Automating the Configuration of Virtual Private Network Servers

by

Yongjun Xu
B.Sc., Guangdong University of Foreign Studies, 2014

A Master's Project Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

Automating the Configuration of Virtual Private Network Servers

by

Yongjun Xu
B.Sc., Guangdong University of Foreign Studies, 2014

Supervisory Committee

Dr. Yvonne Coady, Supervisor
(Department of Computer Science)

Dr. Sudhakar Ganti, Departmental Member
(Department of Computer Science)

Supervisory Committee

Dr. Yvonne Coady, Supervisor
(Department of Computer Science)

Dr. Sudhakar Ganti, Departmental Member
(Department of Computer Science)

ABSTRACT

The challenge of consistent and reliable deployment of a distributed application on a large scale is significant, in particular if all of the steps must be executed manually. This project explores an automated approach to populate a distributed environment using a freely available tool called *Chef*. In particular, we focus on configuring cloud servers into a Virtual Private Network (VPN) of service providers. To demonstrate a fully implemented distributed VPN service, we present an infrastructure including a web interface, payment service and database integration. The prototype system allows for one-line command setup for VPN servers, leveraging an automated deployment framework. Furthermore, a preliminary evaluation and analysis on the automated approach is presented, concretely demonstrating the advantages and disadvantages of automated deployment within the setup process on a large scale.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank:

My family, especially my parents for their unconditional love and full support.

Dr. Yvonne Coady, for mentoring, support, encouragement, and patience.

Luxing Huang for contributing to the project and all the teaching along the way.

And anyone else who has helped me and shared their moments with me. It is an honor to meet you all.

Yongjun Xu

DEDICATION

I dedicate this project report to my parents who always support me.

Chapter 1

Introduction

Deploying modern applications of any size is complex as it involves many steps. Manual deployment process consists of separate and atomic steps, each performed by an individual or team. Often times, the deployment process must be executed repeatedly, leaving it prone to human error. With manual deployment, errors can also be hard to track down since they are subject to each execution environment involved. Moreover, frequent documentation requirements on the details of the process setup is tedious, time consuming, and often overlooked.

This project proposes the idea of automating the deployment process, be it software, or configuration setup. To show the proof-of-concept, the project deploys the configuration code from develop environment, onto a production environment involving cloud servers. The configuration code is designed to set up a Virtual Private Network (VPN) service on cloud servers. This project demonstrates an easy solution to set up a VPN service provider from scratch, by automating the deployment process of configuration policies. To present the case, a front-end interface and other third-party applications are integrated in the implementation, which makes it more practical to use. Based on the nature and scope of the project, Chef [3] is chosen as the automated deployment tool. With automated deployment approach, configuring a server is only one command line away.[1]

[1] provided valid configuration instruction code is in place.

1.1 Structure of the Project

This section provides a map of the report and the content from each Chapter is summarized as follows:

Chapter 2 explains background knowledge and describes in detail the challenges of manual deployment process, followed by an evaluation of different automated deployment tools.

Chapter 3 provides the design principles of the project, which includes but not limited to authentication method, automated tools, etc. It serves as a guideline for project implementation.

Chapter 4 draws a full picture of the project at a higher level and highlights its key features.

Chapter 5 includes the evaluation of the service presented above and the improvement implemented in the project. It also mentions a possible future direction that moves the project to next level.

Chapter 6 concludes the purpose and solution of the project.

Chapter 2

Related Work

This chapter begins by defining Virtual Private Network (VPN) services in general, and how they can be configured in a cloud computing setting particularly. Next, challenges of setting up cloud servers on a large scale are identified. Finally, a survey of the state-of-the-art in popular deployment tools provides the context within which we identify our proposal to automate key and common setup activities.

2.1 Virtual Private Network (VPN)

Strictly speaking, a virtual private network (VPN) is a communications environment in which access is controlled to permit peer connections only within a defined community of interest. A VPN is constructed though some form of partitioning of a common underlying communications network, where this underlying communication provides services to the network on a non-exclusive basis [21]. Simply put, VPN allows the provisioning of private network services for organizations over a public or shared infrastructure such as the Internet.

There are several motivations for building VPNs, but the major one is the desire by individuals and organizations to make some portion (sometimes all) of the communications invisible to external parties, while still be able to benefit from a common communication infrastructure. A VPN creates an encrypted tunnel through the public Internet, ensuring that traffic cannot be intercepted. As the Internet becomes less secure, VPNs are becoming increasingly popular. Examples include allowing employees to work securely from home or mobile devices, possibly over public WiFi networks. VPN service in this project is to prevent all user communications from

being intercepted by external observers.

2.2 Cloud Computing

Cloud computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the data centers that provide those services [15]. P Mell et al. [29] introduce three major service models in cloud computing:

- *Software as a Service (SaaS).* The capability provided to the consumer is to use the providers applications running on a cloud infrastructure. A demand-led software market in which businesses assemble and provide services when needed to address a particular requirement. The key of SaaS is to separate the possession and ownership of software from its use [31].

- *Platform as a Service (PaaS).* The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider.

- *Infrastructure as a Service (IaaS).* The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications.

The goal of this project is to easily configure existing cloud servers into machines that provide a Virtual Private Network (VPN) as a service. Our proposed strategy utilizes a SaaS service model. Simply put, cloud servers that run in a cloud computing environment as software-independent units can be configured to offer packet forwarding functionality in a secured manner. We show how the setup process can be simplified and improved, especially on a large scale, by implementing the configuration policy using current tools for automated deployment.

2.3 Challenges of Scalable VPN Server Setups

Building *one* VPN server only takes a number of steps, which may be manageable manually. However, building such servers on a large scale is not equivalent to repeat-

ing the same manual processes. J Humble et al. [26] explain drawbacks of deploying manually overall:

- The requirement to provide extensive, detailed documentation that describes the steps to be taken.
- Reliance on manual testing to confirm that the application is running correctly.
- Unpredictability in outcomes of releases, which easily result in unforeseen problems.
- Frequent corrections to re-deploy processes for environments that differ in their configuration.

Based on principles above, the following sections uncover the two main difficulties of setting up a set of *distributed* VPN Servers.

2.3.1 Management Overhead

Setting up servers requires installing packages for dependencies and overwriting configuration files. For scalability purposes, setup commands can be summarized in script files and be run once for all. However, as the number of machines involved grow, configuring each individual server is labor intensive and error prone. Moreover, incremental modification to configuration files will result in repeating re-deployment processes. Version control of configuration files can be challenging if these files are not put together in a central workstation. Fortunately, automation management tools can automate the complete configuration with minimal human interaction. With deployment tools, such as chef, existing configuration policies can be leveraged and automatically deployed on new servers [11]. In general, setting up *one* VPN server could take the same amount of effort as doing so for *a million* servers.

