user can test the remote login to see if it has been successful or not. The remote login is optional for users who intend to use the poll method in addition to the default push method KRISEM uses. After the user clicks to add a server, the entered information will be stored in a secured environment with appropriate encryption. Users can add as many servers as they need to monitor their entire system deployment. Users will need to install a KRISEM client monitor into each of the servers they need to monitor.

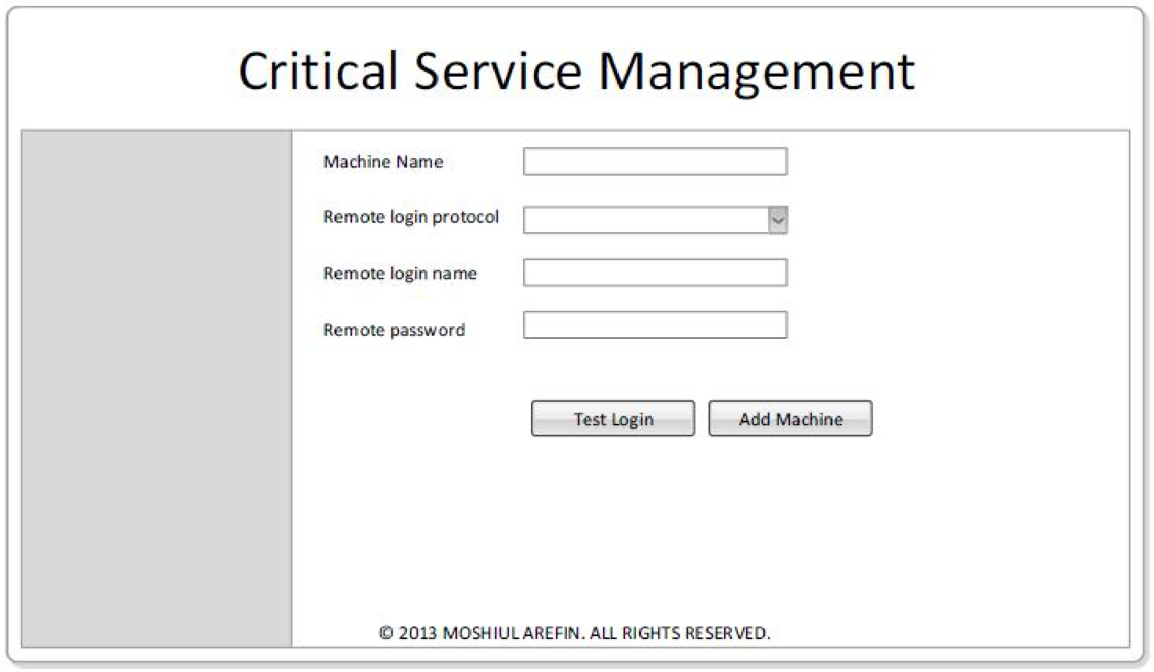


Figure 5: Add server screen

2.2.4 Add Service

Figure 6 shows how a new service on an existing server, from the user's system deployment, can be added into KRISEM for monitoring. It prompts the user to enter service name as well as various parameters to start or retrieve status information of the service. Additional parameters allows KRISEM to be compatible on different platforms and to resolve dependencies before it tries to restart a service. Users can click any of the test buttons if they want to implement a poll in addition to the default push method.

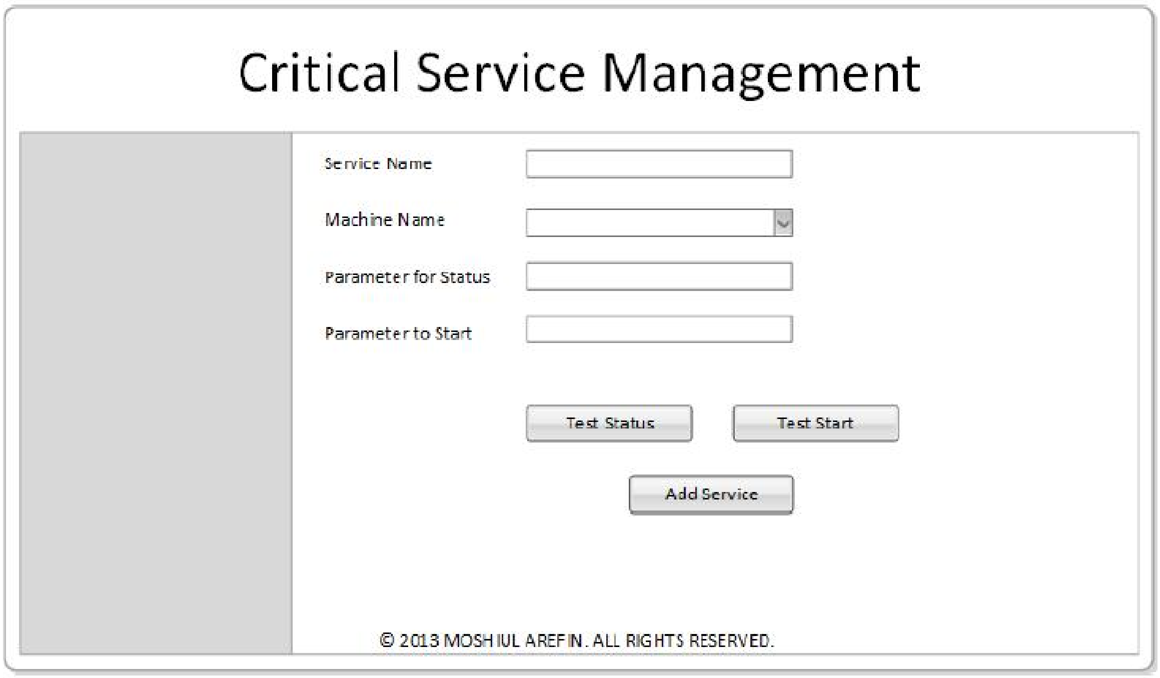


Figure 6: Add new service screen

2.2.5 Add User

Figure 7 shows how a new user can be added to allow additional users to monitor services using KRISEM. Only the administrator can exercise this feature. It prompts the user to add desired user name, password, reenter password and click on submit to add a new user.

