CS Capstone Requirements Document

April 27, 2020

ChatBot for Load Balancer Infrastructure

Prepared for

OSU Information Services

Stacy Brock

Prepared by

Group 63

Nitro ChatBot

Jack Barnes

Sarun Pitaksuteephong

Cheng Xie

Abstract

This document is an SRS (Software Requirement Specification) which outlines the client requirements for the proposed chatbot software. The chatbot will enable users to directly query the status, and modify configurations, for their load balanced resources. The requirements within this document serve as a contract with the client and will be used to judge the quality and completeness of the software upon it's 1.0 release.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Change History

Revision	**Date**	**Changes**
1.0	10/26/2019	First Draft
1.1	11/25/2019	Added language to require direct messages for user commands. Modified language to allow for an additional authentication step. Added mention of a 60 second command response time limit.
1.2	12/03/2019	Fixed broken symbol. Correct spelling error. Created table for user commands. Updated both figures.
1.3	04/27/2020	• 1.3.2 Removed first product function (Chatbot has it's own app window and does not need to listen for it's name). • Removed references to pools (replaced with generalized term: resources). • Removed requirement for updating certificates. • Removed usage of the university's federated login system, instead built-in authentication in Microsoft Teams is used. • Removed reference to stateless functionality, which does not work for this type of application. • 2.1 Testing suites target input checking (as the application does not easily allow for live testing with it's web services implementing additional built-in security to safeguard traffic). • 2.2 Removed reference to additional persistent storage (this requirement is unneeded). • 2.3 Updated Table 1 (Available user commands). Added: list, listall, listbound, disablenow, help, request-auth, Removed: updatecert. • 2.5 Removed reference to additional authentication step (This is unneeded for the used authentication pattern). • Changed reference to ”remove” commmand to ”disable”. • Removed ”Logical database requirements” section (formerly 2.7), as no database is needed.

1 Introduction

1.1 Purpose

The software will provide a chatbot interface for users to query the status of their load balanced resources within the Oregon State University network. Users with the correct permissions will be able to quickly access the status and perform common configuration tasks on the University's load balancer.

1.2 Scope

Two primary software components will need to be developed. The first is the chatbot that will accept commands from a user. The second software will act as the internal relay, receiving requests from the chatbot, sending commands to the load balancer and then returning the results. The relay will maintain authentication for active users and perform the direct interaction with the load balancer API (Application Program Interface). This will allow the chatbot to provide quick access to the status and configuration options for the server pools that the users manage.

1.3 Product overview

1.3.1 Product perspective

Chatbots are typically used in dialog systems for various practical purposes including customer service or information acquisition [1]. This proposed implementation of the chatbot exists on top of an existing system and is meant to provide convenient access for the target users. The chatbot itself will operate as the user interface for users to direct message. The internal relay will operate as a software interface with the Citrix NetScaler, using it's REST API [2].

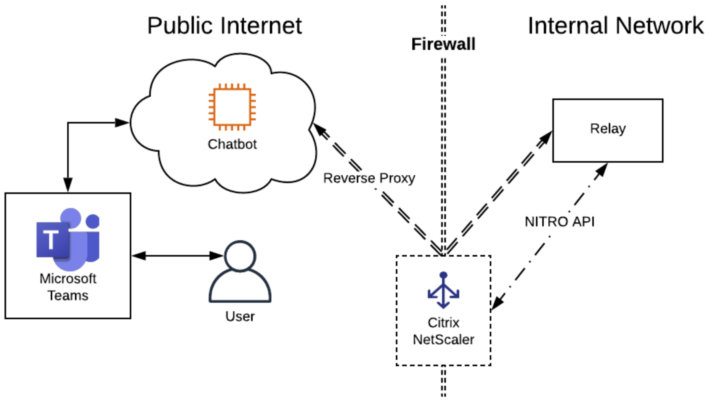

Fig. 1. Basic overview of the chatbot, and how it will allow users access to load balancer configurations

