```
# Execute ls -ltr command on local machine.
def calling_local():
  local("ls -ltr")

# Creates directory, creates files and list them on remote machine's.
def test_run():
  run("mkdir -p /tmp/testdir && ls /tmp/testdir")
  run("touch /tmp/testdir/file86")
  run("ls -ltr /tmp/testdir")

# Installs tree package on remote machine with sudoers privilege.
def test_sudo():
  sudo("apt-get install tree")

# Push file devfile4 to a remote machine on path /home/vagrant/ like scp.
def test_put():
  put("devfile4", "/home/vagrant/")

# Takes user input & pull file from remote machine to local machine.
def test_get():
  filepath=prompt("Enter file path to download")
  get(filepath, ".", use_sudo=True)
```

Calling Fab functions.

When we call fab functions from shell we may or may not need to pass some arguments depending on what is running inside the fab function. For example if I am calling a local method then it just runs OS tasks/commands but if I am calling run or sudo method then it needs the remote server IP and credentials to create an SSH session.

If we have to pass remote server IP and user/pass we use below syntax.

```
fab -H <IP> -u <username> -p '<password>' functioname
```

This remote server information can also be added into the fabfile by using **env.hosts, env.user and env.password** variables which we will see in next example.

```
File  Edit  View  Search  Terminal  Help
imran@DevOps:.../fabric-scripts$ fab calling_local
[localhost] local: ls -ltr
total 16
-rw-rw-r-- 1 imran imran 3034 Jun  1 06:37 Vagrantfile
-rw-rw-r-- 1 imran imran   33 Jun  1 06:40 devfile4
-rw-rw-r-- 1 imran imran  451 Jun  1 06:44 fabfile.py
-rw-rw-r-- 1 imran imran 1360 Jun  1 06:44 fabfile.pyc

Done.
imran@DevOps:.../fabric-scripts$ fab -H 192.168.1.8 -u vagrant -p 'vagrant' test_run
[192.168.1.8] Executing task 'test_run'
[192.168.1.8] run: mkdir -p /tmp/testdir && ls /tmp/testdir
[192.168.1.8] out: file86
[192.168.1.8] out:

[192.168.1.8] run: touch /tmp/testdir/file86
[192.168.1.8] run: ls -ltr /tmp/testdir
[192.168.1.8] out: total 0
[192.168.1.8] out: -rw-rw-r-- 1 vagrant vagrant 0 Jun  1 01:19 file86
[192.168.1.8] out:


Done.
Disconnecting from 192.168.1.8... done.
imran@DevOps:.../fabric-scripts$ fab -H 192.168.1.8 -u vagrant -p 'vagrant' test_sudo
[192.168.1.8] Executing task 'test_sudo'
[192.168.1.8] sudo: apt-get install tree
[192.168.1.8] out:
[192.168.1.8] out: Reading package lists... 0%
[192.168.1.8] out:
```

```
imran@DevOps:.../fabric-scripts$ fab -H 192.168.1.8 -u vagrant -p 'vagrant' test_put
[192.168.1.8] Executing task 'test_put'
[192.168.1.8] put: devfile4 -> /home/vagrant/devfile4

Done.
Disconnecting from 192.168.1.8... done.
imran@DevOps:.../fabric-scripts$ fab -H 192.168.1.8 -u vagrant -p 'vagrant' test_get
[192.168.1.8] Executing task 'test_get'
Enter file path to download /var/log/syslog
[192.168.1.8] download: /tmp/fabric-scripts/syslog <- /var/log/syslog

Done.
Disconnecting from 192.168.1.8... done.
```

Fabric Helpers.

`cd` context manager allows keeping the directory state (i.e. where the following block of comments are to be executed). It is similar to running the *cd* command during an SSH session and running various different commands.

Usage examples:

```
# executing command from a particualr directory.

with cd("/tmp/"):

    items = sudo("ls -l")
```

The `lcd` context manager (local cd) works very similarly to one above (cd); however, it only affects the local system's state.

