ARCHITECTING HIGH AVAILABILITY LINUX ENVIRONMENTS WITHIN THE RACKSPACE CLOUD

A detailed exploration into the technical requirements and business implications of Cloud High Availability

By: The Rackspace Cloud Engineering Team

rackspace® HOSTING

ARCHITECTING HIGH AVAILABILITY CLOUD ENVIRONMENTS

I. SUMMARY

Over the past eighteen months, cloud computing has made significant strides in transforming itself from a niche technology to a key element and quite often an extension of enterprise production environments. As Infrastructure as a Service (IaaS) is much more than virtualization, corporations have begun moving past a test/development scenarios role and have started replacing traditional dedicated resources, forcing IT decision-makers into a whole new set of considerations.

This technical whitepaper assists IT managers and their teams with the concepts needed for planning, designing, and deploying best-practice High Availability IaaS environments with Rackspace Linux Cloud Servers. While most of the information presented here is Rackspace specific, much of the theory and discussion can apply for all virtualized IaaS solutions.

II. WHAT IS HIGH AVAILABILITY (HA)?

While Wikipedia defines **High Availability** as "A system design approach and associated service implementation that ensures a prearranged level of operational performance will be met during a contractual measurement period," we will focus this paper on cloud configurations that remove as many single points of failure as possible and that are inherently designed with a specific focus on operational continuity, redundancy, and fail-over capability.

High Availability can be achieved at many different levels including the application level, infrastructure level, datacenter level, and geographic redundancy level. We will focus on the infrastructure level in this white paper.

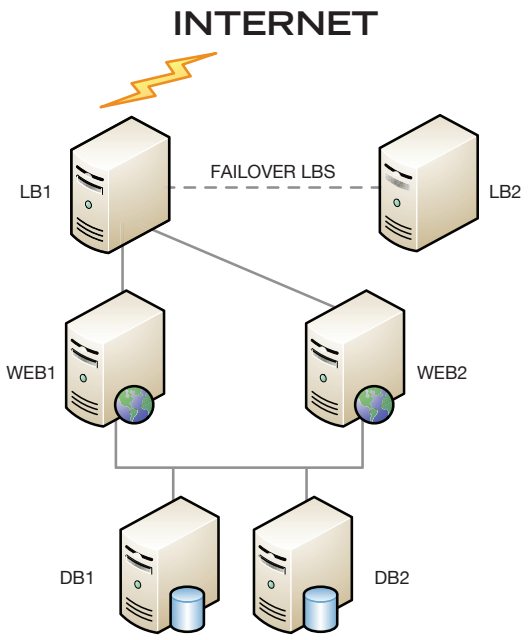
In its most basic form, infrastructure-level HA configurations consist of:

- Two or more Load Balancers
- Two or more Web Servers
- Two or more Database Servers

In this example, active/passive load balancers, multiple web nodes, and replicated DB servers all provide for redundancy at each layer of the configuration.

Of course, advanced HA configurations, including those designed for geographic redundancy across multiple data centers, can become quite a bit more complex than the basic HA example pictured.

EXAMPLE: SIMPLE HA CONFIG



III. DOES YOUR BUSINESS NEED HA?

As our digital dependence increases in our daily lives, downtime in the applications we use most creates a significant amount of angst for users. When Facebook, Gmail, or AT&T experience outages, these events receive national and often international attention.

Not only can downtime in your products turn into a disastrous PR nightmare, but more importantly, it can also seriously tarnish the loyalty of your customer base that depends on you for their financial livelihood. IDC, a leading IT market research firm, estimates that in 2007, "server downtime cost organizations approximately $140 billion in lost revenue and reduced worker productivity."1

"IDC estimates that in 2007, server downtime cost organizations approximately $140 billion in lost revenue and reduced worker productivity."1

Regardless of the size of your organization, if downtime in your internal infrastructure or core product offerings negatively impacts your bottom line, you are a perfect candidate for exploring cloud HA.

High Availability can still be a tricky and expensive proposition in dedicated environments. Fortunately, cloud computing brings HA within the reach of most small- and medium-sized businesses.

IV. RACKSPACE CLOUD HA VS. DEDICATED HA

Three significant advantages make cloud HA a better choice than Dedicated HA. Let's break each of them down.