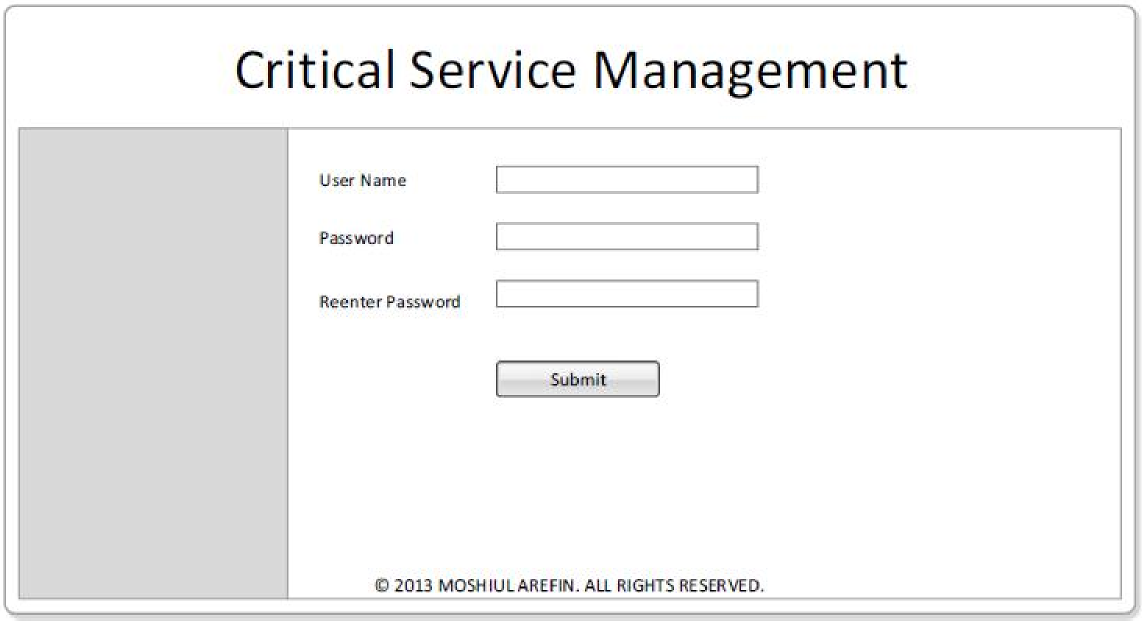

Critical Service Management
User Name
Password
Reenter Password
Submit
© 2013 MOSHIUL AREFIN. ALL RIGHTS RESERVED.

Figure 7: Add new user screen

2.3 Non-functional requirements

The following is a list of non-functional requirements for KRISEM to work as intended.

- Availability: Since KRISEM will be receiving service status from various servers, it should always be available to process that data. Any amount of time KRISEM during which is unavailable means potential loss of important event data.
- Scalability: One of the basic goals of moving a monitoring solution to the cloud is to allow for a dynamic demand on the system. KRISEM should be able to easily scale with demand. This should include cases where multiple servers happen to start sending service status simultaneously.
- Response time: KRISEM should be able to respond with reasonable time

duration when dealing either with servers or end users. In case of servers, a connection should be promptly established and data transferred, stored and processed in a timely manner. In the case of users, KRISEM should appear interactive and responsive.

- Cost: In order to position KRISEM in the market, its recurring costs should be sustainable and appropriate for the amount of servers and services supported.
- Maintainability: KRISEM should be well structured and easy to maintain so that problems can be easily corrected when they arise. It should also be easily integrated with other Internet services. Finally, when updating the user interface, removing any service or adding a new one should not result in extended service downtime or other service degradation.

Chapter 3: Design and Implementation

This chapter provides details about the design and implementation of KRISEM, what are its components, and how they interact with each other.

3.1 Architecture

KRISEM is a cloud-based monitoring solution. Therefore, it made sense to develop it using a client-server architecture style to maintain service quality and sustain its market position. Using a client-server model positively increases its ease of deployment through the usage of enhanced data storage, vast connectivity and reliable application services.

- Improved Data Sharing: Data is retained by usual business processes and manipulated on a server to be made available for designated clients over an authorized access.
- Centralization: Unlike P2P, where there is no central administration, in this architecture there is a centralized control.
- Back-up and Recovery: As all the data is stored on server, it’s easy to make a back-up of it. Also, in case of some failure, if data is lost, it can be recovered easily and efficiently.
- Ease of maintenance: Since client/server architecture is a distributed model, representing dispersed responsibilities among independent computers integrated across a network, it has advantages over other architectures in terms of maintenance. It’s easy to replace, repair, upgrade and relocate a server while clients remain unaffected.
- Security: Servers have better control access and resources to ensure that

only authorized clients can access or manipulate data and server-updates are administered effectively.

Figure 8 shows the top level view of KRISEM and its different components.

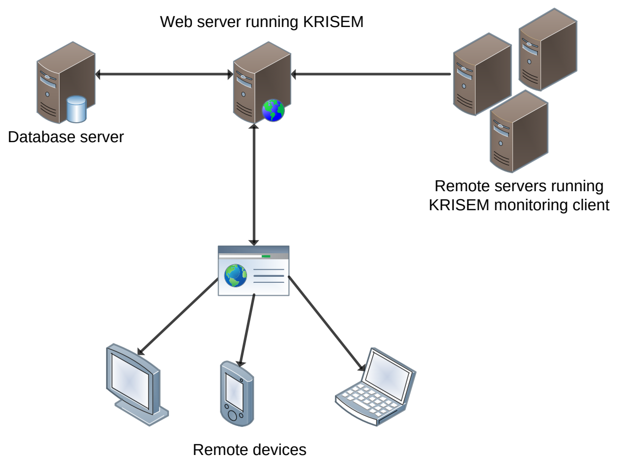

Web server running KRISEM
Database server
Remote servers running KRISEM monitoring client
Remote devices

Figure 8: System architecture

3.2 Design Pattern

In order to simplify testing and maintenance, KRISEM is designed using a model-view-controller (MVC) design pattern [9,10], which divides a given software application into three interconnected parts, so as to separate internal representations of information from the ways that information is presented to, or accepted from the user. The MVC design pattern re-enforces the separation of concerns into three components: the model (data layer), the view (presentation layer), and the controller (business layer):

- A **controller** can send commands to the model to update the model's state (e.g., editing a document). It can also send commands to its associated view to change the view's presentation of the model (e.g., by scrolling through a document).

- A **model** notifies its associated views and controllers when there has been a change in its state. This notification allows the views to produce updated output, and the controllers to change the available set of commands. In some cases an MVC implementation may instead be 'passive' and other components must poll the model for updates rather than being notified.
- A **view** requests information from the model that it uses to generate an output representation to the user.

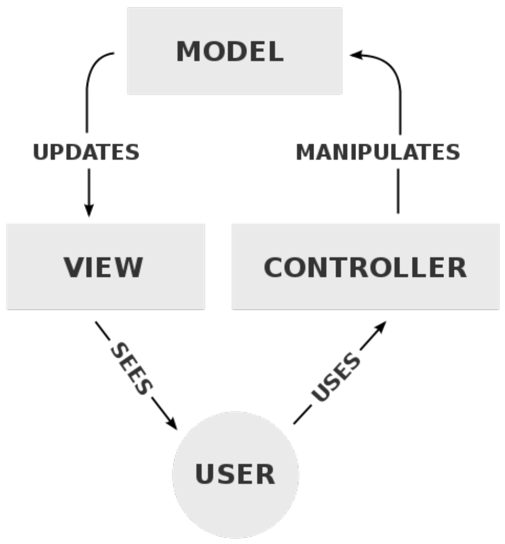


Figure 9: A typical collaboration of the MVC components

3.3 KRISEM Web User Interface

KRISEM's Web-based user interface is responsible for displaying collected events to the users. It has been developed using the ASP.NET MVC2 Web application framework that implements the model-view-controller (MVC) pattern. Figure 10 shows server and services direction graph for KRISEM's models and their respective controllers.

