UNIT-IV

PART-B

Disaster Recovery

Syllabus

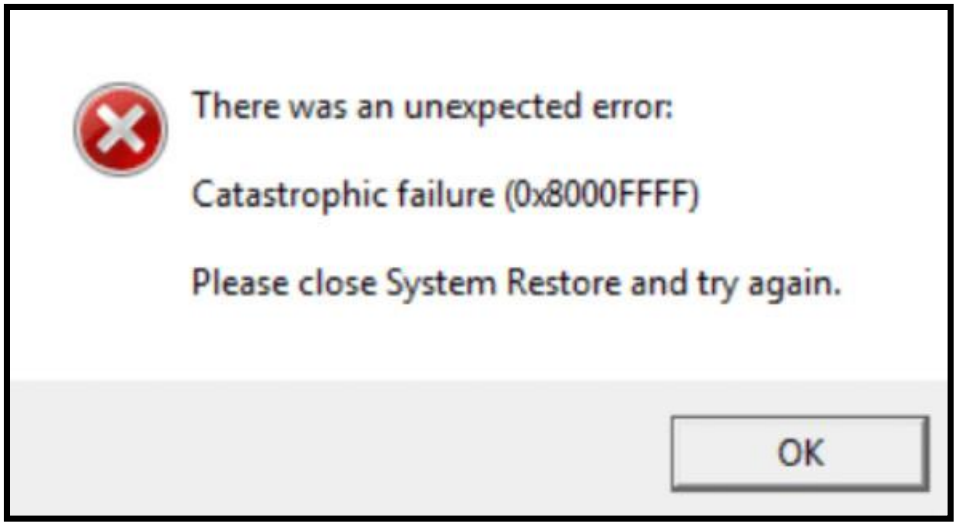
- Disaster Recovery Planning
- Disasters in the Cloud
- Disaster Management

