

# Deployment

## Contents

Introduction .....	2
Approach.....	2
Specification of the Delivery process.....	3
User requirements .....	3
Function requirements .....	3
Risks and Controls .....	4
Design of Delivery process.....	4
Detailed design .....	4
Process Flow.....	4
Architecture .....	5
Build process .....	5
Build Sub Functions.....	5
Step 1: Test the system is healthy .....	5
Step 2: Download Scripts from GitHub .....	7
Step 3: Run the Deployment Script.....	7
Integration process .....	10
Integration Sub Functions.....	10
Test process .....	10
Test Sub functions.....	10
Deployment process .....	11
Deployment sub functions.....	12
Monitoring process .....	12
Monitoring sub functions.....	12
Test Plan.....	16
Integration Testing.....	16
System Testing .....	16
End user testing .....	17
Execution plan.....	17

## Introduction

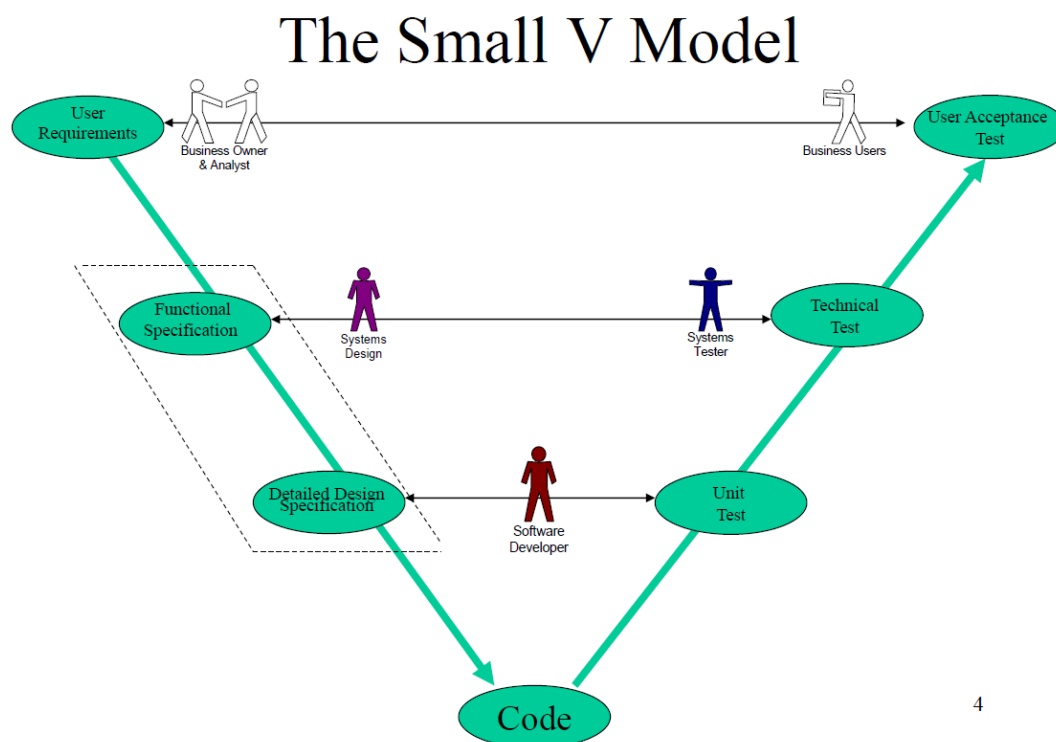
Deployment in large hosting services companies is now treated as part of a development process that supports the Operations of the company and its customers (DevOps). Where before deployment required building of physical servers and configuration of routers, these days deployment is as much about controlling the process programmatically.

The document investigates the best practices for deploying an Ubuntu environment suitable for a web application driven by MySQL and Perl and Apache HTTP web server. The aim is to using as much automation as possible whilst still maintaining a high level of control of what procedures are being run (e.g. what exactly is being installed and what processes are functioning) and also an awareness of how well the web application and the server in general are performing, through testing and monitoring.

