MySQL and Virtualization Guide

MySQL and Virtualization Guide

Abstract

This is the MySQL and Virtualization extract from the MySQL Reference Manual.

Document generated on: 2009-06-02 (revision: 15161)

MySQL and Virtualization

Using virtualization can be an effective way of better utilizing the hardware of your machine when using MySQL, or to provide improved security or isolation of different instances of MySQL on the same machine. In some circumstances, virtualization may be a suitable solution for scaling out your database environment by enabling you to easily deploy additional instances of a pre-configured MySQL server and application environment to new virtualization hosts.

With any virtualization solution there is often a tradeoff between the flexibility and ease of deployment and performance, or between the potential performance advantage and complexities of effectively configuring multiple instances of MySQL to reside within a single physical host.

Different issues are experienced according to the virtualization environment you are using. Virtualization generally falls into one of the following categories:

- **Native virtualization**, including products like VMware Workstation, Parallels Desktop/Parallels Workstation, Microsoft Virtual PC and VirtualBox, all work by acting as an application that runs within an existing operating system environment. Recent versions can take advantage of the virtualization extensions in the Intel and AMD CPUs to help improve performance.

  The application-based solutions have a number of advantages, including the ability to prioritize CPU usage (including multiple CPUs) and easily run multiple virtualized environments simultaneously.

  With these solutions, you also have the ability to easily create a virtualized environment that can be packaged and shared among different virtualization hosts. For example, you can create a MySQL environment and configuration that can be deployed multiple times to help extend an existing scalability or HA environment.

  The major disadvantage of this type of virtualization environment is the effect of the host on the performance of the virtualization instances. Disk storage is typically provided by using one or more files on the host OS which are then emulated to provide physical disks within the virtual instance. Other resources on the host are similarly shared, including CPU, network interfaces and additional devices (USB). It is also difficult to directly share lower-level components, such as PCI devices and that the ability to take advantage of RAID storage solutions.

- **Paravirtualization (Hypervisor)**, including Xen, Solaris xVM (based on Xen), VMware ESX Server, Windows Server 2008 Hyper-V, and Solaris Logical Domains (LDOM), work by running a specialized version of the host operating system. The host OS then allows slightly modified versions of different operating systems to run within the virtualized environment.

  With paravirtualization, the level of performance and the control over the underlying hardware used to support the virtualized environments is higher than native virtualization solutions. For example, using paravirtualization you can dedicate individual CPU cores, RAM, disk drives and even PCI devices to be accessible to individual and specific virtual instances.

  For example, within a paravirtualized environment you could dedicate a physical disk drive or subsystem to a particular virtual environment and gain a performance benefit over a typical file-based solution virtual disk.

- **Operating system-level virtualization**, including BSD jails, and Solaris Containers/Zones, offer methods for isolating different instances of an operating system environment while sharing the same hardware environment. Unlike the other virtualization solutions, operating system level virtualization is not normally used to run other operating systems, but instead to provide a level of security isolation and resource control within the core operating environment.

  The isolation of these different instances is the key advantage of this type of virtualization. Each virtualized instance sees its environment as if it were completely different system. The solution can be an effective method to provide isolated computing resources for different departments or users, or to provide unique instances for testing and development.

The main reasons for using virtualization, particularly with a database or an application stack that includes a database component, include:

- **Security** — separate instances of different operating systems running within a single host but with effective isolation from each other. When used with MySQL, you can provide an increased level of security between different instances of each server.

- **Consolidation** — merging a number of individual systems with a relatively small load onto a single, larger, server. This can help reduce footprint and energy costs, or make more efficient use of a larger machine. Performance is the main issue with this solution as the load of many MySQL databases running in individual virtual instances on a single machine can be considerable.

- **Development/QA/Testing** — by creating different instances of different environments and operating systems you can test your MySQL-based application in different environments.

- **Scalability** — although using virtualization imposes a performance hit, many virtualization solutions allow you to create a packaged version of an environment, including MySQL and the other application components. By distributing the virtualization environment package to new hosts you can often very quickly scale out by adding new hosts and deploying the virtualized en-

vironment.