2.3.2 Error-prone

When VPN setup must be manually reproduced on every server in an application, an important step may be accidentally missed in one particular setup, which could lead to errors in product release. The more repeatable steps are run, the greater the risk of having missteps. Therefore, manual deployments are error-prone.

Unfortunately, VPN servers cannot fully benefit from leveraging system snapshots either. A snapshot is the state of a system at a particular point in time. Though this may provide a template, many small configuration options, e.g. hostname, vary from server to server. To this end, automated deployment tools are introduced to provide a standard, consistent deployment strategy and to eliminate any drawbacks from variability. In the next section we will provide an overview of current state-of-the-art deployment tools.

2.4 Evaluation on Automated Deployment Tools

This section explores automated deployment tools available to date, and explains why Chef is chosen in this project. Configuration deployment, in this project, refers to the process of taking instructions written in-house, copying the related files to cloud servers, then executing those instructions in order. The makeup of server environment is a consideration when selecting the right deployment tool. Different tools are built in different languages, therefore the level of support each provide can vary greatly.

2.4.1 Ansible

Ansible is a free-software platform for configuring and managing computers in a scalable way. It manages nodes over Secure Shell (SSH) or over PowerShell. Modules work over JSON [18] and standard output and can be written in any programming language [2].

Ansible Tower is a commercial web-based tool that centralizes and controls Ansible infrastructure with a simple, straight-forward dashboard. As shown in Figure 2.1, built-in powerful features include job scheduling, role-based access control, cloud inventory management, etc. Ansible is sufficient for easy server provisioning, yet introspection is limited. Moreover, it is strenuous to keep track of values of variables within the playbooks.

2.4.2 Fabric

Fabric is a Python-based tool for streamlining the use of SSH for application deployment or systems administration tasks [9]. It is mostly used in executing tasks across a distributed set of systems, e.g. deployment of an app, system configuration.

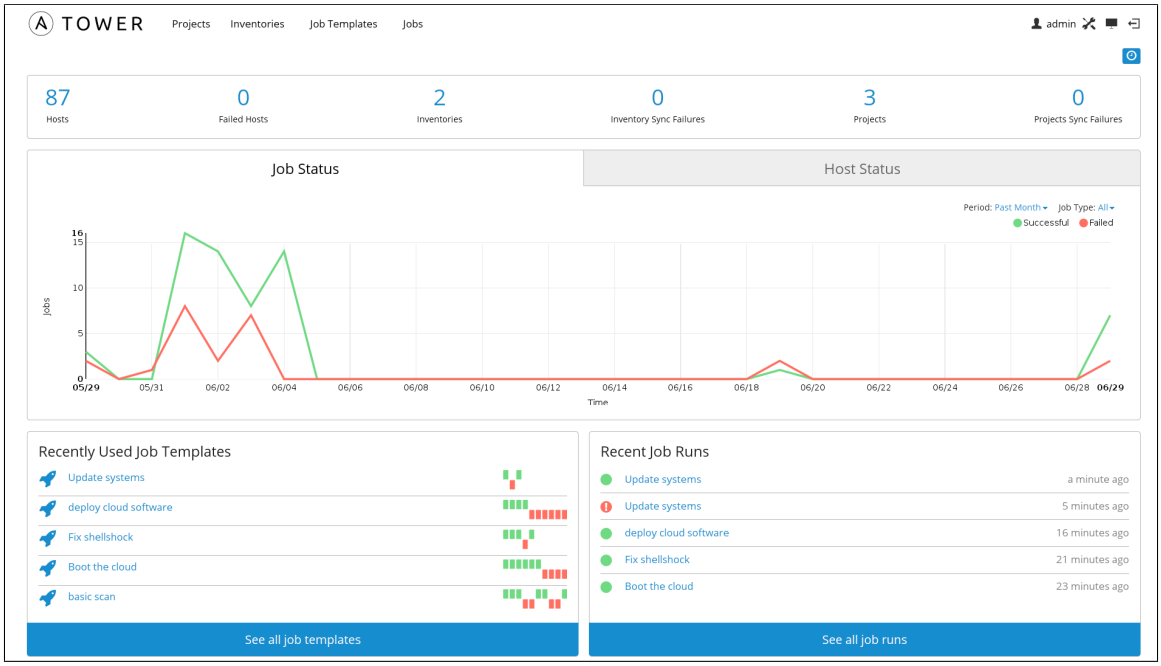

Figure 2.1: The Ansible Tower Dashboard [1]

Figure 2.2 is a fabfile that defines each task with a method. Fabric is relatively easy to deploy, yet with one significant drawback of being single failure point [27], which means that remote nodes are fault intolerant.

2.4.3 Chef

Chef is Ruby-based, which may require some proficiency in the language. However, since it provides a code driven approach which gives more control and flexibility over configurations, it was chosen for this project. Chef infrastructure is composed of three elements: workstation, chef server and remote nodes. Chef infrastructure is illustrated in Figure 2.3 [5]. Chef server works as a master, deploying one single configuration policy written a workstation onto remote clients. One can set up nodes to check periodically with the Chef server to make sure the latest configuration policy applies.

The primary reason of choosing Chef is its code-driven approach, which offers more control and flexibility over configuration policy. Due to large code bases, there is relatively rich collection of modules and mature ecosystem built around Chef. This leads to the other reason of choosing Chef—its Git [10] centered feature. The strong version control capability provided by Chef is paramount because configuration policy

```
from __future__ import with_statement
from fabric.api import *
from fabric.contrib.console import confirm

env.hosts = ['my_server']

def test():
    with settings(warn_only=True):
        result = local('./manage.py test my_app', capture=True)
    if result.failed and not confirm("Tests failed. Continue anyway?"):
        abort("Aborting at user request.")

def commit():
    local("git add -p && git commit")

def push():
    local("git push")
```

Figure 2.2: A example fabfile [9]

may require regular updates from a team of developers. Additionally, Chef is not single point of failure system. In other words, the deployment process continues to operate despite failures of single task execution, e.g. package installation.

2.5 Summary

This chapter reasons about the benefits of an automated deployment approach for scalable server setups by identifying the difficulties of doing so manually. In the next chapter, we will address fundamental principles identified in this project's implementation.

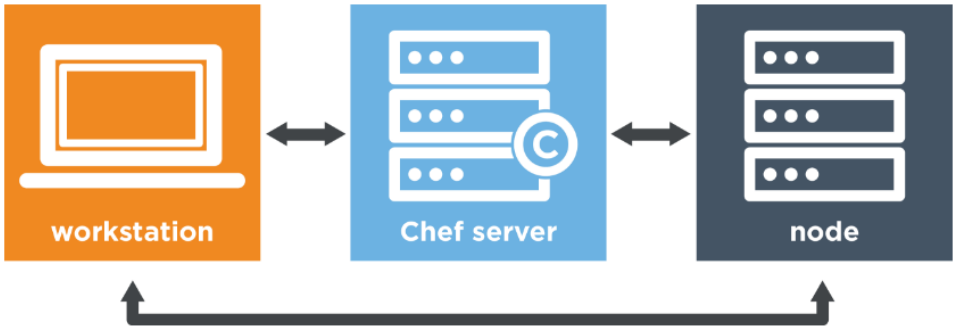

Figure 2.3: Chef infrastructure

Chapter 3

Design Principles

This chapter examines the current state of industrial VPN service providers and discusses the key principles we chose for our automated deployment tool. Chapter Three also looks at the key points of running a VPN service—data security and user legitimacy.

