

HIGH AVAILABILITY IN CLOUD AND DEDICATED INFRASTRUCTURE

AVI FREEDMAN / TECHNICAL ADVISOR

a white paper by





FOREWORD

Even the most robustly architected systems fail from time to time. Fortunately, there are steps you can take to limit your downtime exposure.

This white paper covers manual and automated strategies to help you prevent and remediate downtime. It's the second in a series on dedicated and cloud infrastructure. Click here for the first in the series, titled "[Dedicated vs. Cloud Hosting.](#)"

If you have questions or need advice, you can always reach me at avi@servercentral.com.

Enjoy!

Avi

TABLE OF CONTENTS

Implementing Availability	3
Measuring Availability.	3
MTBF	3
MTTR	3
The Cost Of Unavailability	4
Performing A Root-Cause Analysis.	4
The Cost Of Redundancy.	4
Basic Strategies for MTBF and MTTR	5
Web Servers	5
Database Servers.	5
Load Balancers	6
Network Service IPs	6
Advanced Strategies For MTBF and MTTR	7
Configuration Management Systems	7
Multi-Site Configurations.	7
HA-Awareness In The Application	7
Application-Level Routing	8
Infrastructure Monitoring	8
Stateless Transactions.	9
Queues	9
Self-Healing	9
Web And Application Servers.	9
Database Servers.	10
Network	10
Manual Interaction	10
CMS-Based MTTR Strategies	11
Catasrophic Hardware Failure.	11
Compromised Security	11
Unexpected Traffic Bursts	11
Local Network Overload	11
Architectural Considerations	12
Conclusion.	13
Meet Avi Freedman	14

IMPLEMENTING AVAILABILITY

My application suffers from regular downtime. Sometimes, it's the application. Other times, it's the infrastructure. How can the cloud improve reliability and availability?

A common misconception is that the cloud makes all applications and infrastructures highly available and redundant. While the [cloud is not a panacea](#) for infrastructure and application deployment, when implemented properly, cloud technologies can enable applications to run in a highly available, fully redundant environment at a much lower cost than dedicated infrastructure. The key is building high availability (HA) into both the architecture and the applications. Before we get into how, let's first talk about why.

MEASURING AVAILABILITY

The old adage that “you can't manage what you can't measure” applies—so how do you measure high availability?

Calculate high availability using this equation:

$$\text{Availability Average} = \frac{MTBF}{(MTBF + MTTR)}$$

MTBF = mean time between failures

MTTR = mean time to repair

The key to meeting your availability objectives is maximizing MTBF and minimizing MTTR.

MTBF

In a dedicated infrastructure, MTBF refers to the mean time between individual component failures, like a hard drive malfunction. In a cloud infrastructure, MTBF refers to the mean time between service failures, like a hypervisor reboot.

MTTR

In a dedicated infrastructure, MTTR refers to how long it takes to fix the broken component. If parts are available locally, this can take hours. If parts are ordered from a non-local distribution center, this can take days.

In a cloud infrastructure, MTTR refers to how long it takes the application to come back online. It typically takes minutes to provision another virtual server.

Recovery time is important in measuring applications and is a measurement that is often missed or not properly calculated in application performance.

THE COST OF UNAVAILABILITY

When a web application is down, the effect can be immense:

An e-commerce site that generates \$100,000 per day in gross revenue loses \$4,166.67 per hour of downtime.

Often, the disruption to internal operations far outlasts the outage itself:

That e-commerce site might spend days catching up on shipping, refunds, and customer service calls after just an hour of downtime— and that's without accounting for lost reputation or future order flow.

Despite the business impact, few businesses actually measure how much downtime they rack up in a year.

PERFORMING A ROOT-CAUSE ANALYSIS

Businesses that don't actively plan for and manage failure rates are potentially "flying blind" without knowing their actual level of risk exposure. This is key; you can't fix what you don't know is broken. Risk mitigation and prevention can only be accomplished through understanding why and how each failure occurred. Performing a root-cause analysis on each potential point of failure reveals what needs to be fixed before system reliability can be improved.

An e-commerce company may fix an overloaded network switch by replacing it. But perhaps the root cause wasn't a hardware issue; maybe it was the way the software handled transactions. They might pay for a new switch and schedule downtime to install it when the real answer is a small change in the software configuration (which might also provide the ability to handle many times the traffic or increase redundancy).