Usage examples:

```
# Change the local working directory to /opt/nexus
# and upload a file
with lcd("/opt/nexus"):
    print "Uploading the project archive"
    put("nexus.tar.gz", "/tmp/nexus.tar.gz")
```

Check Fab Documentation for more helpers.

http://docs.fabfile.org/en/1.13/api/core/context_managers.html

Example Fabfile for automating apache setup

In this fabfile we will write functions to setup apache and also cleaning apache setup on a remote machine. We will use env.hosts, env.user and env.password variables to define remote server information. env.hosts is a list and we can add n number of IP/hostnames in the list separated by a comma.

```
from fabric.api import *
env.hosts=['192.168.1.9']
env.user='vagrant'
env.password='vagrant'

def apache_install():
  sudo("apt-get install apache2 -y")

def apache_start():
  sudo("service apache2 start")

def apache_enable():
```

```
  sudo("sudo update-rc.d apache2 enable")


def push_index():
  put("index.html", "/var/www/index.html", use_sudo=True)


def apache_restart():
  sudo("service apache2 restart")


def disable_firewall():
  sudo("service ufw stop")


# Calling all the functions defined above from a single function.
def assemble_apache():
  apache_install()
  apache_start()
  apache_enable()
  push_index()
  apache_restart()
  disable_firewall()


# Define all the fabric methods to clean apache setup in just one function.
def dismantle_apache():
 sudo("service apache2 stop")
 sudo("apt-get remove apache2 -y")
 sudo("sudo update-rc.d apache2 disable")
 sudo("service ufw start")
```

From the above script, we have seen that initially while doing setup we have given only one fabric method in one function. But while dismantling/cleaning apache we have given all the required fabric method in just one function. Both the approach works fine but if I have separate function to start or install apache or setup firewall rule I can also call them individually from bash shell. This is very convenient if I have just start apache on an array of remote host I already have a separate function defined for it. Like this we can reuse this code for some other tasks also.

Example Fabfile for automating Tomcat setup

cat tomcat-user.xml

```
<?xml version='1.0' encoding='utf-8'?>
-->
<tomcat-users xmlns="http://tomcat.apache.org/xml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://tomcat.apache.org/xml tomcat-users.xsd"
version="1.0">

-->
<role rolename="manager-script"/>
<role rolename="manager-gui"/>
<user username="tomcat" password="tomcat" roles="manager-script,manager-gui"/>
</tomcat-users>
**********************************************************************
```

cat context.xml

```
<Context antiResourceLocking="false" privileged="true" >
<!--
<Valve className="org.apache.catalina.valves.RemoteAddrValve"
allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1" />
-->
</Context>
```

cat fabfile.py

```
# Author: Srini Vas

import boto

from fabric.api import *

env.hosts=['public ip of aws Instance']
env.user='ubuntu'
env.key_filename=['~/.ssh/tomcat-keypair.pem']

def update():
  sudo("apt-get update -y")
  sudo("sudo apt-get install default-jdk -y")

def tomcat8():
  sudo("apt-get install tomcat8 -y")

def admin():
  sudo ("apt-get install tomcat8-docs tomcat8-examples tomcat8-admin")

def user_tom():
  put("tomcat-users.xml" , "/var/lib/tomcat8/conf", use_sudo = True)

def user_manage():
  put("context.xml", "/var/lib/tomcat8/webapps/ROOT/META-INF/", use_sudo = True)

def start():
  sudo("systemctl start tomcat8")
```

```
def ufw_stop():
  sudo("service ufw stop")

def restart():
  sudo("systemctl restart tomcat8")

def fab_tom():
  update()
  tomcat8()
  admin()
  user_tom()
  user_manage()
  start()
  ufw_stop()
```

Now Tomcat8 is working
<ip of tomcat>:8080

Example fabfile for setting up Jenkins on a Ec2 instance.

```
# Author: Karunakar G

from fabric.api import *
env.hosts = ['IP']
env.user = 'ubuntu'
env.key_filename = ['~/.ssh/jenkins_server_key.pem']


def add_key_rep():
  sudo("wget -q -O - https://pkg.jenkins.io/debian/jenkins-ci.org.key | sudo apt-key add -")
  sudo("sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ >
/etc/apt/sources.list.d/jenkins.list'")
  sudo("sudo add-apt-repository ppa:openjdk-r/ppa -y")


def apt_get_update():
  sudo("sudo apt-get update")

def jenkins_git_maven_install():
  sudo("sudo apt-get install openjdk-7-jdk -y")
  sudo("sudo apt-get install jenkins -y")
  sudo("sudo apt-get install git -y")
  sudo("sudo apt-get install maven -y")
```

```
def jenkins_start():
  sudo("sudo systemctl start jenkins")


def jenkins_git_maven_status():
  sudo("sudo service jenkins status")

def jenkins_restart():
  sudo("sudo service jenkins restart")


def jenkins_firewall():
  sudo("service ufw stop")
```

```
def jenkins_setup():
  add_key_rep()
  apt_get_update()
  jenkins_git_maven_install()
  jenkins_start()
  jenkins_git_maven_status()
  jenkins_firewall()
```