3.1 Experience with a VPN Service Provider

Before proceeding to the design of our architecture, various industrial VPN applications were examined. State-of-art VPN service providers share basic functions:

- Account management interface, be it a portal website, or computer/mobile application.
- Payment approach, either as an integrated function or linked external tool.
- Information display platform, where a user can obtain data such as network speed, server condition.

The result indicates the basic *client* side functions of a VPN application. However, the closed source nature of industrial VPN products makes it infeasible to exam *server* side functions. Yet, it is evident that most providers either overcharge for dedicated server setups, or fail to offer such service [7]. The reason behind this is the lack of automated deployment approach, causing extra manual steps for repeatable deployment. To this end, this project proposes automated deployment tools for server setups to cut the extra cost.

3.2 Principled Selection of Automated Deployment Tool

The search of a suitable deployment tool is not an easy task. As mentioned in Chapter Two, a variety of alternatives are available in the market. The following principles are followed in order to select the tool that suits the VPN applications we are considering:

- *Code-driven approach.* Complex configuration policies are involved in VPN servers, such as firewall setups. Often times, streamline code alone will not fully accommodate the requirement. Clear instructions in the right order need to be provided for configuration, complete code-based tools offer more control and flexibility over the setup process.

- *Version control feature.* Due to large code bases, along with frequent configuration policy updates, it is of extreme importance to keep track of code component versions. The benefit from version control may be minimal in managing small number of servers; however, as the number of servers grow, matching a new server to its original version of a configuration policy without a version control tool can be an absolute nightmare!

- *Good Integrity with database and other applications.* Machines that offer VPN services work closely with third-party software, such as OpenConnect server (Ocserv) [13]. Ocserv is an SSL-based VPN server. It implements the OpenConnect SSL VPN protocol, and has also (currently experimental) compatibility with clients using the AnyConnect SSL-based VPN protocol. Additionally, information related to users, servers and connections is stored on MySQL database [30], so automated tool needs to integrate tightly with MySQL.

Based on characteristics of the appropriate automated tool analyzed above, it was determined that *Chef* is the best candidate for the VPN project.

3.3 Target Users

As mentioned in previous section, this project focuses on expediting server setup process, which will in turn lower labor cost payable by users. In particular, users that demand dedicated/customized VPN machines are most applicable to our approach.

Many organizations allow remote access by employees to the corporate computer network via the Internet. Such remote connection is commonly implemented using a VPN [16]. These organizations are more likely to purchase dedicated VPN servers, and may benefit from server setup automation.

3.4 Principle of User Authentication

One key purpose of VPN service is to ensure the security of communication between a server and a client. Data packets are strictly private, cannot be eavesdropped by any third parties by any means. In adherence to such a principle, the Transport Layer Security (TLS) [19] protocol is adopted in the project.

TLS guarantees the *identity* of communicating party using public-key cryptography, the *privateness* of communication using data encryption, and the *integrity* of message by the attaching message authentication code to data packets. Table 3.1 illustrates potential attacks without the use of TLS [20]. In handshaking phase, server usually sends its public key in form of a digital certificate to a client. Upon successfully verifying that the public key is not altered, the client adopts TLS protocol throughout all sessions to prevent eavesdropping and tampering. In our project, all communication between gateway servers and clients are secured by the TLS protocol.

Type	**Threats**
eavesdropping	confidentiality
traffic analysis	confidentiality
message stream modification	authenticity, integrity
spurious association	authenticity

Table 3.1: possible attacks through communication

Another key point is to forbid illegitimate connections, i.e., *only* paid users have access to VPN services within subscription days.

There are two ways to authenticate a user—certificate authentication and password authentication. Password based authentication has been a major method over the past years, and it does have its advantages, such as being easy to deploy and easy to administrate. However it provides lower level of security and is vulnerable to password data storage attacks.

In this project, certificate authentication allows for a centralized management

system (CA) to monitor the status of each digital certificates. The cert [17] issued by CA is only valid for the amount of days that a user subscribes; it gets revoked automatically after expires. The revoked certs that is stored in Certificate Revocation List(CRL) [25] distributed to all servers, which prevents VPN services being accessed by users with expired certs. With certificate authentication, user legitimacy can be tracked without human interaction.

Furthermore, unlike password based authentication, certificate based authentication does not require a central database shared by all VPN servers to store sensitive password data. This not only eliminates the risk of data leak, but also saves the resources of sharing files among a large number of servers.

3.5 Summary

This chapter identifies the design principles followed in our VPN project. These principles will serve as a guideline for implementation, as shown in the next Chapter.

Chapter 4

General Project Architecture

This Chapter looks at our VPN service system at a higher level; carefully describing how to initialize a VPN connection from a user's perspective, the necessary steps run by the server behind scenes, and how to keep track the status of server and user's information using a MySQL database.

4.1 Key Features

This section highlights the key features of the project, from a user's perspective. Figure 4.1 demonstrates the general architecture of our proof-of-concept VPN project. Steps to initialize a connection as a new user are listed below:

1. *Account Management.* Users interact with VPN providers via an interface we call the *Gate Portal*. The front-end website is synchronized with the database, and provides close to real-time data, e.g. server availability, connection status, valid service days, etc. Users are able to revoke and request a new certificate through the portal website as well.

2. *Service Purchase.* Users can purchase desired service days through online payment tool—Paypal [14]. Upon receiving payment, a certificate for specific user will be generated.

3. *Email of Authentication Certificates.* A certificate is issued by an server-recognized Certificate Authority (CA), and is only valid for the number of days purchased. Certificates are distributed via an email service, and are disabled upon being triggered by a revoke request.

4. *Connect!* With valid certificate, user can easily connect to whichever server is shown available in the Gate Portal. Based on a device's OS, third-party connect tools may vary.

