DEVOPS

LEARN ONE OF THE MOST POWERFUL SOFTWARE DEVELOPMENT METHODOLOGIES FAST AND EASY!

DEREK RANGEL

DevOps

Learn One of the Most Powerful Software Development Methodologies FAST AND EASY!

By Derek Rangel

Table of contents

Disclaimer

While all attempts have been made to verify the information provided in this book, the author does assume any responsibility for errors, omissions, or contrary interpretations of the subject matter contained within. The information provided in this book is for educational and entertainment purposes only. The reader is responsible for his or her own actions and the author does not accept any responsibilities for any liabilities or damages, real or perceived, resulting from the use of this information.

The trademarks that are used are without any consent, and the publication of the trademark is without permission or backing by the trademark owner. All trademarks and brands within this book are for clarifying purposes only and are the owned by the owners themselves, not affiliated with this document.

Introduction

DevOps is one of the modern software development methodologies which are in use today. The increased popularity of this method is due to the advantages it offers in terms of improved software quality and rapid delivery of software into the market. This explains the need for software developers to learn how to use this method in their activity. This book will guide you on this.

Chapter 1- Definition

DevOps (Developer Operations) is just a process for software development methodology whose emphasis is on collaboration, communication, automation, integration, and a means how the cooperation between the IT professional and the software developers can be measured. In this method of software development, there is interdependency between the members of the software development team. It is due to this that the development team finds it possible to develop and deliver software rapidly and improve the performance of the various operations.

The method is advantageous in that there will be an improved communication and collaboration between the software development team, which means that software of high quality will be produced. The frequency of the software deployment process is also improved, and this will mean that software will reach the market faster.

The methodology was developed so that it can remedy the existing disconnect between the process of development and the operations activity. This disconnect has led to inefficiency and conflicts and thus, the need for introduction of the DevOps method for software development.

Chapter 2- Installation of TomEE from Puppet

With Puppet, the declarative configuration of systems can easily be managed. Our task is just to declare the available resources of the system, and then declare their state too. We then store the description in the Puppet's core files. In this chapter, we will guide you on how to define TomEE as a resource of Puppet and this will mean that it will be automatically installed in all computers which are under the management of Puppet. Note that the TomEE was written in Java programming language, which means that JDK must be installed in your system for you to succeed in this chapter. To easily install the package into your system, just use the package manager.

We need to begin by creating a manifest file named "*init.php"* and then create an exec task which will update the package manager with the list of the available packages. This should have the following code:

 # updating the (outdated) package list

exec { 'update-package-list':

command => 'usrbinsudo usrbinapt-get update',

}

A class should next be defined, and then tasked with the installation of the OpenJDK. In Puppet, a class means several resources put together, and then Puppet will view them as a single unit. This is shown below:

class java {

package { “openjdk-6-jdk”:

ensure => installed,

require => Exec[“update-package-list”],

}

}

The next step should involve installation of the TomEE. Right now, it is not available in your distribution package repository in a software format. This means that a different approach is needed to the one which is followed in OpenJDK. We want to visit the TomEE site, and then download the “*tar.gz*” file which we will extract into our installation directory. The code for doing this is given below:

class tom {

file {“/opt/tomee-1.5.1”:

ensure => directory,

recurse => true,

} ->

exec { “download-tomee” :

command => “/usr/bin/wget http://apache.rediris.es/openejb/openejb-4.5.1/apache-tomee-1.5.1-webprofile.tar.gz -O

/tmp/tomee-1.5.1.tar.gz”,

creates => “/tmp/tomee-1.5.1.tar.gz”,

} ->

exec { “unpack-tomee” :

command => “/bin/tar -xzf /tmp/tomee-1.5.1.tar.gz -C /opt/tomee-1.5.1 —strip-components=1”,

creates => “/opt/tomee-1.5.1/bin”,

}

}

We have created a class named “*tom”* and then the directory in which the TomEE will be installed. The TomEE has been downloaded from the Apache site by use of the “*wget”* command, and the file is downloaded in a compressed format. We have then

uncompressed the file in the directory which we have just created.

At this point, the Apache TomEE has already been installed into the computer, but to start and stop it, this is not done automatically. To make the TomEE available, we must execute the command *"/opt/tomee-1.5.1/bin/startup.sh."* We can change this by use of the service resource. What it does is that an installed service is registered as a service. The next service resource should be defined in the TomEE class as follows:

service { "tomee" :

provider => "init",

ensure => running,

start => "/opt/tomee-1.5.1/bin/startup.sh",

stop => "/opt/tomee-1.5.1/bin/shutdown.sh",

status => "",

restart => "",

hasstatus => false,

hasrestart => false,

require => [Exec["unpack-tomee"], Package["openjdk-6-jdk"]],

}

When it comes to a service resource, one must have the TomEE unpacked and the OpnJDK installed, and this is why we have two declarations in the required attribute. The Puppet will create attributes in the “*exec”* task, and this will determine if a task is to be executed or not.

Chapter 3- Puppet and Packer Immutable Servers

It is recommended that server upgrades or changes on servers should never be done while the servers are live. What you should is that you should create new servers having the upgrades, and then stop using the old servers. The benefit is that you will enjoy immutability as you program at the infrastructure level, and you will not be affected by the configuration drift.

Nodes

Our infrastructure project will be made up of nodes.yalm which will be used for definition of the node names and the AWS security groups which they belong to. This is simple, as it is used in multiple other tools such as the vagrant. The code should be as shown below:

elasticsearch:

group: logging

zookeeper:

group: zookeeper

redis:

group: redis

size: m2.2xlarge

Rakefile

We will use the file *“nodes.yaml”* together with rake for production of packer templates for building out new AMIs. Note that most packer templates usually have similar or related features, so you can manage them as a unit, and this feature will ensure this. The code for this is given below:

```
require ‘erb’

require ‘yaml’

namespace :packer do

task :generate do

current_dir = File.dirname(__FILE__)

nodes = YAML.load_file( “#{current_dir}/nodes.yml”)

nodes.each_key do |node_name|

include ERB::Util

template = File.read(“#{current_dir}/packs/template.json.erb”)

erb = ERB.new(template)

File.open(“#{current_dir}/packs/#{node_name}.json”, “w”) do |f|

f.write(erb.result(binding))

end
```

end

end

end

What we have done is that we have used it together with a simple erb template which will inject the nodename into it. This is shown below:

{

“builders”: [{

“type”: “amazon-ebs”,

“region”: “us-east-1”,

“source_ami”: “ami-10314d79”,

“instance_type”: “t1.micro”,

“ssh_username”: “ubuntu”,

“ami_name”: “<%= node_name %> {{.CreateTime}}”,

“security_group_id”: “packer”

}],

“provisioners”: [{

“type”: “shell”,

“script”: “packs/install_puppet.sh”

}, {

“type”: “shell”,

“inline”: [

“sudo apt-get upgrade -y”,

“sudo sed -i /etc/puppet/puppet.conf -e \“s/nodename/<%= node_name %>-$(hostname)/\””,

“sudo puppet agent —test || true”

]

}]

With the above code, a packer template will be generated for each node, and this will perform the following tasks:

- Install puppet.
- An AMI will be created in us-east-1
- Execute Puppet once for configuration of the system.