1.) Scaling

If your dedicated environment experiences a sudden burst in traffic, do you have the overhead necessary to stay online? What is your scaling strategy? Do you have a method to predict what the anticipated traffic might be during these spikes? Even with a best practices HA config in place on dedicated gear, scaling can be a time-consuming, difficult and costly proposition. Luckily, the cloud mediates a fair amount of these issues. Cloud Servers offer two types of scaling:

a.) Scaling Up (out of the box – for the novice)

Cloud Servers can easily be resized up with a few clicks in a control panel and 5 to 10 minutes of time. Cloud Servers can be provisioned starting at 256MB of RAM (Linux) scaling all the way to 15GB of RAM. These sizing adjustments require brief human intervention using the cloud control panel or can be programmatically executed via the **Cloud Servers API.**

b.) Near-Real-Time Scaling (with the API/3rd party apps – for the expert)

With a cloud HA config, you can leverage dynamic scaling tools such as enStratus to provide near-real-time auto-scaling. enStratus can be set to monitor specific thresholds in your environment including load averages, disk space utilization, etc. It can take a prebuilt web server template image, automatically provision it, and re-configure your load balancer to start serving from the additional web node on the fly. Savvy developers can also use the Rackspace **Cloud Servers API,** to develop their own automated scaling solution.

2.) Management & Tools

Cloud Server management, especially when many nodes are in an environment, is far easier than managing a dedicated environment with multiple servers. Whether using the portal or the API, Cloud Servers can very easily be copied, re-imaged, launched in rescue mode for repair, accessed via a virtual console, and more. Furthermore, an interface like **Cloudkick** takes management a step further and allows for easy Cloud Server resource administration across many providers. This brings all of your virtualized instances, whether from Rackspace or an alternate provider, into a central management console.

V. CAPACITY PLANNING

Even with the inherent flexibility of Cloud Server scaling, there isn't a finite formula for capacity planning for any specific cloud HA configuration, as project requirements are always unique. As with all such calculation endeavors, especially yet-to-be launched projects, a significant amount of estimation based on prior experience is involved in sizing the environment properly. Luckily, with the ease of cloud scaling, estimate miscalculations can easily be rectified. This can prevent customers from over-paying for unutilized resources or the long build times of provisioning dedicated equipment.

A common sense practice, especially with the efficient cloud hourly billing model, suggests building 30-40% past maximum estimated requirements when starting out. It's always easier to scale down unnecessary resources later than to start with insufficient capacity on day one.

To get started sizing your configuration, begin with compiling any information you have available on existing or expected bandwidth utilization, concurrent process and visitor counts, application weight, etc. Some sample data types are presented below.

1. Peak Throughput Analysis: Finding the bandwidth required during peak times is crucial to your calculations. There are three ways to achieve this, in order of accuracy:

a.) **Most accurate:**
If you have access to existing **MRTG** graphs or similar throughput graphing metrics, take note of your maximum throughput in mbits/sec across your firewall (FW) (or all of your web nodes combined) at peak usage times.

b.) **Somewhat accurate:**
If the above isn't available, take the highest usage bandwidth (BW) day over the last few months in megabytes (MB) or gigabytes (GB) and calculate it in mbit/sec. We will take this amount and divide by 8 hours in a day to weigh it towards peak usage. It's important to keep in mind this is a guideline and not a finite calculation.
Example: 5 GB/day: 5GB * 1024 = 5120MB/day / 8 hours = 640 MB/hr / 60 minutes = 10.66 MB/min / 60 seconds = .18 MB/sec * 8 = 1.42 mbit/seconds

c.) Least accurate:

If you only have monthly traffic stats available to you, a general idea can still be reached.
Example: 2TB/mo:
2TB * 1024 = 2048 GB * 1024 = 2,097,152 MB/mo
2,097,152 MB/mo / 31 days = 67,650 MB/day/24 hours = 2818 MB/hr/60 minutes = 47MB/min/ 60 seconds = .78MB/ seconds .78MB/sec * 8 = 6.24 mbit /sec

2. Concurrent Connection Capacity Analysis: Two items should be collected:

- Number of maximum desired concurrent users: This is usually a simple estimate, gleaned from developers or system architects on your team.
- Weight of average user visit: On average, how many pages or resources does a

  user access with each visit? What is the weight of these resources in MB?
  Example: 500 concurrent connections @ .75 MB /visit.
  50 * .75 = 37.5 MB/sec = 300 mbit / sec of throughput needed

3. Application Analysis: Analyzing your application weight is important for estimating how much RAM you may need on your web servers. There are three key metrics which should be noted:

Page Weight = Request weight * requests per page / Sec = Total RAM of all Web Nodes / Request WeightVisits / Sec = Total Web RAM / Page Weight

It's important to stress again that the above analysis only results in a very loose estimate for picking a general starting point for your configuration. With this information, you can determine cost per page and, via web page metrics, discover which pages are costing you the most.