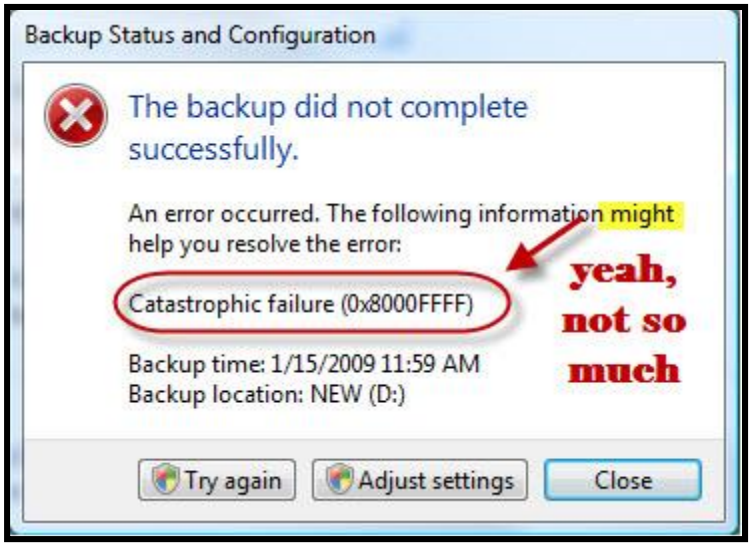
Disaster Recovery Planning

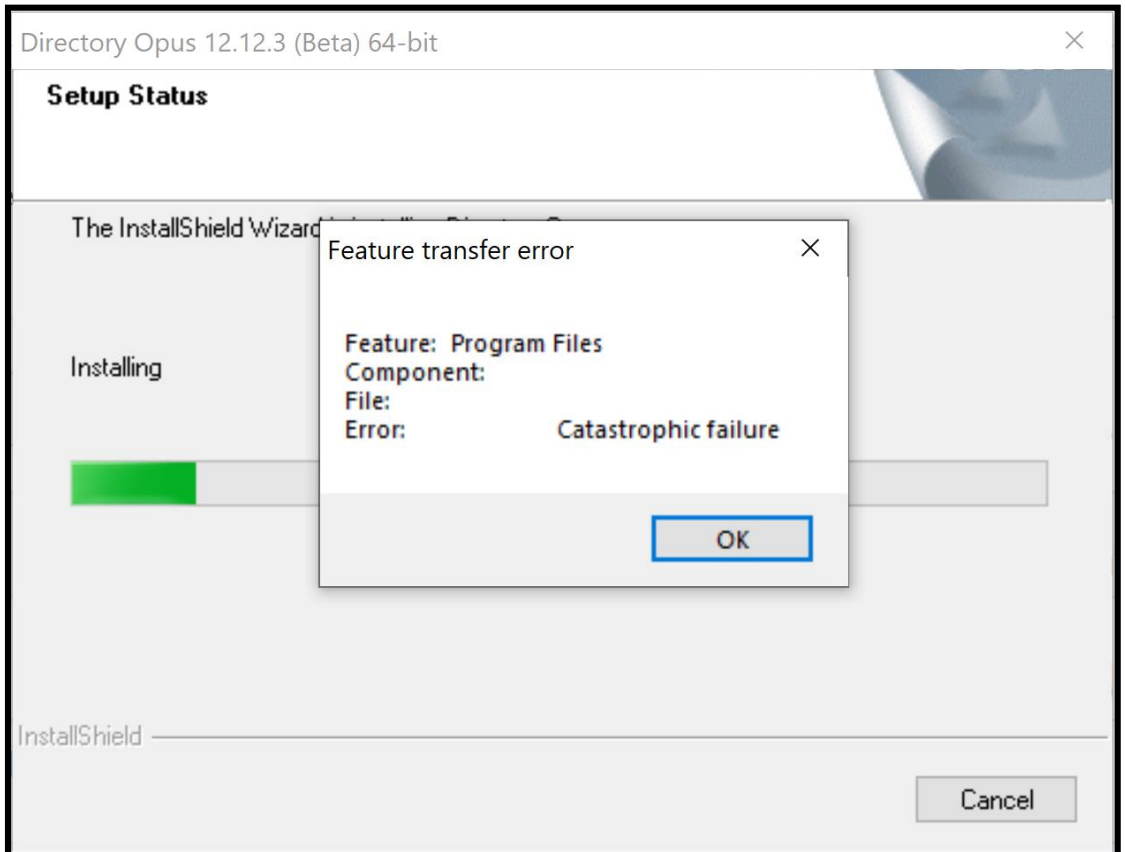
- The Recovery Point Objective
- The Recovery Time Objective

Disaster Recovery Planning

- Disaster recovery deals with **catastrophic** failures that are extremely unlikely to occur during the lifetime of a system.
- If they are reasonably expected failures, they fall under the auspices of traditional availability planning.


There was an unexpected error:
Catastrophic failure (0x8000FFFF)
Please close System Restore and try again.
OK