12. Boto for AWS

Boto is a python library which provides an interface to interact wth AWS services.

Installation

$ sudo pip install boto

Configuration

We need to setup AWS authentication, so boto get authenticated to AWS services.

In order to do that an IAM user has to be created with programmatic access.

Create a ~/.boto file with below syntax:

```
[Credentials]

aws_access_key_id = YOURACCESSKEY

aws_secret_access_key = YOURSECRETKEY
```

Creating a connection for S3

```
>>> import boto

>>> conn = boto.connect_s3()
```

Creating a bucket

```
>>> bucket = s3.create_bucket('boto-demo-%s' % int(time.time()))
```

Accessing A Bucket

```
>>> mybucket = conn.get_bucket('mybucket') # Substitute in your bucket name

>>> mybucket.list()
```

...listing of keys in the bucket..

Deleting A Bucket

Removing a bucket can be done using the `delete_bucket` method. For example:

```
>>> conn.delete_bucket('mybucket') # Substitute in your bucket name
```

Ec2 Interface

Creating a Connection

The first step in accessing EC2 is to create a connection to the service. The recommended way of doing this in boto is:

```
>>> import boto.ec2
>>> conn = boto.ec2.connect_to_region("us-west-2")

At this point the variable conn will point to an EC2Connection object.
```

Launching Instances

To launch an instance and have access to it, you need to first setup security group and key pair.

Now, let's say that you already have a key pair, want a specific type of instance, and you have your security group all setup. In this case we can use the keyword arguments to accomplish that:

```
>>> conn.run_instances(
        '<ami-image-id>',
        key_name='your-key-name-here',
        instance_type='t2.micro',
        security_groups=['your-security-group-here'])
```

Stopping Instances

If you want to stop/shut down an instance, get its instance id and supply that as keyword argument.

```
>>> conn.stop_instances(instance_ids=['instance-id-1','instance-id-2', ...])
```

Terminating Instances

Similar to terminate an instance pass istance id's.

```
>>> conn.terminate_instances(instance_ids=['instance-id-1','instance-id-2',
...])
```

Checking What Instances Are Running

Using Elastic block storage.

EBS Basics

EBS can be used by EC2 instances for permanent storage. Note that EBS volumes must be in the same availability zone as the EC2 instance you wish to attach it to.

```
>>> vol = conn.create_volume(50, "us-west-2")
>>> vol
Volume:vol-0000000
>>> curr_vol = conn.get_all_volumes([vol.id])[0]
>>> curr_vol.status
u'available'
>>> curr_vol.zone
u'us-west-2'
```

We can now attach this volume to the EC2 instance we created earlier, making it available as a new device:

```
>>> conn.attach_volume (vol.id, inst.id, "/dev/sdx")
u'attaching'
```

Working With Snapshots

Snapshots allow you to make point-in-time snapshots of an EBS volume for future recovery. Snapshots allow you to create incremental backups, and can also be used to instantiate multiple new volumes. Snapshots can also be used to move EBS volumes across availability zones or making backups to S3.

Creating a snapshot is easy:

```
>>> snapshot = conn.create_snapshot(vol.id, 'My snapshot')
>>> snapshot
Snapshot:snap-00000000
```

Once you have a snapshot, you can create a new volume from it. Volumes are created lazily from snapshots, which means you can start using such a volume straight away:

```
>>> new_vol = snapshot.create_volume('us-west-2')
>>> conn.attach_volume (new_vol.id, inst.id, "/dev/sdy")
```

```
u'attaching'
```

If you no longer need a snapshot, you can also easily delete it:

```
>>> conn.delete_snapshot(snapshot.id)
True
```

