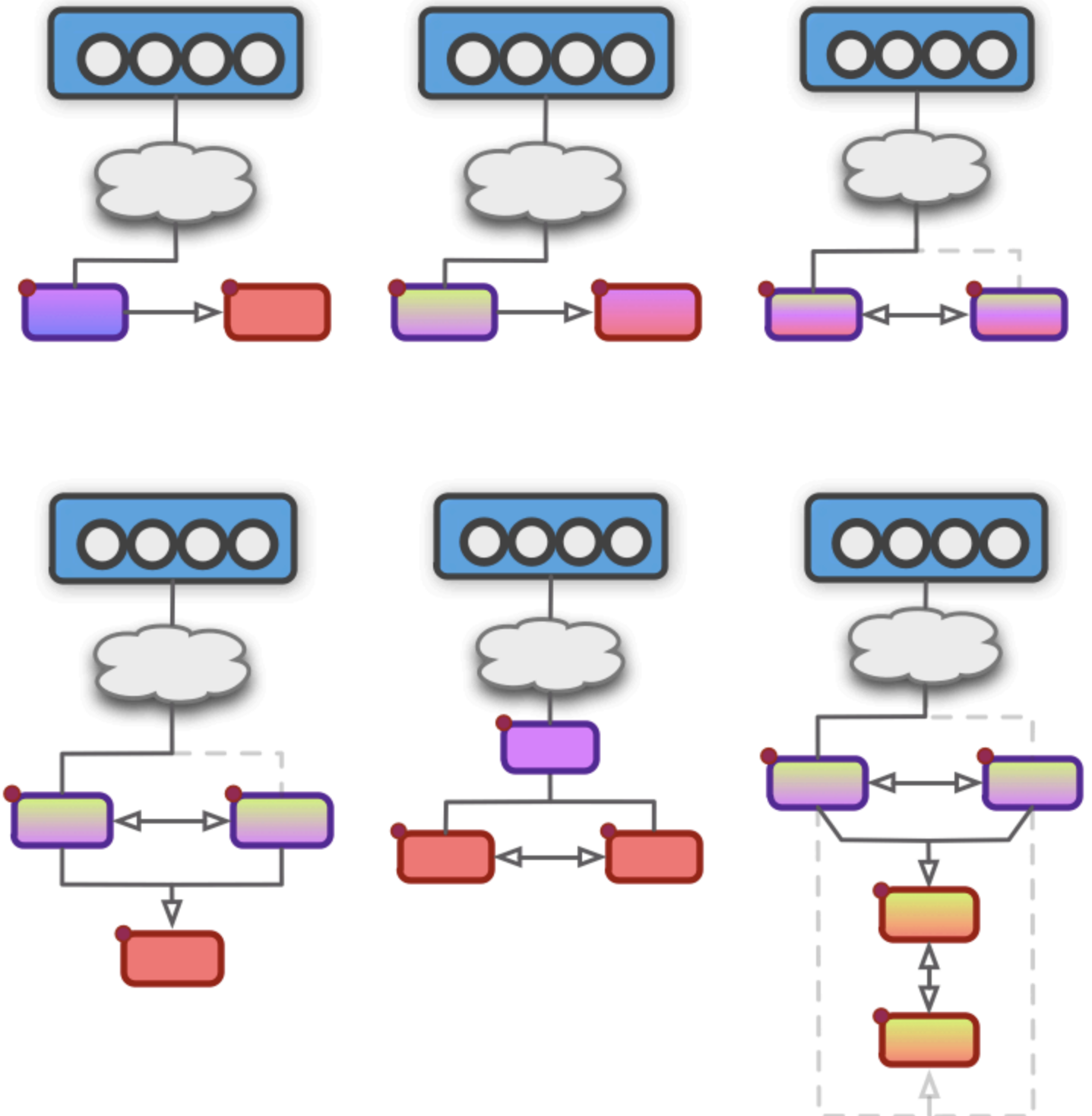


Web Scaling vol. 1

Small Architectures



The Scaling Experts

Web Scaling vol. 1

Copyright

Copyright 2011 © by The Scaling Experts / Alex Williams and can in no way be distributed, printed for commercial use or resold. Please respect the author and contributors by not infringing on the copyright. Thank you.

All rights reserved under International and Pan-American Copyright Conventions.

We strongly believe in DRM-free content, therefore you **may** print this eBook for personal use and you may also create backups of this eBook for **your own personal use**. You may also convert this eBook to audio version for your own personal use. You may **not** create translations of this eBook without our consent. You may not copy graphics, logos, images or text from this eBook to be used in any other materials, whether online, offline or anywhere in between.

Published in Quebec, Canada by Alex Williams, www.AlexWilliams.ca

July 2011, First Edition, v1.1

ISBN: 978-2-924017-00-5

Website:	www.ScalingExperts.com
Email:	scalingexperts@gmail.com
Twitter:	@ScalingExperts

DISCLAIMER

All trademarks are used with respect to their owners and are designated in *italics* wherever possible. We claim no affiliation with any of the people, software, companies or trademarks listed in this eBook, except for our own.

While every precaution has been taken in the preparation of this eBook, the publisher and authors assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein. Please forgive us if some examples don't work as intended. We will try our best to update this eBook with corrections and provide purchasers with a FREE updated version of this edition upon request.

Please visit our website for information on updates and revisions to this eBook.

Table of Contents

- Copyright
- Table of Contents
- Acknowledgements
- Preface
 - Intended audience
 - eBook layout
- 1. Monitoring
 - 1.1. Sysstat
 - 1.2. Munin
- 2. Profiling
 - 2.1. Sysstat
 - 2.2. Other tools
- 3. Small Architecture #1
 - 3.1. Diagram
 - 3.2. Description
 - 3.3. Advantage
 - 3.4. Diagram Explanation
 - 3.5. Failure Scenarios
 - 3.6. How-to
- 4. Small Architecture #2
 - 4.1. Diagram
 - 4.2. Description
 - 4.3. Advantage
 - 4.4. Diagram Explanation
 - 4.5. Failure Scenarios
 - 4.6. How-to
- 5. Small Architecture #3
 - 5.1. Diagram
 - 5.2. Description
 - 5.3. Advantage
 - 5.4. Diagram Explanation
 - 5.5. Failure Scenarios
 - 5.6. How-to
- 6. Small Architecture #4
 - 6.1. Diagram
 - 6.2. Description
 - 6.3. Advantage
 - 6.4. Diagram Explanation
 - 6.5. Failure Scenarios
 - 6.6. How-to
- 7. Small Architecture #5
 - 7.1. Diagram
 - 7.2. Description
 - 7.3. Advantage
 - 7.4. Diagram Explanation
 - 7.5. Failure Scenarios
 - 7.6. How-to
- 8. Small Architecture #6
 - 8.1. Diagram
 - 8.2. Description
 - 8.3. Advantage
 - 8.4. Diagram Explanation (pt.2)
 - 8.4. Diagram Explanation (pt.2)
 - 8.5. Failure Scenarios
 - 8.6. How-to
- 9. Software Installation
 - 9.1. Installing Memcached with PHP support
 - 9.2. Installing Nginx
 - 9.3. Installing Keepalived
 - 9.4. Installing HAProxy

- 10. Scripts and commands
 - 10.1. IP Address Setup
 - 10.2. Database Server Setup
 - 10.3. Transfer Database
 - 10.4. Transfer Website
 - 10.5. Servercheck Setup
 - 10.6. Replication Setup
- 11. Configuration Files
 - 11.1. Memcached Configuration
 - 11.2. Nginx Configuration
 - 11.3.1. Configure Keepalived for Architecture #3 (pt.1)
 - 11.3.2. Configure Keepalived for Architecture #3 (pt.2)
 - 11.4. Configure Keepalived for Architecture #4
 - 11.5. Configure Keepalived for Architecture #5
 - 11.6.1 Configure Keepalived on the web servers for Architecture #6
 - 11.6.2 Configure Keepalived on the database servers for Architecture #6
 - 11.7. MySQL Replication Configuration
 - 11.8. Configure HAProxy
- 12. Code samples
 - 12.1. Use Memcached and MySQL
 - 12.2. Use the local database for Reads, and Master database for Writes
 - 12.3. Use MySQL on an isolated server
 - 12.4. Use a random database server
 - 12.5. Use separate database servers for reads and writes
- Conclusion
- Links
 - Open Source Software
 - Web sites and blogs

Acknowledgements

This eBook could not have been written without the direct and indirect help of some of the IT industry's brightest and skilled infrastructure scaling experts. Many thanks are owed to people and organizations for their support, input, advice, comments and contributions to me and the Open Source community, and to the high availability, scalability, capacity planning resources provided to fuel the creation of this eBook.

My best friend, Patrice LaFlamme for constantly pushing me to DO IT. My previous co-workers at iWeb Technologies. Many thanks to Werner Vogels, James Hamilton, Todd Hoff, Google, Flickr, Twitter, GitHub, 37signals, and other scaling industry experts for sharing your knowledge. Thanks to the creators of Open Source Software such as Linus, Willy Tareau (HAProxy), Alexandre Cassen (Keepalived), and many more. Finally, thank you, the reader, for taking the time to purchase and read this eBook.