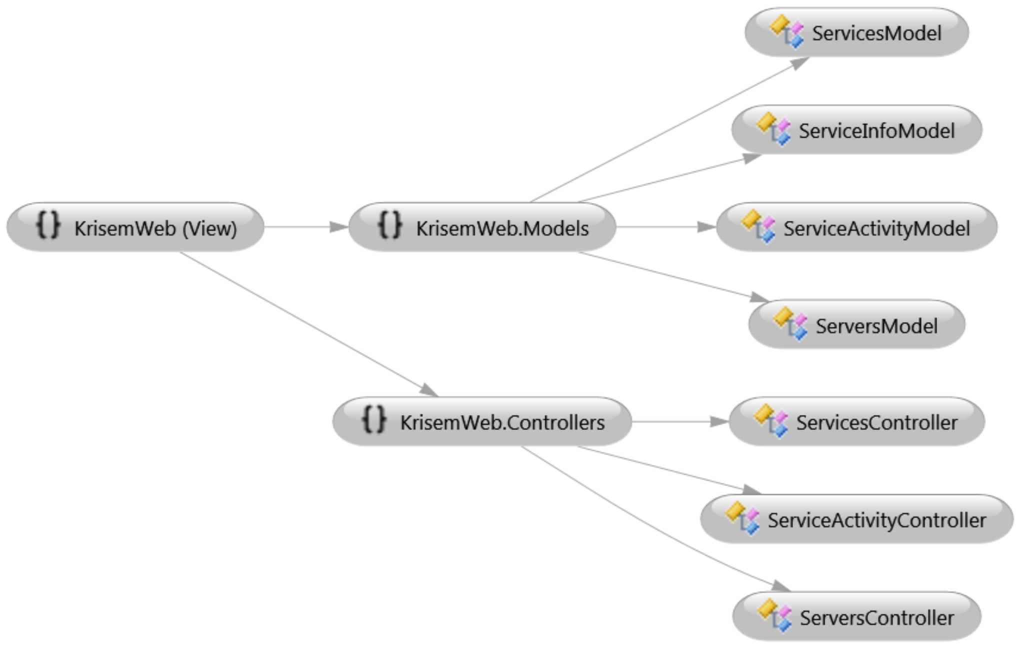
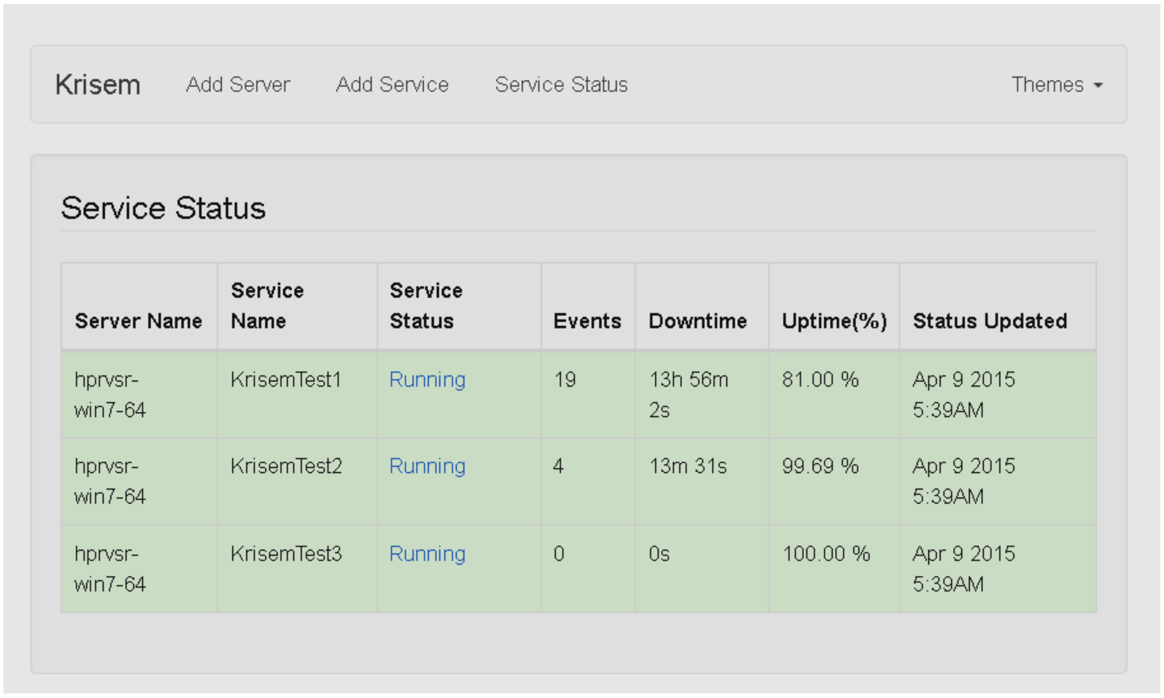


Figure 10: KRISEM models and controllers direction graph

Master Page: To achieve the same look and feel, KRISEM used ASP.NET's master page feature. The master page contained Web user interface components common to all the Web pages in the application. The header navigation in Figure 11 shows the common theme used in the master page.



Service Status

Server Name	Service Name	Service Status	Events	Downtime	Uptime(%)	Status Updated
hprvsr-win7-64	KrisemTest1	Running	19	13h 56m 2s	81.00 %	Apr 9 2015 5:39AM
hprvsr-win7-64	KrisemTest2	Running	4	13m 31s	99.69 %	Apr 9 2015 5:39AM
hprvsr-win7-64	KrisemTest3	Running	0	0s	100.00 %	Apr 9 2015 5:39AM

Figure 11: KRISEM default page

JQuery: KRISEM uses jQuery [6] to make the application compatible across different platforms with different Web browsers. It also makes KRISEM mobile browser friendly. jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript.

Google Chart: Plotting is achieved using Google's Visualization API [3] which takes a dataset created by the user interface and converts it into a graph on the client side, thus offloading some of the computing effort required by the cloud servers. Figure 12 uses Google Chart to show service up and down time in a pie chart as well as number of events on a particular day in a yearly calendar.

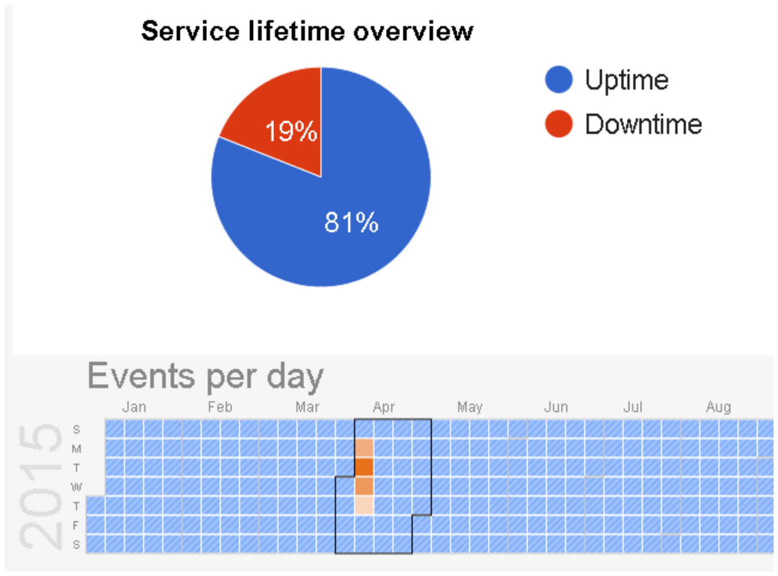


Figure 12: Service health using Google Chart

3.4 KRISEM Monitor Client

Client is an ASP.NET Microsoft Windows service application (similar to Unix daemon), which runs on every remote server to be monitored. It makes request to the server for registered services with KRISEM and connects to services using ServiceController Class [4] for status. In case of a failure client restarts the service, collects failure data using ASP.NET EventLog Class [5] and sends it to the data storage using LINQ (Language Integrated Query) [12]. During this process it also calculates and updates necessary statistical data. KRISEM leverages remote sever resources by doing data processing in the client instead of web server should make for a highly efficient cloud-based deployment. The client is also responsible for sending event-related alerts to the appropriate person. It also sends its own heartbeat at a user-specified frequency, since no one is watching the watchdog.