TIP: Typically with hardware, additional redundancy is needed. With software, refactoring or re-architecting is needed.

THE COST OF REDUNDANCY

In an enterprise environment, computing and network infrastructures rely on redundancy and availability. It's especially critical to engineer high availability and redundancy at all single points of failure. A properly engineered cloud infrastructure designed for HA can operate almost nonstop and recover from subcomponent glitches in minutes.

In a cloud infrastructure, the costs of redundancy can be significantly less than the same solution in a dedicated infrastructure. This is because the costs of high availability routing, switching, load balancing, and hypervisors is shared across the cloud.

However, there are tradeoffs to consider based on the specific requirements of your application and enterprise:

- Many cloud environments cannot support consistent high network and especially disk performance; and
- If you require special load balancing or firewall equipment configurations, you may need to pay for those to be set up in a hybrid environment.

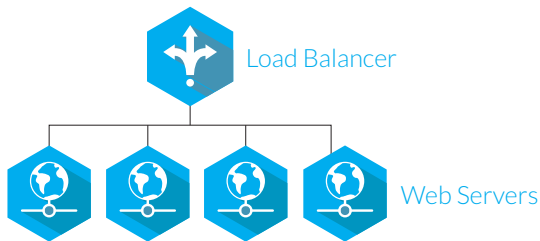
Service recovery times can be reduced significantly through infrastructure redundancy.

BASIC STRATEGIES FOR MTBF AND MTRR

High availability architectures in cloud and dedicated systems should incorporate some or all of the following practices:

WEB SERVERS

Multiple web servers should be incorporated in application infrastructures to provide high availability and redundancy in a production environment. This improves performance because servers will service user requests simultaneously. Depending on the traffic distribution

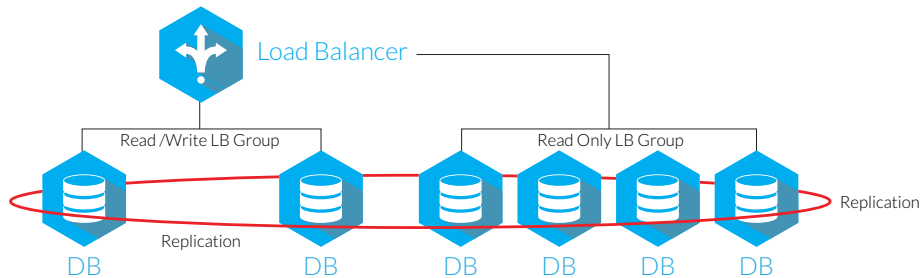


method, a server pool with multiple members should also significantly increase the performance of your application infrastructure.

As more web servers are added to the pool, redundancy increases, MTBF increases, and the ability to service traffic well increases. Load balancing is also important for redundancy and scalability, and when fused properly, the entire web layer can gain elastic scalability.

DATABASE SERVERS

Highly available database deployments can be implemented in many ways. In a more basic configuration, a pair of replicated databases is configured for master-master replication.



For larger traffic volumes or multi-site topologies, complex configurations involving multi-member replication pools with dedicated read and write servers may be required.

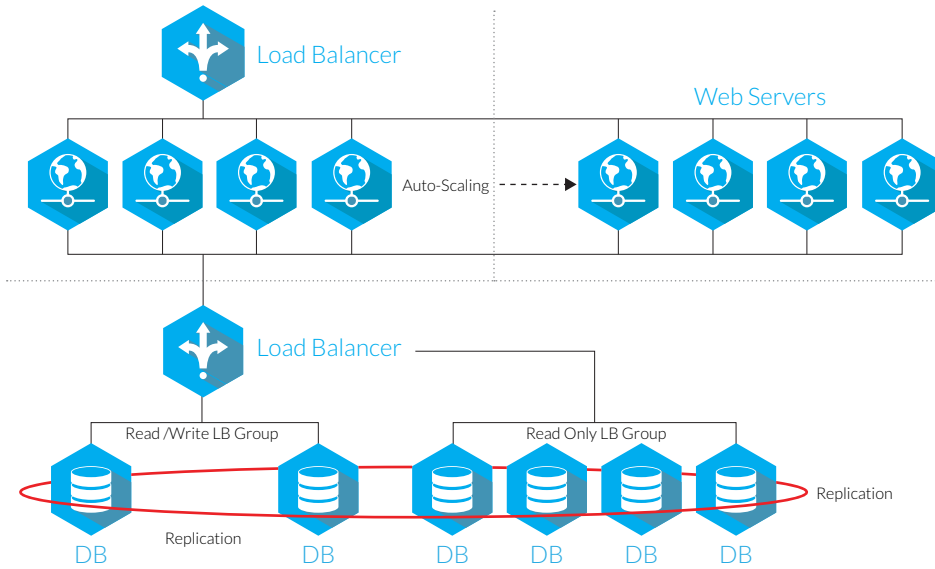
Additional HA and redundancy can be implemented in database server pools by introducing load balancing and implementing application awareness in the application code. Cloud databases should be deployed in multiple member configurations to ensure continued function in the event of a server failure.