The remainder of this chapter looks at common issues with using MySQL in a virtualized environment and tips for using MySQL within different virtualization tools.

For advice on common issues and problems, including performance and configuration issues, when using virtualized instances, see Chapter 1, *Common Issues with Virtualization*.

Chapter 1. Common Issues with Virtualization

There are many issues related to using MySQL within a virtualized environment that are common across the different virtualization types. Most are directly related to the performance or security of the environment in which you are deploying the MySQL server compared to the host on which you are running the virtualization solution.

Before deciding to use virtualization as a solution for your database, you should ensure that the expected load for the server and the expected performance when run in a virtualized environment meet your needs and requirements.

To help you determine the issues and some of the potential solutions, use the following sections:

- For general performance issues, problems and the probable causes, see Section 1.1, “Virtualization Performance Issues”.
- Disk and storage concerns directly affect database storage because most database access is limited by the I/O bandwidth. For some examples and issues, see Section 1.2, “Virtualization Storage Issues”.
- Issues related to network configuration and performance may need more careful planning, especially if you are using network-specific technologies such as MySQL replication. For further examples and details, see Section 1.3, “Virtualization Networking Issues”.

1.1. Virtualization Performance Issues

Often the biggest consideration is the performance of a virtualized environment once hosted. In most cases, the virtualized environment involves some level of emulation of one or more of the hardware interfaces (CPU, network or disk) of the host environment. The effect is to reduce the effective performance of the virtualized environment compared to running an application natively on the host.

Some core resourcing issues to be aware of include:

- Using virtualization does not reduce the amount of CPU required to support a particular application or environment. If your application stack requires 2GB of RAM on an individual machine, the same RAM requirement will apply within your virtualized environment. The additional overhead of the virtualization layer and host operating system or environment often mean that you will need 2.5GB or 3GB of RAM to run the same application within the virtualized environment.

  You should configure your virtualization environment with the correct RAM allocation according to your applications needs, and not to maximize the number of virtualized environments that you can execute within the virtualization host.

- Virtualization of the CPU resources is more complex. If your MySQL database and application stack do not have a high CPU load, then consolidating multiple environments onto a single host is often more efficient. You should keep in mind that at peak times your application and database CPU requirement may need to grow beyond your default allocation.

  With some virtualization environments (Xen, Solaris Containers, Solaris LDOMs) you can dedicate CPU or core to a virtual instance. You should use this functionality to improve performance for database or application loads that have a high constant CPU requirement as the performance benefit will outweigh the flexibility of dynamic allocation of the CPU resources.

- Contention of resources within the host should be taken into account. In a system with high CPU loads, even when dedicating RAM and CPU resources, the I/O channels and interfaces to storage and networking resources may exceed the capacity of the host. Solutions such as Xen and Solaris LDOMs dedicate specific resources to individual virtual instances, but this will not eliminate the effects of the overall load on the host.

- If your database application is time sensitive, including logging and real-time database applications, or you are using MySQL Cluster, then the effects of virtualization may severely reduce the performance of your application. Because of the way the virtualized instances are executed and shared between CPUs and the effects of load on other resources, the response times for your database or application may be much higher than normal. This is especially true if you are running a large number of virtualized instances on a single host.

- Be aware of the limitation of using a single host to run multiple virtualized instances. In the event of a machine or component failure, the problem will affect more than just one database instance. For example, a failure in a storage device could bring down all your virtualized instances. Using a RAID solution that supports fault tolerance (RAID levels 1,3,4,5 or 6) will help protect you from the effects of this.

1.2. Virtualization Storage Issues

Due to the random I/O nature of any database solution, running MySQL within a virtualized environment places a heavy load on

the storage solution you are using. To help keep the performance of your virtualized solution at the highest level, you should use the following notes to help configure your systems.

- Some virtualization solutions allow you to use a physical disk directly within your virtual host as if it were a local disk. You should use this whenever possible to ensure that disk contention issues do not affect the performance of your virtual environment.