A sample Boto script to spin up instance and attach EIP to it.

```
#!/usr/bin/python

import boto

import boto.ec2, time



## Variables

region =

ami_id =

login_key_name =

inst_type =

sbnet_id =

inst_name =


# Establishes connection with aws ec2 service.

conn = boto.ec2.connect_to_region(region)


# Spins up the ec2 instance.

ins = conn.run_instances(ami_id, key_name=login_key_name,
instance_type=inst_type, subnet_id=sbnet_id)


# Gets the instance id.

inst_id = ins.instances[0]

inst = str(inst_id)


# Allocates Elastic IP.

eip = conn.allocate_address(domain='vpc')
```

```
eip = str(eip)


# Prints status information.
print "%s is spiniing, please wait." % inst
status = inst_id.update()
print "Instance state %s" % status


# Checks the status of the instance.
while status == 'pending':
    time.sleep(10)
    status = inst_id.update()
    print "Instance state %s" % status


# If the instance status is running assigns elastic IP to it.
if status == 'running':
    inst_id.add_tag("Name",inst_name)
    conn.associate_address(inst[9:],eip[8:])
    print "<Instance Name> is running"
    print "Elastic IP -- %s" % eip
    print "Instance ID -- %s" % inst
else:
    print('Instance status: ' + status)
```

A Sample Dynamo DB table creation script.

DynamoDB is AWS NOSQL DB service.
Amazon DynamoDB is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability. DynamoDB lets you offload the administrative burdens of operating and scaling a distributed database, so that you don't have to worry about hardware provisioning, setup and configuration, replication, software patching, or cluster scaling.

```
import boto.dynamodb2
from boto.dynamodb2 import *
from boto.dynamodb2.fields import HashKey, RangeKey,KeysOnlyIndex,GlobalAllIndex
from boto.dynamodb2.table import Table
import sys
```

```
conn = boto.dynamodb2.connect_to_region('us-west-2',aws_access_key_id='',
aws_secret_access_key='')


env='Tesla'


def except_func(e):
  e_dict=e.__dict__['message']
  if 'Table already exists' in e_dict:
    print e_dict+', skipping create table.'
    return None
  else:
    print e_dict
    print "exiting db create script"
    sys.exit()


#sprint4
try:
  UserDisLikeProducts = Table.create(env+'AltCurrent',
schema=[HashKey('UserId'),RangeKey('ProductUrl')], connection=conn);
except Exception as e:
  print except_func(e)


print "Hey yo"
```

A Sample boto script to spin Beanstalk Windows .net platform

```
#!/usr/bin/python


import boto
import boto.beanstalk, time
import jenkins
import getpass



### Creating beanstalk instance
```

```
def bean_create(bean_name):


  conn = boto.beanstalk.connect_to_region('us-west-2')

  bean_info = conn.describe_applications(application_names=bean_name)

  bean_list_info =
bean_info['DescribeApplicationsResponse']['DescribeApplicationsResult']['Applica
tions']


  if not bean_list_info:

    print "Creating beanstalk application %s" % bean_name

  else:

    print "%s stalk already exists, please delete the stalk and run the script
again." % bean_name

    exit()


  conn.create_application(bean_name, description=bean_name)

  conn.create_application_version(bean_name, 'Sample_Application')

  conn.create_environment(bean_name, bean_name,
version_label='Sample_Application', solution_stack_name='64bit Windows Server
2012 R2 running IIS 8.5', cname_prefix=bean_name, description=None,
option_settings=[('aws:autoscaling:launchconfiguration', 'Ec2KeyName',
'testdigitaltestApp'), ('aws:autoscaling:launchconfiguration',
'IamInstanceProfile', 'aws-elasticbeanstalk-ec2-
role'),('aws:autoscaling:updatepolicy:rollingupdate', 'RollingUpdateEnabled')],
options_to_remove=None, tier_name='WebServer', tier_type='Standard',
tier_version='1.0')


  time.sleep(10)


  dict = conn.describe_events(application_name=bean_name,
environment_name=bean_name)

  time.sleep(3)

  event1 =
dict['DescribeEventsResponse']['DescribeEventsResult']['Events'][0]['Message']

  event2 =
dict['DescribeEventsResponse']['DescribeEventsResult']['Events'][0]['Message']


  print event1


  while 'Successfully' not in event1:
```

```
    time.sleep(5)
    dict = conn.describe_events(application_name=bean_name,
environment_name=bean_name)


    if 'Error' in event1:
      print event1
      print "Encountered error, please wipe out %s application from AWS
beanstalk and start over again" % bean_name
      exit()
    if event1 != event2:
      print event2
      event1 =
dict['DescribeEventsResponse']['DescribeEventsResult']['Events'][0]['Message']


    event2 =
dict['DescribeEventsResponse']['DescribeEventsResult']['Events'][0]['Message']
```

