# 9-Credit Practicum Proposal

BCIT Bachelor of Technology Program

Network Security and Administration Specialization

Submitted by: ********** (A00******)

June 14, 2009

# NOTICE

This document has been edited from its original to remove personally identifying information. It may be used by BCIT in any capacity that assists education.

Student Experience 4
Education 4
Work Experience 4
Specialization 4
Project Information 5
Project Description 5
Project Background and Context 5
Scope and Depth 6
System Firewall 6
Host Firewalls 7
Redundant Web Servers 7
JEE Connector 7
Web Server Load Balancer 7
Application Server Load Balancer 8
Deployment 8
Security 8
Innovations and Unfamiliar Technology 9
Project Plan 10
Proof of Concept 10
Production Environment Deployment 11
Testing 11
Documentation 11
Test Plan 11
Production Security Testing 11
Production Operational Testing 12
Load Testing 13
Time Estimates 13
Appendix 15

# Student Experience

## Education

2001 - 2002 BCIT Software Systems Development program

2002 - 2009 BCIT Bachelor of Technology program

## Work Experience

I've been building large-scale software applications for 12 years. Most of the projects I've worked on in that time have had a strong emphasis on security and were primarily used in the areas of financial transaction management or identity management and authentication.

In 1998 I helped found a technology startup that managed massive repositories of digital assets for Fortune 500 companies. The experience I gained there in scalable application architectures was invaluable.

In 2002 I worked for a Vancouver company that managed financial transactions for online gaming. Transactions spanned international banking systems and had to be nearly real-time. Security and availability were paramount.

In 2004 I began working at a Vancouver startup focusing on identity management. Partners and clients included US and Canadian government, several Fortune 500 companies and the industry leader in search engine technology. Here I wrote the first software implementations of two well-known identity protocols: ******* and *******. I also designed and implemented a high-availability API service that is today used by over 500,000 users.

## Specialization

Nearly all my work in software development has had a strong focus on large, scalable, and reliable applications that require high levels of security and/or identity management. A large application naturally presents a larger surface for attackers, and managing this threat while providing useful, highly-reliable applications is an interesting problem I enjoy working on. My experience in this domain has been exclusively on the application side of the solution. I chose the Network Administration and Security specialization of the BTech to round out my skills on the the network side of the solution.

# Project Information

## Project Description

The subject of this 9-credit practicum is a portion of the supporting infrastructure required for a secure, highly-available and highly-scalable web-based application.

The components of this supporting infrastructure that fall within the scope of this practicum are firewalls, load balancers, and redundant web servers. These are the traditional components of a scalable, tiered system. The innovations and challenges this practicum will deal with are presented by the environment in which all components are deployed: inside a purely virtualized and on-demand third-party infrastructure. Environments of this type are often dubbed "cloud computing" and have become a cost-effective way to deploy web-centric applications. Examples of such infrastructures are Google's App Engine[1] and Amazon's Amazon Web Services[2] (AWS). This project will be deployed within the Elastic Compute Cloud (EC2) environment of AWS. EC2 provides a XEN[3]-based virtualization environment in which each component runs within a virtual machine, or more accurately, as an instance of a machine image that will be created as part of this project. This adds significant complexity and numerous challenges to a deployed production application such as the one that is the focus of this project. All such issues are outlined below in *Scope and Depth*.

## Project Background and Context

The proposed practicum infrastructure supports a complex financial services application. The details of the application are beyond the scope of this project however a brief description of its nature follows.

The application is an online business-to-business service that helps companies to settle accounts receivable with clients with whom they have an existing business relationship. Application services are available to users via two routes: through a browser-based interface, and through an application programming interface (API). The former allows users to manage operation directly, the latter is for machine interface and provides access to financial and accounting systems.