- The security group will be adjusted to EC2.

The Puppet agent should not be enabled, so that we can avoid polling of updates. Once Puppet has completed, we can then remove it from the server to avoid it being baked in by AMI.

The Script

With packer, the user can specify the shell files and the shell commands which are to be run. When it comes to bootstrapping, this feature is the best, but it is good for the kind of configuration management needed in Puppet. Our packer templates will work by calling a shell script, and this will ensure that we do not use the old version of ruby Linux distros. The server name of the Puppet master will also be specified as part of the installation process. The code is given below:

sleep 20,

wget http://apt.puppetlabs.com/puppetlabs-release-raring.deb

sudo dpkg -i puppetlabs-release-precise.deb

sudo apt-get update

sudo apt-get remove ruby1.8 -y

sudo apt-get install ruby1.9.3 puppet -y

sudo su -c ‘echo ”””[main]

logdir=/var/log/puppet

vardir=/var/lib/puppet

ssldir=/var/lib/puppet/ssl

rundir=/var/run/puppet

factpath=$vardir/lib/facter

templatedir=$confdir/templates

[agent]

server = ip-10-xxx-xx-xx.ec2.internal

report = true

certname=nodename”””” >> /etc/puppet/puppet.conf’

The next step in our process should be to build a new AMI for the redis by running the following command:

```
packer build packs/redis.json
```

Once you execute the above command, the server will be created, configured, imaged, and finally terminated. Note that for each AMI that you create, a cost will be incurred. The cost for a single AMI might be small, but when you have multiple of these, then this will be very costly. This is why the old images have to be cleaned up. This is a very simple task which can be done as shown below:

```
import os

import boto

from fabric.api import task

class Images(object):

def __init__(sf, **kw):

sf.con = boto.connect_ec2(**kw)

def get_ami_for_name(sf, name):

(keys, AMIs) = sf.get_amis_sorted_by_date(name)

return AMIs[0]

def get_amis_sorted_by_date(sf, name):

amis = sf.conn.get_all_images(filters={‘name’: ‘{}*’.format(name)})

AMIs = {}

for ami in amis:

(name, creation_date) = ami.name.split(‘ ‘)

AMIs[creation_date] = ami

# removing the old images!

keys = AMIs.keys()

keys.sort()

keys.reverse()

return (keys, AMIs)
```

```
def remove_old_images(sf, name):

(keys, AMIs) = sf.get_amis_sorted_by_date(name)

while len(keys) > 1:

key = keys.pop()

print(“deregistering {}”.format(key))

AMIs[key].deregister(delete_snapshot=True)

@task

def cleanup_old_amis(name):

”’

Usage: cleanup_old_amis:name={{ami-name}}

”’

images = Images(

aws_access_key_id=os.environ[‘AWS_ACCESS_KEY_ID’],

aws_secret_access_key=os.environ[‘AWS_SECRET_ACCESS_KEY’]

)

images.remove_old_images(name)
```

You can set up the above. It will make sure that the AMI that you have in your system is the latest one. If you need to make sure that your five last AMIs are kept for the purpose of archiving, you can tweak this. If we had data stores, then this would have been made a bit trickier, since we would have to boot each of the replicas of the primary instances,

replicas promoted to primaries, and then old primaries would be retired.

Chapter 4- How to set up a modern web stack in Ubuntu

In this chapter, we will discuss the LERP (Linux, €nginx, Redis, and PHP) stack. With it, all your web needs will be provided for. This chapter will guide you on how to set up the stack, and then create an empty playground having all of the necessary assets needed for experimentation, learning, and building. The set up will be ready for use in a production environment.

My assumption is that you currently have Ubuntu installed on your system. If this is not the case, then download its ISO, and then prepare a bootable media which you will use for installation of the OS. Once the installation process completes, just execute the following command:

```
sudo apt-get update
```

The above command will serve to update your system. The latest version of the LTS is highly recommended, due to its strong support and increased stability.

Nginx (the server)

Nginx can be found from the launch pad

You can now use the following command so as to create a repository, and then refresh the software resources used on your system:

```
sudo add-apt-repository ppa:nginx/stable

sudo apt-get update
```

The install command can be issued as follows:

With the above command, the nginx stable will be installed on your system. If you need to do a verification of this, you can open your browser, and then type in the IP address of your server. The output should be the welcome file for nginx. If the server is being used simply for local development, then you can use the IP address "*127.0.0.1*".

PostgreSQL (The Server)

With Ubuntu, this kind of database comes installed in the system. To verify whether this is the case in your system, just run the following command on your terminal:

```
sudo apt-get install postgresql-9.3
```

To get the latest version of this, one can add the latest version of the postgreSQL APT repository as shown below:

```
sudo deb http://apt.postgresql.org/pub/repos/apt/ trusty-pgdg main
```

This should be followed by these commands:

```
wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | \
  sudo apt-key add -

sudo apt-get update
```

Finally, you can then install the PostreSQL version 9.4 as shown in the figure given below:

It will be good if all of the users install the latest version of the above for stability and strong support. Since here we are working with IT experts, the command line will be a good tool. This is why we are not going to discuss how to install the "*pgAdmin*" which is a graphical utility for this kind of database.

Configuration of PostgreSQL

By default, a user-group named *"postgres"* will be created in the Ubuntu system. However, you need to note that the user in this case will be a superuser, and this is why it is not recommended to use this. This is because they are capable of carrying out any task on the database. Our aim is to create a new user and a database. The user should also be in a position to login, posses a password, and then possess privileges which should allow him or her to access the newly created database only.