Some environments require maximum database performance from day one, while other environments may grow to require this level of performance. Organizations that require it from day one should consider a hybrid infrastructure strategy using dedicated database servers. It's important to determine if and how your cloud/colocation partner(s) can support hybrid infrastructure deployments. It's better to know what's required to make the transition in advance.

The beauty of cloud environments is that a cloud server failure is unlikely to take down an entire application, whereas a failed dedicated server might.

LOAD BALANCERS

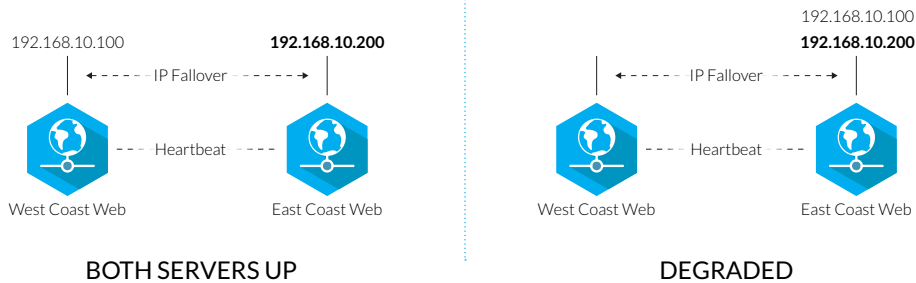
Load balancers do more than distribute load for web servers. They can proxy connections for most types of application servers and distribute traffic to individual servers based on any combination of specialized algorithms. Integrating a load balancer to an application or enterprise provides additional redundancy and elasticity. These enhancements allow an application or server tier to scale horizontally to handle more traffic than the original architecture.



NETWORK SERVICE IPs

Network IP management provides the ability for a published service IP to be moved between machines at the time of a failure. This can be classified as a self-healing process, where two servers are configured to monitor one another. If one server becomes unavailable, the secondary server will assume the roles and processes of the malfunctioning server.

Some packages for Linux that provide this functionality include keepalived and the Linux-HA suite.



An e-commerce site with two database servers can configure the secondary database server to assume the role of the primary server in the event of a hardware failure. This helps guarantee business continuity.

ADVANCED STRATEGIES FOR MTBF AND MTTR

CONFIGURATION MANAGEMENT SYSTEMS

Configuration Management Systems (CMS) are essential tools in any enterprise. When deployed properly in an enterprise infrastructure, a CMS can invoke the ability to remotely execute code, as well as automatically orchestrate and dynamically provision a server or an entire infrastructure. This is extremely helpful in MTBF and MTTR risk mitigation.

For an e-commerce site experiencing heavy traffic load due to viral social networking, a CMS can dynamically scale multiple application tiers in minutes to meet traffic needs.

A CMS also serves as a central repository for all infrastructure and application management for that infrastructure. Specific tasks can be automated or “templated” to reduce MTTR, such as:

- Setting up a test environment, deploying code, and running automated functionality checks
- Adding and removing machines to application layers
- Ensuring that the monitoring systems are updated as infrastructure and software components change

Collecting and analyzing application and infrastructure failure data enables organizations to plan and redesign architectures that are more robust. It can also help determine which CMS to implement, such as Chef, Puppet, Salt, CFEngine, or Ansible. In many cases, your key packages will come bundled with support for one of those engines, which may direct you towards that engine. As a general rule, we advise starting with simpler systems like Salt or Ansible.

