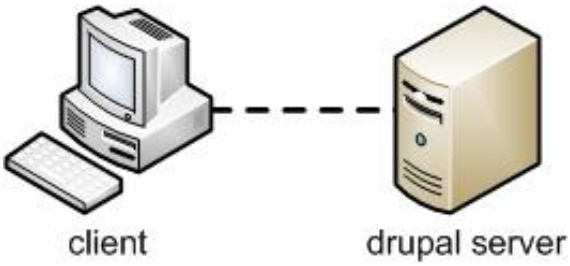
Simple Scalable Architectures

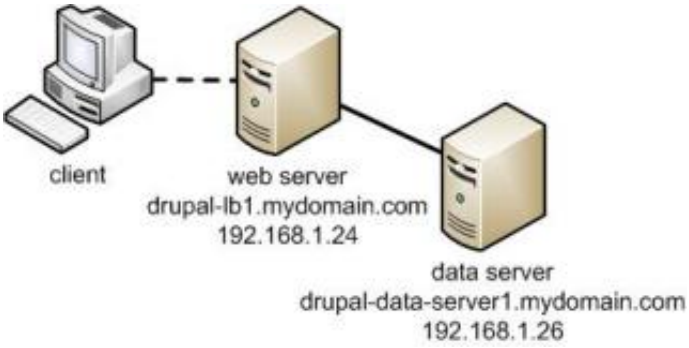
bud@thinkcube.com | twitter @geekaholic

In terms of scaling the web server there are few options.

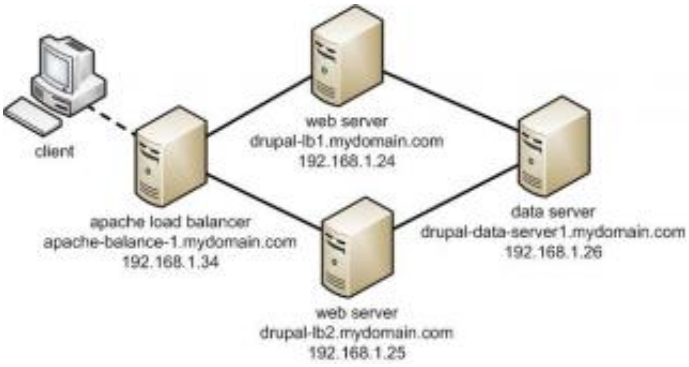
Basically the easiest to setup. Scaling is a matter of buying a better server or upgrading it!



Separate DB from App, as a result each can be scaled separately.



Load balancer *(aka reverse proxy)* will route requests betwen multiple backend HTTP servers while caching results.



.notes: Data scalability is beyond the scope of this presentation.

It is good to isolate the data from the app by hosting it on a separate server. This was the two aspects can be scaled independantly. Some methods to consider:

- Store DB data on MYSQL running on a separate server
- Enable file sharing to share data files using NFS, rsync
- Clustering MYSQL across multiple servers using mysqlcluster
- Cluster file system via DRDB, GFS2 or as Facebook does using Bittorrent

Setting up an HTTP accellerator using Apache

In this setup, the reverse server is what the user will contact while the *real webserver* can be hidden behind a private network.

Enable required modules for caching reverse proxy.

```
a2enmod proxy

a2enmod proxy_connect

a2enmod proxy_http

a2enmod cache
```

vi /etc/apache2/modules-enabled/proxy.conf

```
!apache
<Proxy *>
        AddDefaultCharset off
        Order deny,allow
        Deny from all
        Allow from all
</Proxy>
ProxyVia On
```

Next we configure an empty virtual host that is configured to the public site. But instead of showing the document root we do a reverse proxy.