The concept of roles is supported in PostgreSQL rather the concept of users and user groups. We should begin by a login to the default account, that is, postgres, and then create a new database and a new role from the account. The login can be done as follows:

```
sudo -u postgres psql postgres
```

The above command will log you into the default account. Change the password for the account by use of the following command:

```
\password postgres
```

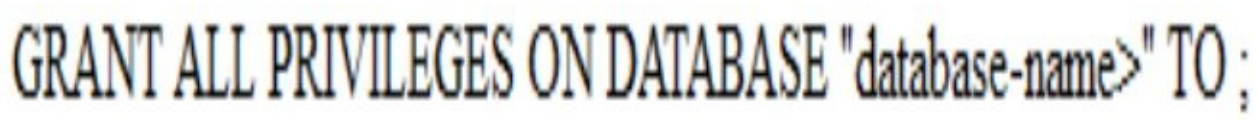
After executing the above command, a prompt will be prompted asking you to provide the new password. You will also be prompted to confirm it. The creation of the new database can now be done, and the normal SQL commands can be used as shown below:

```
CREATE DATABASE <database-name>;
```

We can now create a role on the database, and this will have some limited privileges and this will allow us to perform our operations on the database.

```
CREATE ROLE "username" LOGIN PASSWORD 'password-in-quotes';
```

With the above command, a new role will be created for the username which you specify. In the case of the password, it must be written inside the single quotes. Note that the username in the above case should be similar to the one of your computer. This is how users are managed in postgreSQL. The last step should be to provide the user with some privileges, so that they can be able to operate on the newly created database. This can be done by use of the command given below:



After the above step, your PostgreSQL will be ready for use, since you will be through with the configuration.

Redis (The cache)

It is easy for us to install Redis. With the Ubuntu Trusty repositories, the latest version of this is already available. Just run the command given below:

```
sudo apt-get install redis-server
```

It is after execution of the above command that you can begin to play around with the Redis. Begin by launching the "*redis-cli"* program from your computer's terminal. This can be done by executing the following command on the terminal:

```
$ redis-cli
```

Learning Redis is very easy, and very powerful. If you do not know how to use it, you can consult the tutorials which are available online.

PHP (The Language)

The purpose of the nginx server is to receive all of the requests which are incoming. However, it is unable to process the PHP scripts which are stored in the server. However, "*Common Gateway Interface"* is used in this case. The gateway works by routing all of the requests from the nginx to the PHP engine which is responsible for processing of the script.

We should begin by installing the "*PHP-fpm"* package by running the command given below:

```
sudo apt-get install php5-fpm
```

The installation of the php5-common package will be installed automatically, and this will be responsible for allowing the PHP scripts to be parsed. Before continuing to begin routing of the requests, there are two additional packages which you should install, and they will help you in the process of development. These are "*php5-cli*" and "*php5-xdebug.*" These can be installed by running the following command:

```
sudo apt-get install php5-cli php5-xdebug
```

Once you execute the above command, you will be done. You can now test the command line of the PHP interactively by executing the following command:

```
$ php -a
```

You will be notified that the interactive mode has been enabled. You can then run the "*Hello world"* example to see if it will run. This is shown in the figure given below:

```
php > var_dump('Hello World!');
string(12) "Hello World!"
```

That shows that our set up was successful.

Stitching together the pieces

Now that all of the subcomponents of our web stack have been installed, we need to make them work together. We should begin by installation of the PostgreSQL drivers for PHP, and these will facilitate the connection between the PHP and the PostgreSQL database. Just run the following command, so as to perform the installation of these drivers:

The presence of the PostgreSQL PDO Driver in the system can now be tested by writing the following code:

php -r ‘var_dump(PDO::getAvailableDrivers());’

array(1) {

[0] =>

string(5) “pgsql”

}

We can now set up the nginx so that it can begin to route the requests via the FastCGI to PHP-FPM. Note that the configuration of the nginx has been divided into two parts, one of which is the global configuration, and the other one is a default configuration. The config files of your system have to be stored in the directory “sites-enabled” only. However, the default setting is for all of the files to be stored in this directory and each of the configuration file in this directory will contain the config of a single website. A simple config file looks as follows:

server {

listen 80 default_server;

root /usr/share/nginx/html;

index index.php index.html index.htm;

server_name localhost;

location / {

try_files $uri $uri/ =404;

}

location ~ .php$ {

try_files $uri =404;

fastcgi_pass unix:/var/run/php5-fpm.sock;

fastcgi_index index.php;

fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;

include fastcgi_params;

}

}

The keyword "*listen"* has been used to enable the system to listen to the port number 80, which is the default server. The "*root"* keyword has been used to define where the scripts for the website will be placed. This indicates that all of the files stored in the directory *"/usr/share/nginx/html*" will be accessible from the outside.

The keyword "*index"* will be used for defining the entry point for the website. This means that it will form the first file in the directory of the website. If this file is not found, then the next file will be matched. The keyword "*localhost"* has been used for the purpose of the local development. If the development is not done locally, then the URL of the server should be added here rather than the "*localhost."*

You can then save the file in the directory "/etc/nginx/sites-enabled" and then give it the name "*server.conf."* Once you are done, restart the nginx server by executing the

following command:

sudo service nginx restart

You can next navigate to the directory “/usr/share/nginx/html” and then create a file named “*index.php.”* The following code should be added to the file:

<?php

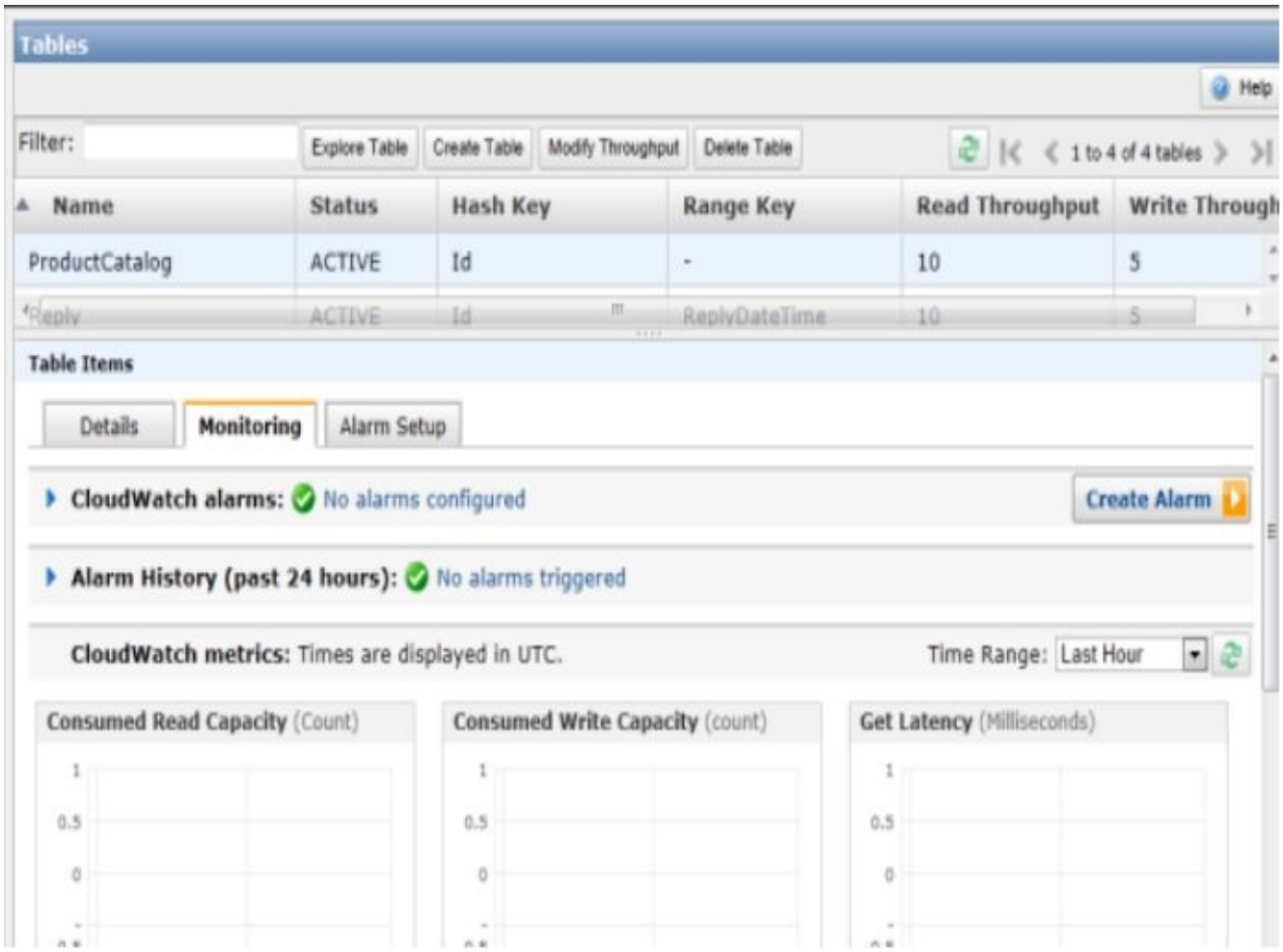
phpinfo();