## Approach

The V model ideology was used to plan and implement the project, as show below. The key advantages of the V Model for the project are:

- It is a highly disciplined model and Phases are completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Simple and easy to understand and use.
- Easy to manage due to the rigidity of the model - each phase has specific deliverables and a review process.



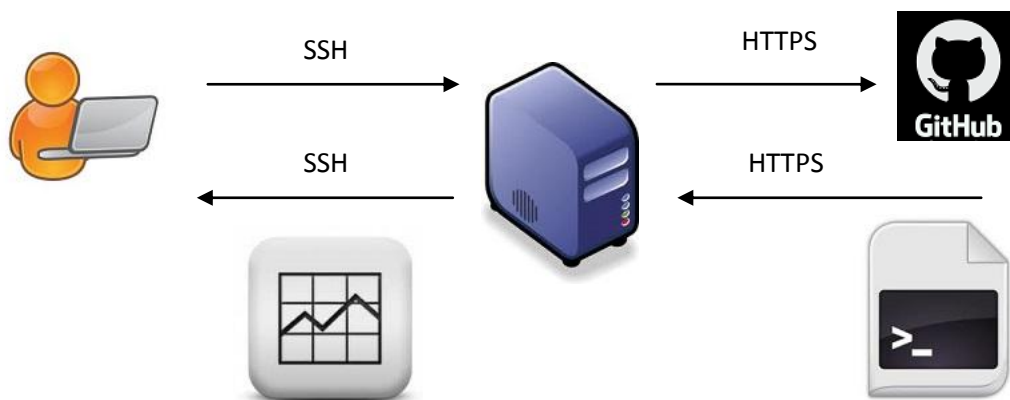
## Specification of the Delivery process

### User requirements

No.	MoSoCo	Statement
1	Mo	It must be possible to deploy content and components with a single command.
2	So	Error logging should be in one place
		Resource monitoring – CPU Usage – the load on the CPU - CPU Computations required per second Ram usage Process resource allocation and resource availability Hard disk usage: Disk input / output Ram usage
3	Mo	Proper unit, integration, and system tests should be in place
4	Mo	Application should be deployed to a web server with public access

### Function requirements

The below Use Case shows how an operator of the Deployment Process can trigger the process to start.



- User connects to server via SSH
- From server terminal, the user downloads from GitHub repository the Bash scripts
- The scripts on the server run the deployment commands, debug procedures, tests, and monitoring functionality which can be reported on back to the user

## ***Risks and Controls***

No.	Impact	Risk Statement	Control
1	High	Corrupted content could replace valid working content on the live platform	Content is backed up prior to deployment of new content. If the deployment fails, the old site is kept in place.
2	High	Failure and downtime of online internal and customer-facing business applications	Services are monitored and processes tested on an ongoing basis
3	High	Inconsistent and degraded user experience on your Webpages	
4	High	Bottlenecks in Webserver performance	
5	High	Difficulty in identifying and diagnosing the root cause of the fault or failure	

## **Design of Delivery process**

### ***Detailed design***

#### **Process Flow**



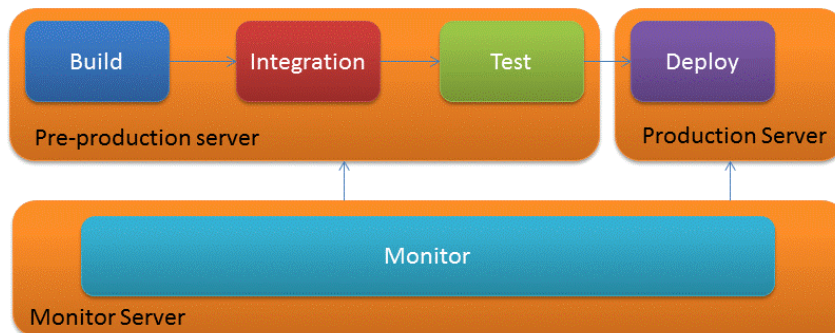
The 'Build' and 'Integration' processes will be ran using script that will be ran only once per server. It will involve the getting the environment ready to install the components and software needed to run the application. Only once the application is has is working on a test environment and all testing is complete it can be deployed to the production server.