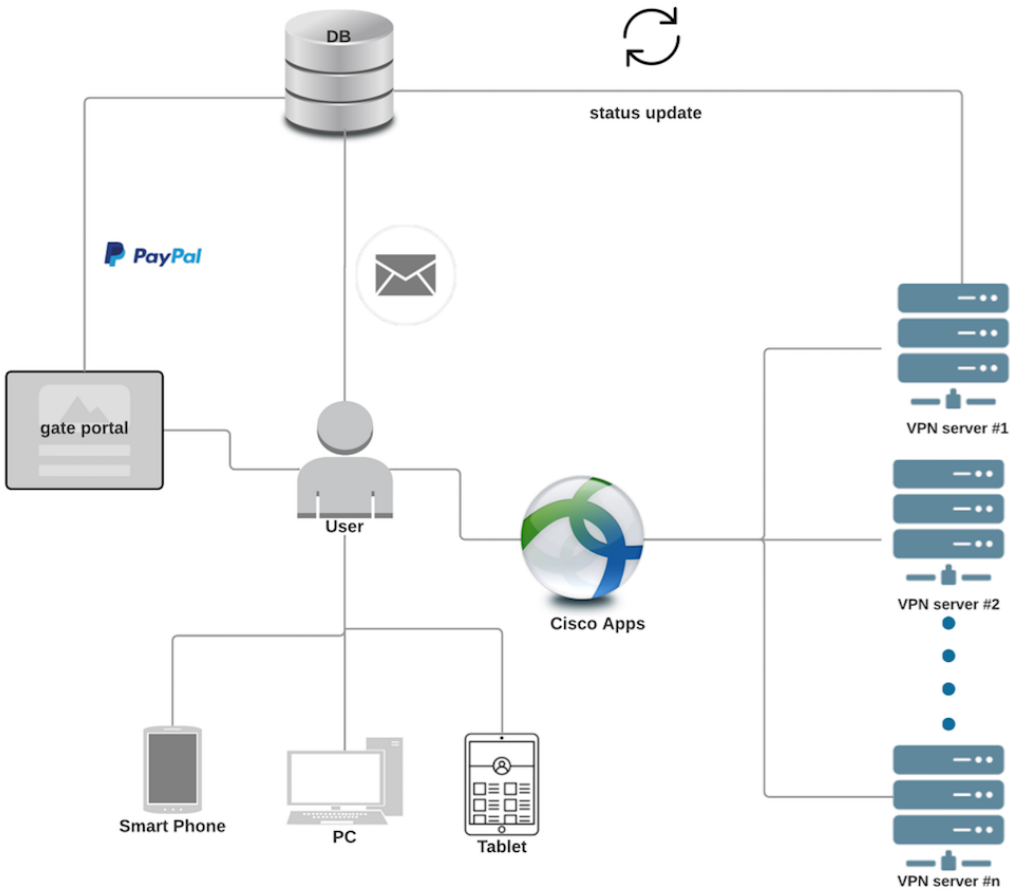

Figure 4.1: General Infrastructure

4.2 Client Side

This section continues with key features described above, detailing the functionality of each feature, and its main purpose in the project.

4.2.1 Gate Portal

The Gate Portal is a website that displays useful information such as status of servers, purchases service days, and a user guide. It is designed to help users manage their accounts and serves as a dashboard to provide useful information. Figure 4.2 captures the dashboard page showing paid service days left, bandwidth usage, available server lists, etc. This data is retrieved from a MySQL database. The Gate Portal is currently hosted on *yonglab.com* at port 8000.

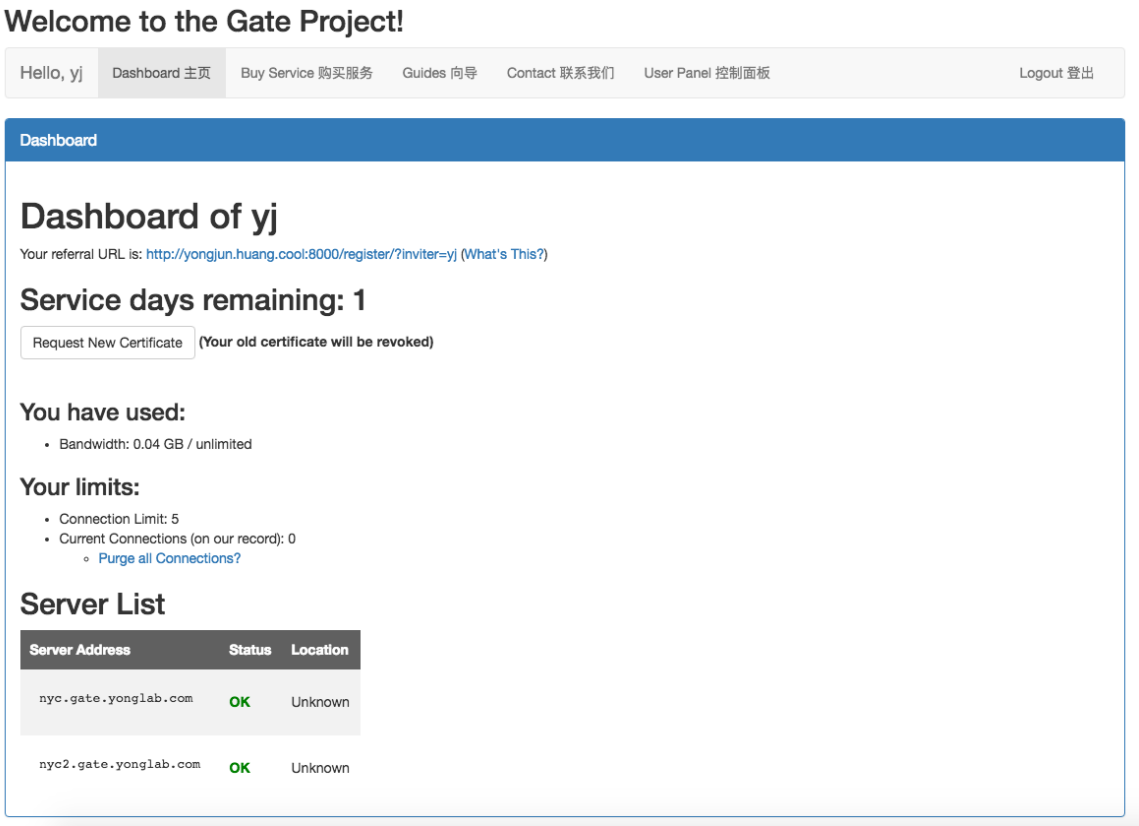

Figure 4.2: Dashboard of user "yj"

4.2.2 Paypal

To obtain a VPN service, the user has to purchase service days via online payment tools. Paypal is easy to integrate with a web portal using REST APIs[28]. What comes with it is good coverage among western countries. Therefore it is chosen as major online payment tool in the project. With a successful transaction, newly

purchased days will be added to database, while certificates with an extended expiry date will be issued.

4.2.3 Certificate Distribution via Email

In order to login to servers, users need a certificate issued by the CA and recognized by all VPN servers. Upon successful payment, a valid certificate will be generated and distributed to paid users via email. Figure 4.3 illustrates the four attached certificates, including private key delivered to intended user via their account email address. There are security concerns in distributing confidential files such as certificates via email, which we will address in next chapter.