1.3.2 Product functions

The software will have the following major functions:

1. The chatbot will accept input from the user and parse input for commands.

2. Upon parsing a command, it will securely transmit command and user information to the internal relay.

3. The relay will manage authentication

4. The relay will send requests to the load balancer through NITRO API

5. The relay will transmit results back to the chatbot.
6. The chatbot will respond appropriately to user commands.
7. The user will not have to authenticate separately outside of Microsoft Teams.
8. Users will be able to query for the status resources (including vservers, servicegroups, services, and servers).
9. Users will be able to query for the status of individual servers.

1.3.3 User characteristics

Users of this chatbot will be system administrators who manage pools of servers at Oregon State University. They have ONID (OSU Network ID) accounts. They have a high-level of technical understanding and are familiar with command line interfaces. The targeted user base is approximately 2 dozen people.

1.3.4 Limitations

The chatbot will operate on a common platform accessible to all targeted users. It will leverage the built-in authentication within Microsoft Teams to identify users. The chatbot must be written in a framework that ensures future portability. The chatbot itself must be stateless, and not requiring persistent storage. The chatbot should be (virtually) always available, running on a cloud service within public internet. A system of CI/CD (Continuous Integration / Continuous Deployment) must also be developed to maintain availability.

1.4 Definitions

Botkit Framework: An open-source developer tool for building chatbot that also allows custom integration for major messaging platforms. Has direct access to platform APIs. It is owned by Microsoft [3].
Chatbot: A chatbot is a piece of software that conducts a conversation via auditory or textual methods [1].
Citrix NetScaler: A an application delivery controller made by Citrix that load balances to make sure that applications can be run as smooth as possible [4].
Duo: It is a vendor for cloud-based two-factor authentication service. It can be integrated with any website, VPN (Virtual Private Network), and cloud services [5].
Load Balancer: Load balancer is a reverse proxy that distributes network or application traffic across the server pool. This will increase the efficiency, maintainability, and, speed of the server pool.
NITRO API: The NetScaler NITRO protocol allows you to configure and monitor the NetScaler appliance programmatically by using Representational State Transfer (REST) interfaces [2].
ONID: Oregon State University ID used for identification
REST: Representational State Transfer. It is a software architect style that defines what could be used for creating web services. The main constraint are the client that uses the API and the resources that the API can provide information to [6] [7].

2 Requirements

2.1 Apportioning of requirements

Two separate pieces of software will be defined in these requirements. This first is referred to as the chatbot, and the second piece of software is referred to as the relay. These separate pieces of software will be developed separately, but concurrently. Both have features that rely on the operation of the other.

To assist with development, tests will be written to check input and parsing of that input. Features and requirements that span both the chatbot and the relay may be implemented in this fashion at first. However the feature will not be considered fully implemented until both component software operate with each other.

2.2 Specified requirements

The first component software is the chatbot. The chatbot will run from from a cloud service and operate within a chat platform that can be equally accessed by all users. The chatbot will be triggered to begin processing input when it's messaged directly, and will not operate by idling within a channel and parsing messages.

When a user messages the chatbot, it will begin parsing input given to it in a manner consistent with command line interfaces. If the input is invalid, it will tell the user. If the input in valid, it will initiate communication with the second component software, the relay. The chatbot will send a request to the relay containing all the necessary information to complete the request, including user information.

When the relay receives a request, it will use the user information to request authentication from the NetScaler. The relay will communicate requests to the NetScaler using it's REST API, NITRO API. The relay will maintain and manage authentication in a manner that meets the software requirements.

After a user is authenticated, the relay will send the command request to the NetScaler. Upon receiving a reply from the NetScaler, the relay will format a reply and send it to chatbot. The chatbot will then format and send the result to the user.