The 'Test' process will involve a single script with predefined tests that can be run manually at any time or set to run automatically at a specified interval, as a CRON scheduled task for example. The installed application will also be manually tested by a user entering data and seeing it added to the database.

The deployment will be run inside a virtual Ubuntu instance, using Virtual Box. Using Bridge Mode will allow us to add the virtual instance to the same network that the host is on. The will give the virtual environment access to the Internet just like in a real environment and also allow is to test its connection, as we can ping it from any point (e.g. the host) on the network.

Once this virtual instance is running without any problems, the same procedure can be hosted on the Amazon Web Services platform. From there we will be able to log onto the application just like any other website on the Internet.

## Architecture



### ***Build process***

The build process will download content from a GitHub repository. It checks that all components and resources are in place for testing. It integrates the static HTML content from two or more files, into one. The build process makes sure that the environment is clean and revisions of necessary components are at the right level.

### **Build Sub Functions**

A sandbox is created that we can download and execute files from without interfering with the rest of the Ubuntu environment outside of the sandbox –

#### ***Step 1: Test the system is healthy***

In the deploymentScripts repository there is a script which checks for the basic health of the system such as connectivity and resource availability.

To run the script type:

```
sudo ./Script1-PreDeploymentCheckScript
```

The script is shown below with comments for each function:

# Level 0 functions <-----

# Returns 1 if the number of processes in param1 is greater than 0

```
function isRunning {
PROCESS_NUM=$(ps -ef | grep "$1" | grep -v "grep" | wc -l)
if [ $PROCESS_NUM -gt 0 ] ; then
    echo $PROCESS_NUM
    return 1
else
    return 0
fi
}
```

**#Return 1 if the number of TCP ports is greater than 1 for processes in param1 in listen state**

**function isTCPListen {**

TCPCOUNT=\$(netstat -tupln | grep tcp | grep "\$1" | wc -l)

if [ \$TCPCOUNT -gt 0 ] ; then

return 1

else

return 0

fi

}

**#Return 1 if the number of UDP ports is greater than 1 for processes in param1 #in listen state**

**function isUDPListen {**

UDPCOUNT=\$(netstat -tupln | grep udp | grep "\$1" | wc -l)

if [ \$UDPCOUNT -gt 0 ] ; then

return 1

else

return 0

fi

}

**#Return 1 if the remote TCP port on IP address param1 is open**

**function isTCPRemoteOpen {**

timeout 1 bash -c "echo >/dev/tcp/\$1/\$2" && return 1 || return 0

}

**#Ping Local IP address**

**# Returns 1 if a single ping is sent and successful received to ip address contained in param1**

**# If isLocalIPalive is true then logWrite Local IP address is alive else logWrite Local IP address is not alive**

**function isIPAlive {**

PINGCOUNT=\$(ping -c 1 "\$1" | grep "1 received" | wc -l)

if [ \$PINGCOUNT -gt 0 ] ; then

return 1

else

return 0

fi

}

**#Return 1 if process in param1 exceeds 50% utilisation**

**function getCPU {**

app\_name=\$1

cpu\_limit="5000"

app\_pid=`ps aux | grep \$app\_name | grep -v grep | awk {'print \$2'}`

app\_cpu=`ps aux | grep \$app\_name | grep -v grep | awk {'print \$3\*100'}`

if [[ \$app\_cpu -gt \$cpu\_limit ]]; then

return 0

else

return 1

fi

}

The script will return an error count and it will be job of the Administrator to investigate the source of each error and see what can be done to fix the issue.