  When running multiple virtual machines, you should use an individual disk for each virtual instance. Using a single disk and multiple partitions, with each partition dedicated to a virtual host, will lead to the same contention issues.

- If you are using standard file-based storage for your virtualized disks:

  - File-based storage is subject to fragmentation on the host disk. To prevent fragmentation, create a fixed-size disk (that is, one where the entire space for the disk file is preallocated) instead of a dynamic disk that will grow with usage. Also be prepared to defragment the disk hosting the files at regular intervals to reduce the fragmentation.

  - Use separate disk files for the operating system and database disks, and try to avoid partitioning a disk file as this increases the contention within the file.

  - Use a high-performance disk solution, such as RAID or SAN, to store the disk files for your virtualized environments. This will improve the performance of what is essentially a large single file on a physical device.

  - When running a number of different virtualized environments within a single host, do not use the same physical host drive for multiple virtual disks. Instead, spread the virtual disks among multiple physical disks. Even when using a RAID device, be aware that each virtual host is equivalent to increasing the load linearly on the host RAID device.

1.3. Virtualization Networking Issues

When running multiple virtual machines on a host, you should be aware of the networking implications of each virtualized instance. If your host machine has only one network card, then you will be sharing the networking throughput for all of your machines through only one card, and this may severely limit the performance of your virtual environments.

If possible, you should use multiple network cards to support your virtualized instances. Depending on the expected load of each instance, you should dedicate or spread the allocation of the virtual network devices across these physical devices to ensure that you do not reach saturation.

If you are using packaged virtual machines as the basis for deployment of your MySQL database solution, you should make sure that the network interfaces are correctly reconfigured. Some solutions duplicate the hardware MAC address which will cause problems when you start up additional instances of the same virtualized environment.

Chapter 2. Using MySQL within an Amazon EC2 Instance

The Amazon Elastic Compute Cloud (EC2) service provides virtual servers that you can build and deploy to run a variety of different applications and services, including MySQL. The EC2 service is based around the Xen framework, supporting x86, Linux based, platforms with individual instances of a virtual machine referred to as an Amazon Machine Image (AMI). You have complete (root) access to the AMI instance that you create, allowing you to configure and install your AMI in any way you choose.

To use EC2, you create an AMI based on the configuration and applications that you want to use and upload the AMI to the Amazon Simple Storage Service (S3). From the S3 resource, you can deploy one or more copies of the AMI to run as an instance within the EC2 environment. The EC2 environment provides management and control of the instance and contextual information about the instance while it is running.

Because you can create and control the AMI, the configuration, and the applications, you can deploy and create any environment you choose. This includes a basic MySQL server in addition to more extensive replication, HA and scalability scenarios that enable you to take advantage of the EC2 environment, and the ability to deploy additional instances as the demand for your MySQL services and applications grow.

To aid the deployment and distribution of work, three different Amazon EC2 instances are available, small (identified as `m1.small`), large (`m1.large`) and extra large (`m1.xlarge`). The different types provide different levels of computing power measured in EC2 computer units (ECU). A summary of the different instance configurations is shown here.

	Small	**Large**	**Extra Large**
Platform	32-bit	64-bit	64-bit
CPU cores	1	2	4
ECUs	1	4	8
RAM	1.7GB	7.5GB	15GB
Storage	150GB	840GB	1680GB
I/O Performance	Medium	High	High

The typical model for deploying and using MySQL within the EC2 environment is to create a basic AMI that you can use to hold your database data and application. Once the basic environment for your database and application has been created you can then choose to deploy the AMI to a suitable instance. Here the flexibility of having an AMI that can be re-deployed from the small to the large or extra large EC2 instance makes it easy to upgrade the hardware environment without rebuilding your application or database stack.

To get started with MySQL on EC2, including information on how to set up and install MySQL within an EC2 installation and how to port and migrate your data to the running instance, see Section 2.1, “Setting Up MySQL on an EC2 AMI”.