2.3 Functions

The perspective of the user, the functioning of the software will be quiet simple, with only a small number of commands available. The intention of the software is to perform these complex configuration tasks as seamlessly as possible. The chatbot will parse commands from a user when a user directly messages it. The available functions are detailed below:

TABLE 1
Available user commands

Command	**Description**	**Response**
list	List all vservers	Returns a list of all vservers
listall	List all resources	Returns a list of all resources
listbound <resource>	Lists all resources bound to the specified resource	Returns a list of resources, or an error if it's a server
status <resource>	Query the status of a resource	Returns a list of resource properties, including it's state
enable <resource>	Enable the server within it's pool	Confirmation of success
disable <resource> [delay]	Disables a resource gently, with an optional delay	Confirmation of success
disablenow <resource> [delay]	Disables a resource (not gently), with an optional delay	Confirmation of success
help [command]	Displays a usage text, or optionally, for one command	Returns all usage text, or a single usage text
request-auth [message]	Sends a request to the admin for access (message is optional)	Confirmation that the log was sent

If a user asks to perform a task without proper having permissions, then the bot will respond to the user that they do not have permission to perform the requested task.

2.4 Performance requirements

The chatbot is to be, essentially, always available. It is expected to run continuously on a cloud server, responding to a user when it's messaged directly. Each user command should be received, processed, and be responded to in a manner of seconds (with 60 seconds being an upper limit). The chatbot should process commands concurrently, never needing to finish previous requests before new requests are received. The chatbot should be able to receive command input from multiple sources simultaneously without losing received requests (<1% dropped input).

2.5 Usability requirements

For the user, sending commands through the chatbot should be a seamless experience, minimizing the number of required steps to perform a task. To achieve this, the chatbot must minimize user interaction while maintaining security.

The chatbot will accept commands from authorized users in a similar style to command line utilities. The formatting of the interaction will be first the command, then any applicable flag, followed by additional information. For example, a user may ask the bot to disable webserver2 by inputting: ”disable webserver2”.

2.6 Interface requirements

For proper operation and integration of the chatbot and it's relay, both have a number of existing systems that they must interface with.

The chatbot must be deployed to a cloud service to maintain availability. This includes designing a system of continuous integration/deployment to allow for remote deployment. The chatbot must also interface with the chosen chat platform. Finally, the chatbot must remotely interface with it's relay using a secure connection and common language, through which, it can communicate with the relay.

The relay, the second piece of software, will eventually be deployed internally within the Oregon State University network. To achieve this, it will likely be deployed as a virtual machine, however the details of this deployment are not statically specified at this time. However, for development, the relay will be deployed to a cloud instance.

The relay must accept a secure connection from the chatbot (which is deployed remotely). The relay must interpret user information contained within requests from the chatbot to identify and authenticate against the load balancer (Citrix NetScaler). The relay will interpret commands and send requests to the Citrix NetScaler using it's REST API, NITRO API.

2.7 Design constraints

The chatbot will be deployed to a cloud service in a manner that allows continuous deployment and continuous integration. The relay must utilize the Nitro API to communicate with the NetScaler. The chatbot must have a separate relay so that it does not communicate directly with the load balancer.

2.8 Software system attributes

2.8.1 Reliability

The chatbot or the relay should not crash or produce unspecified states regardless of user input or data received from any interface. If the software does halt, or reach an unspecified state, it must be deployed it such a manner that it restarts and restores itself with minimal loss of data or downtime.

2.8.2 Availability

The chatbot is intended to be always available. One exception is the availability of the cloud server/service it's deployed to. An additional factor in availability, will be the implementation of a messaging system between the chatbot and the relay. This system must account for connection instability resulting from any number of factors.

2.8.3 Security

Security is paramount with the implementation of this software system, as the chatbot will be indirectly communicating with vital network resources. Users must never be able to request information about, or reconfigure, network resources they don't have explicit permission to modify.

All reconfiguration and status requests must be logged and made accessible to only administrative users.

The relay must interpret requests from the chatbot, never directly passing queries from the chatbot. The relay must minimize the amount of information that is sent to the chatbot, relaying only what is necessary.