## ***Step 2: Download Scripts from GitHub***

#Create a sandbox with random salt added to name

```
cd /tmp
mkdir sandbox_${RANDOM}
echo Using sandbox $SANDBOX
cd $SANDBOX/
ERRORCHECK=0
```

Components needed by the operating system, Ubuntu Linux to process Git commands will be downloaded and installed using:

```
sudo apt-get install git
```

The repository folder called deploymentScripts will be downloaded from the folder's GitHub address into the webpackage folder:

```
cd /webpackage
git clone https://github.com/fredyfontaine/deploymentScripts.git
```

As the files inside the folder are scripts they will need a permission level that will allow them to execute, such as permission 777:

```
cd /deploymentScripts
chmod 777 deploymentScripts/*
```

## ***Step 3: Run the Deployment Script***

If the system is in a healthy state with no underlying problems that could affect deployment (for example, checking that resources available are sufficient to run Apache) and there is nothing else that can be seen that would cause future problems, the deployment script is ran using the command:

```
sudo ./Script1-OneTimeScript
```

Below is an excerpt from the beginning of the script which details how the system was prepared for installation by removing any previously installed versions of Apache and MySQL and then updating the package repository.

#Stop apache & mysql (if they are running)

```
sudo service apache2 stop
sudo service mysql stop
```

# Clean Apache and Mysql environment

```
sudo apt-get -q -y --purge remove apache
sudo apt-get -q -y --purge remove mysql-server mysql-client mysql-common
sudo apt-get -q -y autoremove
sudo apt-get -q -y autoclean
```

##(Purge command removes apache package including all configuration files)

#Refresh apt-get package repository

```
sudo apt-get update
```

Once the operating system is at a 'clean', up to date state, Apache and MySQL can be installed, as the commands from the Bash script show, below.

**#Install Apache2**

```
sudo apt-get -q -y install apache2
```

**#Install MySQL**

```
echo mysql-server mysql-server/root_password password password | debconf-set-selections
echo mysql-server mysql-server/root_password_again password password | debconf-set-selections
apt-get -q -y install mysql-server mysql-client
```

Once Apache and MySQL were installed the Perl libraries and modules were installed so that Apache could handle Perl extensions, allowing Perl files to execute commands to the MySQL database.

**#Below command enables 'make' command on Ubuntu which is needed to install the libraries.**

```
sudo apt-get install build-essential
```

**#Install perl library helper routines:**

```
sudo apt-get -q -y install curl gcc-4.7
sudo curl -L http://cpanmin.us | perl - --sudo App::cpanminus
```

**# Install Perl CGI handling module:**

```
sudo cpanm CGI
```

**# Install Perl database connector:**

```
sudo cpanm DBI
```

Once the perquisites for the web application are running, we can download the application from GitHub to a sandbox;

```
cd /tmp
#
SANDBOX=sandbox_$(RANDOM)
mkdir $SANDBOX
cd $SANDBOX
ERRORCHECK=0
#
# Make the process directories
mkdir build
mkdir integrate
mkdir test
mkdir deploy
#
cd /webpackage
```

**# Download Web app from Github into Sandbox**

```
git clone https://github.com/FSlyne/NCIRL.git
```

**# Tar up the webpackage and call it webpackage\_preBuild.tgz**

**# Check if the MD5 checksum of the file has changed**

**# Store MD5 checksum in the file –**

**# If MD5 checksum hasn't changed then exit the script. Otherwise proceed**



```

tar -zcvf webpackage_preBuild.tgz webpackage
MD5SUM=$(md5sum webpackage_preBuild.tgz | cut -f 1 -d' ')
PREVMD5SUM=$(cat /tmp/md5sum)
FILECHANGE=0
if [[ "$MD5SUM" != "$PREVMD5SUM" ]]
then
    FILECHANGE=1
    echo $MD5SUM not equal to $PREVMD5SUM
else
    FILECHANGE=0
    echo $MD5SUM equal to $PREVMD5SUM
fi
echo $MD5SUM > /tmp/md5sum
if [ $FILECHANGE -eq 0 ]
then
    echo no change in files, doing nothing and exiting
    exit
fi
#