For tips and advice on how to create a scalable EC2 environment using MySQL, including guides on setting up replication, see Section 2.3, “Deploying a MySQL Database Using EC2”.

2.1. Setting Up MySQL on an EC2 AMI

There are many different ways of setting up an EC2 AMI with MySQL, including using any of the pre-configured AMIs supplied by Amazon.

The default *Getting Started* AMI provided by Amazon uses Fedora Core 4, and you can install MySQL by using `yum`:

```
shell> yum install mysql
```

This will install both the MySQL server and the Perl DBD::mysql driver for the Perl DBI API.

Alternatively, you can use one of the AMIs that include MySQL within the standard installation.

Finally, you can also install a standard version of MySQL downloaded from the MySQL website. The installation process and instructions are identical to any other installation of MySQL on Linux. See Installing and Upgrading MySQL.

The standard configuration for MySQL places the data files in the default location, `/var/lib/mysql`. The default data directory on an EC2 instance is `/mnt` (although on the large and extra large instance you can alter this configuration). You must edit `/etc/my.cnf` to set the `datadir` option to point to the larger storage area.

> **Important**
>
> The first time you use the main storage location within an EC2 instance it needs to be initialized. The initialization

process starts automatically the first time you write to the device. You can start using the device right away, but the write performance of the new device is significantly lower on the initial writes until the initialization process has finished.

To avoid this problem when setting up a new instance, you should start the initialization process before populating your MySQL database. One way to do this is to use `dd` to write to the file system:

```
root-shell> dd if=/dev/zero of=initialize bs=1024M count=50
```

The preceding will create a 50GB on the file system and start the initialization process. You should delete the file once the process has finished.

The initialization process can be time-consuming. On the small instance, initialization will take between two and three hours. For the large and extra large drives, the initialization will be 10 or 20 hours, respectively.

In addition to configuring the correct storage location for your MySQL data files, you should also consider setting the following other settings in your instance before you save the instance configuration for deployment:

- Set the MySQL server ID so that when you use it for replication the ID information is set correctly.
- Enabling binary logging so that replication can be initialized without starting and stopping the server.
- Set the caching and memory parameters for your storage engines. There are no limitations or restrictions on what storage engines you use in your EC2 environment. Choose a configuration, possibly using one of the standard configurations provided with MySQL appropriate for the instance on which you expect to deploy. The large and extra large instances have RAM that can be dedicated to caching. Be aware that if you choose to install `memcached` on the servers as part of your application stack you must ensure there is enough memory for both MySQL and `memcached`.

Once you have configured your AMI with MySQL and the rest of your application stack, you should save the AMI so that you can deploy and reuse the instance.

Once you have your application stack configured in an AMI, populating your MySQL database with data should be performed by creating a dump of your database using `mysqldump`, transferring the dump to the EC2 instance, and then reloading the information into the EC2 instance database.

Before using your instance with your application in a production situation you should be aware of the limitations of the EC2 instance environment. See Section 2.2, “EC2 Instance Limitations”. To begin using your MySQL AMI, you should consult the notes on deployment. See Section 2.3, “Deploying a MySQL Database Using EC2”.

2.2. EC2 Instance Limitations

There are some limitations of the EC2 instances that you should be aware of before deploying your applications. Although these shouldn't affect your ability to deploy within the Amazon EC2 environment, they may alter the way you setup and configure your environment to support your application.

- Data stored within instances is not persistent. If you create an instance and populate the instance with data, then the data will only remain in place while the machine is running. The data will survive a reboot. If you shut down the instance, any data it contained will be lost.