The file can then be saved, and a navigation to the url *http://localhost* done. This should give you the PHP info page.

At this point, only one thing will be remaining, which is using the Redis with PHP. For this to be done, a Redis Client library together with PHP has to be used. The installation of this will depend, and it will be determined by the client that you choose.

Chapter 5- Migration of MongoDB to DynamoDB

DynamoDB is one of the non-Structured Query Language (NoSQL) databases which are currently in use today. Scaling of this database is very easy, and it offers no overhead in terms of administration. However, it has a limitation when it comes to the design of the schema. Once you have migrated your data from MongoDB to DynamoDB, you will notice that the task of administering the data will be reduced, and that it will be possible for you to archive the old data. The archiving here can mean that the data which is not queried more often by the database can be moved and stored in a slow storage. Each of the components can have a single table. After setting up the tables, the read and write operations can be specified for each of the tables, and this will be determined by how often the table is being accessed.



The database reserves up to 300 seconds of unused read and write capacity.

However, the index is one of the limitations that DynamoDB has. You are allowed to use either a hash key, or a combination of the hash key and the range key. This means that a multiple-key index is not supported. The following is an example of the search result entry in our system:

{

name : “john”,

sex : “male”,

country : “US”,

results : “…”,

added_on : 2015-07-03T11:00:00Z

}

The solution to the limitation is to combine fields, and then split out the tables. When the tables are split, then this means that we will be having more tables in our system, and this will help in improving our control over capacity planning for the different data which we are handling. After combining the fields and splitting the tables, we ended up having search results which looks as follows:

{

name : “john”,

sex : “male”,

results : “…”,

created_on : 2015-07-03T11:00:00Z

}

…

{

name : “hellen”,

sex : “female”,

results : “…”,

created_on : 2015-08-04T12:00:00Z

}

However, you need to be careful when dealing with this, as it has to match your use case and even the product. This means that it might not be an obvious solution to all of the similar problems, but it has guided you on how you can think outside the box and get things done.

You also need to note that in DynamoDB, native Date or Date Time object is not supported. This means that you have to come up with ways on how to handle these. The solution to this is to convert this into the Linux timestamp, and then store the number. An example of this is given below:

{

name : “john”,

sex : “male”,

results : “…”,

added_on : 1248764145

```
}
```

What happens is that the conversion of the date into timestamp is done at the application layer, and before we can query the DynamoDB. The sorting can also be performed at this point.

How to query Dates

You application can need to query the database for data, include the field for the date. In this case, you will have to Query the database rather than using the "*GetItem"* command. An example of this is given below:

.find({

"keyword" : "john",

"added_date" : new Date(2015, 1, 2)

});

In DynamoDB, I might need to query the following command:

"name" : {

"AttributeValueList" : [{ "S" : "john" }],

"ComparisonOperator" : "EQ"

},

"added_date" : {

"AttributeValueList" : [

{ "N" : 1520170510 },

{ “N” : 1620256897 }

],

“ComparisonOperator” : “BETWEEN”

}

The querying can be done, but there are some problems associated with it. First, the Query command is slower when compared to the “*GetItem”* command, which is straight forward. In the latter case, the user is given both the hash key and the range key for matching.

DynamoDB also provides us with the “*BatchGetItem”* which can be used to get the search results for multiple keywords which are frequently used in applications. Each of the API requests to the DynamoDB can lead to an overhead which can add up whenever we are handling the names the application is requesting.

Storing Data as String

One can choose to store their data which has been formatted as a string. Consider the example given below:

{

name : “john”,

sex : “male”,

country : “US”,

results : “…”,

created_on : “2015-02-04”

}

We can then use the “*GetItem”* object so as to get our data more quickly. This is shown in the example given below:

“name” : {

“S” : “john”,

},

“added_date” : {

“S” : “2015-02-04”),

}

With the above, we will be in a position to fetch the data in batches. When you use the DynamoDB web console, the data will also become human-readable, which means that the user will save some time.

Now that you using DynamoDB, you will notice how it offers an effortless scaling and zero maintenance. Some creativity is needed when designing the tables and breaking out of the old paradigm. Other than concentrating on configuring your MongoDB, your efforts will only be needed in development of the product.

Chapter 6- MongoDB and Tree Structures

In MongoDB, we can store tree structures. Most projects usually deal with tree structures, and this is why you need to know how to store these in MongoDB. However, when dealing with tree structures in the MongoDB, we should be able to perform operations in the tree which include inserting, updating, and removal of nodes, calculate the path which leads to a particular node, and then get all the descendants of a particular node.

For us to operate with the tree, some calculations will be needed for changing the position of a particular node together with its siblings.