MULTI-SITE CONFIGURATIONS

With a well-designed architecture, in the unlikely event of a catastrophic hardware failure, resources can be redeployed to a secondary location in minutes and with little planning. Data replication and resource availability is present in the secondary location and the just-in-time deployment of entire application infrastructures are measured in minutes, not hours or longer.

When architected and implemented properly, an e-commerce company can redeploy their entire infrastructure into a new datacenter in the event of a natural disaster.

An organization that cannot tolerate downtime in their application infrastructure will most likely opt for a multi-site configuration. In this configuration, the additional site is a completely independent datacenter that will host an independent copy of the primary site infrastructure. Depending on how the site application is configured, the additional site can be in an active-active configuration that will service a portion of the traffic coming into the site.

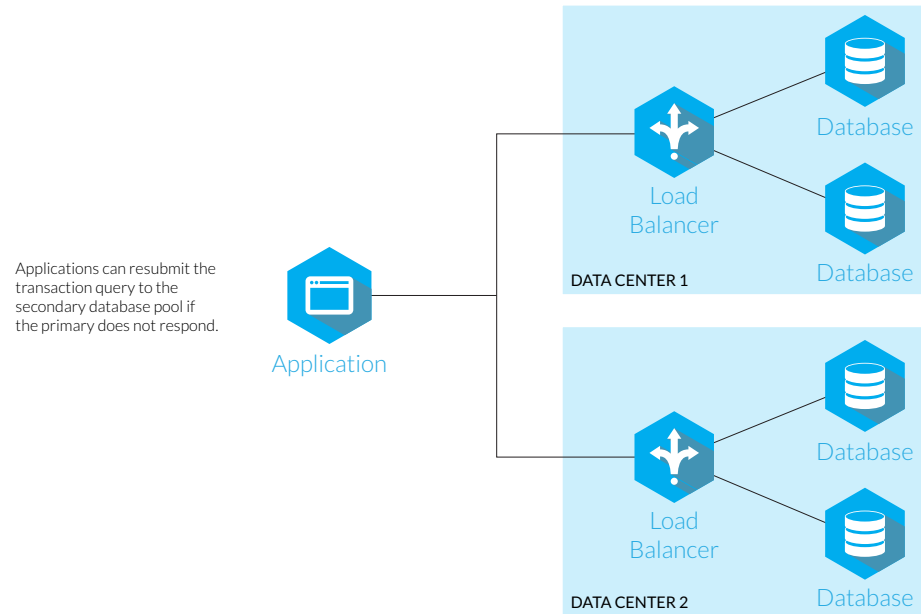
A multi-site configuration can also be a primary-failover site that will not serve traffic, but sit idle while continuously replicating data from the primary.

HA-AWARENESS IN THE APPLICATION

Application architecture can significantly improve robustness and availability in a multi-site deployment. In the event of a system failure, a well-architected application that is cloud-aware will improve transaction completion rates. In more advanced systems, application components (or central monitors) can even request additional infrastructure resources be made available at a predetermined threshold of load, traffic, and/or performance.

APPLICATION-LEVEL ROUTING

In the event of a transaction failure, cloud-aware applications can be engineered to intelligently route transactions to a secondary service point. Applications can then be engineered to resubmit the transaction to a backup infrastructure which is completely transparent to the end user.



INFRASTRUCTURE MONITORING

System and infrastructure monitoring is a critical part of MTBF/MTTR risk mitigation. A well-integrated monitoring package can provide insight into an application infrastructure at discrete detail levels. Alerts can be configured to perform automated self-healing tasks on the infrastructure and open a support ticket with the Network Operations Center (NOC) automatically. This provides insight into an application and its current function, monitoring error-rates that exceed a pre-determined threshold.

Some widely available monitoring packages are Nagios, Cacti, Zabbix, and Icinga. When looking at monitoring packages, it is important to note your business and technical objectives so that all stakeholders are represented in the process. Outages or periods of negative performance equally impact the business and technology.

An e-commerce site can monitor a payment gateway and alert the NOC if credit card authorization transactions exceed a 20% failure rate, which shows potential problems in the authorization software.