Summary:

- Python is an interpreted language, its highly extensible and comes with lot of goodies.
- Python is very easy to read and write compared to any other programming language.
- Numbers, Strings, List, Tuples & Dictionary are python data types. Python comes with lot of methods that be applied on these datatypes to manipulate or present data in different ways.
- Decision making(if/elif/else) and loops(for&while) in python has a very simple syntax and block of code inside them are indented mostly two or three spaces.
- Python functions are used when we want to group some code together and have it accessible from any other python code. We can use these functions also as modules and import them into other python scripts. This gives us a very modular structure for our complex code.
- OS python library is used to run system commands from python scripts like cd, mkdir, chmod, cp, mv etc. There are other libraries as well for system tasks like sub process and commands.
- Fabric is a python library used to run system commands on local and remote system. Most of the python code is abratracted from us, we just call fabric functions for automating linux tasks on local and remote system.
- Boto is a python system library to call, operate and manage AWS services from python.

Conclusion:

Being so versatile in nature, managing huge python code becomes difficult. When it comes to sharing our code with our team mates or other teams it is a big problem. Everybody have their own style of coding and most of us while doing automation dont follow any best practices which makes code difficult to read and understand. That is the reason world is moving or moved towards configuration management tools for doing automation. Diffrences will be discussed in Ansible chapter.

Some important links.

For practicing python with exercises.
https://www.codecademy.com/learn/python

Posting errors and questions.
https://stackoverflow.com/

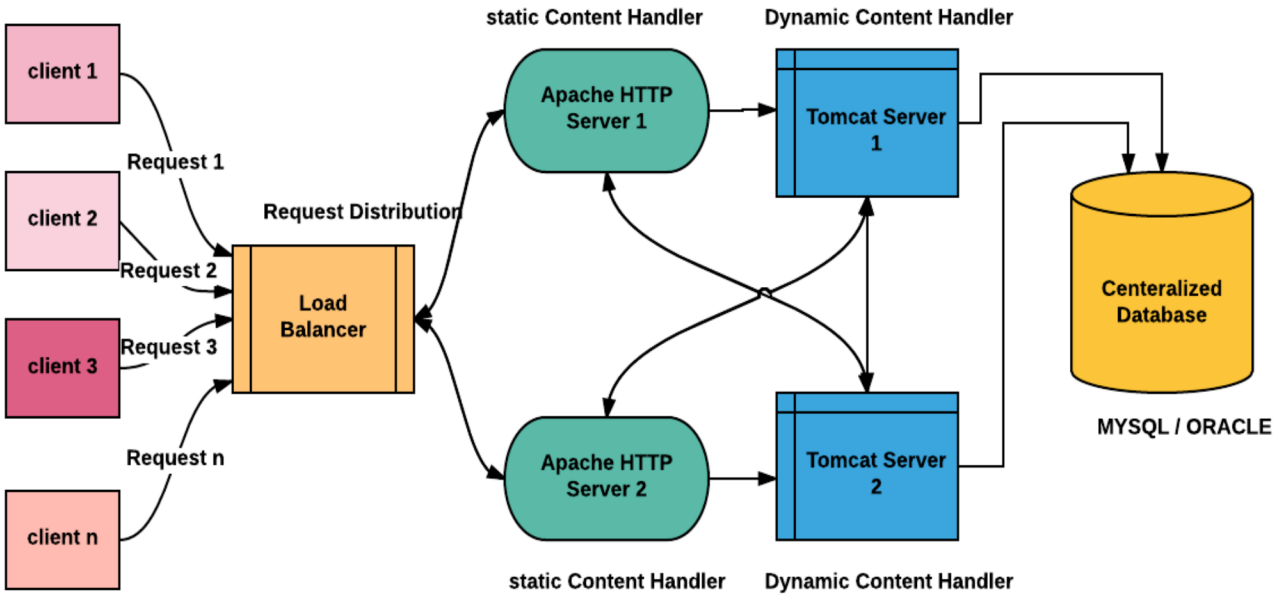
Fabric documentation.
https://docs.fabric.io/

Boto Documentation
https://boto3.readthedocs.io/en/latest/

VII. Web Application Architecture

This Chapter discusses a few architectural designs used for web applications. Each design provides one or more examples architectural solutions for modern web Application.