> vi /etc/apache2/sites-available/public-domain.com

```
!apache
<VirtualHost *:80>

    ServerName your-public-domain.com

    <Proxy *>
        Order deny,allow
        Allow from all
    </Proxy>

    ProxyPass / http://your-private-domain.com/
    ProxyPassReverse / http://your-private-domain.com/

</VirtualHost>
```

> a2ensite public-domain.com
> service apache2 reload

Using Nginx

- Nginx was designed as a reverse proxy first, and an HTTP server second
- Unlike Apache, Nginx uses a non blocking process model

1. Use Nginx for the static content and Apache for PHP

2. Use FastCGI to embed PHP

- Receive request, trigger events in a process
- The process handles all the events and returns the output
- Process handles events in parallel
- *Limitation* is PHP can no longer be embedded (`mod_php`) inside process as PHP is not asynchronous
- Unlike Apache, Nginx doesn't not have an .htaccess equivelant. You need to reload server after making any chage, making it difficult to use for shared hosting

In this setup we put Nginx as the frontend http accellerator and Apache as the backend app server. If you want to run this on the same physical server you'll need to either change the Apache port from 80 to another value or bind and Nginx to their own IP addresses with the same server.

> Listen 8080

or using the ip address

> Listen 127.0.0.1:8080

Now we're ready to install Nginx

> sudo apt-get install nginx

Nginx uses a different format for defining virtual hosts than Apahche.

```
!apache
<VirtualHost>
      DocumentRoot "/usr/local/www/mydomain.com"
      ServerName mydomain.com
```

```
        ServerName mydomain.com
        ServerAlias www.mydomain.com
        CustomLog /var/log/httpd/mydomain_access.log common
        ErrorLog /var/log/httpd/mydomain_error.log
        ...
</VirtualHost>
```

becomes...

```
!nginx
server {
        root /usr/local/www/mydomain.com;
        server_name mydomain.com www.mydomain.com;

        # by default logs are stored in nginx's log folder
        # it can be changed to a full path such as /var/log/...
        access_log logs/mydomain_access.log;
        error_log logs/mydomain_error.log;
        ...
}
```

The following example will server all static content via nginx while redirect dynamic content (php) to Apache

```
!nginx
server {
    listen   80 default;
    server_name  localhost;

    access_log  /var/log/nginx/localhost.access.log;

    location / {
        root   /var/www;
        index  index.php index.html index.htm;
    }

     ## Parse all .php file in the /var/www directory
     location ~ .php$ {
        # these two lines tell Apache the actual IP of the client being
forwarded
        proxy_set_header X-Real-IP  $remote_addr;
        proxy_set_header X-Forwarded-For $remote_addr;

        # this next line adds the Host header so that apache knows which
```

```
vHost to serve
        proxy_set_header Host $host;

        # And now we pass back to apache
        proxy_pass http://127.0.0.1:8080;
    }
}
```

There is some debate as to whether using nginx with php via FastCGI is actually faster than redirecting to Apache. In anycase lets see how we can setup a pure nginx based model.

Unlike Apache, Nginx has has a hands off approach to managing php processes and therefore requires manual intervention. Fortunately as of PHP 5.3.3, there is a built in Front Process Manager (FPM), which looks after the php processes.

> apt-get install php5-fpm

If your on Ubuntu 10.04LTS then you'll need to add a special repository before you can install php5-fpm.

> add-apt-repository ppa:brianmercer/php && apt-get update

Next start the php5-fpm process

> service php5-fpm restart

Finally modify nginx configuration to use fast-cgi to redirect all files having the php extension.

> vi /etc/nginx/sites-available/defaul

vi /etc/nginx/sites-available/defaul

```
!nginx
server {
    listen   80 default;
    server_name  localhost;

    access_log  /var/log/nginx/localhost.access.log;

    location / {
        root   /var/www;
        index index.php index.html index.htm;
    }

     ## Parse all .php file in the /var/www directory
     location ~ .php$ {
        fastcgi_pass   127.0.0.1:9000;
        fastcgi_index  index.php;
        fastcgi_param  SCRIPT_FILENAME  /scripts$fastcgi_script_name;
        includefastcgi_params;
    }
}
```