3.5 KRISEM Database

The database is the part of KRISEM that stores and serves up data upon request from the other components. This component is meant to provide scalability and the ability to easily access the required elements. Microsoft's SQL Server 2012 relational database is used for data storage. This database provides better security, high availability and supports complex query. It also will be able to handle future requirement for additional fields, if needed. KRISEM uses LINQ to SQL to access and manipulate data. Figure 13 shows a diagram of the database schema used by KRISEM.

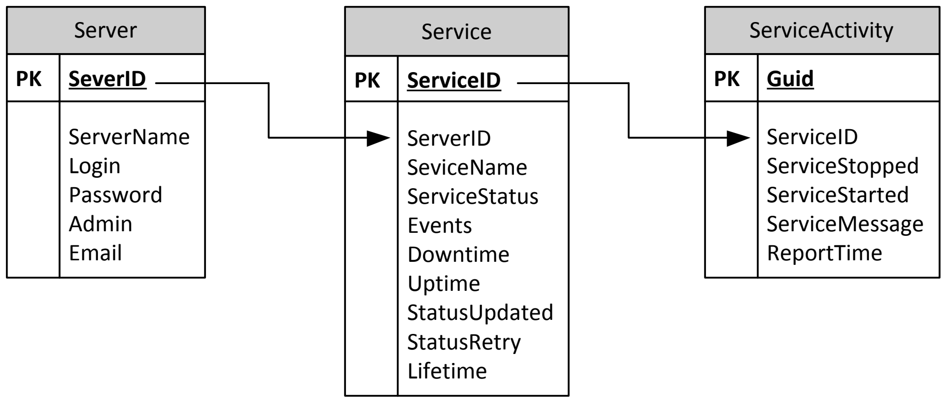


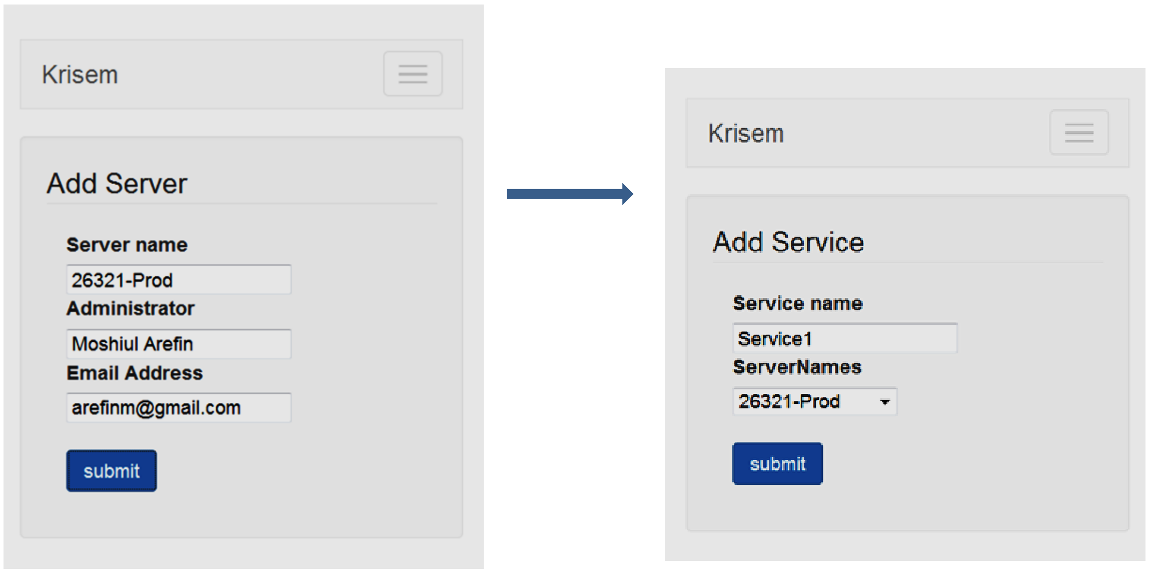
Figure 13: Database schema

Chapter 4: Summary of Results

This chapter analyzes the final implementation of KRISEM. It explores the look and feel of this tool, as well as its performance in Rackspace Private Cloud [14]. It also looks at important software metrics to gauge quality and maintainability of the project.

4.1 Qualitative results

The success of KRISEM can be determined by comparing initial requirements with the end results. In this case, it can be said that KRISEM has fulfilled its most important functional requirements. The following figures show KRISEM in action. Most part of the user interface of KRISEM represents the one specified by mockups. Figure 14 shows setting up a server and a service in KRISEM for monitoring and notification.


Krisem

Add Server

Server name: 26321-Prod
Administrator: Moshiul Arefin
Email Address: arefinm@gmail.com

submit

Krisem

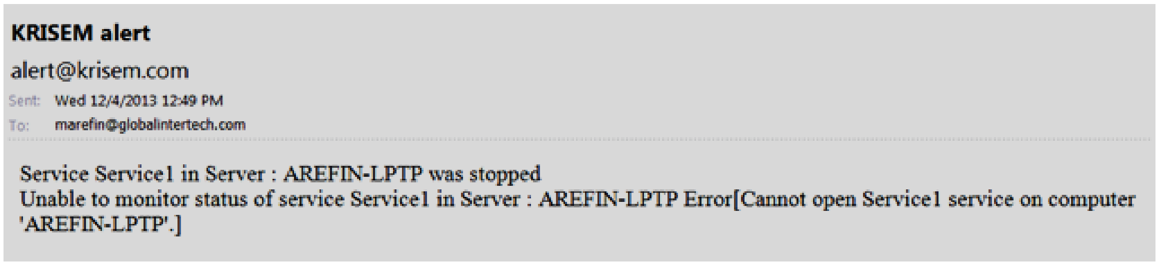
Add Service

Service name: Service1
ServerNames: 26321-Prod

submit

Figure 14: Configuring server and services

Figure 15 is a screenshot of an email alert of a service failure. KRISEM's monitoring client did not attempt to start the service, since it wasn't able to get a handle on the service controller [4] itself, due to a missing parameter in the service configuration.