Preface

| Scalability (n.): The potential for your website to continue to function effectively as its size and popularity increases.

Intended audience

This eBook's focus is mainly for Linux users and assumes you need to scale your Linux system.

If you're reading this eBook, it's most likely because you have a problem. You need to scale.

Well, one solution is to add more RAM, more disk drives, a bigger CPU... but what do you do when your server just can't grow any bigger? You need to scale out (horizontally) by adding additional servers to your architecture.

But how? How do you scale from your tiny little server to an architecture that not only supports more users, but that also allows temporary failures and solves intermittent issues? How do you do this when you have no idea what's out there? Sure there's tons of resources online, if you have time to look for them, and try to understand how they work, and then implement the solutions. But, who has time?

If your server randomly reboots every 12 hours, you need to find out why. You need to monitor your server, you need to profile it and find out exactly what's causing problems, and then you need to do something about it. Quickly!

Many books will teach you the art of scalability. That's extremely important if you're a system/network administrator or IT director, and you have large setups to maintain and manage. We strongly encourage you to purchase those books and learn the art and theory to scaling.

On the other hand, if you need a quick and functional solution, this eBook is for you. We don't claim to have the absolute best solutions, but we have ones that work. We want you to be able to implement one of our Small Architectures and solve your problems immediately. Once that's done, you can go ahead and learn the art to make sure these problems don't re-occur. In the meantime, we hope to guide you in the right direction so that you can move on with your life and get some better sleep at night.

| Failover (n.): The capability to automatically switch to a redundant or standby server during failure of one or more a components.

eBook layout

We chose to provide this guide in eBook format only, because it has some neat features such as inline links. We've used certain conventions throughout this eBook to make life simpler for you.

You don't need to read this eBook from top to bottom.

Chapter 1 is about monitoring. It covers the first thing you should do to your server once it's installed. Setup monitoring in order to accurately track exactly what went wrong, and when. This should be standard practice on all servers you install. We'll provide you with a list of our favorite monitoring tools.

Chapter 2 is about profiling. If your server is having problems, you need tools to find out what's causing them. Profiling tools will give you an instant snapshot of activity, whether it's related to disk IO, memory consumption or CPU usage. These tools combined with monitoring tools will help you quickly identify and fix almost any problem on your Linux servers.

Chapters 3 to 8 are examples of Small Architectures. These *recipes* will show you different ways to scale from 1 to 4 servers. Each chapter contains a short description of the architecture, a diagram and explanations on how it should work. The *How-to* section of each architecture describes steps required to scale to that architecture. Some steps apply to more than 1 setup. If you click the inline-link, you'll be taken to the chapter with example code and commands. That chapter will also provide a link back to the architecture you were reading, this way you can quickly skip back and forth between examples and How-to. Some steps have been excluded because we assume you already know how to do them (such as installing an OS, securing your server, connecting via SSH).

Chapter 9 contains a list of commands for installing certain applications.

Chapter 10 contains a list of scriptable BASH commands for performing certain tasks.

Chapter 11 contains a list of configuration files for different applications.

Chapter 12 contains a list of PHP scripts as examples of how to modify your web application to scale to this architecture. If you need to send database READ requests to 1 server, and WRITE requests to a different one, these examples can help.

At the end of the eBook, we provide a list of links to obtain more information on the Open Source Software used throughout this eBook. It also contains a list of blogs and websites which can and should be viewed for more information on scaling.

1. Monitoring

Is your server constantly crashing, becoming unavailable or spiking in traffic at odd times, yet you have no idea why?

The first thing you should consider is installing some monitoring tools. We'll start by recommending 2 indispensable tools:

1.1. Sysstat

- *Sysstat* is a set of tools to gather performance information about your Linux server. It stores that information for you to go back in time to review.

```
1. apt-get install sysstat
```



Warning

***Sysstat* is not enabled by default in Debian Lenny 5.0, so edit `/etc/default/sysstat` and set the `ENABLED` option to `true`**

```
# From configuration file: '/etc/default/sysstat'  
ENABLED="true"
```

- You can see a lot of past system information with the command-line tool `sar`

```
# Show CPU info  
1. sar -t  
...  
18:25:01      CPU      %user    %nice    %system  %iowait  %steal    %idle  
18:35:01      all       0.05     0.09     0.18     0.03     0.00     99.64  
18:45:01      all       0.06     0.07     0.19     0.02     0.00     99.65  
...  
# Show memory info  
2. sar -r  
...  
18:25:01      kbmemfree kbmemused  %memused kbbuffers  kbcached  kbswpfree  kbswpused  %swpused  
kbswpcad  
18:35:01      116532    318764    73.23    154868    103224    524284     0         0.00  
0  
18:45:01      111292    324004    74.43    154876    108424    524284     0         0.00  
0  
...  
# Show network info for eth0  
3. sar -n DEV | grep eth0  
...  
18:25:01      IFACE    rxpck/s  txpck/s  rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcast/s  
18:35:01      eth0     0.27     0.73     0.03     0.76     0.00     0.00     0.00  
18:45:01      eth0    12.18     7.63    16.23     2.28     0.00     0.00     0.00  
...
```


1.2. Munin

- *Munin* will allow your system to collect data on just about anything running on your system. You install `munin-node` (client daemon) on every system, and install `munin` (server daemon) on one server running a web server software such as *Nginx*. The server daemon will then query the client at regular intervals in order to generate lovely graphs and HTML pages for all your data.

```
# Install munin-node on EVERY server
1. apt-get install munin-node

# Install munin on a server running a web server software
2. apt-get install munin-node
```

- Configuring *Munin* is quite simple.
 - Make sure you edit the `/etc/munin/munin-node.conf` file and only allow your server's local subnet from connecting to it.
 - Also make sure you grab some plugins from `/usr/share/munin/plugins/` and copy them into `/etc/munin/plugins`.



Tip

You don't have to copy EVERY munin plugin, only the ones you'll actually need. If you're not running *Tomcat*, don't copy the `tomcat_` plugins.



Another Tip

Some plugins, such as the `ip_` plugin must be renamed, for example: `ip_192.168.0.11`.

- Modify the `/etc/munin/plugin-conf.d/munin-node` file to reflect the new plugins you've added.
 - Make sure to restart the `munin-node` service once you've made changes to the `munin-node` file.
- The *Munin* server daemon runs as a cron job every 5 minutes. If you're monitoring many servers, you'll want to add them to the `/etc/munin/munin.conf` file, otherwise their graphs will never be generated.



Tip

It's a good idea to protect your public *Munin* directory with a password (`.htpasswd`) and even use SSL to access the graphs page. Those graphs contain a lot of sensitive information about all your servers, so you'll want to keep them protected.

```
# Sample configuration file: '/etc/nginx/sites-enabled/000-default'
server {
    listen 172.16.0.10:443;
    ssl on;
    ssl_certificate /etc/nginx/ssl/site.com.crt;
    ssl_certificate_key /etc/nginx/ssl/site.com.key;
    ssl_protocols SSLv3 TLSv1;
    ssl_ciphers TLSv1+HIGH:!SSLv2:!aNULL:!eNULL:!3DES:@STRENGTH;
    #ssl_ciphers ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP;
    ssl_prefer_server_ciphers on;

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

    location / {
        root /var/www/nginx-default;
        index index.html index.htm;
    }

    location /munin {
        root /var/www;
        expires off;
        auth_basic "Munin";
        auth_basic_user_file /etc/nginx/.htpasswd;
    }
}
```

2. Profiling

2.1. Sysstat

- With *Sysstat*, you can also watch current activity

```
# Show CPU and Disk IO, refresh every 1 second, total 3 times
1. iostat 1 3
...
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.05    0.08   0.18   0.01   0.00   99.67

Device:            tps    Blk_read/s    Blk_wrtn/s    Blk_read    Blk_wrtn
xvda                1.68         0.26         29.43       265282     30515720
xvdb                 0.00         0.00         0.00         336         0
...
```

2.2. Other tools

All commands listed below have tons of different command-line arguments to output different types of information. Please learn them since we only provide some basic arguments.