Addition of a New Node

Consider the example given below:

var excount = db.categoriesPCO.find({parent:‘Linux’}).count();

var norder = (excount+1)*10;

db.categoriesPCO.insert({_id:‘LG’, parent:‘Linux’, someadditionalattr:‘test’, order:norder})

//{ “_id” : “Ubuntu”, “parent” : “Linux”, “someadditionalattr” : “test”, “order” : 40 }

That is a new node that can be added. Very simple!

Updating a Node

Consider the example given below, which shows how the updating of an existing node can be done:

```
excount = db.categoriesPCO.find({parent:‘Linux_Distributions’}).count();
norder = (excount+1)*10;
db.categoriesPCO.update({_id:‘LG’},{$set:{parent:‘Linux_Distributions’,
order:norder}});
//{ “_id” : “Ubuntu”, “order” : 60, “parent” : “Linux_Distributions”,
“someadditionalattr” : “test” }
```

If you need to remove a particular node, then use the following command:

```
db.categoriesPCO.remove({_id:‘Ubuntu’});
```

If you need to get the node children in an ordered manner, then do it as shown below:

```
db.categoriesPCO.find({$query:{parent:‘Linux’}, $orderby:{order:1}})
//{ “_id” : “Ubuntu”, “parent” : “Linux”, “order” : 10 }
```

//{ “_id” : “Our_Main_Products”, “parent” : “Linux”, “order” : 20 }

//{ “_id” : “Linux_Distributions”, “parent” : “Linux”, “order” : 30 }

That is how it can be done. If you need to get the descendants of a particular node, then do it as follows:

```
var desc=[]
var stack=[];
var it = db.categoriesPCO.findOne({_id:“Linux_Distributions”});
stack.push(it);
while (stack.length>0){
var cnode = stack.pop();
var child= db.categoriesPCO.find({parent:cnode._id});
while(true === child.hasNext()) {
var childn = child.next();
desc.push(childn._id);
stack.push(childn);
}
}
```

desc.join(“,”)

Path to a particular Node

Sometimes, you might need to get the path which leads to a particular node. The operation to be involved in this case will be a recursive one, as shown below:

```
var p =[]
var it = db.categoriesPCO.findOne({_id:“RedHat”})
while (it.parent !== null) {
it=db.categoriesPCO.findOne({_id:it.parent});
p.push(it._id);
}
p.reverse().join(‘ / ‘);
```

In this case, indexes can be used as follows:

```
db.categoriesPCO.ensureIndex( { parent: 1, order:1 } )
```

The above operations are for tree structures which have a parent reference. In the next section, we will discuss tree structures which have a child reference.

In this case, an “ID” and a “ChildReference” for each node will be stored. An order field will not be necessary for this case, because the information is provided by the child collection. In most cases, the order of an array is preferred, but if this is not supported in your case, then an additional code will have to be written for your maintaining of the order, meaning that much complexity will be involved.

Addition of a New Node

This can be added as shown below:

db.categoriesCRO.insert({_id:‘Ubuntu’, childs:[]});

db.categoriesCRO.update({_id:‘Linux’},{ $addToSet:{childs:‘Ubuntu’}});

//{ “_id” : “Linux”, “childs” : [“Linux_Distributions”, “Our_Top_Products”, “Linux_Distrutions”, “Ubuntu”] }

Updating a New Node

If you need to rearrange the order under the same parent, then do it as follows:

db.categoriesCRO.update({_id:‘Linux’},{$set:{“childs.1”:‘Ubuntu’,“childs.3”: ‘Our_Top_Products’}});

//{ “_id” : “Linux”, “childs” : [“Linux_Distributions”, “Ubuntu”, “Linux_Distributions”, ” Our_Top_Products ”] }

If you need to move a particular node, then do it as follows:

db.categoriesCRO.update({_id: ‘Linux_Distributions’},{ $addToSet: {childs:‘Ubuntu’}});

db.categoriesCRO.update({_id:‘Linux’},{$pull:{childs:‘Ubuntu’}});

//{ “_id” : “Linux_Distributions”, “childs” : [“RedHat”, “Suse”, “CentOS”, “Mint”, “Kali”, “Fedora”] }

If you need to remove a particular node, then do it as follows:

db.categoriesCRO.update({_id:‘Linux_Distributions’},{$pull:{childs:‘Ubuntu’}})

db.categoriesCRO.remove({_id:‘Ubuntu’});

The above code will remove the node that you specify.

If you need to get the children of a node in an ordered manner, then do it as follows:

var p = db.categoriesCRO.findOne({_id:‘Linux’})

db.categoriesCRO.find({_id:{$in:p.childs}})

However, note that in the above, an additional sorting in the client side will be needed in the parent array sequence.

To get all of the descendants of a particular node, then do it as follows:

```
var desc=[]

var stack=[];

var it = db.categoriesCRO.findOne({_id:“Linux_Distributions”});

stack.push(it);

while (stack.length>0){

var cnode = stack.pop();

var child = db.categoriesCRO.find({_id:{$in:cnode.childs}});

while(true === child.hasNext()) {

var childn = child.next();

desc.push(childn._id);

if(childn.childs.length>0){

stack.push(childn);

}

}

}

desc.join(“,”)
```

Path to a Node

If you need to obtain a path which leads to a particular node, then do it as follows:

```
var p=[]

var it = db.categoriesCRO.findOne({_id:“Ubuntu”})

while ((it=db.categoriesCRO.findOne({childs:it._id}))) {

p.push(it._id);

}

p.reverse().join(‘ / ‘);
```

Indexes

It is recommended that indexes should be used on children. The following syntax should be used:

db.categoriesCRO.ensureIndex({ childs: 1 })

Tree Structure having an Array of Ancestors

In this case, an ID, ParentReferenc and an AncestorReference will be stored for each of the available nodes. The rest of the operations are discussed below:

Addition of a New Node

The new node can be added as follows in this kind of a tree structure:

var ancpath = db.categoriesAAO.findOne({_id:‘Linux’}).ancestors;

ancpath.push(‘Linux’)

db.categoriesAAO.insert({_id:‘Ubuntu’, parent:‘Linux’,ancestors:ancpath});

//{ “_id” : “Ubuntu”, “parent” : “Linux”, “ancestors” : [“Linux”] }

To update a particular node, then do it as follows:

ancpath = db.categoriesAAO.findOne({_id: ‘Linux_Distributions’}).ancestors;

ancpath.push(‘Linux_Distributions’)

db.categoriesAAO.update({_id:‘Ubuntu’},{$set:{parent:‘Linux_Distributions’, ancestors:ancpath}});

//{ “_id” : “Ubuntu”, “ancestors” : [“Linux”, “Linux_Distributions”,“Linux_Distributions”], “parent” : “Linux_Distributions” }

If you need to remove a particular node, then use the following syntax:

db.categoriesAAO.remove({_id:‘Ubuntu’});

For you to get the children of a node in an Unordered manner, then use the following syntax:

db.categoriesAAO.find({$query:{parent:‘Linux’}})

Note that if you need to get the ordered children of a particular node, then an order field must be introduced. That is why you must come up with an approach which will help you to make the children ordered.