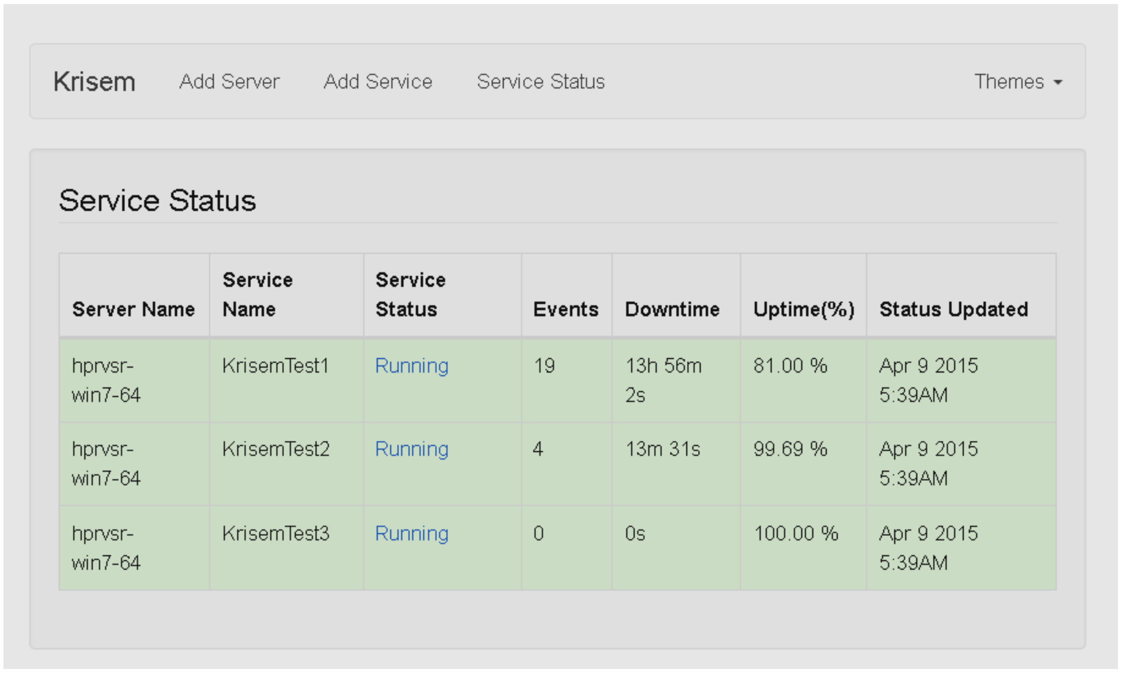


KRISEM alert
alert@krisem.com
Sent: Wed 12/4/2013 12:49 PM
To: marefin@globalintertech.com

Service Service1 in Server : AREFIN-LPTP was stopped
Unable to monitor status of service Service1 in Server : AREFIN-LPTP Error[Cannot open Service1 service on computer 'AREFIN-LPTP'.]

Figure 15: Email alert

Figure 16 is a screenshot taken from krisem.com that shows the service listing, service status, number of events, total downtime and uptime in percent, as well as the time the monitor reported service status.

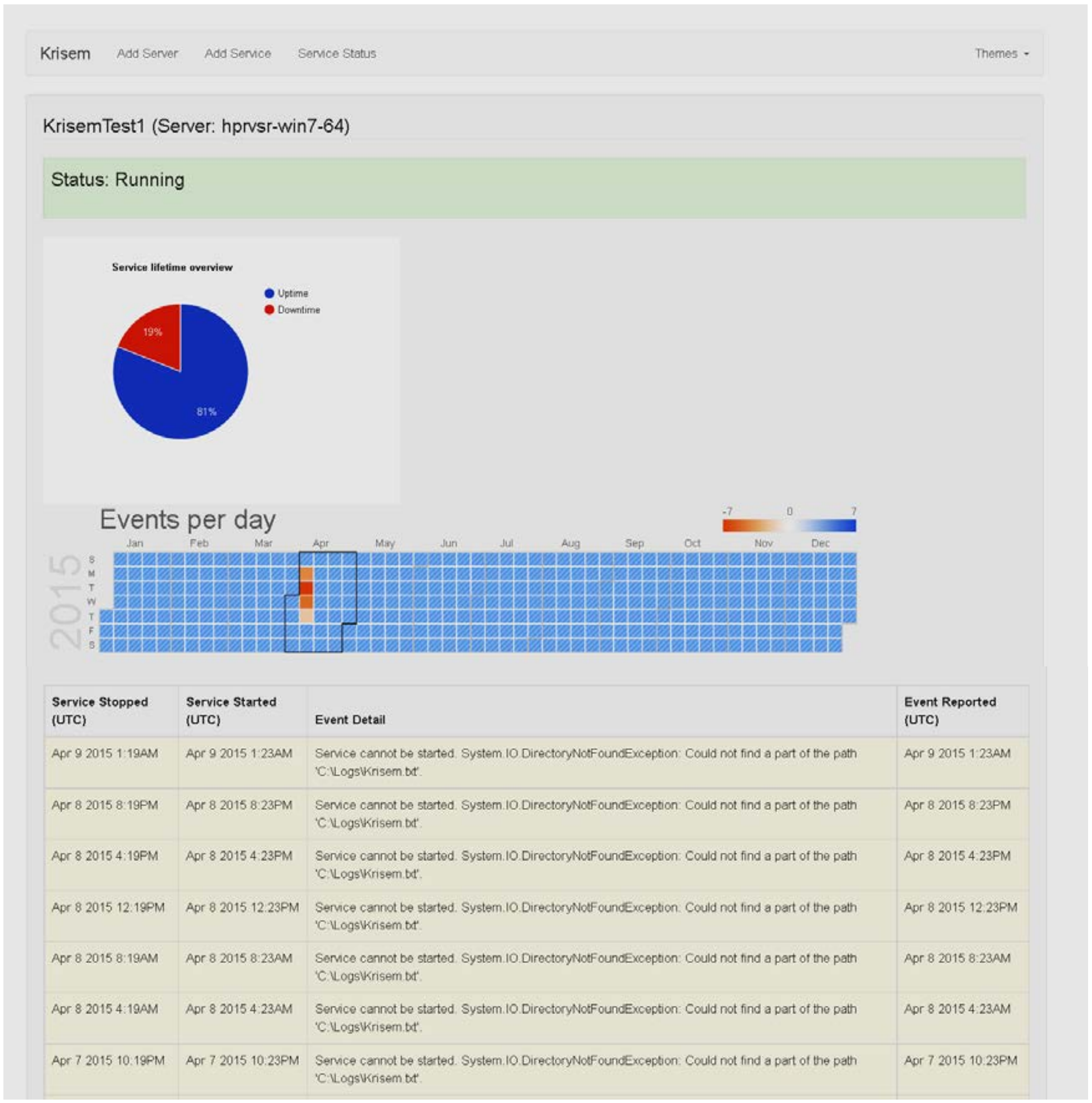


Service Status

Server Name	Service Name	Service Status	Events	Downtime	Uptime(%)	Status Updated
hprvsr-win7-64	KrisemTest1	Running	19	13h 56m 2s	81.00 %	Apr 9 2015 5:39AM
hprvsr-win7-64	KrisemTest2	Running	4	13m 31s	99.69 %	Apr 9 2015 5:39AM
hprvsr-win7-64	KrisemTest3	Running	0	0s	100.00 %	Apr 9 2015 5:39AM

Figure 16: Service status page

Figure 17 is a screenshot of the detail service status page. In this screenshot, KRISEM shows service uptime / downtime using a pie chart and number of events on a particular day for the whole year. It also displays the log of all the event details, including at what time the service stopped, what time the service started, and what time the event was reported by KRISEM.



Service Stopped (UTC)	Service Started (UTC)	Event Detail	Event Reported (UTC)
Apr 9 2015 1:19AM	Apr 9 2015 1:23AM	Service cannot be started. System.IO.DirectoryNotFoundException: Could not find a part of the path 'C:\Logs\Krisem.txt'.	Apr 9 2015 1:23AM
Apr 8 2015 8:19PM	Apr 8 2015 8:23PM	Service cannot be started. System.IO.DirectoryNotFoundException: Could not find a part of the path 'C:\Logs\Krisem.txt'.	Apr 8 2015 8:23PM
Apr 8 2015 4:19PM	Apr 8 2015 4:23PM	Service cannot be started. System.IO.DirectoryNotFoundException: Could not find a part of the path 'C:\Logs\Krisem.txt'.	Apr 8 2015 4:23PM
Apr 8 2015 12:19PM	Apr 8 2015 12:23PM	Service cannot be started. System.IO.DirectoryNotFoundException: Could not find a part of the path 'C:\Logs\Krisem.txt'.	Apr 8 2015 12:23PM
Apr 8 2015 8:19AM	Apr 8 2015 8:23AM	Service cannot be started. System.IO.DirectoryNotFoundException: Could not find a part of the path 'C:\Logs\Krisem.txt'.	Apr 8 2015 8:23AM
Apr 8 2015 4:19AM	Apr 8 2015 4:23AM	Service cannot be started. System.IO.DirectoryNotFoundException: Could not find a part of the path 'C:\Logs\Krisem.txt'.	Apr 8 2015 4:23AM
Apr 7 2015 10:19PM	Apr 7 2015 10:23PM	Service cannot be started. System.IO.DirectoryNotFoundException: Could not find a part of the path 'C:\Logs\Krisem.txt'.	Apr 7 2015 10:23PM

Figure 17: Detail service status page

4.2 Quantitative Results

4.2.1 Availability

The issue of the availability of cloud platform is a bit difficult to ascertain. Many service outages may be entirely unrelated to the availability and stability of the platform on which they run on. For instance, some outages may be caused by a fault in the application itself, while another may be related to only a partial outage on a platform. Rackspace Private Cloud provides service level agreement defining an appropriate level of availability [15] for applications. Uptime of 99.99% [15,16] is assured with refunds offered for higher levels of unavailability. KRISEM never went down since its inception in Rackspace Private Cloud.