✔ lsof

```
# Show all TCP/UDP ports used and listening over IPv4 and IPv6
1. lsof -P -n -i
```

✔ strace

```
# Monitor a process while it is being executed
1. strace -p <pid>
```

✔ df

```
# I'm sure you know this one. Display mounted disks/partitions and sizes/usage in human-readable format
1. df -h
```

✔ free

```
# Display memory and swap usage in human-readable format
1. free -m
```

✔ ps

```
# Display processes and their parents
1. ps faux
```

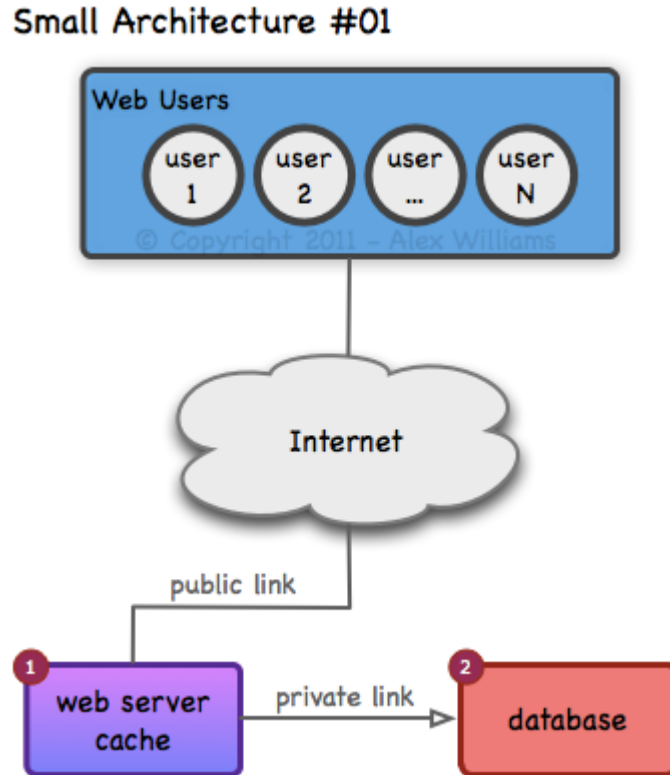


Information

Unfortunately it would require an entire book to list **every** profiling tool out there. It is your job to familiarize yourself with what's available to answer your needs.

3. Small Architecture #1

3.1. Diagram



3.2. Description






- 2 servers
 - 1 x web + cache
 - 1 x database

This architecture is designed to isolate database and web activity. The web server will handle all HTTP requests and cache data from the database. The database server will handle queries coming from the web server.

3.3. Advantage

💡 This setup allows for removing the database load from 1 server and placing it all on its own server.

3.4. Diagram Explanation

Object	Diagram	Description
Web users		People and devices on the internet
Public link		Internet-facing link, accessible by everyone
Private link		Local-facing link, accessible only by the Web and Database servers
Web server Cache		Hosts your web application, caching and web server software
Database server		Hosts your database software and data

3.5. Failure Scenarios

Warning



This architecture provides no failover and no load-balancing. There is no point planning for failure without a redundant configuration.

3.6. How-to

The following changes are required for scaling to this architecture:



Setup a 2nd server to host your database:

- [Database Server Setup](#)



Configure a local private IP address on both servers:

- [IP Address Setup](#)



Enable NAT or routing on the web server (optional)



Tip

It is best to avoid NAT or routing to your database server. This prevents it from accessing the internet, thus improving security. It can be enabled at a later date if you want to perform updates or setup automatic off-site backups.



Setup caching on the web server:

- [Installing Memcached + PHP support](#)
- [Memcached Configuration](#)



Remove the public IP address on the database server



Transfer your data to the database server:

- [Transfer Database](#)

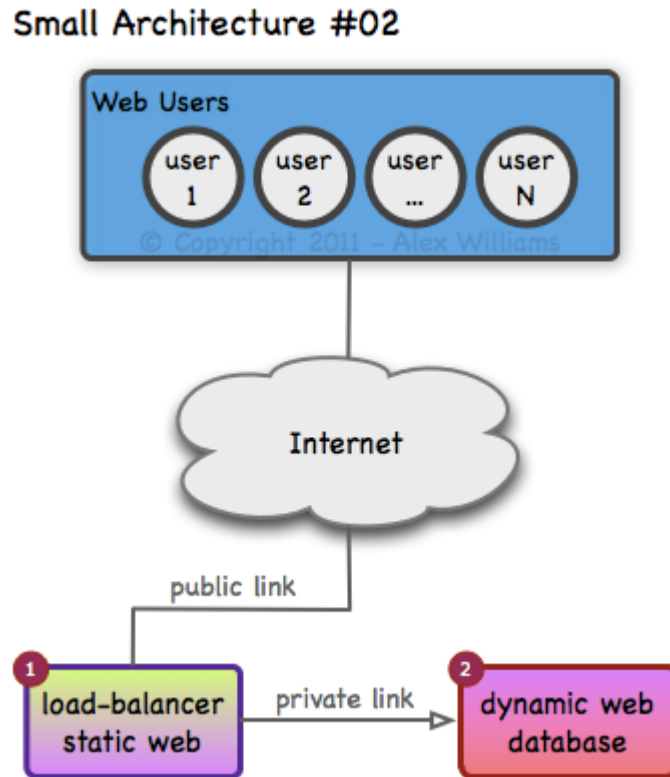


Modify your web application:

- [Use Memcached and MySQL](#)

4. Small Architecture #2

4.1. Diagram



4.2. Description




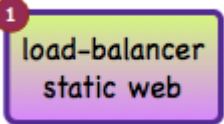
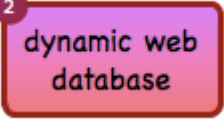
- 2 servers
 - 1 x load-balancer + static web
 - 1 x dynamic web + database

This architecture is designed to isolate static web objects from dynamic web objects. The static web server will serve files which rarely change, such as .jpg, .gif, and .html pages. It will forward all other requests to the dynamic web server. The dynamic web server will process requests for .php pages containing information pulled from the database.

4.3. Advantage

- 💡 This setup allows for quicker access to non-dynamic data, and lowers the requests processed by the dynamic web server.

4.4. Diagram Explanation

Object	Diagram	Description
Web users		People and devices on the internet
Public link		Internet-facing link, accessible by everyone
Private link		Local-facing link, accessible only by the Static and Dynamic Web servers
Load-balancer Static Web server		Hosts your static files, load-balancing software and static web server software
Dynamic Web server Database server		Hosts your dynamic files and data, database and dynamic web server software

4.5. Failure Scenarios

Warning



This architecture offers no failover and no load-balancing. There is no point planning for failure without a redundant configuration.

4.6. How-to

➕ Setup a 2nd server as a load-balancer and static web server:

- [Installing Nginx](#)

➕ Configure a public IP address on the static web server:

- [IP Address Setup](#)

➕ Configure a local private IP address on both servers:

- [IP Address Setup](#)

➖ Remove the public IP address from the dynamic web server

➕ Setup proxy forwarding on the static web server:

- [Nginx Configuration](#)

➕ Transfer your website to the static web server:

- [Transfer Website](#)

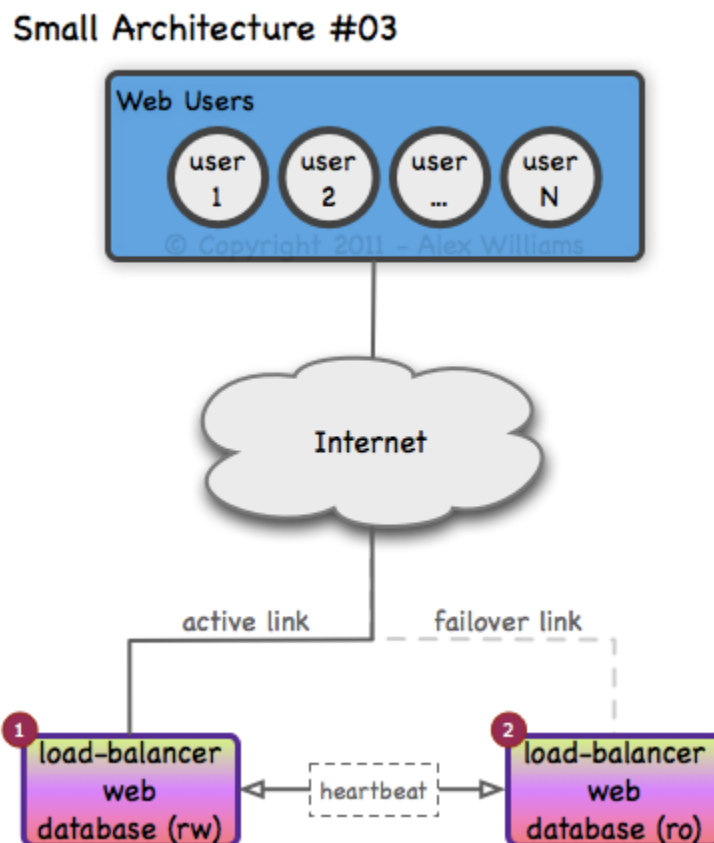


Tip

It is not necessary to delete the non-static files from the static web server. Nginx will only serve static content.

5. Small Architecture #3

5.1. Diagram



5.2. Description


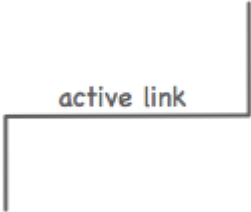


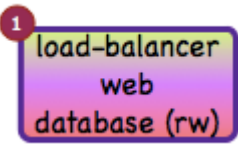
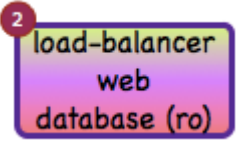
- 2 servers
 - 2 x load-balancer + web + database

This architecture is designed to provide 2 identical servers running concurrently. One server will be on the active link receiving all HTTP requests. Each server will load-balance requests to itself and its replica. Only one server will accept database write requests, but both will take reads.