1. WEB APPLICATION ARCHITECTURE 1



2. The Client

Web development is all about communication and sharing Information . In this case, communication between two (2) parties, over the HTTP protocol:

The Server - This party is responsible for serving pages.

The Client - This party **requests pages** from the Server, and displays them to the user. In most cases, the client is a web browser.

a. The User - The user uses the Client in order to surf the web, fill in forms, watch videos online, etc.

3. Load balancing

Load balancing refers to efficient and skilful distribution of incoming network traffic across a group of backend servers, also known as a server farm or server pool.

Modern high-traffic websites must serve hundreds of thousands, if not millions, of concurrent requests from users or clients and return the correct text, images, video, or application data, all in a fast and reliable manner. To cost-effectively scale to meet these high volumes, modern computing best practice generally requires adding more servers.

4. Load balancer

A load balancer acts as the “traffic cop” that act as **traffic controller** in front of your servers and routing client requests across all servers capable of fulfilling those requests in a manner that maximizes speed and capacity utilization and ensures that no one server is **overloaded or overworked**, which could degrade performance of server or may let down the whole server. If a single server goes down, the load balancer redirects traffic to the remaining online servers. When a new server is added to the server group, the load balancer automatically starts to send requests to it.
In this manner, a load balancer performs the following functions:

Distributes client requests or network load efficiently across multiple servers
Ensures high availability and reliability by sending requests only to servers that are online
Provides the flexibility to add or subtract servers as demand dictates

5. Load Balancing Algorithms

There are different load balancing algorithms are available and each provide different benefits; the choice of load balancing method depends on your requirement:

- Round Robin – Requests are distributed across the group of servers sequentially.
- Least Connections – A new request is sent to the server with the fewest current connections to clients. The relative computing capacity of each server is factored into determining which one has the least connections.
- IP Hash – The IP address of the client is used to determine which server receives the request.

6. Dynamic Configuration of Server Groups

For a fast-changing application require new servers to be added or remove on a constant basis. This is common in environments such as the Amazon Elastic Compute Cloud (EC2), which enables users to pay only for the computing capacity they actually use, while at the same time ensuring that capacity scales up in response traffic spikes. In such environments, it greatly helps if the load balancer can dynamically add or remove servers from the group without interrupting existing connections.

7. Hardware vs. Software Load Balancing

Vendors of hardware-based solutions load proprietary software onto the machine they provide, which often uses specialized processors. To cope with increasing traffic at your website, you have to buy more or bigger machines from the vendor. Software solutions generally run on commodity hardware, making them less expensive and more flexible. You can install the software on the hardware of your choice or in cloud environments like AWS EC2.
Note : Here we include Nginx as load balancer.

8. NGINX / NGINX Plus As Load Balancer

NGINX Plus and NGINX are used by many developers for load balancing solutions for high-traffic websites such as Dropbox, Netflix, and Zynga. NGINX Plus and NGINX to deliver their content quickly, reliably, and securely. When you include NGINX Plus as a load balancer in front of your application and web server farms, it increases your website's efficiency, performance, and reliability. NGINX Plus helps you maximize both customer satisfaction and the return on your IT investments.

How To Setup Nginx As Load Balancer?