```

## **# BUILD**

**# Move Webpackage tar file to build directory**

```

mv webpackage_preBuild.tgz build
rm -rf webpackage
cd build
#
# Untar the webpackage-preBuild
tar -zxvf webpackage_preBuild.tgz
#
cd NCIRL/

```

Once they have been downloaded into the sandbox, they are then copied into the default Apache folders for HTML and CGI files respectively –

```

#Copy all files (i.e. HTML files) from NCIRL/Apache/www (downloaded from
GitHub) to the local Apache www folder
sudo cp -r Apache/www/* /var/www

```

**#Copy all files (i.e. script files) from NCIRL/Apache/cgi-bin (downloaded from GitHub) to the local Apache cgi-bin folder**

```

sudo cp -r Apache/cgi-bin/* /usr/lib/cgi-bin

```

**#set permissions appropriately**

```

cd /
sudo chmod a+x /usr/lib/cgi-bin/*
sudo chmod a+x /var/www/*

```

**#Start Apache and MySQL services**

```

sudo service apache2 start
sudo service mysql start

```

**# Tar/zip up webpackage, call it webpackage\_preIntegrate.tgz**

```

tar -zcvf webpackage_preIntegrate.tgz webpackage
#

```

```
#Set ERRORCHECK if any errors
ERRORCHECK=0
```

## Integration process

Integration integrates the static content with the components that provide the dynamic content, so as to create the overall content.

### *Integration Sub Functions*

The integration process could involve taking a snapshot of the live production server and saving it as a copy of the production server. If we ran the build processes on this copy, we could see how the web application would integrate with the production server instead of a server which may have nothing on it and therefore no dependencies.

#### **# INTEGRATE:**

```
# Move Webpackage tar file to integrate directory
mv webpackage_preIntegrate.tgz ../integrate
rm -rf webpackage
cd ../integrate
#
# Untar/unzip file
tar -zxvf webpackage_preIntegrate.tgz
#
# Script commands as per the build process
#
# Tar/zip up webpackage, call it webpackage_preTest.tgz
tar -zcvf webpackage_preTest.tgz webpackage
#
# Set ERRORCHECK if any errors
ERRORCHECK=0
```

## Test process

The test process makes sure that the static content is properly constructed (HTML tags etc.), and that the dynamic content functions as required.

### *Test Sub functions*

Once the Apache and MySQL services were running, we could test the system by populating the web application with data.

Commands had to be 'piped' into MySQL, which let us integrate SQL commands into the Bash script. A test user, called 'dbtestuser', below is created which will be the user account under which customer related data will be added to the database. In this test case that data will be a customer name and address which will be added to a customer table we will create.

#### **# TEST**

```
# Move Webpackage tar file to the test directory
mv webpackage_preTest.tgz ../test
rm -rf webpackage
cd ../test
```

```

#
# Untar/unzip file
tar -zxvf webpackage_preTest.tgz
#
#Connect to MySQL
cat <<FINISH | mysql -uroot -ppassword

#Create database
drop database if exists dbtest;
CREATE DATABASE dbtest;

#Create user
CREATE USER 'dbtestuser'@'localhost' IDENTIFIED BY 'dbpassword';
GRANT ALL PRIVILEGES ON dbtest.* TO 'dbtestuser'@'localhost' IDENTIFIED BY
'dbpassword';

#Create table
use dbtest;
drop table if exists custdetails;
create table if not exists custdetails
(
name VARCHAR(30)    NOT NULL DEFAULT '',
address VARCHAR(30)    NOT NULL DEFAULT ''
);

#Insert test data into table
insert into custdetails (name,address) values ('John Smith','Street
Address');

select * from custdetails;
FINISH

#Restart services
sudo service apache2 restart
sudo service mysql restart

#Cleanup - remove Sandbox directory
cd /tmp
rm -rf $SANDBOX

# Tar/zip up webpackage, call it webpackage_preDeploy.tgz
tar -zcvf webpackage_preDeploy.tgz webpackage
#
# Set ERRORCHECK if any errors
ERRORCHECK=0
#

```

## Deployment process

Deployment ensures that all components (content, packages etc) and resources (memory, disk, I/O etc) are in place for production. It unpacks the content and moves it to its proper location on the production server. It backs up the content prior to deployment of new content. If the deployment fails, the old site is kept in place.