5.3. Advantage

💡 This setup is to ensure if one of the servers crashes, the other one can takeover. Both servers can handle requests concurrently if both are online.

5.4. Diagram Explanation

Object	Diagram	Description
Web users		People and devices on the internet
Active link		Active internet-facing link, accessible by everyone
Failover link		Failover internet-facing link, accessible by everyone
Heartbeat		Local-facing link, accessible only by the 2 servers Shared/floating IPs (VIP) assigned to the <u>Master</u> server
Load-balancer Web server Database (read+write)		Hosts your web application and data, load-balancer, heartbeat monitor, web and database software
Load-balancer Web server Database (read-only)		Hosts your web application and data, load-balancer, heartbeat monitor, web and database software

5.5. Failure Scenarios

This architecture allows for the full failure of 1 server, or partial failure of both servers.

Detection of failures occurs through the heartbeat over a private link between the 2 servers.

Full server failure

- In the event of a full server failure, the active link moves to the functional server. This server will bind the public and private floating IP addresses, and provide all services to the web users.

Partial server failure: web

- If the website is placed down for maintenance (by modifying or removing the `/servercheck.txt` file, the load-balancer on the active link will remove it from the load-balanced pool of web servers.
- If the web server software on the active link becomes unavailable, the other server will bind the public floating IP address, become the active link, and provide load-balancing services of web requests to the web users. This will also occur if an error is detected on the public network interface.


Partial server failure: database

- If the database server software on the Master database server becomes unavailable, the other server will bind the private floating IP address and serve read+write database requests. This will also occur if an error is detected on the private network interface.


5.6. How-to

 Setup a 2nd server identical to your current server:


- [Database Server Setup](#)
- [MySQL Replication Configuration](#)

 Configure a local private and public IP addresses on both servers:

- [IP Address Setup](#)

 Setup the load-balancer:

- [Installing Keepalived](#)
- [Keepalived Configuration](#)

 Setup a server check file:


- [Servercheck Setup](#)

 Transfer your data to the 2nd server:

- [Transfer Database](#)
- [Transfer Website](#)

 Setup database replication:

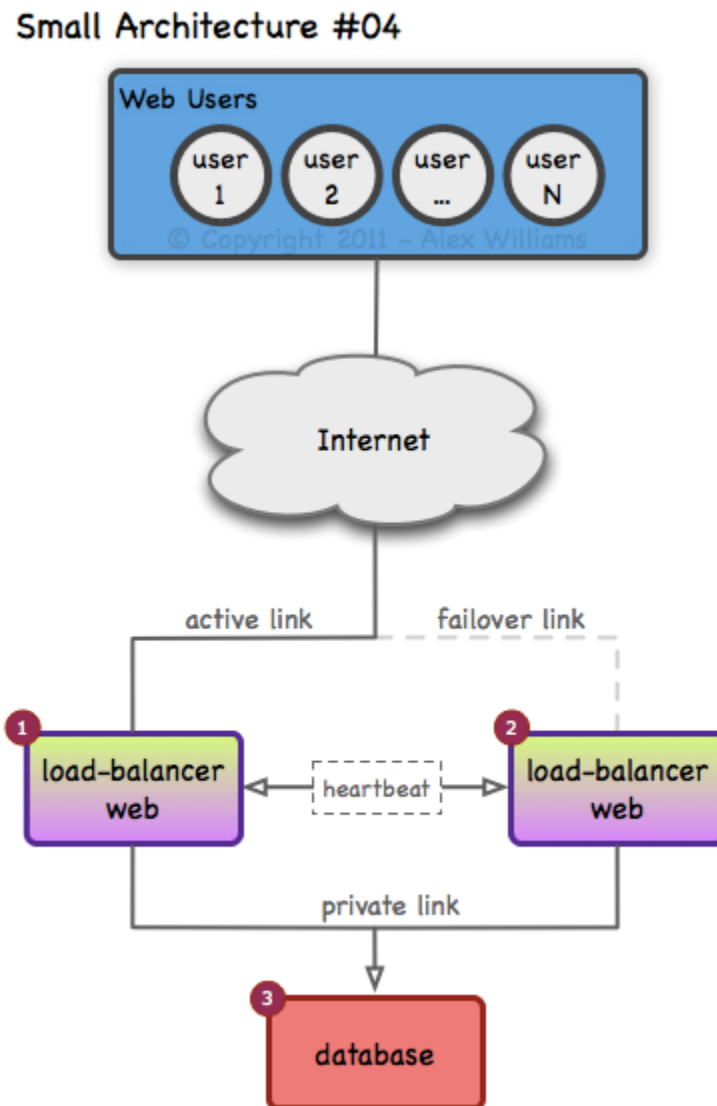
- [Replication Setup](#)

 Modify your web application:

- [Use the local database for Reads, and Master database for Writes](#)

6. Small Architecture #4

6.1. Diagram



6.2. Description


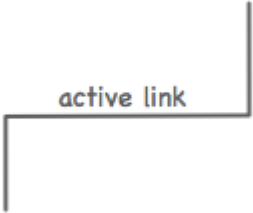


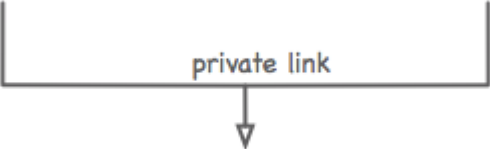
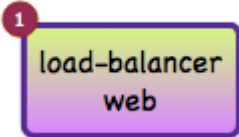
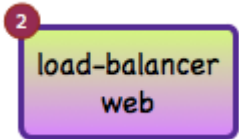
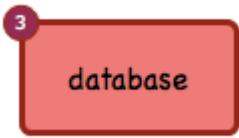
- 3 servers
 - 2 x load-balancer + web
 - 1 x database

This architecture is designed to isolate database and web activity while providing failover through 2 identical web servers. One server will be on the active link receiving all HTTP requests. Each server will load-balance requests to itself and its replica. The database server will accept database read and write requests.

6.3. Advantage

💡 This setup allows more concurrent web requests to be accepted.

6.4. Diagram Explanation

Object	Diagram	Description
Web users		People and devices on the internet
Active link		Active internet-facing link, accessible by everyone
Failover link		Failover internet-facing link, accessible by everyone
Heartbeat		Shared/floating IP (VIP) assigned to the active server
Private link		Local-facing link, accessible only by all 3 servers
Load-balancer Web server		Hosts the web application, load-balancer and web server software
Load-balancer Web server		Hosts the web application, load-balancer and web server software
Database (read+write)		Hosts the database server software and data

6.5. Failure Scenarios



Information

This architecture provides failover and load-balancing for the web servers, but not for the database

This architecture allows for the full failure of 1 web server, or partial failure of both web servers.

Detection of failures occurs through the heartbeat over a private link between the 2 web servers.



Full web server failure

- In the event of a full web server failure, the active link moves to the functional server. This server will bind the public floating IP address, and provide load-balancing and web services to the web users.



Partial server failure: web

- If the web server software, or the website is placed down for maintenance (by modifying or removing the `/servercheck.txt` file, the load-balancer on the active link will remove it from the load-balanced pool of web servers.
- If the load-balancing software on the active load-balancer becomes unavailable, the other server will bind the public floating IP address, become the active link, and provide load-balancing services of web requests to the web users. This will also occur if an error is detected on the public or private network interfaces.

6.6. How-to



Setup a 2nd server (web2) identical to your current server



Setup a 3rd server to host your database:

- [Database Server Setup](#)



Configure a local private and public IP addresses on both web servers:

- [IP Address Setup](#)



Configure a local private IP address on the database server:

- [IP Address Setup](#)



Configure the floating IP address on both web server's loopback adapter:

- [IP Address Setup](#)



Setup the load-balancer on both web servers:

- [Installing Keepalived](#)
- [Keepalived Configuration](#)
- [Installing HAProxy](#)
- [HAProxy Configuration](#)



Setup a server check file:

- [Servercheck Setup](#)



Transfer your data to the 2nd server:

- [Transfer Database](#)
- [Transfer Website](#)