- **Install nginx**

The steps require the user to have root privileges on your virtual private server (VPS).Prior to setting up nginx load balancing, you should have nginx installed on your VPS. You can install it quickly with apt-get:

```
$ sudo apt-get install nginx
```

- **Upstream Module**

In order to set up a round robin load balancer, we will need to use the nginx upstream module. We will incorporate the configuration into the nginx settings.
Go ahead and open up your website's configuration (in my examples I will just work off of the generic default virtual host):

```
$sudo nano /etc/nginx/sites-available/default
```

We need to add the load balancing configuration to the file.
First we need to include the upstream module which looks like this:

```
upstream backend  {
  server backend1.example.com;
  server backend2.example.com;
  server backend3.example.com;
}
```

We should then reference the module further on in the configuration:

```
server {
  location / {
    proxy_pass  http://backend;
  }
}
```

- **Restart nginx**

```
$ sudo service nginx restart
```

As long as you have all of the virtual private servers in place you should now find that the load balancer will begin to distribute the visitors to the linked servers equally.

❖ **Directives**

The previous section covered how to equally distribute load across several virtual servers. However, there are many reasons why this may not be the most efficient way to work with data. There are several directives that we can use to direct site visitors more effectively.

Weight

One way to begin to allocate users to servers with more precision is to allocate specific weight to certain machines. Nginx allows us to assign a number specifying the proportion of traffic that should be directed to each server.

A load balanced setup that included server weight could look like this:

```
upstream backend  {
  server backend1.example.com weight=1;
  server backend2.example.com weight=2;
  server backend3.example.com weight=4;
}
```

The default weight is 1. With a weight of 2, backend 2.example will be sent twice as much traffic as backend 1, and backend 3, with a weight of 4, will deal with twice as much traffic as backend 2 and four times as much as backend 1.

Hash

IP hash allows servers to respond to clients according to their IP address, sending visitors back to the same VPS each time they visit (unless that server is down). If a server is known to be inactive, it should be marked as down. All IPs that were supposed to routed to the down server are then directed to an alternate one.

The configuration below provides an example:

```
upstream backend {
  ip_hash;
  server   backend1.example.com;
  server   backend2.example.com;
  server   backend3.example.com  down;
 }
```

Max Fails

Based on default round robin settings, nginx will continue to send data to the virtual private servers, even if the servers are not responding. Max fails can automatically prevent this by rendering unresponsive servers inoperative for a set amount of time.

There are two factors associated with the max fails: max_fails and fall_timeout. Max fails refers to the maximum number of failed attempts to connect to a server should occur before it is considered inactive. Fall_timeout specifies the length of that the server is considered inoperative. Once the time expires, new attempts to reach the server will start up again. The default timeout value is 10 seconds.

A sample configuration might look like this:

```
upstream backend  {
  server backend1.example.com max_fails=3  fail_timeout=15s;
  server backend2.example.com weight=2;
  server backend3.example.com weight=4;

}
```

9. Apache HTTP Server

The Apache HTTP server i s free and open-source cross-platform web server software, released under the terms of Apache. Apache is developed and maintained by an open community of developers under the Apache Software Foundation. The project develops and maintains an open-source HTTP server for modern operating system including UNIX and Windows . The Apache HTTP web server is still the most used web server worldwide.

Windows:
Download Installer Apache HTTP server here: (http://httpd.apache.org/download.cgi)
Be sure to download the apache_2.2.16-win32-x86-no_ssl MSI.
Note :the Apache HTTP server version may be different at the time of your installation

Linux:
On Ubuntu you can install the Apache HTTP server with the following command.
$ sudo apt-get install apache2

10. Tomcat Server

Tomcat, is also developed by Apache (www.apache.org), is a standard reference implementation for Java servlets and JSP. It can be used standalone as a Web server or be plugged into a Web server like Apache, Netscape Enterprise Server, or Microsoft Internet Information Server. There are many versions of Tomcat. Here we uses Tomcat 5.5.9 as an example. The same steps should also apply to all later versions of Tomcat.

Note : For Using Tomcat we need JDK installed in our system

JDK Setup:

- This step involves downloading an implementation of the Java Software Development Kit (SDK) and setting up the PATH environment variable appropriately.
- You can download SDK from Oracle's Java site − Java SE Downloads.
- Once you download your Java implementation, follow the given instructions to install and configure the setup. Finally set the PATH and JAVA_HOME environment variables to refer to the directory that contains java and javac, typically java_install_dir/bin and java_install_dir respectively.
- If you are running Windows and install the SDK in C:\jdk1.8.0_60, you need to add the following line in your C:\autoexec.bat file.
- Set PATH = C:\jdk1.8.0_60\bin;%PATH%
- set JAVA_HOME = C:\jdk1.8.0_60