  To ensure that you do not lose information, take regular backups using `mysqldump`. If the data being stored is critical, consider using replication to keep a “live” backup of your data in the event of a failure. When creating a backup, write the data to the Amazon S3 service to avoid the transfer charges applied when copying data offsite.
- EC2 instances are not persistent. If the hardware on which an instance is running fails, then the instance will be shut down. This can lead to loss of data or service.
- If you want to use replication with your EC2 instances to a non-EC2 environment, be aware of the transfer costs to and from the EC2 service. Data transfer between different EC2 instances is free, so using replication within the EC2 environment does not incur additional charges.
- Certain HA features are either not directly supported, or have limiting factors or problems that may reduce their utility. For example, using DRBD or MySQL Cluster may not work. The default storage configuration is also not redundant. You can use software-based RAID to improve redundancy, but this implies a further performance hit.

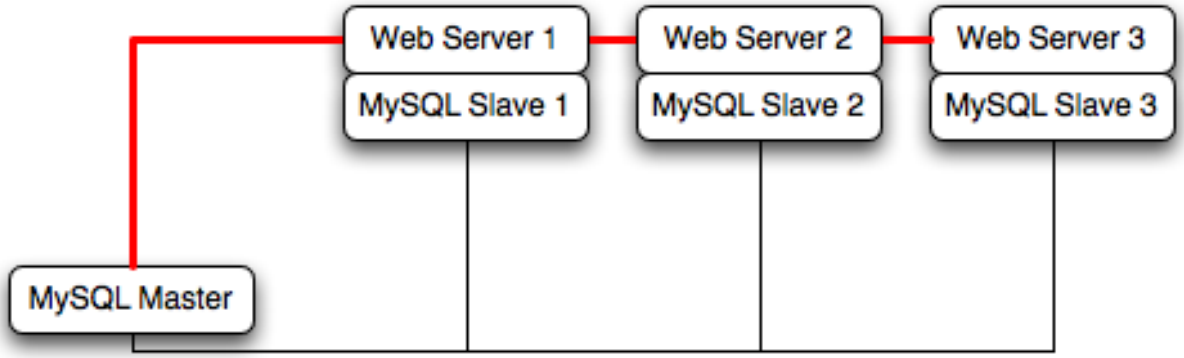
2.3. Deploying a MySQL Database Using EC2

Because you cannot guarantee the uptime and availability of your EC2 instances, when deploying MySQL within the EC2 environment you should use an approach that enables you to easily distribute work among your EC2 instances. There are a number of ways of doing this. Using sharding techniques, where you split the application across multiple servers dedicating specific blocks of your dataset and users to different servers is an effective way of doing this. As a general rule, it is easier to create more EC2 instances to support more users than to upgrade the instance to a larger machine.

The EC2 architecture means that you should treat the EC2 instances as temporary, cache-based solutions, rather than as a long-term, high availability solution. In addition to using multiple machines, you should also take advantage of other services, such as `memcached` to provide additional caching for your application to help reduce the load on the MySQL server so that it can concentrate on writes. On the large and extra large instances within EC2, the RAM available can be used to provide a large memory cache for data.

Most types of scale out topology that you would use with your own hardware can be used and applied within the EC2 environment. However, you should be use the limitations and advice already given to ensure that any potential failures do not lose you any data. Also, because the relative power of each EC2 instance is so low, you should be prepared to alter your application to use sharding and add further EC2 instances to improve the performance of your application.