If you need to get all of the descendants of a particular node, then do it as follows:

```
var ancs = db.categoriesAAO.find({ancestors:“Linux_Distributions”},{_id:1});
while(true === ancs.hasNext()) {
var element = ancs.next();
desc.push(element._id);
}
desc.join(“,”)
```

One can also achieve the above by using the aggregations framework which is well known in MongoDB. This is shown below:

```
var agancestors = db.categoriesAAO.aggregate([
{$match:{ancestors:“Linux_Distributions”}},
{$project:{_id:1}},
{$group:{_id:{},ancestors:{$addToSet:”$_id”}}}
])
```

desce = agancestors.result[0].ancestors

desc.join(“,”)

Tree Structures with a Materialized Path

In this case, we have to store the “*ID”* and the “*PathToNode.”*

Addition of a New Node

This can be done as follows:

var ancpath = db.categoriesMP.findOne({_id:‘Linux’}).path;

ancpath += ‘Linux,’

db.categoriesMP.insert({_id:‘LG’, path:ancpath});

//{ “_id” : “Ubuntu”, “path” : “Linux,” }

To update or move a particular node, then do it as follows:

ancpath = db.categoriesMP.findOne({_id: ‘Linux_Distributions’}).path;

ancpath +=’ Linux_Distributions,’

db.categoriesMP.update({_id:‘Ubuntu’},{$set:{path:ancpath}});

//{ “_id” : “Ubuntu”, “path” : “Linux, Linux_Distributions’, Linux_Distributions’,”

}

To remove a particular node, then use the following syntax:

db.categoriesMP.remove({_id:‘Ubuntu’});

Chapter 7- Configuration of Apache for Multiple Domains

Sometimes, you might need your Apache web server to handle multiple names for domains, and to deliver the correct site to the visitors. You need to know how to create Apache virtual hosts, test the names of domains so as to be sure that the web server they are pointing to is the correct one, and then perform a configuration on the Apache virtual host files so that the names for the domains can be pointing to a specific folder.

Configuration of the Apache vhost

When the domains are working effectively and as expected, we should configure the Apache so that it can route the domain names to the site directory. This can be done by following the steps given below:

1. Locate and navigate to the directory having your Apache configuration. For Ubuntu users, then can be found at "*/etc/apache2.*" For other types of servers, this can be found at "*/etc/http.*"

2. You can then locate the vhost configuration. For Ubuntu users, this should be the directory "*sites-available.*" For the users of other types of servers, then they might have to edit the file "*httpd.conf.*"

3. You can then open or create the vhost configuration. For Ubuntu users, just create a new file in the directory "*sites-available.*" The file can be given the same name as the name of the domain. However, you can choose the name that you want for the file, provided you can recall it.

4. A new vhost record can then be added. The Apache directives can then be added to the file. However, one has to have ServerAlias, ServerName, and DocumentRoot directives for a particular host. An example of this is given below:

 # The vhost record can now be started on the default HTTP port 80

<VirtualHost *:80>

 # The vhost name.

ServerName udrupal.com

 # The alternative names for the same vhost.

 # The other domains can be added here. These will be moved to the same place.

ServerAlias news.udrupal.com

ServerAlias udrupalalumni.com

 # This is the Directory where the code for the website lives.

DocumentRoot /home/udrupal/www

<Directory />

Options FollowSymLinks

AllowOverride All

</Directory>

</VirtualHost>

5. The changes made to the file can now be saved.

6. The site can now be enabled. This is to make sure that the Apache web server applies the newly made changes to the configuration.

7. Just open the command prompt, and then run the following command:

sudo a2ensite udrupal

What will happen is that you will be notified that the site is being enabled, and then the reload command will be issued.

8. The Apache can then be reloaded or restarted. However, the Apache web server will not immediately notice the changes. This is why you have to restart the

configuration files for Apache. The command for doing this will depend on the type of system that you are using. In most systems, the command should have the "*sudo*" command in front of it as shown below:

```
sudo /etc/init.d/apache2 reload
```

The system should now be set up. You can open up your browser, and then type one of the domain names. The directory of the site should be observed loading.

Chapter 8- Reverse Cache Proxy in Nginx

After reading this chapter, you will understand the importance and know how to set up in Nginx. With the reverse cache proxy in Nginx, the performance of the system can be greatly improved, and it makes it possible for the system to handle multiple concurrent users on the landing pages.

However, most people do not know what Nginx is. It is just an Open Source Http Web server and a reverse web server. It is currently being used today for the powering of websites, ranging from simple to complex ones. This web server helps in handling users, and especially multiple concurrent users. When it comes to websites, users usually issue requests to the system, and then wait for the feedback from it.

The problem comes when all the users are new to the site, and they all issue requests. In this case, servicing the requests for all of these users becomes a bit tricky. The solution to this problem is to come up with the strategy of caching. Consider a situation whereby all of the concurrent users are requesting the same page. In this case, the page can be placed in the cache, in which case once any user requests it, then it will be issued to them directly.

Configuration and Setup

For users of Ubuntu, the configuration and setup can be done as follows:

Begin by opening the file "*/etc/nginx/nginx.conf*" in the text editor of your choice. Under the definition for "*http {*", add the following lines:

proxy_cache_path /var/www/cache levels=1:2 keys_zone=my-cache:8m max_size=1000m inactive=600m;

proxy_temp_path /var/www/cache/tmp;

real_ip_header X-Forwarded-For;

With the first two lines in the above code, a cache directory will be created in your system.

The next step should be the creation of a virtual host under "/*etc/nginx/sites-available/website.*" This is shown below:

server {

listen 80;

server_name _;

```
server_tokens off;

location / {

proxy_pass              http://127.0.0.1:8080/;

proxy_set_header        Host                $h;

proxy_set_header        X-Real-IP           $r_addr;

proxy_set_header        X-Forwarded-For     $p_add_x_forward_for;

proxy_cache  my-cache;

proxy_cache_valid 3s;

proxy_no_cache $cookie_PHPSESSID;

proxy_cache_bypass $cookie_PHPSESSID;

proxy_cache_key         “$scheme$host$request_uri”;

add_header X-Cache $upstream_cache_status;

}

}


server {

listen   8080;

server_name _;

root /var/www/root_for_document/;

index index.php index.html index.htm;

server_tokens off;
```

location ~ .php$ {

try_files $uri /index.php;

fastcgi_pass 127.0.0.1:9000;

fastcgi_index index.php;

fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;

include /etc/nginx/fastcgi_params;

}

location ~ /.ht {

all denied;

}

}

To enable the above, you can do the following:

cd

ln -s /etc/nginx/ available-sites/website /etc/nginx/enabled-sites/website

/etc/init.d/nginx restart

The first definition of the server is for the reverse cache proxy which runs at port number

80. The next one should be the backend one. With the proxy pass http://127.0.0.1:8080/, the traffic will be forwarded to port 8080.