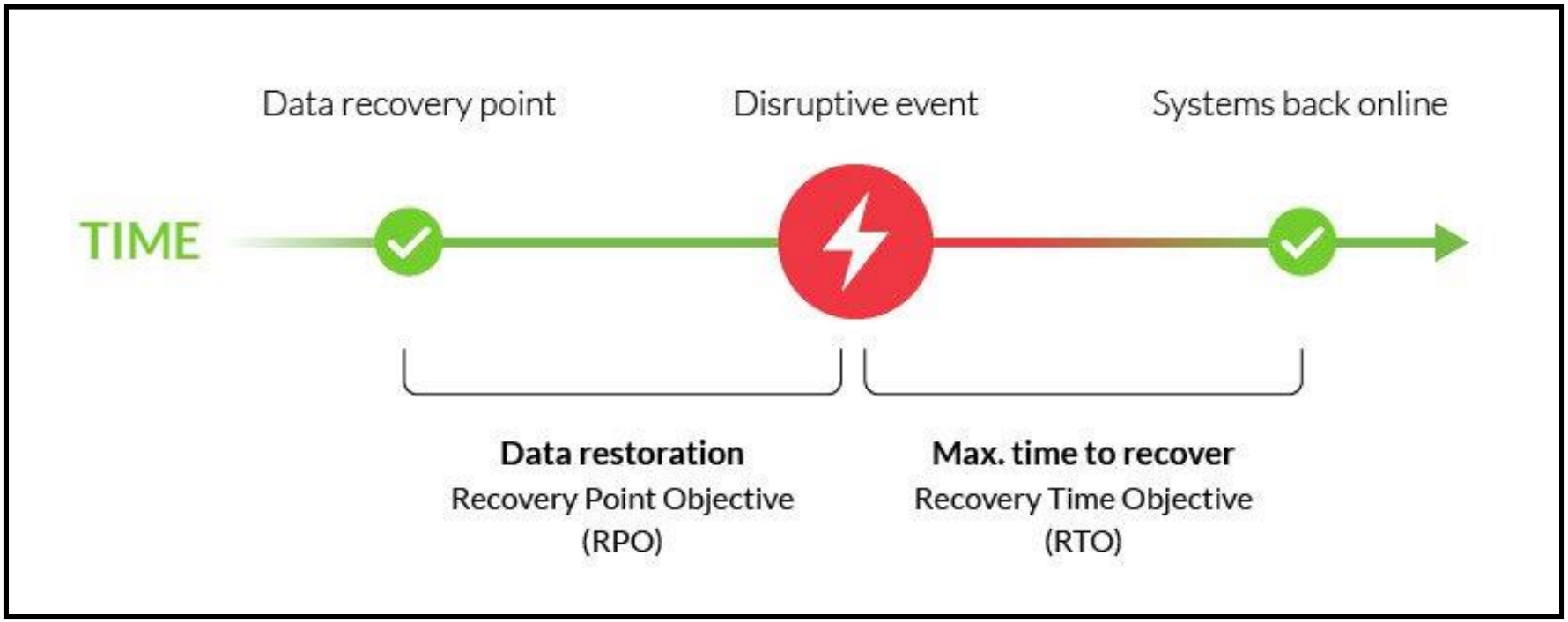

Backup Status and Configuration
The backup did not complete successfully.
An error occurred. The following information might help you resolve the error:
Catastrophic failure (0x8000FFFF)
yeah, not so much
Backup time: 1/15/2009 11:59 AM
Backup location: NEW (D:)
Try again
Adjust settings
Close


Directory Opus 12.12.3 (Beta) 64-bit
Setup Status
The InstallShield Wizard
Installing
Feature transfer error
Feature: Program Files
Component:
File:
Error: Catastrophic failure
OK
InstallShield
Cancel

Disaster Recovery Planning

- Defining a disaster recovery plan involves **two key metrics**:
- **Recovery Point Objective (RPO)** - The recovery point objective identifies **how much data** you are willing to **lose** in the event of a disaster. This value is typically specified in a number of hours or days of data.
  - For example, if you determine that it is OK to lose 24 hours of data, you must make sure that the backups you'll use for your disaster recovery plan are never more than 24 hours old.
- **Recovery Time Objective (RTO)** - The recovery time objective identifies **how much downtime** is **acceptable** in the event of a disaster.
  - If your RTO is 24 hours, you are saying that up to 24 hours may elapse between the point when your system first goes offline and the point at which you are fully operational again.