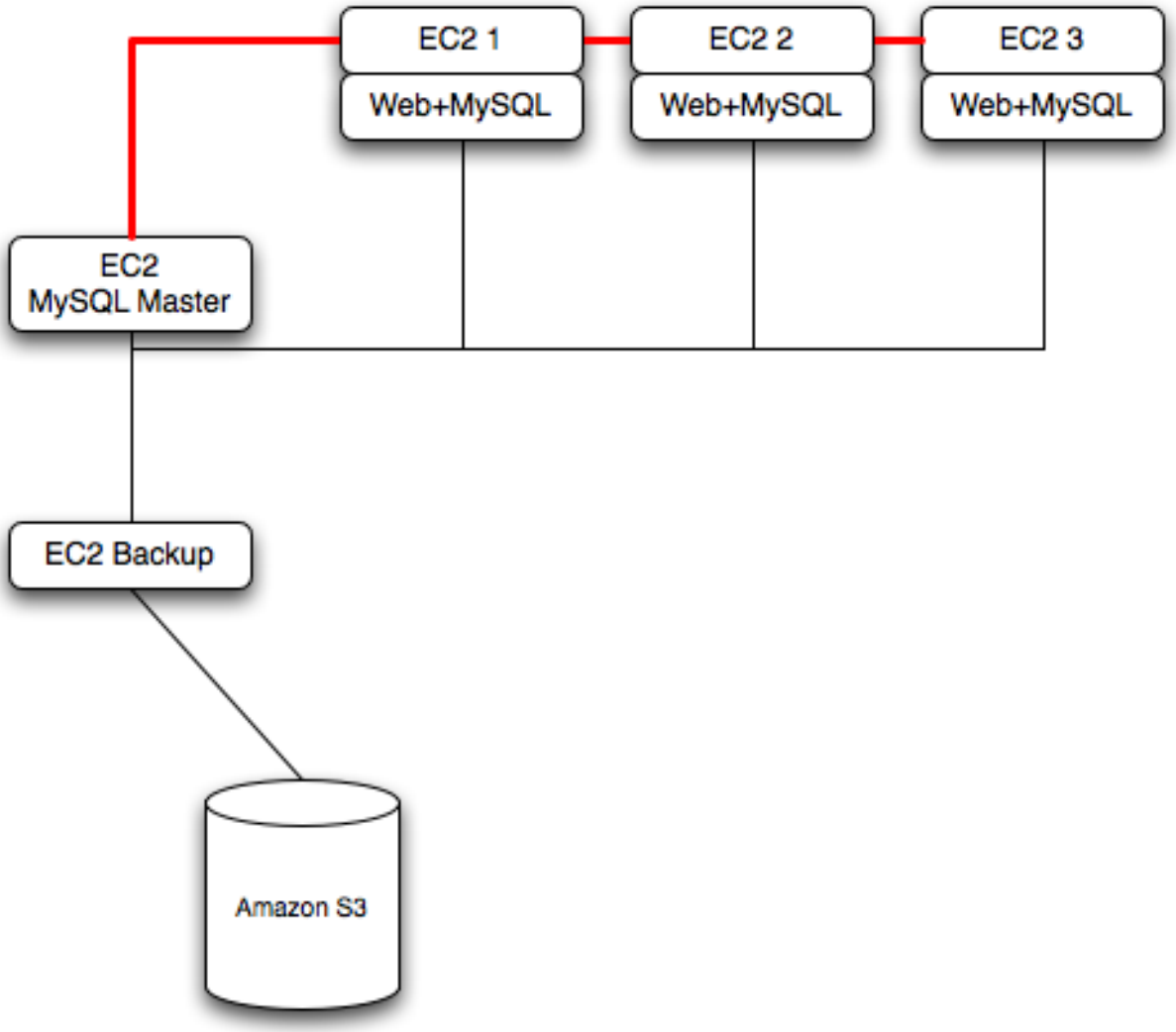
For example, take the typical scale-out environment shown following, where a single master replicates to one or more slaves (three in this example), with a web server running on each replication slave.



You can reproduce this structure completely within the EC2 environment, using an EC2 instance for the master, and one instance for each of the web and MySQL slave servers.

> **Note**
>
> Within the EC2 environment, internal (private) IP addresses used by the EC2 instances are constant. You should always use these internal addresses and names when communicating between instances. Only use public IP addresses when communicating with the outside world - for example, when publicizing your application.