4.2.2 Scalability

One of the benefits of using a cloud platform is the ability to maintain performance as demand grows. As such, KRISEM is expected to properly scale to handle varying traffic loads. The more servers and services it monitors, the more resources it will require for its processing. The Web user interface is running in IIS 7 with the MS SQL 2012 database, also making KRISEM very scalable.

4.2.3 Platform Compatibility

KRISEM is a cloud-based service management solution making it highly compatible with different operating system. Users only need an Internet connection and a compatible Web browser to setup or retrieve service status information. Currently KRISEM can only monitor services running in windows based operating system. KRISEM monitor client needs to be extended to support additional protocols (e.g., SNMP, Net-SNMP [13]) to able to monitor services or processes running Linux (or UNIX variant).

4.2.4 Performance

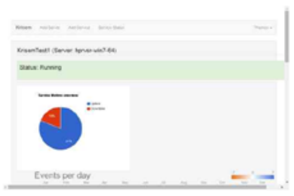
This section explores KRISEM performance by monitoring three test services running in Windows 7, Intel I5 3.20 GHz processor and 3.00 GB of available RAM. Test services are set to crash in random order with different types of program failures such as file not found, SqlExcetpion etc. KRISEM used stopwatch methods [17] to record time to monitor services. Table 2 shows average time KRISEM took to monitor and report each incident.

Table 2: Processing time

Scenario	Operation	Time (ms)
Service running	Calculate and update statistical information in KRISEM database.	236.31
Service stopped	Restart successful, retrieve event failure log, calculate, update statistical information in KRISEM database and send alert notification.	2450.08
Service stopped	Restart failed, retrieve event failure log, calculate, update statistical information in KRISEM database and send alert notification.	2948.76

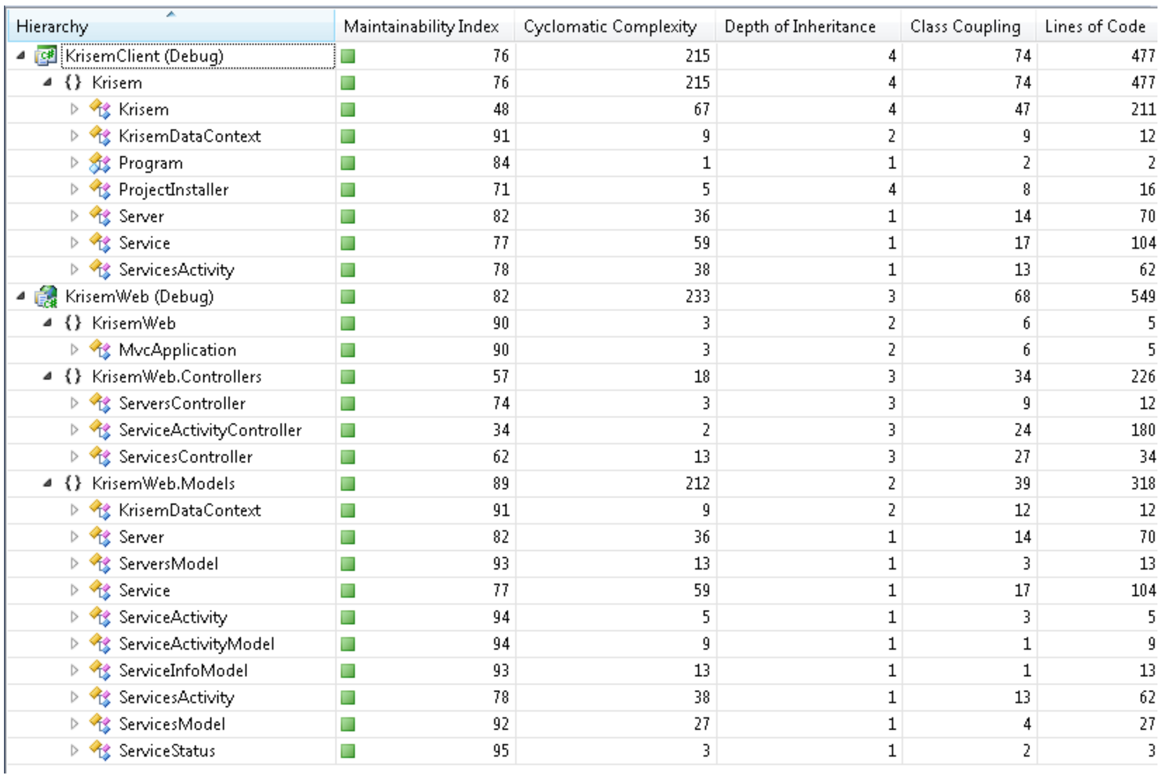
KRISEM web interface was also tested using WEBPAGETEST [18] web page performance test. Average minimum page load time is 0.984 second and maximum is 2.326 seconds. Page that took the longest uses Google Chart API [3] to display plots. Table 3 contains screenshots from different page load performance test.

Table 3: Web page performance comparison

KRISEM status page	KRISEM service status detail page	Google home page
Fully Loaded Document Complete (0.983 sec)	Fully Loaded  Document Complete (2.326 sec)	Fully Loaded Document Complete (1.806 sec)

4.2.5 Software Metrics

Software metrics were computed for KRISEM for measuring the quality and complexity of code. Figure 19 shows the code metrics generated by Visual Studio 2010.



Hierarchy	Maintainability Index	Cyclomatic Complexity	Depth of Inheritance	Class Coupling	Lines of Code
KrisemClient (Debug)	🟩 76	215	4	74	477
Krisem	🟩 76	215	4	74	477
Krisem	🟩 48	67	4	47	211
KrisemDataContext	🟩 91	9	2	9	12
Program	🟩 84	1	1	2	2
ProjectInstaller	🟩 71	5	4	8	16
Server	🟩 82	36	1	14	70
Service	🟩 77	59	1	17	104
ServicesActivity	🟩 78	38	1	13	62
KrisemWeb (Debug)	🟩 82	233	3	68	549
KrisemWeb	🟩 90	3	2	6	5
MvcApplication	🟩 90	3	2	6	5
KrisemWeb.Controllers	🟩 57	18	3	34	226
ServersController	🟩 74	3	3	9	12
ServiceActivityController	🟩 34	2	3	24	180
ServicesController	🟩 62	13	3	27	34
KrisemWeb.Models	🟩 89	212	2	39	318
KrisemDataContext	🟩 91	9	2	12	12
Server	🟩 82	36	1	14	70
ServersModel	🟩 93	13	1	3	13
Service	🟩 77	59	1	17	104
ServiceActivity	🟩 94	5	1	3	5
ServiceActivityModel	🟩 94	9	1	1	9
ServiceInfoModel	🟩 93	13	1	1	13
ServicesActivity	🟩 78	38	1	13	62
ServicesModel	🟩 92	27	1	4	27
ServiceStatus	🟩 95	3	1	2	3

Figure 18: Code metrics from Visual Studio 2010

- Maintainability Index (MI) for each of the module is >20. So the project source is highly maintainable as expected from a client-server architecture and model-view-controller design pattern.
- Cyclomatic Complexity varied for different modules. KRISEM can use more automated testing for better code coverage.
- Depth of Inheritance for each module is very low which indicates simple to follow class hierarchy.
- Class Coupling is very low among different modules indicating a higher reuse potential and easy to maintain.
- Lines of Code per module is in the low range indicating less complexity and higher maintainability.