Figure 4.3: An sample email with certificates attached

4.2.4 Connect Tool

Suppose a set of servers that provide a VPN service are available and a user has obtained their valid certificate to logon. It is necessary to associate the servers with a user using a connect tool. A connect tool joins a designated VPN by providing a gateway address, based on the successful verification of certificates. Devices of various platforms need access to VPN services, so a connect tool must be available on all of these platforms. Table 4.1 illustrates commonly used tools for each of the corresponding platforms.

Platform	**Connect Tool**
Android	Cisco AnyConnect[22]
Windows	Cisco AnyConnect
Linux	OpenConnect[12]
iOS	Cisco AnyConnect
Mac OSX	Cisco AnyConnect

Table 4.1: Platforms vs Connect Tools

4.3 Server Side

This section describes components related to set up a dedicated VPN server, using fully automated deployment by Chef. It starts off by listing applications integrated with Chef, then focuses on detailing Chef infrastructure.

4.3.1 Overview

- *DigitalOcean* provides virtual servers in cloud infrastructure to software developers. Note, since configuration is targeted at CentOS in our system, the virtual server needs to be of the same type.

- *Git* provides version control for Chef source code. Figure 4.4 lists part of configuration code for this project on Gitlab.

- *CloudFlare* provides the Domain Name Server (DNS) service in this project. CloudFlare is responsible for populating domain name specified in environment json files with IP address of newly boosted server.

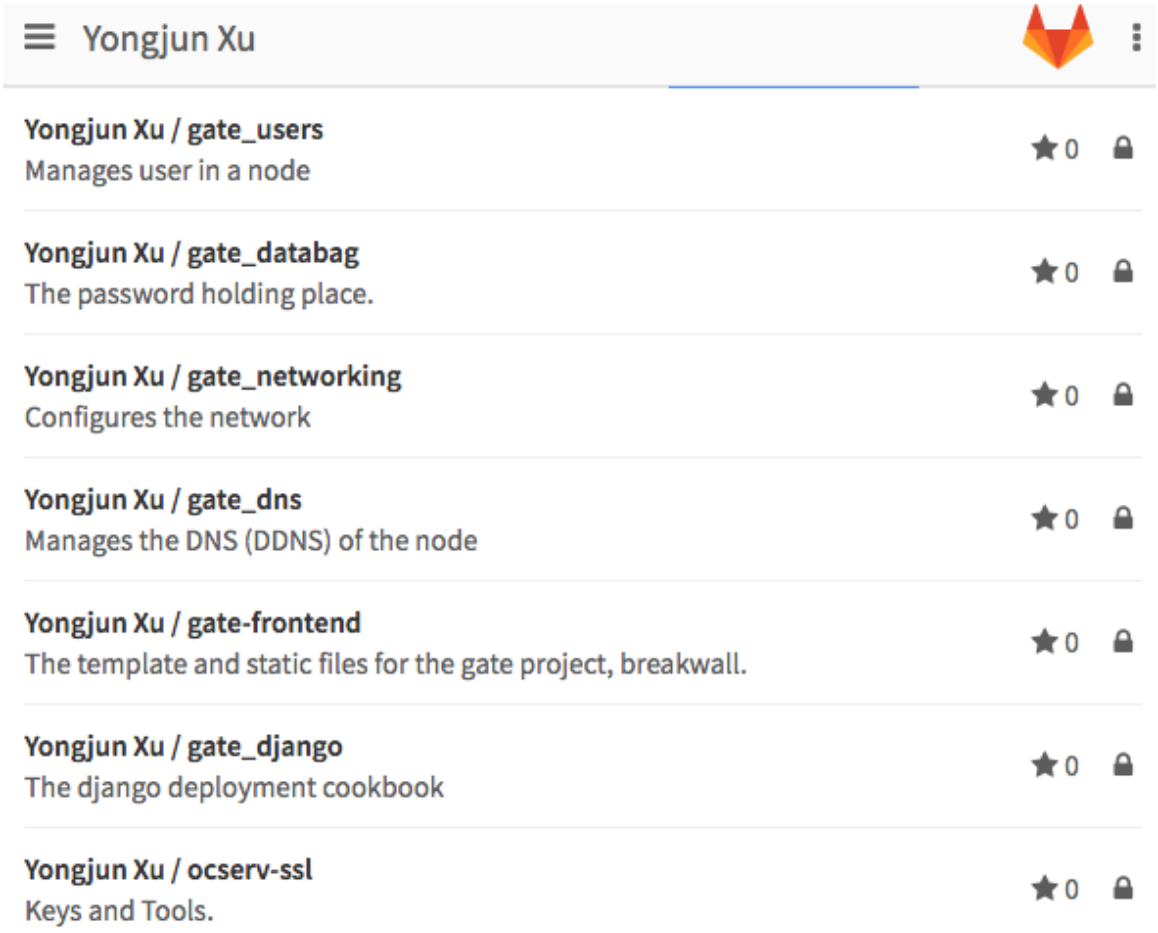

Figure 4.4: Cookbooks on Gitlab

4.3.2 Chef Infrastructure

This section covers basic elements in Chef infrastructure and definition of terms and how they apply in the context of this project.

Concepts & Terms

There are a list of basic terms used by Chef, refer more at Chef official document [4].

- *Cookbook* defines a scenario and contains necessary things required to support a given scenario. Things like recipes, that specify the resources to use and the order of execution, attribute values and so on. Simply put, it is the container where instruction code or configuration policies are put.
- *Environment* is a way to define the variable that can be configured and managed

for a *single* node. In this project, each VPN sever has a unique environment JSON file, to specify customized attributes. A perfect example attribute is the server's hostname.

- *Role* is a way to define certain patterns that apply *across* nodes in an organization instead of one single node. In this project, typical attributes are configuration path, shared secret file[1], etc.

Chef Workflow

Typically chef is comprised of three elements: workstation, Chef server and nodes.

- *A Workstation* is the computer which you author your cookbooks in, and administrate on daily basis. All cookbooks on a workstation are managed by Git for version control. The hostname of the workstation for this VPN project is yonglab.com.
- *Chef server* acts as a central repository for your cookbooks as well as for information about every node it manages. For example, the Chef server knows a node's fully qualified domain name (FQDN) and its platform [5]. The Chef server in this project is hosted at *https://chef-server.gate.huang.cool/*.
- *A Node* is any physical or virtual machine that is managed by a Chef server. In our case, it is the VPN machine. The Chef server communicates with every node through the Chef client[2].

Chef workflow is illustrated in Figure 4.5 [6]. Chef code that defines configuration policy are written from workstation, then uploaded onto Chef server using the Knife command. Chef server applies that policy to a node, via the Chef-client installed on that node.