Once the application was working in the virtual test environment, the scripts were downloaded again from GitHub but this time to the production server, hosted by **Amazon Web Services**, running Ubuntu, hosted at:

<http://ec2-54-194-155-139.eu-west-1.compute.amazonaws.com/form>

### ***Deployment sub functions***

The same routines and tests carried out again, initiated like before, checking the error count and then untaring the files:

```
# Tar/zip up webpackage, call it webpackage_preDeploy.tgz
tar -zcvf webpackage_preDeploy.tgz webpackage
#
# Set ERRORCHECK if any errors
ERRORCHECK=0
#
```

Security rules were set to allow an SSH from the virtual Ubuntu test system to the Amazon Ubuntu server, i.e., the production server. A rule was also created that would allow HTTP traffic from all IP addresses, as the web application should be accessible by the public just like a usual website.

### **Monitoring process**

Monitoring ensures that the site is functioning from a HTML, HTTP and other socket layer perspective. It should make sure that 6 key parameters (e.g. memory, I/O etc ) are within thresholds. It should report errors.

### ***Monitoring sub functions***

The following Bash script outlines how the system will be monitored.

```
#!/bin/bash
# Reference: fslyne 2013

ADMINISTRATOR=brianwebberley@yahoo.ie
MAILSERVER=mail1.eircom.net

# Level 1 functions <-----

#Function to check if Apache is running
# returns value of isRunning apache2
function isApacheRunning {
    isRunning apache2
    return $?
}

#Function to check if Apache is listening for TCP traffic on port 80
# Returns value of isTCPListen 80
function isApacheListening {
    isTCPListen 80
    return $?
}
```

**#Function to check if MySQL is listening for TCP traffic on port 3306**

**# Returns value of isTCPListen 3306**

```
function isMysqlListening {  
    isTCPListen 3306  
    return $?  
}
```

**# Returns value of isTCPremoteOpen localhost 80**

```
function isApacheRemoteUp {  
    isTCPremoteOpen 127.0.0.1 80  
    return $?  
}
```

**# Function to check if MySQL is running**

**# Returns value of isRunning mysqld**

```
function isMysqlRunning {  
    isRunning mysqld  
    return $?  
}
```

**# Returns value of isTCPremoteOpen localhost 3306**

```
function isMysqlRemoteUp {  
    isTCPremoteOpen 127.0.0.1 3306  
    return $?  
}
```

**# Functional Body of monitoring script <-----**

**# If isApacheRunning is true then logWrite Apache process is Running else logWrite Apache process is not Running**

```
isApacheRunning  
if [ "$?" -eq 1 ]; then  
    echo Apache process is Running  
else  
    echo Apache process is not Running  
    ERRORCOUNT=$((ERRORCOUNT+1))  
fi
```

**# If isApacheListening is true then logWrite Apache is Listening else logWrite Apache is not Listening**

```
isApacheListening  
if [ "$?" -eq 1 ]; then  
    echo Apache is Listening  
else  
    echo Apache is not Listening  
    ERRORCOUNT=$((ERRORCOUNT+1))  
fi
```