STATELESS TRANSACTIONS

Engineering an application to perform transactions in a stateless manner significantly improves availability. In a stateless model, any machine only keeps state (data on) transactions that are 'in fly,' but after a transaction is completed, any machines that die or degrade have no effect on the state or memory of historic transactions. Clients are therefore not limited to server dependence, and the loss of any pool member in a tier ensures the client session is not interrupted due to a hardware or application fault on a discrete pool member. The key is to avoid storing permanent state (i.e., transactions, inventory, user data) on individual logical or physical servers.

[Amazon.com, the world's largest e-commerce store, leverages stateless transactions with a static key-value store to persist shopping carts indefinitely.](#)

QUEUES

Ordered, first-in first-out lists (implemented as an HA service) increase availability and performance by distributing a transaction to the first-available member for processing in a distributed environment. RabbitMQ and ZeroMQ are leading open source packages that implement HA queuing systems.

[An e-commerce application can leverage queues by quickly performing checkout and search operations and routing shopper requests to the first-available pool member. Even a 300 millisecond delay is noticeable to a shopper.](#)

SELF-HEALING

Advanced monitoring solutions can be implemented into self-healing application infrastructures. With transaction-level monitoring, applications can effectively identify components that have failed or become non-responsive and recover them through a pre-determined process.

In a dedicated infrastructure, additional resources have to be available for automated provisioning in concert with provisioning and load balancing systems to implement the strategies that involve replacing resources.

WEB AND APPLICATION SERVERS

Web and application servers can be monitored based on the expectation of a correct response and transactional latency. It is important to implement this kind of monitoring because it yields the data necessary to feed load balancing systems and enables them to automatically resolve issues that normally require human intervention. This can decrease the reliance on DevOps for automatable tasks, as well as improve net uptime and service levels to your customers.

An application monitor that measures a preprogrammed function on an individual web server can integrate with the backend automation systems to initiate several responses, such as:

- Restarting the server process and dependencies
- Starting up additional server processes on the existing server to handle the additional load
- Removing a degraded web and automatically rebuilding and redeploying based on a machine template pre-determined in the provisioning system

DATABASE SERVERS

Enterprise database deployments should always include a redundancy strategy. As database failure causes complete application failure in most cases, a redundant database implementation provides increased performance as well as risk mitigation in an enterprise application deployment. In cases where performance is heavily reliant on a database, a hybrid solution of dedicated database servers combined with cloud application and web servers provides maximum performance with data redundancy.

Redundancy in a production database serves the primary purpose of data availability, which allows application and web servers to provide business continuity in the event of a failure.

[An e-commerce site with one slow database member could potentially cause a price drop of inventory items to zero \(as documented by Best Buy and Walmart\).](#)

Examples of self-healing processes for databases include:

- Resynchronizing a database member due to database staleness
- Adding additional database instances to the cluster
- Providing a snapshot of database states and performing nightly or hourly backups
- Monitoring transactional lag on individual cluster members and removing the discrete members from the production pool until they are in sync with the other cluster members

NETWORK

It is critical to have high availability at the network layer. For dedicated infrastructures, ServerCentral usually deploys aggregated Ethernet links to each server or storage node, connected to different uplink switches. This allows a “rolling upgrade” of the network layer to fix bugs or security issues, without affecting uptime of the applications supported by the infrastructure.

In the cloud, you can generally assume that the network has been implemented redundantly, but it’s best to ask:

- How is redundancy implemented from the server to the Internet?
- How large/small are the network failure zones? In general, the smaller, the better.
- Does the provider use virtual switching and tunneling? In many cases, particularly with larger zones, virtual switching and tunneling necessitates a forced reduction in throughput available between any network clients.

MANUAL INTERACTION

In many infrastructures, manual interaction is required for unexpected scenarios. MTBF and MTTR strategies such as redundant teams and cross training are advantageous in human-interactive scenarios.

CMS-BASED MTTR STRATEGIES

Configuration Management Systems (CMS) allow for ease of implementation in MTTR mitigation, as any component can be redeployed as needed. Below are some valuable CMS-based MTTR strategies.

CATASPROPHIC HARDWARE FAILURE

In the unlikely event of a catastrophic failure, resources can be redeployed to a secondary location in minutes with little planning. Data replication and resource availability is present in the secondary location and the just-in-time deployment of entire application infrastructures is measured in minutes, not hours or longer.