4.4 Bridge In Between: Database

While Django [8] framework comes with a built-in database, for reliability concerns, an independent MySQL database is chosen to prevent any data crash due to website misuse. Table 4.2 illustrates part of the information collected from both the gate

[1] can be treated as password file

[2] an application needs to be installed on the Node to talk to Chef server

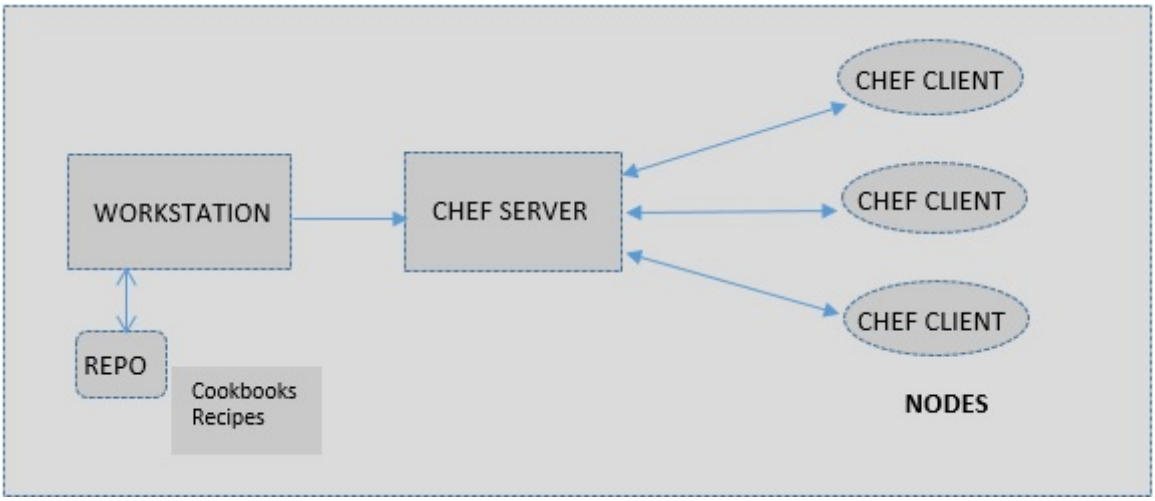

Figure 4.5: Chef workflow

portal and the node server. This data is collected close to real-time through the web portal and scripts run on VPN servers.

For instance, the connect/disconnect scripts are deployed on VPN servers during the setup process; they collect data such as bandwidth usage, current active users upon connection/disconnection. Such information is later displayed in the web interface for users. On the other hand, scenarios where the server needs to determine whether total connections from one user reaches its maximum number, the database comes into play by providing this information. The database not only stores essential information on both users and servers, it also acts as a middle-man in between.

Entity	**Attribute**	**Note**
user	username, email, password, etc.	registered user
server	hostname, port, max_connection, etc.	available servers
quota	bandwidth, connection_limit, etc.	VPN usage per user
connection	server_id, ip address etc.	active connection entries

Table 4.2: Table structure in DB

4.5 Summary

This chapter provides an overall architecture of our proof-of-concept VPN project. The next chapter will showcase our VPN server setup and analyze its behaviour.

Chapter 5

Evaluation and Future Insight

This Chapter explores both the proposal of automated deployment approach and its proof-of-concept implementation, the VPN project.

5.1 Contributions

The advancement we explore is the change of deployment approach. This project attempts to automate system deployment process for VPN service, in order to eliminate the cost of building a new environment on existing configuration. Part of our implementation involves testing the successful deployment of a valid service. The VPN service is the most fundamental networking functionality, thus was chosen in presenting the result of automated deployment. Additionally, there are some side components of project: portal website, payment tool, certificate distribution by emails and VPN connect tools. Industrial VPN providers may bundle these components into a presentable and cohesive application, which we did not attempt to do in this proof-of-concept.

5.2 Manual versus Automated Deployment

This section demonstrates the easy setup of our VPN server using automated deployment, compared to doing so manually.

Manual configuration of a VPN is a lengthy process. Some necessary steps are highlighted as follows.

1. Install Connect Application: OpenConnect server.

2. Configure OpenConnect server, e.g. default DNS server, redirecting traffic through OpenVPN.

3. Generate Keys & Certificates. Install easy-RSA to generate server key and certificate, place it under OpenVPN directory.

4. Networking Configuration. Install IPtables service and configure routing tables accordingly. Enable IP forwarding in sysctl.

5. System Configurations. Set hostname, time zone, configure fail2ban, etc.

6. Synchronization with DB. Add server entry, log on server data.

7. Synchronization with Certificate list.

8. Install dependency packets, e.g. crond, rsync, net-tools.

9. Update Domain Name System with new IP address.

Each step requires more than just one line command to execute, and constant checking of system failures. Moreover, manually setting up large numbers of servers requires running all these steps repeatedly. This will increase the chance of having accidental misstep, which can be hard to trace and reproduce.

On the other hand, automated deployment setup only requires one-time design. It takes little to none effort to configure another server in the same way. The following demonstrates how easy it is to get a new VPN server up and running.

1. *Knife command.* Interface between local Chef-repo and Chef-server. Update configuration policies from workstation to Chef server: cookbooks, environment, role, as illustrated in Figure 5.1.

2. *DigitalOcean—Cloud server purchase.* Note, select OS distribution that is compatible with configuration requirements in Chef-repo. Figure 5.3 shows a newly booted machine, along with its system configuration.

3. *One-line command VPN setup.* Make sure the IP address is accurate, then anything else will follow. The code snippet shows the one-line command. Figure 5.4 captures part of package dependencies installing on new server.