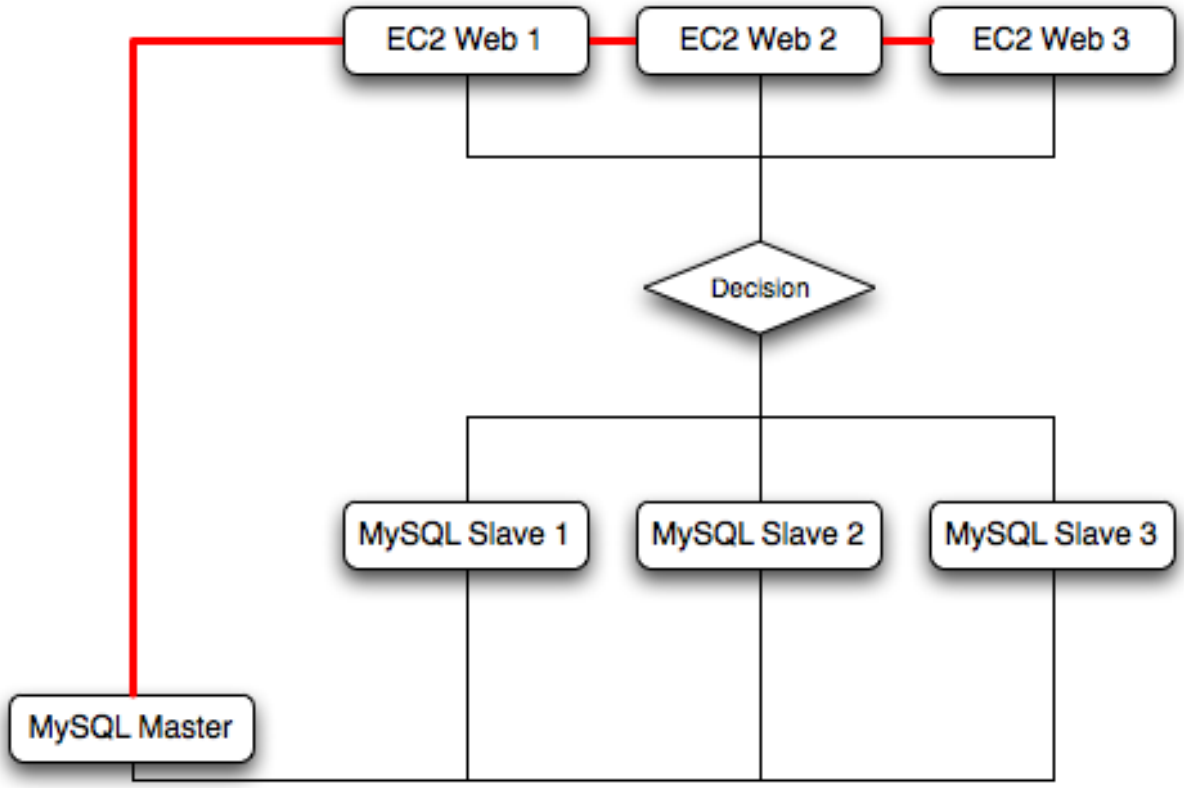
To ensure reliability of your database, you should add at least one replication slave dedicated to providing an active backup and storage to the Amazon S3 facility. You can see an example of this in the following topology.



Using `memcached` within your EC2 instances should provide better performance. The large and extra large instances have a significant amount of RAM. To use `memcached` in your application, when loading information from the database, first check whether the item exists in the cache. If the data you are looking for exists in the cache, use it. If not, reload the data from the database and populate the cache.

Sharding divides up data in your entire database by allocating individual machines or machine groups to provide a unique set of data according to an appropriate group. For example, you might put all users with a surname ending in the letters A-D onto a single server. When a user connects to the application and their surname is known, queries can be redirected to the appropriate MySQL server.

When using sharding with EC2 you should separate the web server and MySQL server into separate EC2 instances, and then apply the sharding decision logic into your application. Once you know which MySQL server you should be using for accessing the data you then distribute queries to the appropriate server. You can see a sample of this in the following illustration.



Warning

With sharding and EC2 you should be careful that the potential for failure of an instance does not affect your application. If the EC2 instance that provides the MySQL server for a particular shard fails, then all of the data on that shard will be unavailable.

Chapter 3. Virtualization Resources

For more information on virtualization, see the following links:

- MySQL Virtualization Forum
- Amazon Elastic Compute Cloud (Amazon EC2)
- MySQL and Cloud Computing
- MySQL Enterprise for Amazon EC2