COMPROMISED SECURITY

In the event that server or resource is compromised, a new instance of the application can be deployed in minutes in the cloud, rather than manually deploying the new resource. Of course, the new deployment must be installed on top of systems with pre-mitigated security vulnerabilities. Firewalls and access control lists can be updated to include mitigating a compromised resource.

An e-commerce company with compromised payment systems affecting 1800 stores nationwide can rebuild and redeploy all systems within minutes. They don't have to worry about code integrity after a firewall strategically isolates the affected system.

UNEXPECTED TRAFFIC BURSTS

Servers can be deployed or resized in a functional tier manually through CMS tier management strategies. The size of the functional pools may also be manually adjusted.

An e-commerce company can dynamically expand their infrastructure to handle an expected or unexpected increase in traffic due to a change in market conditions.

LOCAL NETWORK OVERLOAD

Data can be redirected to utilize a separate, discrete path, and entire application infrastructures can be migrated to a different physical location through network bridges. A CMS enables automated configuration, testing, and inserting of new services into the application layer(s), allowing the underlying network redundancy to be utilized so both sites continue to service traffic simultaneously.

TIP: Dedicated infrastructures can benefit from redundancy by re-provisioning specific resources in an infrastructure, as opposed to redirecting all traffic to an ancillary site.

Applications will continue to be available throughout a physical component failure and remediation period.

ARCHITECTURAL CONSIDERATIONS

Architectural considerations for MTBF and MTTR are important in any infrastructure, and specific strategies can be implemented in your application to increase availability.

What if the infrastructure fails? What else can I do?

You should expect that that every infrastructure will fail. No matter how many things you do right, failure is inevitable. How and when are just variables.

In some cases, a software error causes cascading failure in an entire infrastructure. For example, poorly developed web code can cause an entire infrastructure to fail.

What you can do is evolve your risk mitigation strategies alongside your applications and infrastructure. While this is time consuming, companies like ServerCentral can help.

How can I change my infrastructure without increasing human resource capital?

Enterprise storage and backup solutions significantly reduce MTTR times. These efficient recovery strategies often return infrastructures to full service with very limited outages—usually in minutes.

Engineering applications specifically for the cloud can significantly improve uptime, availability, and performance.

Can I prevent downtime?

Unfortunately, the answer is almost always no. Even the most robustly architected systems fail from time to time. You can, however, limit downtime exposure. Experienced hosting providers can help you implement automated mitigation strategies that help prevent and remediate downtime. As an example, here are some of the risks that ServerCentral addresses with customers:

RISK	MITIGATION
PLANNED DOWNTIME	
<ul style="list-style-type: none"> Hardware Software OSs Devices 	<ul style="list-style-type: none"> Manage hardware and software mix to reduce updates Use server clustering and manual failovers to upgrade without downtime
COMPONENT FAILURES	
<ul style="list-style-type: none"> Hardware Software Programming Viruses File corruption 	<ul style="list-style-type: none"> Availability SLAs Server clusters and load balancing Website version control and testing Security management Data backup Spare parts on site
HUMAN ERROR	
<ul style="list-style-type: none"> File deletion Unskilled operation Access management 	<ul style="list-style-type: none"> Training Test and development servers Staff certification requirements
MALICIOUS ATTACKS	
<ul style="list-style-type: none"> DDoS Hackers 	<ul style="list-style-type: none"> Managed security service

Application and infrastructure redundancy significantly increases MTBF by mitigating failed components in the overall system. Provisioning parallel redundant systems that can concurrently perform the same functions in an infrastructure not only provides a reduced failure rate, it increases system capacity.

CONCLUSION

MTTR mitigation strategies implemented with a services-based recovery approach can improve response times and service availability. They also enhance existing architectures with the ability to scale horizontally as resource utilization requirements change. In many cases, a well-managed MTBF/MTTR strategy increases revenue by:

- Reducing the risk of financial and opportunistic revenue loss
- Enabling horizontal scalability as resource utilization requirements change

Regardless of your chosen method in pursuit of HA architectures, it's critical to begin working towards understanding your MTBF and MTTR immediately. If you would like to discuss this topic or need help with your MTBF and MTTR calculations, give us a call at +1(312) 829-1111 or email us at sales@servercentral.com. We welcome the opportunity to learn about your business and IT environment and are happy to share our advice.