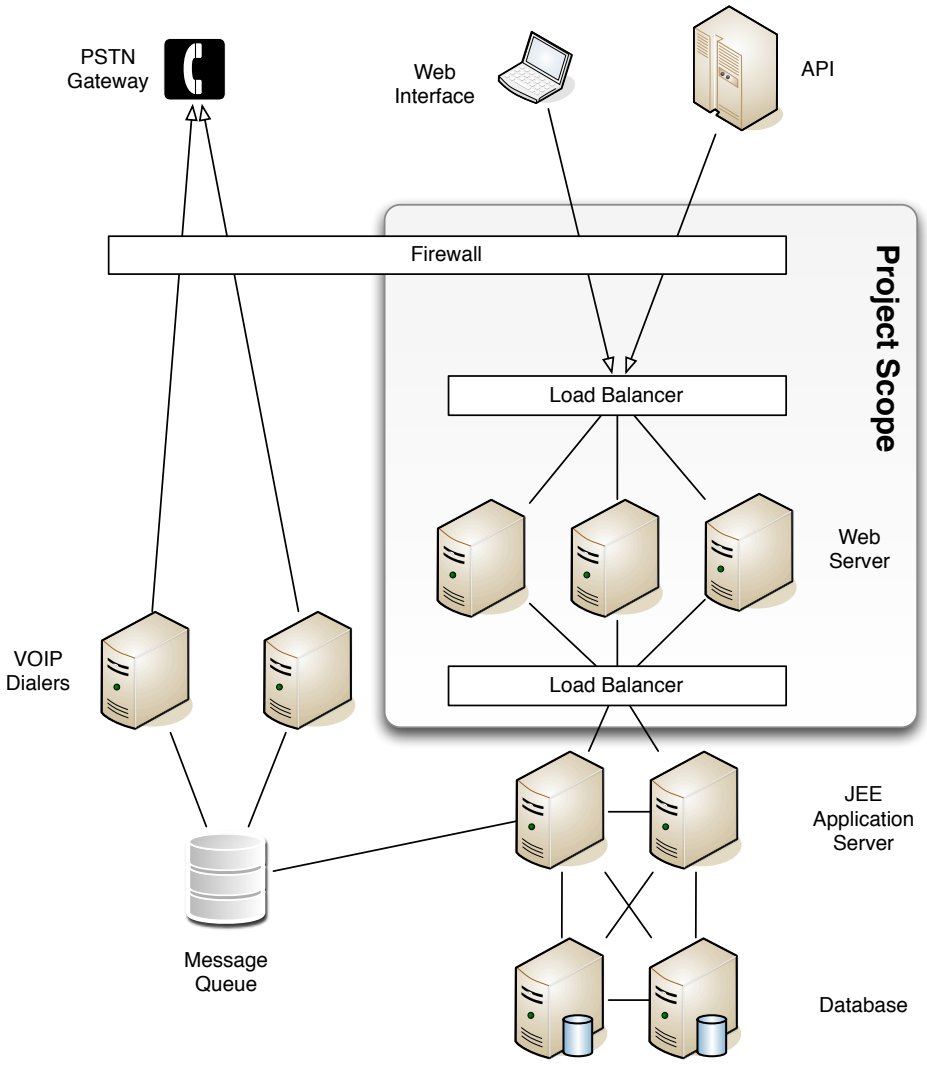
---

[1] http://code.google.com/appengine/

[2] http://aws.amazon.com/

[3] http://www.xen.org/

# Scope and Depth




The scope of this practicum is illustrated above. It will focus on the following components of this application architecture:

## System Firewall

This is the mandatory firewall required by EC2. It provides typical packet filtering abilities, as well as access control based on EC2-specific access control lists (ACLs). This firewall resides outside of any virtual machine instance.

## Host Firewalls

Individual machine instances will also be firewalled by iptables for defense-in-depth.

## Redundant Web Servers

Each web server is an instance of the same machine image. The host operating system will be Debian Linux. The web server has yet to be determined but will be selected based on the following three criteria: security, stability, performance. The two candidates are the Apache webserver[4] or Nginx[5].

Webservers are deployed in parallel with minimum of two hosts running at any time, but this number is intended to be dynamic. More instances will be made available to serve requests as load increases, and instances will be terminated as load decreases. The three servers shown in the architecture diagram serves as an example only. In production, the actual number could be lower or higher depending on the demand at the time. This dynamic host availability is one of the key benefits of virtualized deployment.