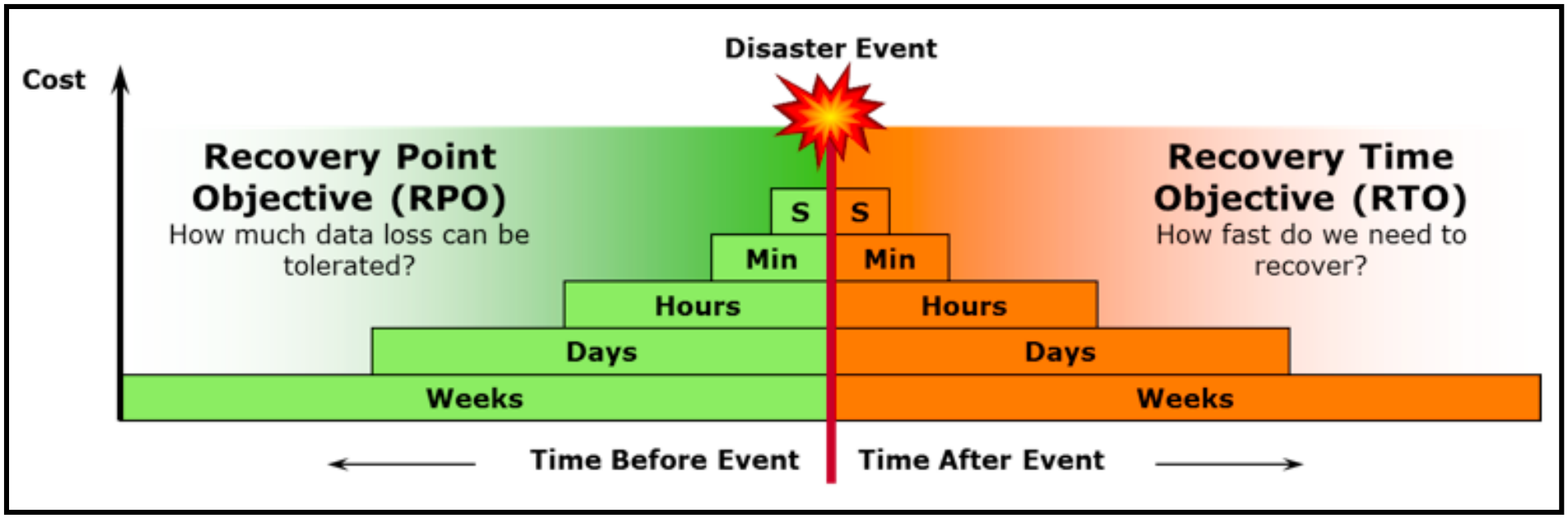
Disaster Recovery Planning



The Recovery Point Objective

- Your RPO is typically governed by the way in which you save and back up data:
  - **Weekly off-site** backups will survive the loss of your data center with a week of data loss. **Daily off-site** backups are even better.
  - **Daily on-site** backups will survive the loss of your production environment with a day of data loss plus replicating transactions during the recovery period after the loss of the system. **Hourly on-site** backups are even better.
  - A **NAS/SAN** will survive the loss of any individual server, except for instances of data corruption with no data loss.
  - A clustered database will survive the loss of any individual data storage device or database node with no data loss.
  - A clustered database across multiple data centers will survive the loss of any individual data center with no data loss.

RPO vs RTO


Disaster Event
Cost
Recovery Point Objective (RPO)
How much data loss can be tolerated?
Recovery Time Objective (RTO)
How fast do we need to recover?
S
S
Min
Min
Hours
Hours
Days
Days
Weeks
Weeks
Time Before Event
Time After Event

The Recovery Time Objective

- Having up-to-the-second off-site backups does you no good if you have no environment to which you can restore them in the event of failure.
- The ability to assemble a replacement infrastructure for your disasters—including the data restore time—governs the RTO.

The Recovery Time Objective

- What would happen if your managed services provider closed its doors tomorrow? If you have a number of dedicated servers, it can be days or weeks before you are operational again unless you have an agreement in place for a replacement infrastructure.

The Recovery Time Objective

- In a traditional infrastructure, a rapid RTO is very expensive. As discussed earlier, you would have to have an agreement in place with another managed services provider to provide either a backup infrastructure or an SLA for setting up a replacement infrastructure in the event your provider goes out of business.
- Depending on the nature of that agreement, it can nearly double the costs of your IT infrastructure.
- The cloud—even over virtualized data centers—alters the way you look at your RTO.

Disasters in the Cloud

- Backup Management
  - Fixed data strategy
  - Configuration data strategy
  - Persistent data strategy (aka database backups)
  - Backup security
- Geographic Redundancy
  - Spanning availability zones
  - Operating across regions
    - DNS management
    - Database management
    - Regulatory Issues
- Organizational Redundancy

Disasters in the Cloud

- Assuming unlimited budget and capabilities, we focus on three key things in disaster recovery planning:
  - Backups and data retention
  - Geographic redundancy
  - Organizational redundancy

Backup Management

- Your ability to recover from a disaster is limited by the quality and frequency of your backups.
- In a traditional IT infrastructure, companies often make full weekly backups to tape with nightly differentials and then ship the weekly backups off-site.
- You can do much better in the cloud, and do it much more cheaply, through a layered backup strategy.
- In disaster recovery, persistent data is generally the data of greatest concern.
- We can always rebuild the operating system, install all the software, and reconfigure it, but we have no way of manually rebuilding the persistent data.

Table: Backup requirements by data type

Kind of data	Description
Fixed data	Fixed data, such as your operating system and common utilities, belong in your AMI. In the cloud, you **don't back up** your AMI, because it has no value beyond the cloud.
Transient data	File caches and other data that can be lost completely without impacting the integrity of the system. Because your application state is not dependent on this data, **don't back it up**.
Configuration data	Runtime configuration data necessary to make the system operate properly in a specific context. This data is not transient, since it must survive machine restarts. On the other hand, it should be easily reconfigured from a clean application install. **This data should be backed up semi-regularly.**
Persistent data	Your application state, including critical customer data such as **purchase orders**. It changes constantly and a database engine is the best tool for managing it. Your database engine should store its state to a block device, and you should be performing constant backups. Clustering and/or replication are also critical tools in managing the database.

Fixed data strategy

- If you are fixated on the idea of backing up your machine images, you can download the images out of **S3** and store them outside of the Amazon cloud.
- If S3 were to go down and incur data loss or corruption that had an impact on your AMIs, you would be able to upload the images from your off-site backups and reregister them.
- It's not a bad idea and it is not a lot of trouble, but the utility is limited given the uniqueness of the failure scenario that would make you turn to those backups.