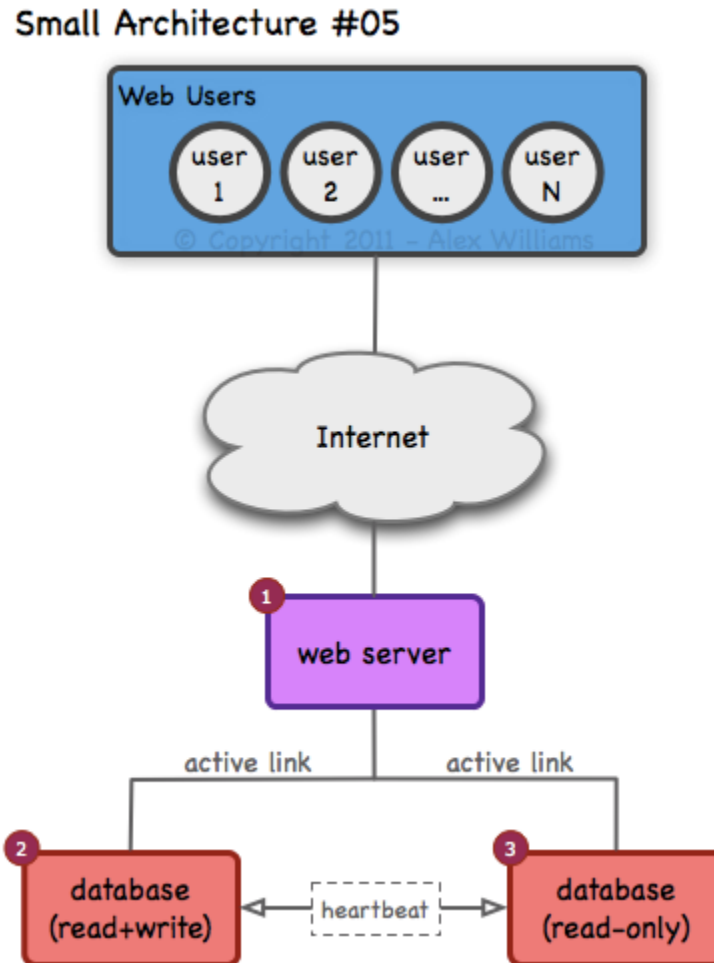


Modify your web application:

- [Use MySQL on an isolated server](#)

7. Small Architecture #5

7.1. Diagram



7.2. Description



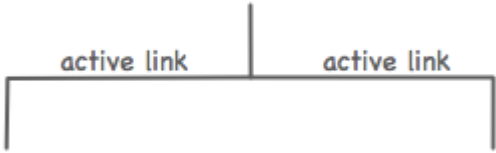
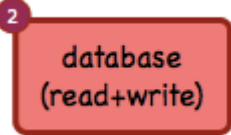
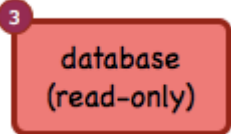
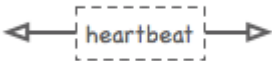
- 3 servers
 - 1 x web
 - 2 x database
 - 1 x read+write
 - 1 x read-only

This architecture is designed to isolate web and database activity while providing failover through 2 identical database servers. Both servers will be on an active link receiving read requests, while only one server will receive write requests.

7.3. Advantage

💡 This setup allows your website to process more database requests, and provides failover in the event one database server becomes unavailable.

7.4. Diagram Explanation

Object	Diagram	Description
Web users		People and devices on the internet
Web server		Hosts the web application and web server software
Active link		Local-facing link, accessible only by all 3 servers
Database (read+write)		Hosts the database server software and data
Database (read-only)		Hosts the database server software and data
Heartbeat		Shared/floating IP (VIP) assigned to the read+write database server

7.5. Failure Scenarios



Information

This architecture provides failover for the database servers, but not for the web application

This architecture allows for the full or partial failure of 1 database server.

Detection of failures occurs through the heartbeat over a private link between the 2 database servers.



Full or partial database server failure

- In the event of a database server failure, the functional server will bind the private floating IP address, and accept read+write requests from the web server. The web users should not be affected by this failure, as long as the database server is not over-capacity.



Tip

In any redundant configuration, it's a good idea to ensure the failure of 1 server doesn't create a snowball effect by overloading the 2nd server.

7.6. How-to



Setup 2 database servers with identical configurations:

- [Database Server Setup](#)
- [MySQL Replication Configuration](#)



Configure a local private IP address on both database servers:

- [IP Address Setup](#)



Configure a public IP address on the web server:

- [IP Address Setup](#)



Setup the heartbeat monitor on both database servers:

- [Installing Keepalived](#)
- [Keepalived Configuration](#)



Transfer your data to the database servers:

- [Transfer Database](#)

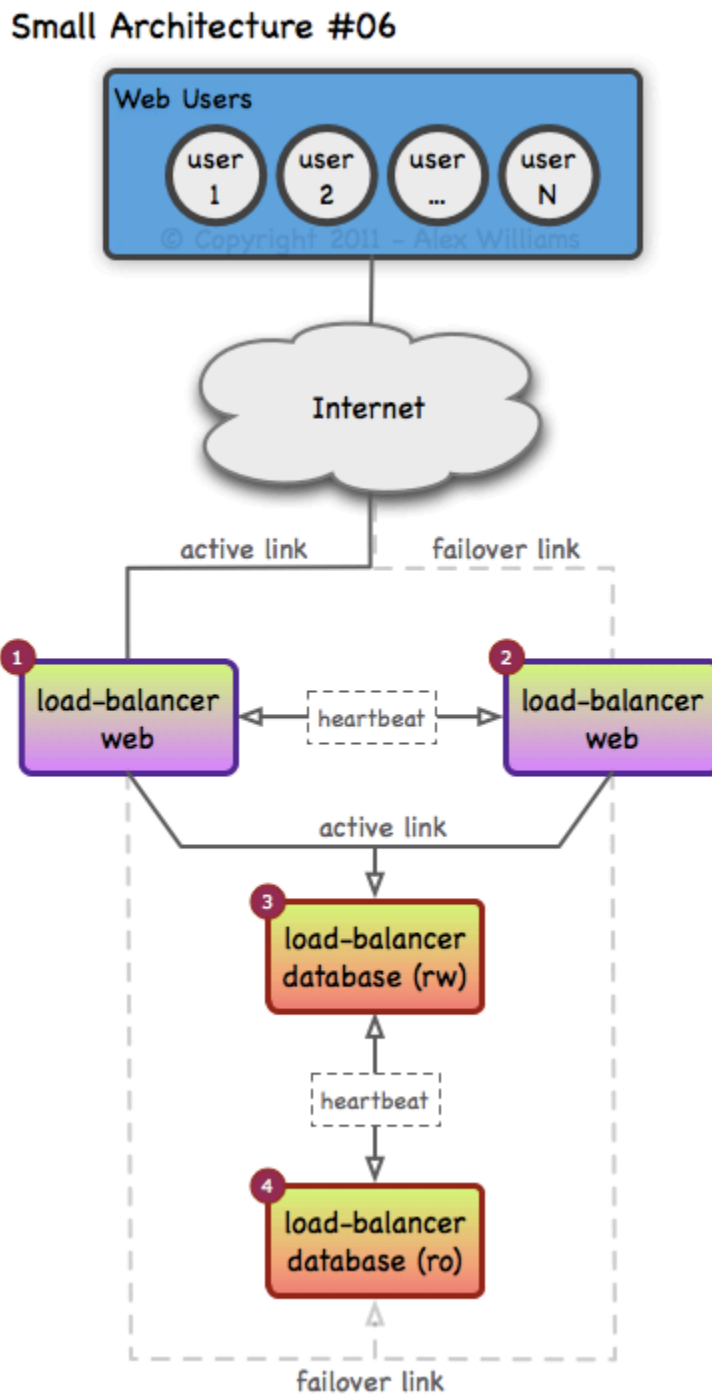


Modify your web application:

- [Use a random database server](#)

8. Small Architecture #6

8.1. Diagram



8.2. Description

- 4 servers
 - 2 x load-balancer + web
 - 2 x load-balancer + database


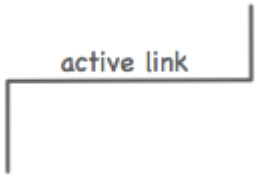


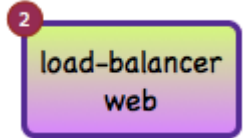
This architecture is designed to isolate database and web activity while providing failover through 2 identical web and database servers.

- One web server will be on an active link receiving all HTTP requests. Each web server will load-balance requests to itself and its replica.
- One database server will be on an active link receiving all DB requests. Each database server will load-balance read requests to itself and its replica, ensuring all writes only go to the active server.

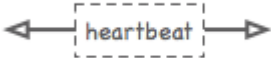
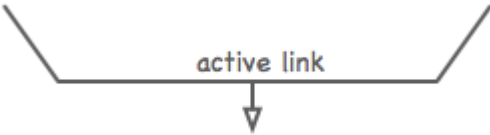
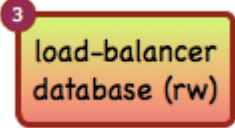
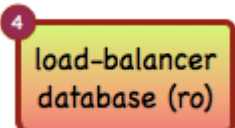

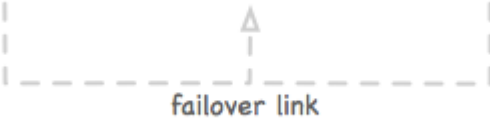
8.3. Advantage

💡 This setup allows more concurrent database requests AND web requests, and also provides failover.

8.4. Diagram Explanation (pt.2)

Object	Diagram	Description
Web users		People and devices on the internet
Active link #1		Active internet-facing link, accessible by everyone
Failover link #1		Failover internet-facing link, accessible by everyone
Load-balancer Web server		Hosts the web application, load-balancer and web server software
Load-balancer Web server		Hosts the web application, load-balancer and web server software

8.4. Diagram Explanation (pt.2)

Object	Diagram	Description
Heartbeat #1		Shared/floating public IP (VIP) assigned to the active server
Active link #2		Active local-facing link, accessible by the web and database servers
Load-balancer Database server (read+write)		Hosts the database server software and data
Load-balancer Database server (read-only)		Hosts the database server software and data
Heartbeat #2		Shared/floating private IPs (VIP) assigned to the active server
Failover link #2		Failover local-facing link, accessible by the web and database servers

8.5. Failure Scenarios

Load-balancer failure

- If the load-balancing software on the active load-balancer becomes unavailable, the other server will bind the public floating IP address, become the active link, and provide load-balancing services of web requests to the web users. This will also occur if an error is detected on the public or private network interfaces.