## JEE Connector

Web servers only serve static content. Application logic is handled by Java Enterprise Edition (JEE) servers that are located behind the web servers. Although these JEE servers are outside of the scope of this project, traffic to them is proxied through the web servers via a JEE connector. These proxy-enabling connectors are within the scope of the project. Which connector will be used will depend on the selection of the web server, but it must meet the following requirements:

1. Only specific URL components must be proxied. If the sought resource is also not a static resource that can be handled by the web server, a 404 error must be returned.
2. Proxied responses must in no way appear different than those served directly by the web servers.
3. A proxy failure must not interfere with the serving of static resources.

## Web Server Load Balancer

This load balancer resides in front of the web servers and distributes load among them. Which implementation of load balancer will be used here will be the subject of research. Options include a proprietary fee-based offering from Amazon which is essentially a virtualized hardware load balancing appliance, as well as host-based software implementations such as HAProxy[6]. Which option is chosen will depend on which best satisfies the following requirements:

1. Balancing must scale dynamically according to the number of web server instances deployed.

---

[4] http://httpd.apache.org/

[5] http://nginx.net/

[6] http://haproxy.1wt.eu/

2. Load balancing must use an intelligent load-based algorithm (not round-robin).

## Application Server Load Balancer

This load balancer resides between the web servers and the application servers. The Amazon implementation is not available here so a host-based solution will be chosen. It must meet the same requirements as for the web server load balancer.

## Deployment

Unlike a traditional hardware-based server, virtual machine instances are ephemeral. If an instance is terminated deliberately or accidentally, all information in memory and all information written to disk since system boot are lost. This is problematic for a system that requires persistence for databases, logs, system and application updates, etc.

Considerable research will go into choosing and implementing acceptable solutions to this situation. Details of the solutions will be presented as part of the practicum deliverables.

## Security

The proposed application will run entirely on third-party infrastructure, adding to the already large security measures that must be taken to secure applications, traffic, and sensitive information in an online application. The following are the security measures that will be implemented:

1. Machine images will be created on local hardware from trusted installation sources.
2. SSH certificates with passwords will be the only means of accessing instances. Root SSH access will be denied.
3. Hard drives, databases, logs and backups will be encrypted using AES-256
4. Hosts will be hardened using Bastille[7]
5. Only the minimum required services will be running. Only the minimum required software will be installed.
6. Hosts cannot operate in promiscuous mode in the EC2 virtualized environment, so no dedicated IDS can be deployed. I will undertake testing to determine the performance costs of running an IDS such as Snort on each production host and determine the feasibility of doing so in production.
7. Firewalls will permit only the minimum essential services and will deny all other traffic. Unusual, malformed and obviously malicious traffic will also be blocked.

---

[7] http://bastille-linux.sourceforge.net/

8. Amazon's security white paper will be studied and any additional recommendations and best practices will be implemented.

## Innovations and Unfamiliar Technology

The innovative aspects of this project are derived from implementing the standard components of a typical secure, reliable and scalable web application in a virtualized environment. Virtualization introduces several challenges not found in other deployments. As described in *Scope and Depth*, several unique issues arise in the areas of security, scalability and deployment.

I will be using several technologies in this project that are currently unfamiliar to me:

- Virtualized servers, or "cloud computing", especially EC2 instances of servers running on Xen.
- Load balancing in general. Load balancing on virtualized hardware or software balancers in particular.
- Running redundant servers.
- JEE proxying via web servers.
- Production-scale firewalling.
- Production-scale intrusion detection system (IDS) feasibility assessment and deployment.
- Security analysis of a production software system, especially one exposed to the internet.

# Project Plan

This project has been broken down into four stages. Each stage offers an overview of key points and milestones. For a more detailed stage breakdown, see the accompanying project plan in the appendix.

## Proof of Concept

Given the number of unknown technologies that must eventually be incorporated into a production environment, the safest and most efficient approach is to research and explore these technologies in a proof-of-concept environment. Since virtual machines must be built from scratch into a special machine image without the Linux kernel and then uploaded into AWS, it is far more efficient to build, upload and deploy the initial image, and then make changes to it running in the EC2 environment rather than build and upload a new image after change. Each component of the system will be added iteratively in this manner until all unfamiliar technologies have been tried and shown to work, and all alternate technologies have been tested and a best option selected.