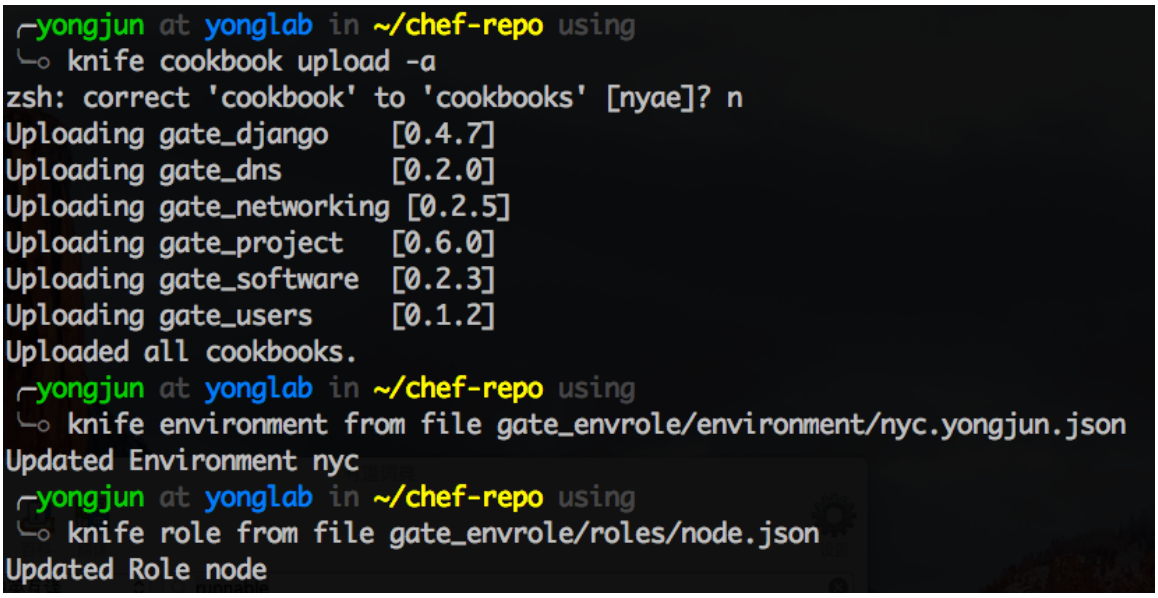

Figure 5.1: Knife command for update

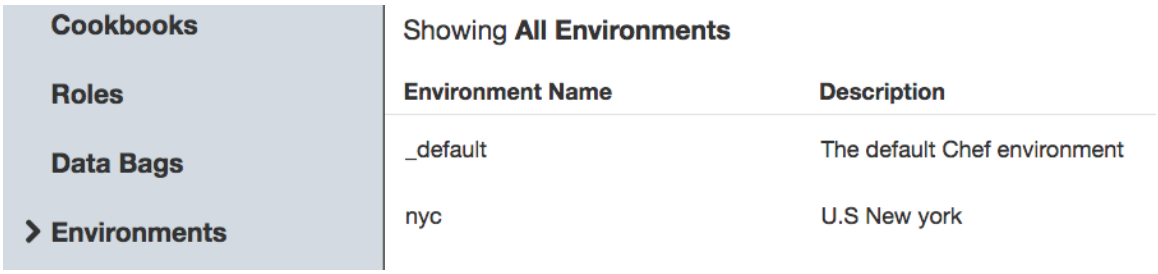

Figure 5.2: Web interface showing cookbooks in Chef server

```
sudo knife bootstrap 67.205.139.156 -E 'nyc' -N 'nyc' -x root
    --secret-file /home/yongjun/chef-repo/gate_databag/gate/secret
    --node-ssl-verify-mode none -r 'role[node] recipe[gate_project]'
```

These three steps are all you need to set up a brand new working VPN server. The new server automatically put the valid/revoked user on record. Figure 5.5 shows all available servers user has assess to (only one in this case). Automated deployment also takes care of DNS registration with new IP address in CloudFlare, adding new server entry in database, bandwidth auditing upon connect/disconnect, etc.

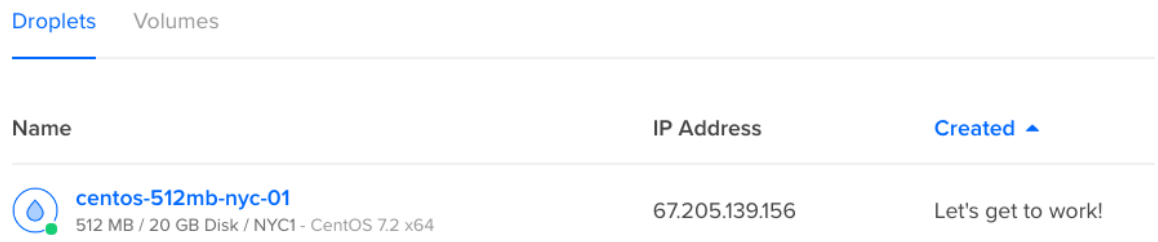

Figure 5.3: A droplet available

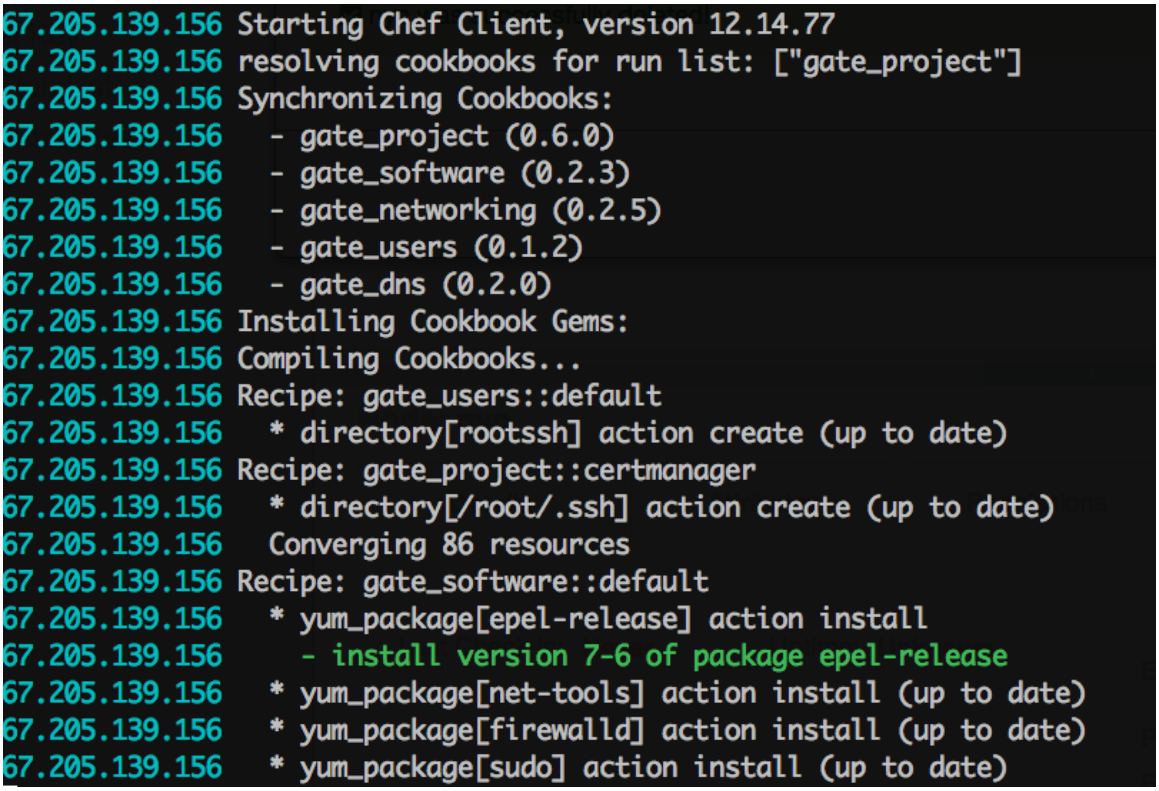

Figure 5.4: System output from VPN server shown in workstation

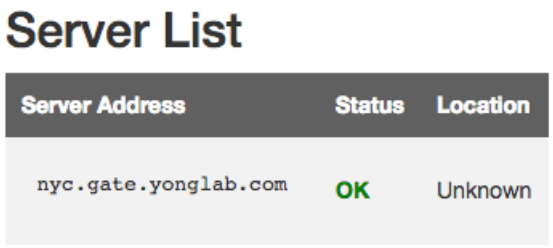

Figure 5.5: Server list in user dashboard

5.3 Limits of this VPN proof-of-concept Project

The project is successful in building basic VPN functionality and providing a proof-of-concept for an automated deployment approach. Our experience shows this is indeed an improvement to existing manual approaches. However, with the benefits in terms of time and resource constraints, there are still some security vulnerabilities that need to be addressed.