In the next section, we will take these inputs and build a best-estimate architecture. Once the architecture is constructed, and the application is deployed in test mode, we will run a set of rigorous performance benchmarking utilities to validate our calculations and see how accurate our estimates turned out.

VI. ARCHITECTING THE SOLUTION

Once capacity planning is completed, we need to start designing our cloud HA configuration at the load balancing layer and work through the web and database layers. For load balancing, we will use two Cloud Servers, each of which will act as a load balancer, utilizing open-source utilities such as HAProxy, nginx, etc. These servers will be designed to poll each other for instant fail-over utilizing **Heartbeat**, and **shared IPs**. If a node or host hypervisor fails, the second load balancer will take over in real time.

You can read more about this setup by referencing the following docs in the **Rackspace Cloud Server Wiki:**

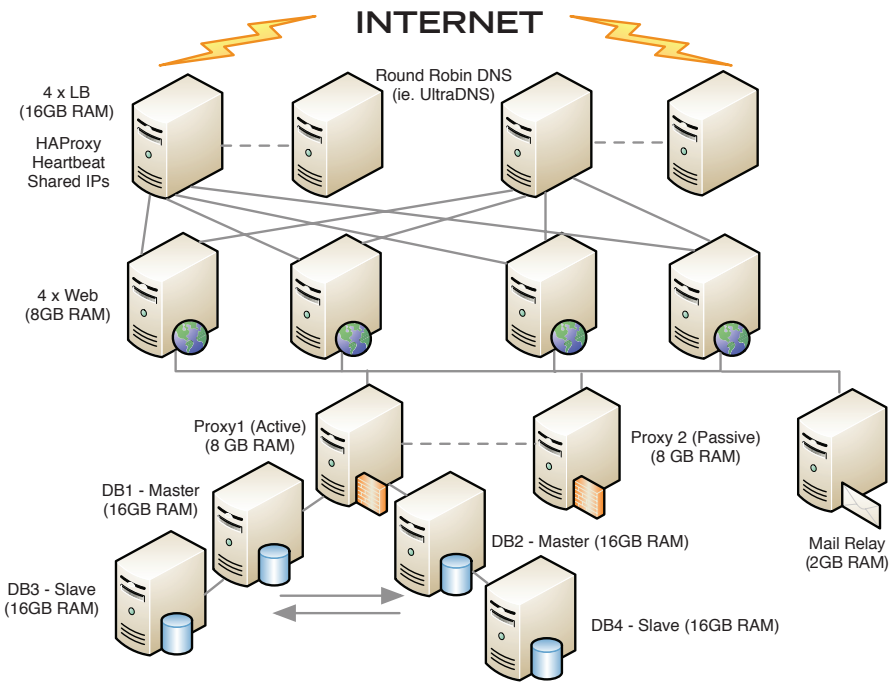
IP_Failover_-_High_Availability_Explained
IP_Failover_-_Setup_and_Installing_Heartbeat

Load Balancing Layer Design

1.) Decide if SSL termination is needed at the LB layer. If yes, use nginx. If no, or if SSL pass-through is sufficient, use HAProxy.
2.) Request an additional IP as a "floating IP" from support via the customer support ticketing system. This will function as the VIP (virtual IP/movable IP), which you will point DNS to for your sites.
3.) Select Cloud Servers based on the outbound mbit/sec requirements calculated above at peak times for the public interface. Reference this chart for the proper sizing:

Cloud Server Size	Network Throughput (public/private)
256MB RAM / 10GB HD	10 Mbps / 20 Mbps
512MB RAM / 20GB HD	20 Mbps / 40 Mbps
1024MB RAM / 40GB HD	30 Mbps / 60 Mbps
2048MB RAM / 80GB HD	40 Mbps / 80 Mbps
4096MB RAM / 120GB HD	50 Mbps / 100 Mbps
8192MB RAM / 320GB HD	60 Mbps / 120 Mbps
15872MB RAM / 620GB HD	70 Mbps / 140 Mbps

4.) If your LB throughput needs are higher than 70mbit/sec, you will need to consider multiple load balancer pairs. To achieve this, a round-robin DNS solution, such as **UltraDNS** would need to be used.