**# If isApacheRemoteUp is true then logWrite Remote Apache TCP port is up is up else logWrite Remote Apache TCP port is down**

```
isApacheRemoteUp  
if [ "$?" -eq 1 ]; then  
    echo Remote Apache TCP port is up
```

```

else
    echo Remote Apache TCP port is down
    ERRORCOUNT=$((ERRORCOUNT+1))
fi

#If isMysqlRunning is true then logWrite Mysql process is Running else logWrite Mysql process is not
Running
isMysqlRunning
if [ "$?" -eq 1 ]; then
    echo Mysql process is Running
else
    echo Mysql process is not Running
    ERRORCOUNT=$((ERRORCOUNT+1))
fi

# If isMysqlListening is true then logWrite Mysql is Listening else logWrite Mysql is not Listening
isMysqlListening
if [ "$?" -eq 1 ]; then
    echo Mysql is Listening
else
    echo Mysql is not Listening
    ERRORCOUNT=$((ERRORCOUNT+1))
fi

#If isMysqlRemoteUp is true then logWrite Remote Mysql TCP port is up else logWrite Remote Mysql
TCP port is down
isMysqlRemoteUp
if [ "$?" -eq 1 ]; then
    echo Remote Mysql TCP port is up
else
    echo Remote Mysql TCP port is down
    ERRORCOUNT=$((ERRORCOUNT+1))
fi

if [ $ERRORCOUNT -gt 0 ]
then
    echo "There is a problem with Apache or Mysql" | perl sendmail.pl
$ADMINISTRATOR $MAILSERVER
fi

```

The below Perl mailer utility would be used to email a log of the everything 'echoed' above.

```

#!/usr/bin/perl
# Reference: fslyne 2013
use Net::SMTP;

my $subj="Mailer message - ".convdatetimenow();
my $mailserver='maill.eircom.net';
my $to=shift @ARGV;
my $from=$to;
my $m = shift @ARGV;
$mailserver=($m) ? $m : $mailserver;

```

### # set up access to mailserver

```
$smtp = Net::SMTP->new($mailserver);
$smtp->mail($from);
$smtp->to($to);
$smtp->data();
$smtp->datasend("From: $from\n");
$smtp->datasend("To: $to\n");
$smtp->datasend("Subject: $subj\n");
$smtp->datasend("\n");
while(<STDIN>) {
    $smtp->datasend($_);
}
$smtp->dataend();
$smtp->quit;

exit;

sub convdatetimenow {
    return convdatetime(time());
}

sub convdatetime {
    my $time = shift;
    return convdate($time)." ".convtime($time);
}

sub convdate {
    my $time = shift;
    my ($sec,$min,$hour,$day,$mon,$year,$yday,$isdst)=localtime($time);
    $year = "1900"+$year;
    $mon = $mon+1; $mon = "0".$mon if ($mon<10);
    $day = "0".$day if ($day<10);
    return "$year-$mon-$day";
}

sub convtime {
    my $time = shift;
    my ($sec,$min,$hour,$day,$mon,$year,$yday,$isdst)=localtime($time);
    $hour= "0".$hour if ($hour<10);
    $min = "0".$min if ($min <10);
    $sec = "0".$sec if ($sec <10);
    return "$hour:$min:$sec";
}

exit 0
```

## Test Plan

### Unit Testing

The previous monitoring and testing script can be run at predefined intervals using a tool such as Cron.

### Integration Testing

#### Connect to Database with perl script:

The following perl script verifies that the mysql database is accessible both through unix domain sockets and through network tcp sockets.

```
#!/usr/bin/perl

use DBI;

print "connecting to mysql using UNIX domain sockets\n";
my $dsn='DBI:mysql:dbtest', 'dbtestuser';
$dbh = DBI->connect($dsn, 'dbtestuser', 'dbpassword'
                    ) || die "Could not connect to database: $DBI::errstr";
my $sth = $dbh->prepare('SELECT * FROM custdetails');
$sth->execute();
$result = $sth->fetchrow_hashref();
print "Value returned: $result->{name}\n";

print "connecting to mysql using TCP sockets\n";
my $dbhost='127.0.0.1'; my $dbport=3306;
my $dsn="DBI:mysql:dbtest;host=$dbhost;port=$dbport";
$dbh = DBI->connect($dsn, 'dbtestuser', 'dbpassword'
                    ) || die "Could not connect to database: $DBI::errstr";
my $sth = $dbh->prepare('SELECT * FROM custdetails');
$sth->execute();
$result = $sth->fetchrow_hashref();
print "Value returned: $result->{name}\n";
```

#### The script produces the output:

```
connecting to mysql using UNIX domain sockets
Value returned: John Smith
connecting to mysql using TCP sockets
Value returned: John Smith
```