Full server failure: web

- In the event of a full web server failure, the active link moves to the functional server. This server will bind the public floating IP address, and provide load-balancing and web services to the web users.


Partial server failure: web

- If the web server software crashes, or the website is placed down for maintenance (by modifying or removing the `/servercheck.txt` file, the load-balancer on the active link will remove it from the load-balanced pool of web servers. The active link will remain the same.


Database server failure

- In the event of a database server failure, the functional server will bind the private floating IP addresses, and accept read+write requests from the web servers. The web users should not be affected by this failure, as long as the database server is not over-capacity.


8.6. How-to

 Setup 2 database servers with identical configurations:


- [Database Server Setup](#)
- [MySQL Replication Configuration](#)

 Configure a local private and public IP addresses:


- [IP Address Setup](#)

 Configure the floating IP address on both web server's loopback adapter:

- [IP Address Setup](#)

 Setup a server check file on both web servers:


- [Servercheck Setup](#)

 Setup the heartbeat monitor on all servers:


- [Installing Keepalived](#)
- [Keepalived Configuration on the web servers](#)
- [Keepalived Configuration on the database servers](#)

 Setup the load-balancer on the web servers:

- [Installing HAProxy](#)
- [HAProxy Configuration](#)

 Transfer your data to the web and database servers:

- [Transfer Database](#)
- [Transfer Website](#)

 Modify your web application:

- [Use separate database servers for reads and writes](#)

9. Software Installation




Information

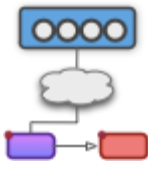
The commands and scripts below have ONLY been tested on Debian Lenny v5.0.

9.1. Installing Memcached with PHP support



Applies to:

 **Architecture #1**



see: [How-to #1](#)


```
# Install memcached
1. apt-get install memcached


# Add Memcached + PHP support
2. apt-get install php5-dev php5-memcache
```

9.2. Installing Nginx



Applies to:

 **Architecture #2**



see: [How-to #2](#)

```
1. apt-get install nginx
```

9.3. Installing Keepalived

✔ Applies to:

👍 Architecture #3	👍 Architecture #4	👍 Architecture #5	👍 Architecture #6
see: How-to #3	see: How-to #4	see: How-to #5	see: How-to #6

⚠ **Warning**
Debian Lenny 5.0 is distributed with an old and slightly dysfunctional version of Keepalived. It is recommended to use a more recent and manually compiled version obtained from the Keepalived website.

```
1. apt-get install keepalived
```

9.4. Installing HAProxy

✔ Applies to:

👍 Architecture #4	👍 Architecture #6
see: How-to #4	see: How-to #6







⚠ **Warning**
Debian Lenny 5.0 is distributed with an old and slightly dysfunctional version of HAProxy. It is recommended to use a more recent and manually compiled version obtained from the HAProxy website.

```
1. apt-get install haproxy
```

10. Scripts and commands

10.1. IP Address Setup

✔ Applies to:

👍 Architecture #1	👍 Architecture #2	👍 Architecture #3	👍 Architecture #4	👍 Architecture #5	👍 Architecture #6
					
see: How-to #1	see: How-to #2	see: How-to #3	see: How-to #4	see: How-to #5	see: How-to #6

- Public IP addresses will allow web users to communicate with load-balancers and web servers over the public link.
- Local private IP addresses will allow web and database servers to communicate over the private link. They will also prevent web users from accessing the database server directly over the internet.
- Loopback IP addresses will allow services to start without being accessible outside the loopback interface.

The examples below are provide with support for IPv4 or IPv6 configurations.

IPv4

```
# IPv4 public
1. ip addr add 172.16.0.11/16 dev eth0 # on web1
2. ip addr add 172.16.0.12/16 dev eth0 # on web2

# IPv4 local private
3. ip addr add 192.168.0.11/24 dev eth1 # on web1
4. ip addr add 192.168.0.12/24 dev eth1 # on web2
5. ip addr add 192.168.0.21/24 dev eth1 # on database1
6. ip addr add 192.168.0.22/24 dev eth1 # on database2

# IPv4 loopback
7. ip addr add 172.16.0.10/16 dev lo # on web1
8. ip addr add 172.16.0.10/16 dev lo # on web2
```

IPv6

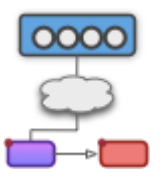
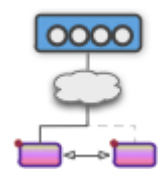
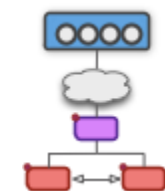
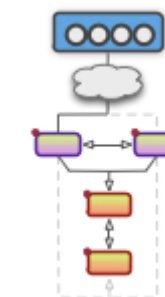
```
# IPv6 public
1. ip -6 addr add 2001:db8::ffff:11/64 dev eth0 # on web1
2. ip -6 addr add 2001:db8::ffff:12/64 dev eth0 # on web2

# IPv6 local private
3. ip -6 addr add 2001:db8::11/64 dev eth1 # on web1
4. ip -6 addr add 2001:db8::12/64 dev eth1 # on web2
5. ip -6 addr add 2001:db8::21/64 dev eth1 # on database1
6. ip -6 addr add 2001:db8::22/64 dev eth1 # on database2

# IPv6 loopback
7. ip -6 addr add 2001:db8::ffff:10/64 dev lo # on web1
8. ip -6 addr add 2001:db8::ffff:10/64 dev lo # on web2
```


10.2. Database Server Setup

✔ Applies to:

👍 Architecture #1	👍 Architecture #3	👍 Architecture #5	👍 Architecture #6
			
see: How-to #1	see: How-to #3	see: How-to #5	see: How-to #6

Information
Your new database server should be setup with a separate partition or drive for your data. This will allow you to take quick snapshots and backups of your data without placing the server offline

Before you proceed, you will need to:

- Measure the current disk space used by your database

```
1. du -hs /var/lib/mysql
```

- Ensure your database server has a separate partition or drive

```
1. sfdisk -l -s -uM # displayed in Bytes  
2. fdisk -l # displayed in MB or GB
```

- Make sure the drive or partition isn't already mounted/used

```
# In this example, we'll use /dev/sdb  
1. df -h /dev/sdb  
2. umount /dev/sdb
```



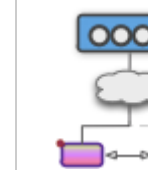

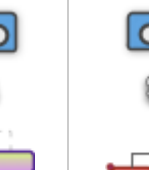

- Setup an LVM partition for your database

```
1. pvcreate /dev/sdb  
  
# Add /dev/sdb to the LVM volume group called 'data'  
2. vgcreate data /dev/sdb  
  
# Create an LVM volume using 90% of the disk space and add it to the 'data' volume group  
3. lvcreate -l 90%FREE --name mysql data  
  
# Format and mount your MySQL partition using ext3, add it to fstab  
4. mkfs.ext3 /dev/data/mysql  
5. echo "/dev/data/mysql /var/lib/mysql ext3 rw,noatime 0 1" >> /etc/fstab  
6. mount /dev/data/mysql
```

✔ **Tip**
Sufficient care should be taken to ensure this server has ample memory and disk space.

10.3. Transfer Database

✔ Applies to:

👍 Architecture #1	👍 Architecture #2	👍 Architecture #3	👍 Architecture #4	👍 Architecture #5	👍 Architecture #6
					
see: How-to #1	see: How-to #2	see: How-to #3	see: How-to #4	see: How-to #5	see: How-to #6



Information


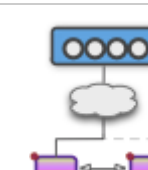
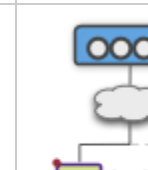

We recommend using SCP (Secure Copy over SSH) to transfer your data from the source server to the new database server.

Depending on the volume/activity of your server, it might be necessary to **SHUTDOWN MySQL** before transferring the data.

```
1. scp -r /var/lib/mysql/* root@192.168.0.21:/var/lib/mysql
```

10.4. Transfer Website

✔ Applies to:

👍 Architecture #2	👍 Architecture #3	👍 Architecture #4	👍 Architecture #6
			
see: How-to #2	see: How-to #3	see: How-to #4	see: How-to #6

Transfer the entire website to the other web server.