The following key activities must be completed:

1. Build test AMI machine image. This is primarily a research activity. The AWS APIs, the EC2 environment, Xen virtualization, instance configuration and communication, etc are all complex topics with which I must become familiar before I can get a host running.

2. Select and deploy a web server into the test image. A web server must be chosen to serve static content and proxy application requests to the JEE server. The lightweight server Nginx is being considered in addition to Apache because of the constrained operating environment. Security, reliability and performance will be evaluated through research and first-hand load testing.

3. Select and deploy a web server load balancing solution. A virtualized hardware service offered by Amazon is being considered, as are software solutions such as HAProxy. See requirements under *Scope and Depth.* Research and load testing will be used to select the best option.

4. Select and deploy an app server load balancer. The same requirements and procedures as for the web server load balancer apply here, except that the Amazon solution is not available as an option.

5. Select and deploy a JEE connector. See requirements at *Scope and Depth*. The proxy connector chosen depends on the web server chosen. Research and load testing will be used to select the best alternative.

6. Configure firewalls. There are two types of firewall relevant to the scope of this practicum: the mandatory system firewall provided by Amazon, and the host firewalls on each web server. Details of their requirements are listed under *Scope and Depth* and *Security*.

7. Select and deploy system and application deployment solutions. Virtual machine instances boot from the original image each time, so all information created between boots is lost unless written to external media. A significant amount of effort will go into researching solutions to this problem. Options include revision control systems such as SVN or CVS, mountable devices, mirroring and Amazon's S3 service. The best option will be selected based on security and reliability.

8. Select and deploy image persistence solution. The same options and criteria applied to persistence in system and application deployment apply here.

9. Secure test image. Here options such as drive encryption, IDS, host hardening and secure communications will be implemented, or researched and evaluated for feasibility in a virtualized environment.

### Production Environment Deployment

The production environment takes what was learned during the proof-of-concept stage and applies it in the construction of a production-ready environment. The environment is hardened from a security standpoint and optimized for reliability and performance. Specifically, unneeded services and binaries are removed, web servers are chrooted, Bastille is run against the system, setuid permissions are removed where unnecessary, etc. The host is then put into service as a production-capable component of the overall system.

### Testing

This phase tests the production environment from both a security and an operational standpoint. A great deal of effort goes into this phase as it provides the best insight of all phases into the risks borne by the system. See *Test Plan* and the the project planner in the appendix for more details on this phase.

### Documentation

During this phase all the notes and observations compiled during the previous activities are organized into formal system documentation. The most relevant artifact produced by this phase as far as the practicum is concerned is the final report.

## Test Plan

### Production Security Testing

These tests recreate the procedures performed by crackers in an attempt to assess risk to the production system.

1. Vulnerability assessment. Vulnerability scanners such as Nessus[8] and Nikto[9] will be used to identify known vulnerabilities in the internet-facing web servers.

---

[8] http://www.nessus.org/

[9] http://cirt.net/nikto2

2. Network scanning. Recreating reconnaissance with malicious intent, I'll attempt to determine as much as possible about the production network using commonly-available means such as network scanners and custom packet crafting.

3. Reconnaissance assessment. I'll attempt to gather as much information about the system using public information repositories such as domain registrars, web searches and directories, and social engineering techniques against partners, providers, vendors, etc.

4. System Firewall testing / ACL functionality. Using port scans, custom-crafted packets and buffer overflow attempts, I'll try to gain unauthorized access to resources blocked by the Amazon system firewall.

5. Host Firewall Testing. Using the same techniques used against the system firewall, I'll try to gain access to unauthorized resources on the web servers.

6. Exploit testing. I'll attempt to execute each known exploit against any vulnerabilities uncovered during the earlier assessment.

7. DOS testing. I'll attempt to prevent the system from serving legitimate requests by sending it crippling levels of traffic, malformed-packets, buffer overflows, etc.