4.2.6 Engineering efforts

The effort involved maintaining any application usually represents a significant portion of its cost. As such, the following numbers provide a quantification of this complexity.

4.2.6.1 Source Lines of Code

The following tables show statistics about KRISEM's source base. Total written code for .NET Framework [20] is 1886 lines, which is about 3.8% of the code. Also only 0.6% of the code is test code totaling 292 lines of code. Most of the testing was done manually. Total code for other web technologies (JavaScript [35] and CSS [36]) is 1913 lines, which is about 3.84%. In summary, total lines of code written is 3907 (7.84%) and total lines of library code is 45912 (92.16%).

Table 4: Lines of code per language

Language	Lines of code	Percent of code
.js	35644	71.5%
.css	12181	24.5%
.cs	1029	2.1%
.aspx	691	1.4%
.sql	166	0.3%
.xml	108	0.2%
Total	49819	100.00%

Table 5: Lines of code per type

Type	Lines of code	Percent of code
Blank lines	8516	12.2%
Comment lines	9511	13.6%
Lines of code	48820	71.3%
Designer files	2051	2.9%

Table 6: Lines of test code

Type of code	Lines of code	Percent of code
Test code	292	0.6%
Production code	49528	99.4%

4.2.6.2 Source Control

The two developed versions of KRISEM are kept in separate branches in a GIT repository. Most of the development was performed in the cloud-based version. In total, there are 29 commits. The second version of KRISEM that only monitors and generate alerts has a total of 18 commits. Figure 20 is a screenshot of impact graph of GIT commit history since inception. It may be worth mentioning that best practices were not always followed, as there were several long periods of time with numerous changes were made without a code commit.

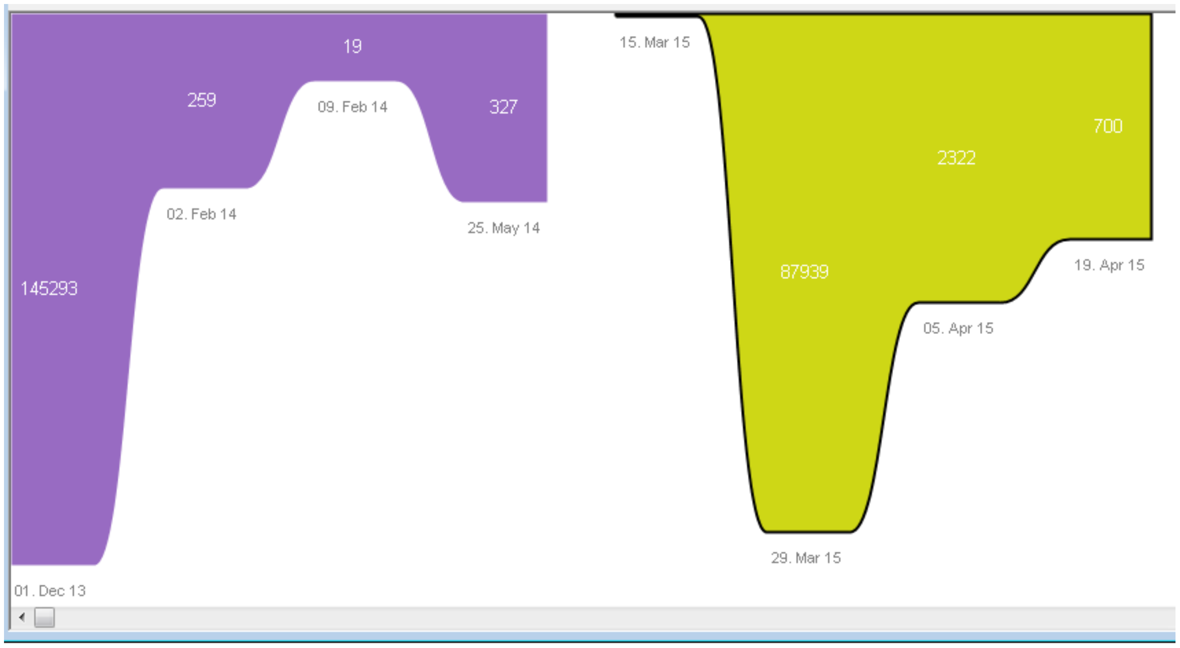


Figure 19: Commit history since inception

Chapter 5: Conclusions

5.1 Summary

This report presented KRISEM, a cloud-based information technology service management application. This report has explored the vision behind this tool, as well as its requirements and design. The tool was successfully completed and tested manually as well as automatically by inserting three dummy services for the whole system. With complete results, it can be concluded that KRISEM consumes relatively modest resources on servers; and, therefore, is proven to be feasible to implement across a broad variety of systems and circumstances.

KRISEM was able to successfully perform all the required functions. It was able to receive, process and store data from servers, as well as send alerts when required. Moreover, it did so while providing adequate levels of scalability, availability, maintainability, and response time while costing a reasonable amount.

KRISEM needs additional polishing before it can be marketed for end-customers, but it has been proved that the design will accomplish the required functions. The next move will be to improve and expand the tool's capabilities in an effort to bring its benefits to wide variety of customers. The tools and concepts behind KRISEM can be used to help improve uptime for many operational solutions that are experiencing repeated service failures.

5.2 Future Work

KRISEM has been designed and built in such a way that it can be easily extended later. In future, I expect to enhance features of KRISEM to provide analytics for failures and predict solutions to improve troubleshooting efficiency. Another useful feature would be to combine all the services as well as the servers to generate a plot showing the overall system health. The tool is functional, but it could use a number of performance refinements as well security enhancements before being released to end-customers.

5.3 Lesson Learned

Five things that worked during the development of KRISEM:

- MVC design pattern and Bootswatch made the user interface development very simple. The overall time spent on the user interface compared to other ASP.NET web application was lot less.
- JQuery made the user interface very mobile browser friendly and do not require any mobile application for the user.
- Google Chart API provided the plots needed for the service status visualization and took very little time to integrate.
- LINQ to SQL [12] provided a run-time infrastructure for managing relational data as objects in C#.
- Use tools with good support: Community support for tools is important when developing a new application. When there is a problem, often the best source of a solution is the community that uses the tool and supports it

by providing answers to questions. Community supported tools will also have a longer life than unsupported tools.