When it comes to static content, Nginx is very fast in serving this. An example of this content is the single page which is described earlier on. The improved performance is due to the use of the cache which makes the processing easy. The benchmark for this is given below:

```
ab -n 1000 -c 100 http://127.0.0.1:80
```

With the above command, 1,000 requests, which are 100 concurrent, will be sent to our reverse cache proxy which is on port number 80.

Consider the command given below:

```
ab -n 1000 -c 100 http://127.0.0.1:8080
```

What happens with the above command is that 1,000 requests, having 100 concurrent will be send to the backend at port number 8080. For the case of the port number 80, it will take 0.2 seconds for the 1,000 requests to be run while for the port number 8080, it will

take 2.5 seconds for the same number of requests to run. This translates to be 12.5 times faster.

On port 80, 4,300 requests will be processed in a second, while in port number 8080, only 400 requests will be processed per second. This translates to 10.7 times faster.

Although PHP accelerators can be very useful, it might not be effective in certain scenarios when compared to the reverse cache proxy. The PHP accelerator works by caching the content of PHP scripts which has been compiled so as to improve on performance. This normally happens in environments where shared memory is being used so as to avoid the concept of recompiling the source code for each request which is made. Whenever the source code of the PHP script is changed, then the OpCode which is stored is changed to the appropriate one.

Varnish is also a good tool to use for a reverse cache proxy. You need to know that its focus is mainly on HTTP. Nginx can act like a web server, a mail server, a Reverse Cache Proxy, and a load balancer. However, this is not the case with Varnish. The two tools are good in reverse cache proxying. The good thing with Varnish is that it can be configured more easily. However, it takes more memory and CPU. The process of setting up Nginx as a backend or as a reverse cache proxy is much easier, as you will not be required to install anything. With the latter, when the infrastructure grows in size, then the process of adding

or installing new software will not be easy. This is why the use of Varnish is not very recommended compared to Nginx.

It can be concluded that once Nginx has been set up as a reverse cache proxy, the system will exercise an improved performance when it comes to certain scenarios. The process of setting this up is very easy.

Chapter 9- Setting Up LAMP on Ubuntu hosted on AWS

Begin by booting up an instance of the AWS Ubuntu server, and then log into it by use of MobaXterm. Use the username "*ubuntu*" to log into the system. Once you are logged in, execute the following command:

```
sudo apt-get update
```

The next step should be execution of the following command:

```
sudo apt-get upgrade
```

The LAMP server can then be installed by use of the following command:

```
sudo apt-get install lamp-server^
```

Note that the password for root in MySQL should not be forgotten.

Our web root directory will be "*/var/www/html*," so there is a need to give ourselves permissions so that we can work from there. The following sequence of commands can be used for that purpose:

```
sudo chgrp www-data /var/www/html
sudo chmod 775 /var/www/html
sudo chmod g+s /var/www/html
sudo usermod -a -G www-data ubuntu
sudo chown ubuntu /var/www/html/
```

You have to install curl, as it will be needed for the LAMP server to work. It can be

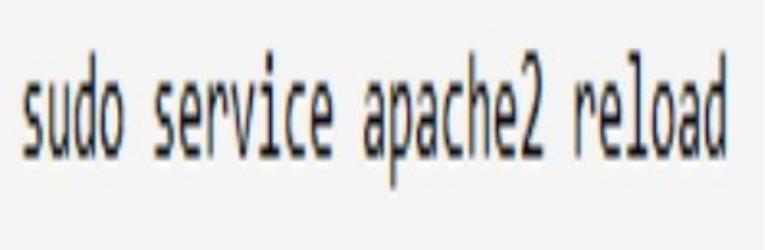
installed as follows:

```
sudo apt-get install php5-curl
```

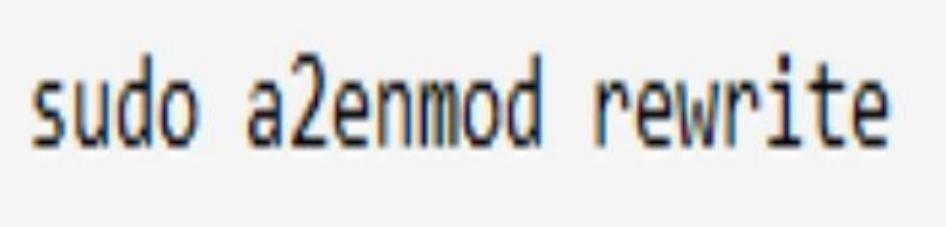
Mcrypt will also be needed, so it can be installed as follows:

```
sudo apt-get install php5-mcrypt
sudo php5enmod mcrypt
```

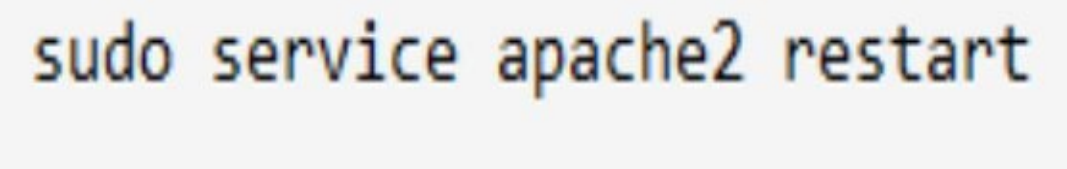
Once you are through with the installation, reboot the Apache by use of the following command:



The next step should involve sorting out of the Mod-rewrite. The following command can be used for this purpose:



The Apache can then be restarted so that the above change can take effect or can be applied. The following command can be used for this purpose:



You can then navigate to the web root by use of the command given below:

```
cd /var/www/html/
```

You will then be done, meaning that the LAMP server will be ready for use on your system.