### System Testing

#### Command to connect and download:

```
cd /tmp
wget 127.0.0.1
```

#### The script produces the output:

```
ubuntu@ip-172-31-31-111:/tmp$ wget 127.0.0.1
--2014-02-12 20:42:50-- http://127.0.0.1/
Connecting to 127.0.0.1:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 177 [text/html]
Saving to: `index.html'
```



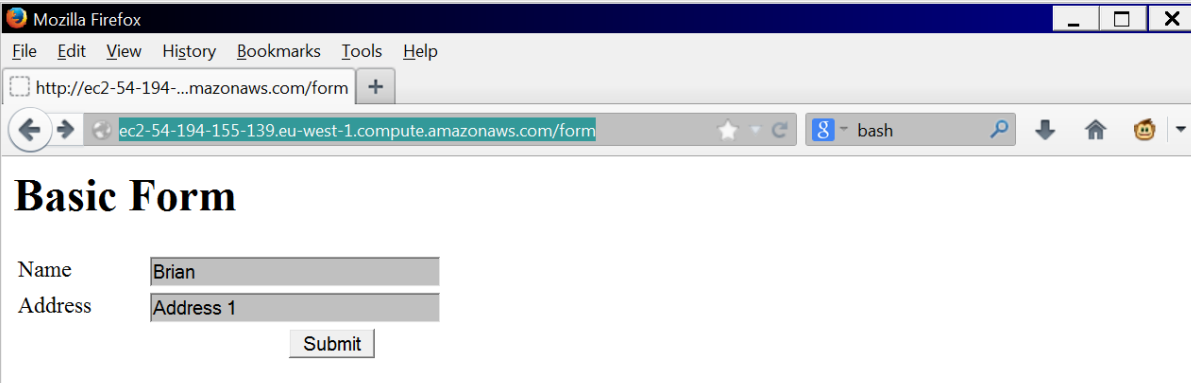
```
100%[=====>] 177          --.-K/s    in 0s  
2014-02-12 20:42:50 (16.4 MB/s) - `index.html' saved [177/177]
```

## ***End user testing***

Should map back to the initial User requirements

The end user can log onto the below site and perform a basic user input exercise and see that the data they entered will be saved to the database.

<http://ec2-54-194-155-139.eu-west-1.compute.amazonaws.com/form>



A screenshot of a Mozilla Firefox browser window. The address bar shows the URL `http://ec2-54-194-155-139.eu-west-1.compute.amazonaws.com/form`. The page title is "Basic Form". The form contains two input fields: "Name" with the value "Brian" and "Address" with the value "Address 1". A "Submit" button is located below the address field.

The data the user entered is return to the on them on the next page:



A screenshot of a Mozilla Firefox browser window. The address bar shows the URL `ec2-54-194-155-139.eu-west-1.compute.amazonaws.com/cgi-bin/accept_form.pl`. The page content displays the message "inserting name:Brian and address:Address 1 into Database" followed by "Showing the contents of the Database".

## **Execution plan**

These are instructions on how to test the process for the required outcomes as well as demonstration of

**Step 1:** Log onto the server which you would like to install the web application. We will call this the target server. For security reasons the log-on should be made via SSH:

```
SSH Target server IP address  
Enter password
```

**Step 2:** Install Git components on target server

```
sudo apt-get install git
```

**Step 3: Make a Sandbox for the Git repository**

```
cd /tmp
mkdir sandbox_$RANDOM
cd $SANDBOX/
```

**Step 4: Clone the repository that contains the Bash scripts:**

```
git clone https://github.com/fredyfontaine/deploymentScripts.git
This creates a folder called deploymentScripts
```

**Step 5: Make the Bash scripts inside the newly downloaded or 'cloned' repository which is executable:**

```
cd /deploymentScripts
chmod 777 deploymentScripts/*
```

**Step 6: Run the script with sudo privileges to avoid having to enter passwords as the script runs:**

```
sudo ./Script1-OneTimeScript
```