## Production Operational Testing

These tests simulate normal and abnormal operational states that may be encountered during production operations.

1. Load/stress testing. Forces the system to work under greater and greater load until its operational limits are met or exceeded. Performance and failure behavior is noted. See *Load Testing* for details.

2. Failover/availability testing on web server load balancer. Test load balancing capability by monitoring traffic to individual hosts during heavy load. Test failover by taking a web server offline and verify that its traffic is routed to available web servers.

3. Failover/availability testing on app server load balancer. Same test procedure as for web server load balancer testing.

4. Test dynamic web server scaling on load. Add new web server hosts to a cluster already under load. Verify that traffic is routed to it in even proportion.

5. Test database persistence on instance crash. Verify persistent storage by terminating a machine instance and restarting from the image. No loss of data may occur in the database.

6. Test log persistence on instance crash. Verify persistent log storage by terminating a machine instance and restarting from the image. No loss of logs may occur.

7. Test system update persistence on instance crash. Verify that any system and application updates are automatically reapplied after an instance is terminated and subsequently restarted from its image.

8. Integration testing of web UI and API. Use JMeter [10] and Selenium[11] to verify the proper operation of the system's interfaces.

### Load Testing

As part of verifying the level of service the production system is capable of handling, load testing or stress testing will be used to send high levels of traffic to the system. This traffic will be distributed between static resource requests and application requests which generally require larger system resources to satisfy. Load will be increased until the component fails. Tools used for load testing are JMeter and httperf[12]

## Time Estimates

Below are summary estimates for the various components of this practicum. A detailed breakdown of time distribution and a Gantt chart are provided in the appendix.

| Component | Effort (hours) |
|---|---|
| Build test AMI machine image | 27h |
| Select and deploy a web server into the test image | 27h |
| Select and deploy a web server load balancing | 30h |
| Select and deploy an app server load balancer | 21h |
| Select and deploy a JEE connector | 8h |
| Configure firewalls | 9h |
| Select and deploy system and application | 22h |
| Select and deploy image persistence solution | 23h |
| Secure test image | 25h |

[10] http://jakarta.apache.org/jmeter/

[11] http://seleniumhq.org/

[12] http://www.hpl.hp.com/research/linux/httperf/

| Component | Effort (hours) |
|---|---|
| **Proof of Concept Complete** | |
| Deploy production web server clusters and security | 35h |
| Deploy production load balancing and persistence | 40h |
| **Production Deployment Complete** | |
| Production security testing | 62h |
| Production operational testing | 73h |
| **Testing Complete** | |
| Documentation | 36h |
| **Project Complete** | |
| Total Effort | 438h |