Five things that did not worked during the development of KRISEM

- Conversion from MVC2 to MVC4 is close to impossible. MVC2 has to be converted to MVC3 before converting to MVC4 [40]. Spend lot of time trying to convert MVC2 to MVC3 and finally gave up.
- Visual Studio performance analyzer [41] supposed to let developers measure, evaluate, and target performance-related issues but the report was very difficult to understand.
- KRISEM monitor client runs into permission issues in some of the machines and did not fully understand the root cause.
- JSON implementation did not go well due to lack of understanding about the technology and time limitation.
- Server clustering implementation failed but was able to resolve this by using a third party cloud [14].

Five things that would be nice to have:

- KRISEM should use MVC4 instead of MVC2: it provides very responsive view engine, chart, crypto as well as better JQuery support [38].
- Analytics: the application needs to have the ability to identify the root cause of any past failures as well as predict possible future failures, and suggest a possible resolution.

- KRISEM need to have a monitor client to able to monitor services or processes running Linux (or UNIX variant) to be widely accepted monitoring application.
- SMS Alert: the application needs to be extended to allow SMS alert notification in addition to email for events that needs immediate attention.
- Automated testing: Manual testing is time consuming and might not get the needed coverage, should have used automated testing from the beginning of the development.

Bibliography

[1] Nagios Core 4.x Documentation, 2015. Available: http://nagios.sourceforge.net/docs/nagioscore/4/en/toc.html

[2] S.U.Muthunagai C.D. Karthic S. Sujatha, "Efficient Access of Cloud Resources through Virtualization Techniques" in IEEE International Conference on Recent Trends in Information Technology, 2012

[3] Google Visualization API Reference. 2015. Available: https://developers.google.com/chart/

[4] Microsoft .NET Service Controller. 2015. Available: https://msdn.microsoft.com/secontroller.aspx

[5] Microsoft .NET EventLog Entry. 2015. Available: https://msdn.microsoft.com/eventlog.aspx

[6] The JQuery Foundation. 2015. Available: https://jquery.com/

[7] ASP.NET Ajax: Enhanced Interactivity. 2015. Available: http://www.asp.net/ajax

[8] Bootswatch Themes for Bootstrap. 2015. Available: https://bootswatch.com/

[9] Model-View-Controller. 2015. Available: https://msdn.microsoft.com/en-us/library/ff649643.aspx

[10] Elisabeth Freeman, Head First Design Patterns. O'Reilly Media. 2004.

[11] Len Bass, Paul Clements. Software Architecture in Practice. Addison-Wesley. 2010.

[12] Language-Integrated Query (LINQ). 2015. Available: https://msdn.microsoft.com/en-us/library/bb397926.aspx

[13] Simple Network Management Protocol (SNMP). 2015. Available: http://www.net-snmp.org/

[14] Rackspace Private Cloud (RPC). 2015. Available: http://www.rackspace.com/cloud/private

[15] Implementing High Availability (HA) For Rackspace Private Cloud. 2015. Available: http://www.rackspace.com/blog/implementing-high-availability-ha-for-rackspace-private-cloud/

[16] Rackspace adds 99.99 uptime guarantee to private cloud service. 2015. Available: http://www.zdnet.com/article/rackspace-adds-99-99-uptime-guarantee-to-private-cloud-service/

[17] Stopwatch Class. 2015. Available: https://msdn.microsoft.com/en-us/library/system.diagnostics.stopwatch%28v=vs.110%29.aspx

[18] WEBPAGETEST. 2015. Available: http://www.webpagetest.org/

[19] Code Metrics Values. 2015. Available: https://msdn.microsoft.com/en-us/library/bb385914.aspx

[20] .NET Framework. 2015. Available: https://msdn.microsoft.com/en-us/vstudio/aa496123.aspx

[21] Oscar Cabrera, Xavier Franch, “A Quality Model for Analysing Web Service Monitoring Tools”, Research Challenges in Information Science (RCIS), 2012 Sixth International Conference on, 2012, pp. 1-12.

[22] M. P. Papazoglou, K. Pohl, M. Parkin and A. Metzger, "Service research challenges and solutions for the future Internet: S-Cube – towards engineering, managing and adapting service-based systems,” vol. 6500, Germany: Springer-Verlag Berlin Heidelberg, 2010.

[23] L. Baresi and S. Guinea, “Towards dynamic monitoring of WS-BPEL processes,” Third International Conference in Service Oriented Computing (ICSOC), 2005, pp. 269–282.

[24] L. Baresi, S. Guinea, and P. Plebani, “WS-Policy for service monitoring,” 6th International Workshop, TES, 2005, pp. 72–83.

[25] ManageEngine, “Monitoring Tool”, 2015. Available: http://www.manageengine.com/

[26] Nagios, “Monitoring Tool”, 2015. Available: http://www.nagios.org/

[27] WebInjectCorey, Goldberg, “Monitoring Tool”, 2015. Available: http://www.webinject.org

[28]WS-I Web Services Interoperability Organization, “Monitoring Tool”, 2015. Available: http://www.wsi.org/

[29] Dotcom-Monitor, “Monitoring Tool”, 2015. Available: http://www.dotcom-monitor.com

[30] ORACLE, “Monitoring Tool”, 2015. Available: http://www.oracle.com/index.html

[31] SmartBear, “Monitoring Tool”, 2015. Available: http://www.soapui.org

[32] Neustar Webmetrics, “Monitoring Tool”, 2015. Available: http://www.webmetrics.com/

[33] Apache Software Foundation, “Monitoring Tool”, 2015. Available: http://axis.apache.org/

[34] Sophon Mongkolluksamee, Panita Pongpaibool, Chavee Issariyapat, “Strengths and Limitations of Nagios as a Network Monitoring Solution”, Proceedings of the 7th International Joint Conference on Computer Science

and Software Engineering (JCSSE 2010) Vol. 1, pp. 96-101, Bangkok, Thailand, May 2010

[35] JavaScript, 2015. Available: http://en.wikipedia.org/wiki/JavaScript

[36] Cascading Style Sheets (CSS), 2015. Available: http://en.wikipedia.org/wiki/Cascading_Style_Sheets

[37] ASP.NET Razor View Engine, 2015. Available: http://en.wikipedia.org/wiki/ASP.NET_Razor_view_engine

[38] Difference between MVC2, MVC3, MVC4, 2015. Available: http://freefeast.info/tutorials-for-beginners/dotnet-tutorials/difference-between-mvc2-mvc3-mvc4-mvc2-vs-mvc3-vs-mvc4/

[39] Selenium Browser Automation, 2015. Available: http://www.seleniumhq.org/

[40] Migrate from MVC2 to MVC4, 2015. Available: http://stackoverflow.com/questions/12107191/migrating-from-asp-net-mvc2-to-mvc4

[41] Analyzing Application Performance by Using Profiling Tools, 2015. Available: https://msdn.microsoft.com/en-us/library/z9z62c29.aspx

[42] Machine to Machine (M2M), 2015. Available: http://en.wikipedia.org/wiki/Machine_to_machine

[43] Nighthawk Cellular M2M Control, 2015. Available: http://nighthawkcontrol.com/m2m/

Vita

Moshiul Arefin is a key member of Nighthawk's smart grid meter team, leading a team of software engineers in many aspects of product and overall system design, development, testing and production oversight. He is also responsible for wireless communication gateways, user facing applications and infrastructure development for Nighthawk.

He received his Bachelor in Electrical Engineering from University of Texas at Austin in December 2003. His focus was on VLSI Design and Embedded Systems in his undergraduate studies. Currently he is pursuing his Masters in Software Engineering at the University of Texas at Austin and expected to finish by May 2015.

Address: arefinm@gmail.com

This report was typed by the author.