Additionally, the chatbot must never monitor messages that aren't directed at it.

2.8.4 Portability

In an attempt to future-proof the chatbot, it will be written using a portable framework. This will allow the chatbot to be easily portable to different messaging networks, minimizing the work required if an organizational transition occurs.

3 Appendices

3.1 Feature implementation timeline

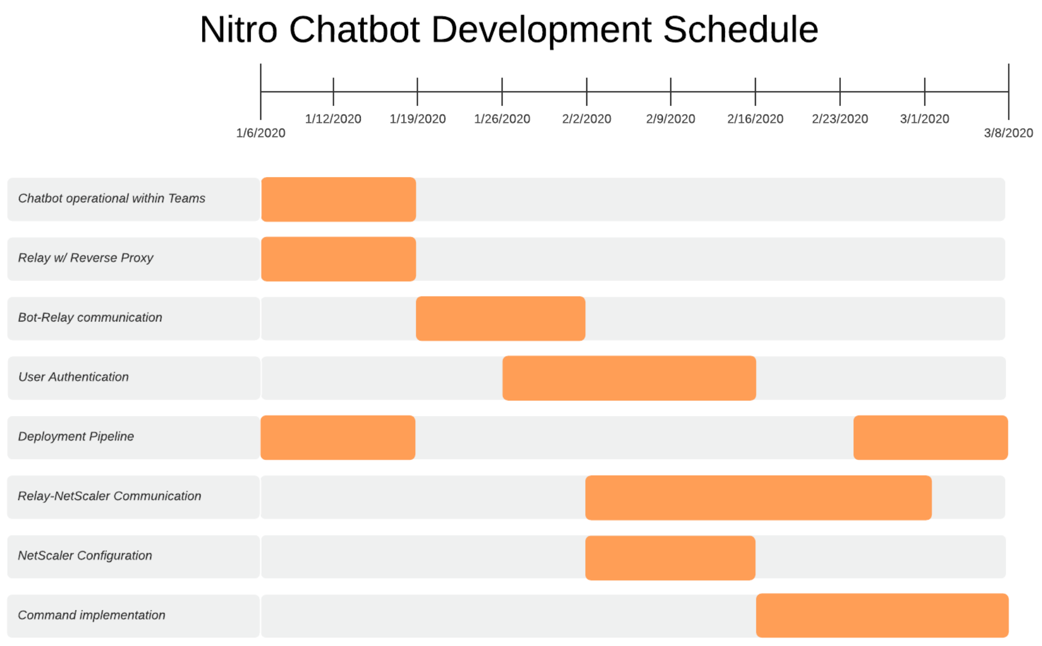

Fig. 2. This is tentative development schedule for Winter term 2020. This timeline give a rough estimation or when features will be implemented.

References

[1] (2019, Oct.) Chatbot. Wikipedia, Wikimedia Foundation. [Online]. Available: https://en.wikipedia.org/wiki/Chatbot

[2] Citrix NetScaler 12.0 REST APIs - NITRO. Citrix. [Online]. Available: https://developer-docs.citrix.com/projects/netscaler-nitro-api/en/12.0/

[3] J. Church. (2017, Nov.) Picking a Chatbot Framework: Botkit vs Microsoft Bot Builder. [Online]. Available: https://medium.com/@jonchurch/picking-a-chatbot-framework-botkit-vs-microsoft-bot-builder-1088db3d075a

[4] V. King. (2019, Aug.) What is Citrix Netscaler? Whitehat Virtual. [Online]. Available: https://www.whitehatvirtual.com/what-is-citrix-netscaler/

[5] M. Rouse. (2014, Dec.) What is Duo Security? Tech Target. [Online]. Available: https://searchsecurity.techtarget.com/definition/Duo-Security

[6] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[7] (2017, Jun.) REST API Tutorial. [Online]. Available: https://restfulapi.net/