Chapter 10- Using Nginx with a Web Application

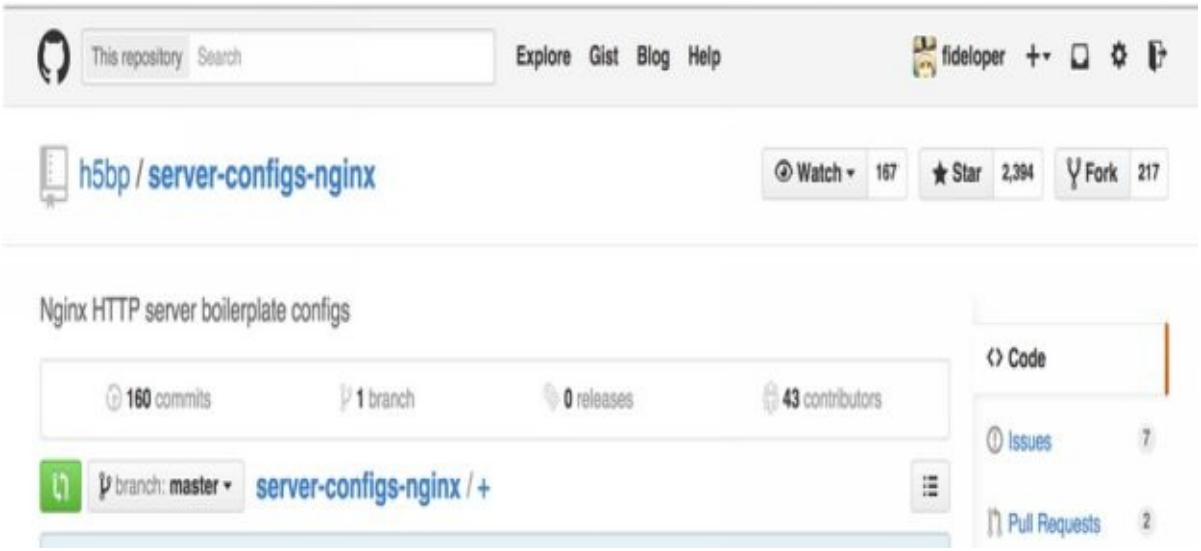
With Nginx, communication with web applications which have been developed dynamically is made possible and a bit easy. It can also be used for distribution of traffic among backend servers. Just like Varnish, it can also be used for the caching of web content.

Configuration of Nginx Server

Consider the code given below:

```
server {
listen 80 default_site;
root /var/www;
index index.html index.htm; !
server_name server_name.com www.server_name.com; !
location / {
try_files $uri $uri/ =404;
}
}
```

The above code shows a basic configuration of Nginx so as to serve files on a particular website. It listens on port number 80, which is just a regular HTTP port. We must tell it what the default site is. This is the site where Nginx will go if it receives a request which has no website which has been specified. The root of the web also has to be set, and this is where we will store our files for the web application.



For the purpose of server configuration setup, it is recommended that you use H5BP. Once the Nginx has been installed on the distributions of Linux such as Debian, Ubuntu, Fedora, and others, the sites will be enabled by default, and the structure of the sites will be available.

With Nginx, requests can be sent to HTTP listeners, to WSGI listeners, to fastCGI listeners, and communication with memcache can be done directly. Some fancy caching can be done in HTTP as shown below:

```
location /static {
try_files $uri $uri/ =404;
}
location / {
proxy_pass 127.0.0.1:9000;
proxy_param APPENV production;
include proxy_params;
}
```

The above code shows how one can proxy their request from Nginx to another application. Whenever you have an application which is using Nginx, then the Nginx will be placed at the front of the application so as to receive requests from users, but you come up with a way to tell or guide it on how to handle static files. You should also tell it the time that it should send out a request from the application.

Consider the code given below:

```
location ~ .php$ {
```

fastcgi_split_path_info ^(.+.php)(/.+)$;

fastcgi_pass 127.0.0.1:9000;

 # Or:

#fastcgi_pass unix:/var/run/php5-fpm.sock;

fastcgi_index index.php;

fastcgi_param APPENV production;

include fastcgi.conf;

}

Note that with PHP-FPM, listening to ports is done by use of fastCGI rather than using HTTP. However, in our case above, we are listening to the port number 9000 which is located on the local host. An HTTP request will not be accepted in the above case. Only requests which are made in fastCGI will be accepted, and this specifies the standard way how PHP files are processed.

For the case of HTTP proxies, non-static URLs are matched and then passed into an application. In PHP, this only happens with files having a "*.php"* extension, as they are the ones which are matched and then passed into the application.

The type of server information which PHP is to use has also been specified.

location /static {

try_files $uri $uri/ =404;

}

location / {

uwsgi_pass 127.0.0.1:9000;

uwsgi_param APPENV production;

include uwsgi_params;

}

Lastly, the HTTP request is taken away by WSGI Nginx, in which it is converted into the correct protocol that the gateway uses, and then it is sent off to the gateway. The gateway will then communicate with the application where the request is processed, and then the response is sent back.

Nginx as a Load Balancer

Nginx makes a good load balancer, just like other software such as HAProxy. It can be configured as follows for the purpose of load balancing:

```
upstream myapp {
zone backend 64k;
least_conn;
server 127.0.0.1:9000 max_fails=2 fail_timeout=30s;
server 127.0.0.1:9001 max_fails=2 fail_timeout=30s;
server 127.0.0.1:9002 max_fails=2 fail_timeout=30s;
}
server {
location / {
health_check;
include proxy_params;
proxy_pass http://myapp/;
# handling of the web socket connections
proxy_http_version 1.1;
proxy_set_header Upgrade $hupgrade;
```

proxy_set_header Connection “upgrade”;

}

}

Upstream are the load balancers to be load balanced within. Note that we have only three servers, and all are listening to port number 9000. Consider the code given below:

server {

location / {

health_check;

include proxy_params;

proxy_pass http://myapp/;

 # Handling of the Web Socket connections

proxy_http_version 1.1;

proxy_set_header Upgrade $hupgrade;

proxy_set_header Connection “upgrade”;

}

Our aim is to stop the Nginx from sending its requests to the servers which might have broken down. That is why we have the above parameter. The Health check is responsible for checking this.

Conclusion

DevOps (Developer Operations) is a software development methodology which was introduced so as to bring collaboration, communication, and cooperation between the software developers and other professionals in an organization. With DevOps, the degree of cooperation among the software developers and other IT professionals in the organization is improved.

This will translate to the fact that software will be rapidly developed and released into the market, which is an advantage to the software organization. The software will also be of high quality, due to the increased degree of cooperation between the software developers. With the earlier software development methodologies, a disconnect existed between the processes of development and operations activity.

This usually led to conflicts between the software developers, which translated to software of poor quality being developed and released into the market. The software was also of poor quality. This is why DevOps was introduced, and it effectively solves these problems. With Puppet, we find it possible for us to configure systems declaratively. Our task is just to declare the resources which we need or the ones that we have, and then declare their state. We will then be done. The configuration is then stored in system files. Note that we are not encouraged to perform a configuration when our servers are live. What we should do is to create new servers having the upgrades and the updates, and then start using them rather than using the old servers.