EXAMPLE: ROUND ROBIN DNS HA CONFIG

Web Layer Design

Designing the Cloud Server web layer is very straightforward as Cloud Servers mimic dedicated gear from many administration perspectives. Whether you are utilizing **Apache**, **Tomcat**, **lighttpd**, or some other web server, designing this layer will be familiar territory for an intermediate-to-advanced administrator. Remember to keep in mind that a scripting scheme will need to be developed to replicate your content to each web node when the codebase is updated. Either **Git** or **rsync** via a recurring **cron** job are both good options here.

Data Base Layer Design

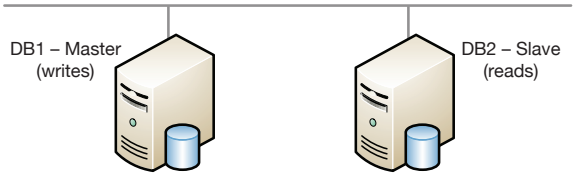
Cloud Servers provide for a flexible and capable DB environment. Let's explore a few different possible configurations and cover some pitfalls and important design notes.

Master/Slave:

It's not a true HA environment in every sense, since fail-over requires some manual intervention. Still, master-slave replication still provides two key advantages:

- **Performance:** By executing DB inserts (writes) on the Master and DB selects (reads) to the slave, the general DB load is spread and can yield additional performance for busy R/W sites. Another benefit is that the data availability is maintained, with a reduced RPO (recovery point objective), should the master server suffer a data loss failure.

- **Backups:** By performing backups on the slave, your site is not affected during the backup process. With a single server and a large database, backups could potentially cause a brief slowdown in your sites from **mysqldump** -induced lag as tables are locked. Instead, you can just take the slave(s) out of rotation, run the backups off the slave, then start the slave again for a more seamless backup experience.

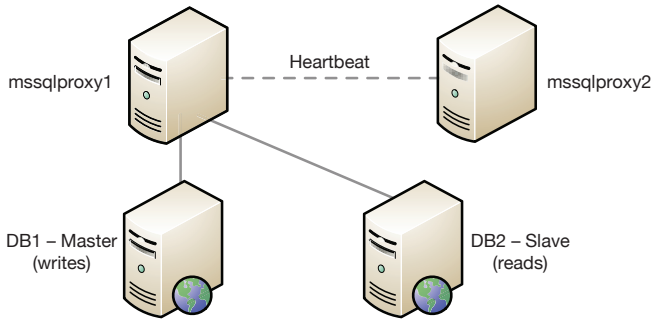


One downfall about Master/Slave is that you usually need to ensure your application is designed to split reads and writes. A bit of re-coding may be needed if this isn't the case.
Horizontally scaling your Master/Slave configuration with additional slaves can also be accomplished by randomizing read sources in your application code or by placing an HAProxy instance in front of the slaves to balance the requests.

Master/Slave with MySQL Proxy:

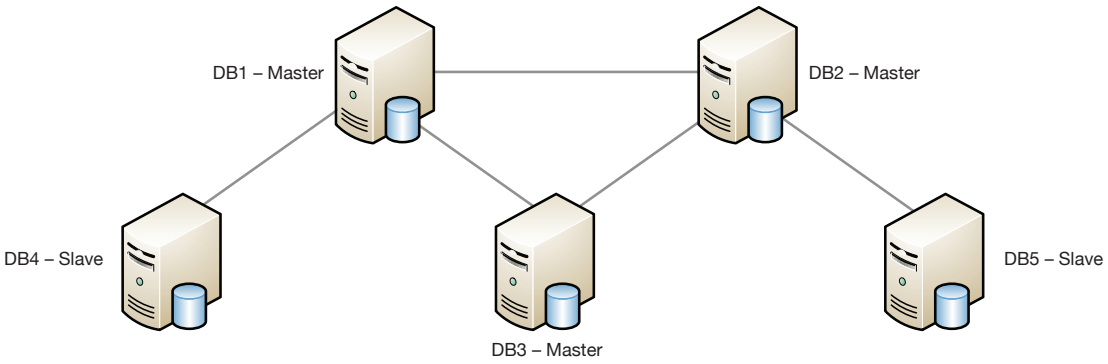
The great benefit behind implementing MySQL Proxy is the ability to incorporate read/write transaction splitting of the master/slave solution described above, but without the need to re-write your application. However, a single MySQL Proxy instance is also now a single point of failure and does not meet the requirements of HA. To solve this, we can deploy a DB structure with a failover MySQL Proxy utilizing heartbeat as the second diagram shows below.

A potential challenge here is the need to leverage a floating IP, much as in the HA LB layer example above. However, this IP needs to be private to avoid bandwidth charges and currently the Cloud Servers product doesn't offer private floating IPs. Workarounds can be achieved by creating custom routes and a custom subnet for the cloud servers to use.