First and foremost, SMTP [23] servers and clients normally communicate in the clear over the Internet. In many cases, this communication goes through one or more routers that are not controlled or trusted by either entity. Such an untrusted router might allow a third party to monitor or alter the communications between the server and client. Therefore, distributing login certificate via email is subject to active attacks and monitoring.

A compromised solution is to use an SMTP Extension—Secure SMTP over Transport Layer Security [24]. This way, only trusted routers or SMTP agents are able to communicate with one another. However, this approach is still vulnerable to Man-in-the-middle attacks; moreover, the fact that content is not encrypted in email communication makes the content easily intercepted.

A better solution is to download the certificate as an attachment in portal website. All communication is protected by HTTPS protocol, therefore encrypted and authentication required. This feature is only enables with valid login credential, which guarantees the certificate is received by a legitimate user only.

As described in Chapter 3, certificate based authentication allows for more secure login. However, what comes with security is extra setup procedures. For instance, X.509 certificate validity cannot be extended without re-issue. The reason is that the certificate's hash is calculated after the rest of the certificate is written, editing validity date field would cause the certificate's hash to change. If the hash is changed anyone else checking the certificate will know it has been altered. For this reason, users are required to install a new certificate each time they want to extend the expired date. Currently there is no better alternative except downgrading to a password based login.

5.4 Future Insight

Regular VPN providers do not distinguish nationwide traffic from global traffic. A request for local website takes up VPN resources even it does not need to bypass Chinese Firewall. One future focus is to separate requests for global websites from local websites in China. This advanced feature could be implemented based on IP address. Not only does this save unnecessary cost, it will also improve surfing performance as packets travel shorter distance.

5.5 Summary

This chapter describes the initiative of the project, discusses the improvement accomplished through automation, and identifies what else can be done in the future. The next and final chapter concludes the project.

Chapter 6

Conclusions

The purpose of this VPN project is to explore the idea of automated deployment. We show a proof-of-concept by automating the process of deploying configuration policies onto cloud servers, so that they are setup properly to provide an application specific VPN service. An automated deployment tool, Chef, is used along with other helper services including Git and CloudFlare.

We have shown how automating deployment processes greatly minimizes accidental misconfiguration and lowers the cost of repeatable tasks. We discovered that dedicated VPN server setup, in particular, can benefit from automated deployment, and that such a setup service is lacking in the market. This project was evaluated within a live system. Our infrastructure includes client interface portal website, payment modules and intermediate database storage. The particular deployment is opensource and available for public use[1].

This project is an attempt at automating configuration deployments, with a concrete example of easily configuring cloud servers into VPN service providers. In the future, load balancing strategies between VPN servers can be developed for performance improvement as the number of users grow. It is important to note that the functionality of a targeted server does is not limited to a VPN service only, but open to all kinds of possibilities.

[1] https://gate-project.com/

Bibliography

[1] Ansible Tower App Screenshot. https://www.ansible.com/. Accessed: 2016-09-25.

[2] Ansible Wikipedia. https://www.wikiwand.com/en/Ansible_(software). Accessed: 2016-09-25.

[3] Chef. https://www.chef.io/. Accessed: 2016-09-25.

[4] Chef Document. https://docs.chef.io/cookbooks.html. Accessed: 2016-09-25.

[5] Chef infrastructure. https://learn.chef.io/manage-a-node/rhel/. Accessed: 2016-09-22.

[6] Chef Workflow. http://cloudacademy.com/blog/the-5-best-tools-for-aws-deployment/. Accessed: 2016-09-23.

[7] Dedicate VPN server overcharge. https://blog.serverdensity.com/saving-500k-per-month-buying-your-own-hardware-cloud-vs-colocation/. Accessed: 2016-09-25.

[8] Django. https://www.wikiwand.com/en/Django_(web_framework). Accessed: 2016-09-25.

[9] Fabric. http://www.fabfile.org/. Accessed: 2016-09-25.

[10] Git. https://git-scm.com/. Accessed: 2016-09-25.

[11] How Chef Performs Configuration Management. https://www.chef.io/solutions/infrastructure-automation. Accessed: 2016-10-19.

[12] OpenConnect Application. `http://www.infradead.org/openconnect/`. Accessed: 2016-09-25.

[13] OpenConnect VPN Server. `http://www.infradead.org/ocserv/`. Accessed: 2016-09-13.

[14] Paypal. `https://www.paypal.com`. Accessed: 2016-09-13.

[15] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy H Katz, Andrew Konwinski, Gunho Lee, David A Patterson, Ariel Rabkin, Ion Stoica, et al. Above the clouds: A berkeley view of cloud computing. 2009.

[16] Chen Yehezkel Burshan. Connecting vpn users in a public network, July 28 2009. US Patent 7,568,220.

[17] Dave Cooper. Internet x. 509 public key infrastructure certificate and certificate revocation list (crl) profile. 2008.

[18] Douglas Crockford. The application/json media type for javascript object notation (json). 2006.

[19] Tim Dierks. The transport layer security (tls) protocol version 1.2. 2008.

[20] Kevin R Fall and W Richard Stevens. *TCP/IP illustrated, volume 1: The protocols*. addison-Wesley, 2011.

[21] Paul Ferguson and Geoff Huston. What is a vpn?, 1998.

[22] Americas Headquarters. Cisco anyconnect secure mobility client administrator guide. 2010.

[23] Paul Hoffman. Smtp service extension for secure smtp over transport layer security. 2002.

[24] Paul Hoffman. Smtp service extension for secure smtp over transport layer security. 2002.

[25] Russell Housley, W Polk, Warwick Ford, and David Solo. Internet x. 509 public key infrastructure certificate and certificate revocation list (crl) profile. Technical report, 2002.

[26] Jez Humble and David Farley. *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education, 2010.

[27] Gary S Lynch. *Single point of failure: The 10 essential laws of supply chain risk management*. John Wiley and Sons, 2009.

[28] E Michael Maximilien, Hernan Wilkinson, Nirmit Desai, and Stefan Tai. A domain-specific language for web apis and services mashups. In *International Conference on Service-Oriented Computing*, pages 13–26. Springer, 2007.

[29] Peter Mell and Tim Grance. The nist definition of cloud computing. 2011.

[30] AB MySQL. Mysql, 2001.

[31] Mark Turner, David Budgen, and Pearl Brereton. Turning software into a service. *Computer.*, 36(10):38–44, 2003.