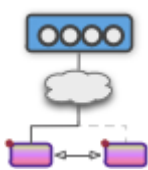
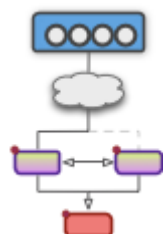
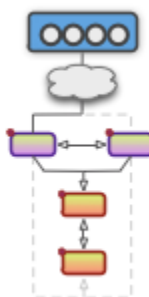
```
1. scp -r /var/www/site.com/* root@192.168.0.12:/var/www/site.com

# Login to the static web server
2. cd /var/www/site.com

# Delete all non-static files (this step is not necessary)
3. find . ! -iregex ".*\.(jpg|png|js|ico|gif|txt|css|swf|zip|pdf|avi|mp3|html)" -print0
| xargs -0 rm
```

10.5. Servercheck Setup

✔ Applies to:

👍 Architecture #3	👍 Architecture #4	👍 Architecture #6
		
see: How-to #3	see: How-to #4	see: How-to #6

- A file must be placed on each web server for the load-balancers to verify its availability.

```
1. echo "up" > /var/www/site.com/servercheck.txt
```

- An MD5 hash/digest of the file must be obtained and placed in the *Keepalived* configuration file.

```
1. genhash -s 127.0.0.1 -p 80 -u /servercheck.txt
```



Tip

If you want to perform maintenance on a web server, you can simply delete the `servercheck.txt` file. *Keepalived* will automatically remove the server from the load-balancing pool.

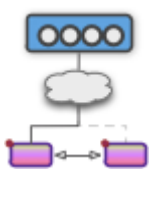
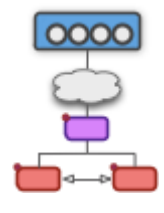
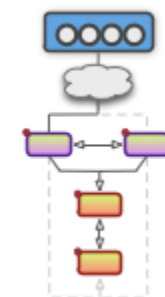


Warning

If you want to re-add the server to the load-balancing pool, don't forget to re-create the `servercheck.txt` file and verify its digest.

10.6. Replication Setup

✔ Applies to:

👍 Architecture #3	👍 Architecture #5	👍 Architecture #6
		
see: How-to #3	see: How-to #5	see: How-to #6

This will allow the web servers from making read+write requests to the Master database server, and read-only requests to one of the two database servers (or locally).

Before you proceed, you will need to:

- Create a replication user.

```
1. mysql> GRANT REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO 'replication'@'192.168.0.%'  
-> IDENTIFIED BY 'password';
```

- Start the replication
Change the commands on the database2 and database1 servers to point to each other.

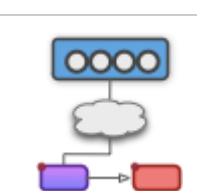
```
# on database2  
1. mysql> CHANGE MASTER TO MASTER_HOST='192.168.0.21', MASTER_USER='replication',  
-> MASTER_PASSWORD='password', MASTER_LOG_FILE='mysql-bin.000001', MASTER_LOG_POS=0;  
2. mysql> START SLAVE;  
  
# on database1  
3. mysql> CHANGE MASTER TO MASTER_HOST='192.168.0.22', MASTER_USER='replication',  
-> MASTER_PASSWORD='password', MASTER_LOG_FILE='mysql-bin.000001', MASTER_LOG_POS=0;  
4. mysql> START SLAVE;
```

11. Configuration Files

11.1. Memcached Configuration

✔ Applies to:

👍 Architecture #1



see: How-to #1

Caching will allow the web server to locally store results for database queries in memory, to provide quicker access to subsequent requests.



Warning


The cache should NOT be accessible over the public link. It should run directly on a socket or localhost.

```
# Sample configuration in '/etc/memcached.conf'  
-d  
logfile /var/log/memcached.log  
-v  
-m 64  
-p 11211  
-u nobody  
-l 127.0.0.1
```

11.2. Nginx Configuration

✔ Applies to:

👍 Architecture #1




see: How-to #1

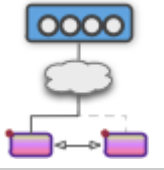
Proxy forwarding will allow the static web server to parse HTTP requests and forward them to the correct server (itself, or the dynamic web server).

```
# Sample configuration in '/etc/nginx/nginx.conf'
server {
    listen 172.16.0.11:80;
    server_name site.com www.site.com;
    location / {
        proxy_pass http://192.168.0.12:80;
        proxy_redirect off;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_connect_timeout 120;
        proxy_send_timeout 90;
        proxy_read_timeout 90;
        client_max_body_size 8m;
    }
    location ~* ^.+\. (jpg|png|js|ico|gif|txt|css|swf|zip|pdf|avi|mp3|html)$ {
        root /var/www/site.com;
    }
}
```

11.3.1. Configure Keepalived for Architecture #3 (pt.1)

✔ Applies to:

 **Architecture #3**



see: [How-to #3](#)

```
# Sample configuration in '/etc/keepalived/keepalived.conf'
global_defs {
    router_id WEB-DB
}
vrrp_script chk_mysql {
    script "killall -0 mysqld"
    interval 2
}
vrrp_script chk_apache {
    script "killall -0 apache2"
    interval 2
}
vrrp_instance VI_ETH0 {
    interface eth0
    virtual_router_id 01
    nopreempt
    priority 90
    advert_int 1
    state BACKUP

    virtual_ipaddress {
        172.16.0.10
    }
    track_script {
        chk_apache
    }
    track_interface {
        eth0
    }
}
vrrp_instance VI_ETH1 {
    interface eth1
    virtual_router_id 02
    nopreempt
    priority 90
    advert_int 1
    state BACKUP

    virtual_ipaddress {
        192.168.0.10
    }
    track_script {
        chk_mysql
    }
    track_interface {
        eth1
    }
}
# [TRIMMED] Sample configuration in '/etc/keepalived/keepalived.conf'
```

11.3.2. Configure Keepalived for Architecture #3 (pt.2)


The active load-balancer will bind the floating IPs, take HTTP requests and distribute them equally between the 2 web servers. A heartbeat monitor will watch for failed network interfaces and failed local services.

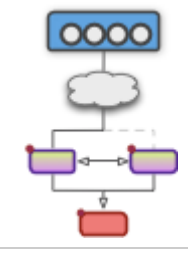
```
# [CONTINUED] Sample configuration in '/etc/keepalived/keepalived.conf'
virtual_server 172.16.0.10 80 {
    delay_loop 15
    lb_algo wrr
    lb_kind NAT
    persistence_timeout 0
    protocol TCP

    real_server 192.168.0.11 80 {
        weight 1
        HTTP_GET {
            url {
                path /servercheck.txt
                digest 67839e1bba5029447aa47d2e4b280b37
            }
            connect_timeout 5
            nb_get_retry 3
            delay_before_retry 5
        }
    }
}
real_server 192.168.0.12 80 {
    weight 1
    HTTP_GET {
        url {
            path /servercheck.txt
            digest 67839e1bba5029447aa47d2e4b280b37
        }
        connect_timeout 5
        nb_get_retry 3
        delay_before_retry 5
    }
}
}
```


11.4. Configure Keepalived for Architecture #4

✔ Applies to:

 **Architecture #4**



see: [How-to #4](#)

The active load-balancer will bind the public floating IP. A heartbeat monitor will watch for failed network interfaces and failed *HAProxy* service.

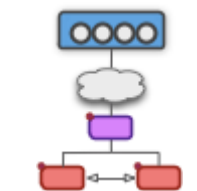
```
# Sample configuration in '/etc/keepalived/keepalived.conf'
global_defs {
    router_id WEB
}
vrrp_script chk_haproxy {
    script "killall -0 haproxy"
    interval 2
}
vrrp_instance VI_ETH0 {
    interface eth1
    virtual_router_id 01
    nopreempt
    priority 90
    advert_int 1
    state BACKUP

    # public floating IP
    virtual_ipaddress {
        172.16.0.10 dev eth0
    }
    track_script {
        chk_haproxy
    }
    track_interface {
        eth0
        eth1
    }
}
```

11.5. Configure Keepalived for Architecture #5

✔ Applies to:

👍 Architecture #5



see: [How-to #5](#)


A heartbeat monitor will watch for failed network interfaces and failed local services. It will also provide the ability to bind floating IP addresses.