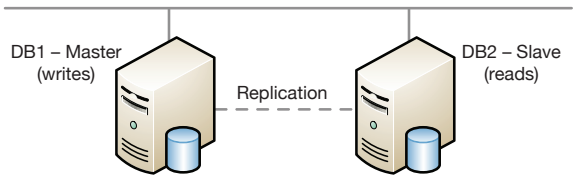
tMulti Master Mode:

This configuration is something that generally isn't advisable in a Cloud Server environment due to the potential of the DB nodes failing to stay synchronized. When this happens, replication breaks, and resolving it can cause application downtime. Consider this solution only if there is a very specific need for it and understand all of the risks and implications.



Master-Master:

This configuration is something that generally isn't advisable in a Cloud Server environment due to the potential of the DB nodes failing to stay synchronized. When this happens, replication breaks, and resolving it can cause application downtime. Consider this solution only if there is a very specific need for it and understand all of the risks and implications.



These are just some of the many database architecture choices that can be made when designing an HA configuration. High-traffic and database intensive sites will benefit the most from careful choices in this area.

VII. SECURING AN HA ENVIRONMENT

Securing an HA cloud environment shouldn't be a daunting task. By using standard server hardening best practices, customers can ensure they have a set of protected machines. Common practices such as closing off unused ports, running services on alternate ports, disabling root ssh access, keeping your systems patched, etc., are all just as important on Cloud Servers as they are on Dedicated Servers.

Linux Cloud Servers come with two network interfaces by default: a public (eth0) which connects the Cloud Server to the outside world and a private (eth1) which runs on a backend network referred to as ServiceNet. ServiceNet is intended for Cloud Server to Cloud Server communication without any BW fees. Stringent iptables rules on both of these interfaces should be a primary focus of a security plan. Great documentation to assist you in configuring iptables for Cloud Servers can be found in our Knowledge Base **here.**

Another security configuration option to consider is using your Load Balancing instances as Firewalls/ Gateways and disabling any public access (eth0 interfaces) to your web and DB nodes. These "backend" servers will then communicate with the LBs via the private Rackspace (eth1) ServiceNet network, removing another possible attack vector from intruders.

Finally, while Cloud Servers can be configured to be nearly every bit as secure as a Dedicated Server, many regulatory compliance standards haven't yet been re-written to include cloud technology. Even though Cloud Servers are easily secured and hardened, customers seeking HIPAA, PCI, and other regulatory compliancy need to understand that the cloud environments in general do not yet meet these compliance standards. It can also be said that some of these standards (that forbid multi-tenancy for example) will need to be updated, and we expect they will be over time, and Rackspace participates in standards committees to such end. Suffice it to say that you should be highly familiar with all compliance requirements prior to utilizing the cloud.

VIII. TESTING AND GOING LIVE

After your configuration is built, testing is the next crucial step in determining if real-world results are in line with the capacity estimates calculated in the design phase.

Stress testing can be performed either with open source tools such as **ApacheBench**, **JMeter**, **http_load**, and **Sieve**, or with advanced services such as **SOASTA** and **LoadStorm**. While the free tools can absolutely yield some helpful comparative test results between your current environment and your cloud environment, these tools don't always tell the whole story. When budgets are able, advanced services such as SOASTA and their team of experts can help simulate geographically dispersed traffic with various test cases and give the best estimation of what you can expect.

> **Tip:** Deploying monitoring and performance logging tools (such as **SAR**, **Munin**, and mql-slow-query-logging) prior to stress testing can be instrumental in helping to show areas in the environment which may be candidates for optimization.

A significant portion of your test plan should also include infrastructure functionality testing. Necessary exercises in due diligence include:

- verifying that your Load Balancers fail-over properly;
- ensuring web nodes balance requests correctly;
- and inspecting databases for proper replication functionality.

Finally, creating a documented plan outlined with specific steps for corrective action in the case of a disaster, and testing this plan prior to launch, should be a top priority.

FINAL THOUGHTS

This technical whitepaper merely skimmed the surface on designing and deploying HA Linux Cloud Server architectures. It discussed some crucial Rackspace-specific information and hopefully provided a bit of insight into the tools and methodology to consider when taking on the task of creating HA solutions on The Rackspace Cloud. Cloud computing offers a monumental shift in increased scalability and substantial cost-savings over traditional environments. The cloud is ready for enterprise production environments and organizations who have not yet embraced the technology should consider making it part of their IT strategy at this time.