# Appendix

## Project Time Estimates

| Task | Effort |
|---|---|
| **1) Build test AMI machine image** | 27h |
| 1.1) Research AMI machine images | 6h |
| 1.2) Research EC2 environment | 7h |
| 1.3) Build Linux images | 6h |
| 1.4) Build web server OS environment | 5h |
| 1.5) Deploy test image to S3 | 2h |
| 1.6) Instantiate in EC2 | 1h |
| **2) Select and deploy a web server into the test image** | 27h |
| 2.1) Research security considerations of using Nginx | 4h |
| 2.2) Research performance considerations of using Apache in ... | 4h |
| 2.3) Research the scalability features of both servers | 5h |
| 2.4) Deploy both web servers into running test instance | 3h |
| 2.5) Research stress/load testing tools | 3h |
| 2.6) Stress/load test web servers | 6h |
| 2.7) Evaluate all results and make selection | 2h |
| **3) Select and deploy a web server load balancing solution** | 30h |
| 3.1) Research Amazon's Auto Scaling load balancer | 4h |
| 3.2) Research HAProxy software load balancer | 4h |
| 3.3) Deploy the selected load balancer into the test... | 5h |
| 3.4) Research and deploy Amazon's IP address pool | 3h |
| 3.5) Deploy multiple instances of the test image, configure... | 7h |
| 3.6) Stress/load test the balancer | 5h |
| 3.7) Evaluate results, make selection | 2h |
| **4) Select and deploy an app server load balancer** | 21h |
| 4.1) Research JEE alternatives to HAProxy | 4h |
| 4.2) Deploy the chosen balancer into the test image... | 4h |
| 4.3) Deploy multiple instances of the app server and test... | 6h |
| *Includes jee environment* | |
| 4.4) Stress/load test the balancer | 5h |
| 4.5) Evaluate results, make selection | 2h |
| **5) Select and deploy a JEE connector** | 8h |
| 5.1) Research available JEE connectors based on the web serve... | 4h |
| 5.2) Deploy JEE connector into the test image environment | 4h |
| **6) Configure firewalls** | 9h |
| 6.1) Configure the Amazon firewall | 4h |
| 6.2) Configure iptables firewalls on the web server and app... | 5h |
| **7) Select and deploy system and application deployment**... | 22h |
| 7.1) Research deployment options: S3, SVN, CVS, mounted... | 10h |
| 7.2) Implement chosen option | 8h |
| 7.3) Test system and application deployment using chosen... | 4h |
| **8) Select and deploy image persistence solution** | 23h |
| 8.1) Research image (database, logs, etc) persistence options | 6h |
| 8.2) Implement chosen option | 12h |
| 8.3) Test image persistance using chosen option | 5h |
| **9) Secure test image** | 25h |
| 9.1) Research virtualized drive encryption | 6h |
| 9.2) Deploy drive encryption onto test images and peristent... | 5h |
| 9.3) Research Snort resource compatibility on virtual machines | 3h |
| 9.4) Deploy Snort into test machines | 2h |
| 9.5) Run Bastille in test images | 2h |
| 9.6) Set up certificate-based SSH authentication | 2h |
| 9.7) Study Amazon security white paper, implement... | 5h |
| 10) Proof of Concept Complete | |
| **11) Deploy production web server clusters and security** | 35h |
| 11.1) Build production Linux images | 6h |
| 11.2) Build production web server OS environment | 5h |
| 11.3) Harden production OS | 5h |
| 11.4) Optimize binaries and configuration | 4h |
| 11.5) Configure production web server firewalls | 4h |
| 11.6) Configure JEE connectors | 2h |
| 11.7) Configure production system firewall | 4h |
| 11.8) Deploy production web server image to S3 | 2h |
| 11.9) Instantiate multiple instances in EC2 | 3h |
| **12) Deploy production load balancing and persistance** | 40h |
| 12.1) Configure production web server load balancing | 6h |
| 12.2) Configure production app server load balancing | 6h |
| 12.3) Build persistence solutions for system and application... | 12h |
| 12.4) Build persistence solutions for databases and logs | 12h |
| 12.5) Build run-time configuration injection | 4h |
| 13) Production Environment Configured | |
| **14) Production security testing** | 62h |
| 14.1) Vulnerability assessment | 6h |
| 14.2) Network scanning | 7h |
| 14.3) Reconnaissance assessment | 16h |
| 14.4) Host Firewall Testing | 6h |
| 14.5) System Firewall testing / ACL functionality | 8h |
| 14.6) Custom packet generation | 6h |
| 14.7) Exploit testing | 8h |
| 14.8) DOS testing | 5h |
| **15) Production operational testing** | 73h |
| 15.1) Load/stress testing | 10h |
| 15.2) Failover/availability testing on web server load balancer | 6h |
| 15.3) Failover/availability testing on app server load balancer | 6h |
| 15.4) Test dynamic web server scaling on load | 8h |
| 15.5) Test database persistence on instance crash | 5h |
| 15.6) Test log persistence on instance crash | 5h |
| 15.7) Test system update persistence on instance crash | 5h |
| 15.8) Integration testing of web UI and API | 10h |
| 15.9) Performance benchmarking | 8h |
| 15.10) Performance profiling | 10h |
| 16) Testing Complete | |
| **17) Documentation** | 36h |
| 17.1) Architecture documentation | 10h |
| 17.2) Procedures and policy documentation | 8h |
| 17.3) BCIT practicum report | 18h |
| 18) Project Complete | |