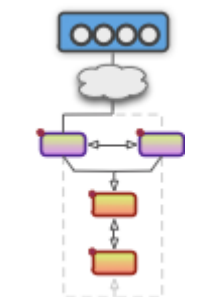
```
# Sample configuration in '/etc/keepalived/keepalived.conf'
global_defs {
    router_id DB
}
vrrp_script chk_mysql {
    script "killall -0 mysqld"
    interval 2
}
vrrp_instance VI_ETH0 {
    interface eth0
    virtual_router_id 01
    nopreempt
    priority 90
    advert_int 1
    state BACKUP

    # local private floating IP
    virtual_ipaddress {
        192.168.0.20 dev eth0
    }
    track_script {
        chk_mysql
    }
    track_interface {
        eth0
    }
}
```

11.6.1 Configure Keepalived on the web servers for Architecture #6

✔ Applies to:

 **Architecture #6**



see: [How-to #6](#)

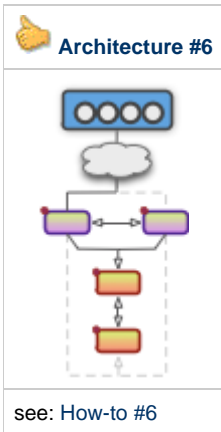
Configure *Keepalived* to assign a floating public IP address.

```
# Sample configuration in '/etc/keepalived/keepalived.conf'
global_defs {
    router_id WEB
}
vrrp_script chk_haproxy {
    script "killall -0 haproxy"
    interval 2
}
vrrp_instance VI_ETH0 {
    interface eth0
    virtual_router_id 01
    nopreempt
    priority 90
    advert_int 1
    state BACKUP

    # public floating IP
    virtual_ipaddress {
        172.16.0.10 dev eth0
    }
    track_script {
        chk_haproxy
    }
    track_interface {
        eth0
        eth1
    }
}
```

11.6.2 Configure Keepalived on the database servers for Architecture #6

✔ Applies to:



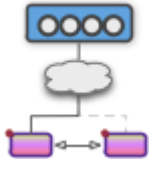
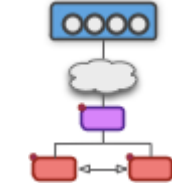
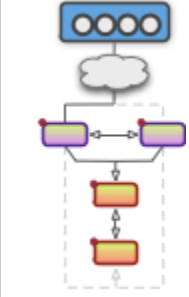
```
# Sample configuration in '/etc/keepalived/keepalived.conf'
global_defs {
    router_id DB
}
vrrp_script chk_mysqlqd {
    script "killall -0 mysqld"
    interval 2
}
vrrp_instance VI_ETH0 {
    interface eth0
    virtual_router_id 02
    nopreempt
    priority 90
    advert_int 1
    state BACKUP

    virtual_ipaddress {
        192.168.0.30 dev eth0 # database reads
        192.168.0.31 dev eth0 # database writes
    }
    track_script {
        chk_mysqlqd
    }
}
virtual_server 192.168.0.30 3306 {
    delay_loop 15
    lb_algo wrr
    lb_kind NAT
    persistence_timeout 0
    protocol TCP

    real_server 192.168.0.21 3306 {
        TCP_CHECK {
            connect_timeout 5
        }
    }
    real_server 192.168.0.22 3306 {
        TCP_CHECK {
            connect_timeout 5
        }
    }
}
```

11.7. MySQL Replication Configuration

✔ Applies to:

👍 Architecture #3	👍 Architecture #5	👍 Architecture #6
		
see: How-to #3	see: How-to #5	see: How-to #6

Database replication will allow the 2 servers to maintain identical copies of the data. Configure *MySQL* with the correct replication log/index file names, server ID and bind address.



Warning

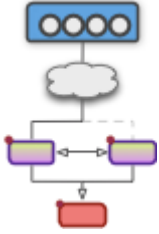
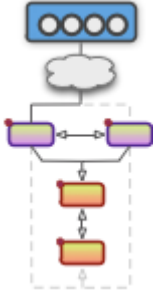
Change the IP addresses to match the IPs of your servers

```
# Server 1 - Sample configuration in '/etc/mysql/my.cnf'
[mysqld]
datadir           = /var/lib/mysql
log_bin           = mysql-bin
log_bin_index     = mysql-bin.index
relay_log         = mysql-relay-bin
relay_log_index   = mysql-relay-bin.index
server-id         = 1
bind-address      = 192.168.0.21

# Server 2 - Sample configuration in 'my.cnf'
[mysqld]
datadir           = /var/lib/mysql
log_bin           = mysql-bin
log_bin_index     = mysql-bin.index
relay_log         = mysql-relay-bin
relay_log_index   = mysql-relay-bin.index
server-id         = 2
bind-address      = 192.168.0.22
```

11.8. Configure HAProxy

✔ Applies to:

👍 Architecture #4	👍 Architecture #6
	
see: How-to #4	see: How-to #6

The active load-balancer will take HTTP requests and distribute them equally between the 2 web servers.

```
# Sample configuration in '/etc/haproxy/haproxy.cfg'
frontend web
  bind 172.16.0.10:80
  mode http
  default_backend site.com

backend site.com
  mode http
  option httpchk GET /servercheck.txt
  server web1 192.168.0.11:80 weight 1 check port 80 inter 2s rise 2 fall 5
  server web2 192.168.0.12:80 weight 1 check port 80 inter 2s rise 2 fall 5
```

12. Code samples




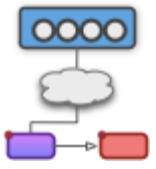
Information

The code below has only been tested in PHP 5.3 with Apache 2.0

12.1. Use Memcached and MySQL



Applies to:

 Architecture #1

see: How-to #1

Modify your web application to get/store cache data in *Memcached* and use the *MySQL* database on the database server.

```
# PHP Example
<?php
function get_data($id) {
    $cache = memcache_connect('127.0.0.1', 11211); // connect to Memcached locally

    $result = memcache_get($cache, "key_" . $id); // try to get the data from the cache


    if (!$result) {
        // $db = mysql_connect("localhost", "user", "password");
        $db = mysql_connect("192.168.0.2", "user", "password"); // connect to MySQL on the database1
server

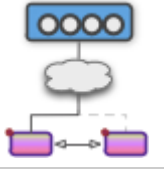
        mysql_select_db("test", $db);
        $result = mysql_query("SELECT * FROM " . $id);

        memcache_set($cache, "key_" . $id, $result, 0, 3600); // add to the cache for 1 hour
    }
    return $result;
}
$data = get_data('users');
?>
```

12.2. Use the local database for Reads, and Master database for Writes

✔ Applies to:

 **Architecture #3**




see: [How-to #3](#)

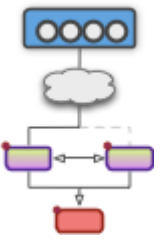
Modify your web application to use the local private floating IP for *MySQL* database write requests, and localhost for read requests.

```
# PHP Example
<?php
$db_write = mysql_connect("192.168.0.10", "user", "password"); // floating IP
$db_read = mysql_connect("127.0.0.1", "user", "password"); // local IP
if (!$db_read)
    $db_read = mysql_connect("192.168.0.10", "user", "password"); // local database is unavailable
?>
```

12.3. Use MySQL on an isolated server

✔ Applies to:

 **Architecture #4**




see: [How-to #4](#)

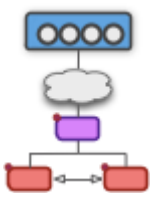
Modify your web application to use the database on the database server.

```
# PHP Example
<?php
// $db = mysql_connect("localhost", "user", "password");
$db = mysql_connect("192.168.0.21", "user", "password");
?>
```


12.4. Use a random database server

✔ Applies to:

 **Architecture #5**



see: [How-to #5](#)

The web application should get data from a random database server, and write data to the Master database server.


```
# PHP Example
<?php
// $db = mysql_connect("localhost", "user", "password");
$db_write = mysql_connect("192.168.0.20", "user", "password"); // floating IP
$db_read_array = array('192.168.0.21', '192.168.0.22');

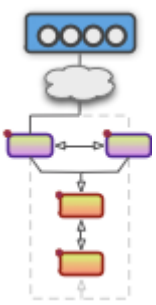
shuffle($db_read_array); // randomize the databases

$db_read = mysql_connect($db_read_array[0], "user", "password"); // use the 1st database server
if (!$db_read)
    $db_read = mysql_connect($db_read_array[1], "user", "password"); // use the 2nd database server
?>
```

12.5. Use separate database servers for reads and writes

✔ Applies to:

 **Architecture #6**



see: [How-to #6](#)

Modify your web application to use the floating local private IP addresses for read and write requests.

```
# PHP Example
<?php
// $db = mysql_connect("localhost", "user", "password");
$db_read = mysql_connect("192.168.0.30", "user", "password");
$db_write = mysql_connect("192.168.0.31", "user", "password");
?>
```

Conclusion

We hope this eBook has been valuable to you. If you've been able to scale your architecture successfully thanks to us, or simply fix some important server problems, then we've done our job.

When you're ready to scale beyond the scope of this eBook, please visit our website <http://www.ScalingExperts.com> as we're soon going to publish 2 more eBooks for even larger, more elaborate, and automated architectures, along with virtualization examples, scripts and much more.

The author, Alex Williams, is also available for consulting on your scaling projects.

Links



Information

The following links may not be available anymore.

Open Source Software

✔ HAProxy haproxy.1wt.eu

✔ Keepalived keepalived.org

✔ Nginx nginx.org

✔ Munin munin-monitoring.org

✔ MySQL mysql.com

✔ Apache httpd.apache.org

✔ PHP php.net

✔ Memcached memcached.org

✔ Redis redis.io

- Although we haven't provided any examples using *Redis* in this eBook, we love *Redis* and our biased views have prompted us to include a link for Salvatore.

Web sites and blogs

✔ GitHub github.com

✔ High Scalability highscalability.com

✔ All Things Distributed (Werner Vogels) allthingsdistributed.com

✔ Perspectives (James Hamilton) perspectives.mvdirona.com

✔ Google Publications research.google.com

✔ Free Software Foundation fsf.org

✔ Atlassian atlassian.com