Configuration data strategy

- A good backup strategy for configuration information comprises two levels.
  - The first level can be either a regular filesystem dump to your cloud storage or a filesystem snapshot
  - The second approach is to check your application configuration into a source code repository outside of the cloud and leverage that repository for recovery from even minor losses

Configuration data strategy

- At some point, you do need to get that data out of the cloud so that you have off-site backups in a portable format. Here's what we recommend:
  - **Create regular**—at a minimum, daily—snapshots of your configuration data
  - **Create semi-regular**—at least less than your RPO—filesystem archives in the form of ZIP or TAR files and move those archives into Amazon S3
  - **On a semi-regular basis**—again, at least less than your RPO—copy your filesystem archives out of the Amazon cloud into another cloud or physical hosting facility

Persistent data strategy (aka database backups)

- It is recommended that using a relational database to store customer information and other persistent data.
- After all, the purpose of a relational database is to maintain the consistency of complex transactional data.
- The challenge of setting up database backups is doing them regularly in a manner that does not impact operations while retaining database integrity.
- MySQL, like all database engines, provides several convenient tools for backups, but you must use them carefully to avoid data corruption.

Persistent data strategy (aka database backups)

- The first line of defense is either **multimaster replication or clustering**.
  - A multimaster database is one in which two master servers execute write transactions independently and replicate the transactions to the other master. A clustered database environment contains multiple servers that act as a single logical server. Under both scenarios, when one goes down, the system remains operational and consistent.
- Instead, you can perform **master-slave replication**.
  - Master-slave replication involves setting up a master server that handles your write operations and replicating transactions over to a slave server. Each time something happens on the master, it replicates to the slave.

Persistent data strategy (aka database backups)

- Replication in itself is not a “first line of defense,” since replication is not atomic with respect to the transactions that take place on the master. In other words, a master can crash after a transaction has completed but before it has had time to replicate to the slave. To get around this problem, generally do the following:
  - Set up a master with its data files stored on a block storage device.
  - Set up a replication slave, storing its data files on a block storage device.
  - Take regular snapshots of the master block storage device based on my RPO.
  - Create regular database dumps of the slave database and store them in S3.
  - Copy the database dumps on a semi-regular basis from S3 to a location outside the Amazon cloud.

Persistent data strategy (aka database backups)

- Snapshots are available in most cloud environments and provide an important approach for maintaining database integrity without completely shutting down application processing—even with large data sets in databases such as MySQL.
- You need to freeze the database only for an instant to create your snapshot. The process follows these steps:
  - Lock the database.
  - Sync the filesystem (this procedure is filesystem-dependent).
  - Take a snapshot
  - Unlock the database.
- The whole process should take about one second.

Persistent data strategy (aka database backups)

- On Amazon EC2, you will store your snapshots directly onto block storage. Unfortunately, the snapshots are not portable, so you can't use them for off-site storage. You therefore will need to do database dumps.
- The steps for creating the database dump are:
  - Execute the database dump
  - When complete, encrypt the dump and break it into small, manageable chunks
  - Move the dump over to S3
- Amazon S3 limits your file size to 5 GB. As a result, you probably need to break your database into chunks, and you should definitely encrypt it and anything else you send into Amazon S3.
- Now that you have a portable backup of your database server, you can copy that backup out of the Amazon cloud and be protected from the loss of your S3 backups.

Backup security

- Your filesystems are encrypted to protect the snapshots you are making for your backups from prying eyes. The harder part is securing your portable backups as you store them in S3 and move them off site.
- We typically use PGP-compatible encryption for my portable backups. You need to worry about two issues:
  - Keeping your private decryption key out of the cloud.
  - Keeping your private decryption key some place that it will never, ever get lost.

Backup security

- You really have no reason for ever giving out the private decryption key to an instance in the cloud unless you are automating the failover between two different cloud infrastructures. The cloud needs only your public encryption key so it can encrypt the portable backups.
- You can’t store your decryption key with your backups. Doing so will defeat the purpose of encrypting the backups in the first place. Because you will store your decryption key somewhere else, you run the risk of losing your decryption key independent of your backups.
- On the other hand, keeping a bunch of copies of your decryption key will make it more likely it will fall into the wrong hands.

Backup security

- The best approach? Keep two copies:
  - One copy stored securely on a highly secure server in your internal network.
  - One copy printed out on a piece of paper and stored in a safety deposit box nowhere near the same building in which you house your highly secure server.
- More than one person should know the locations of these copies. A true disaster can unfortunately result in the loss of personnel, so personnel redundancy is also important for a disaster recovery plan.
- If you are automating the recovery from portable backups, you will also need to keep a copy of the private decryption key on the server that orchestrates your automated recovery efforts.

Geographic Redundancy

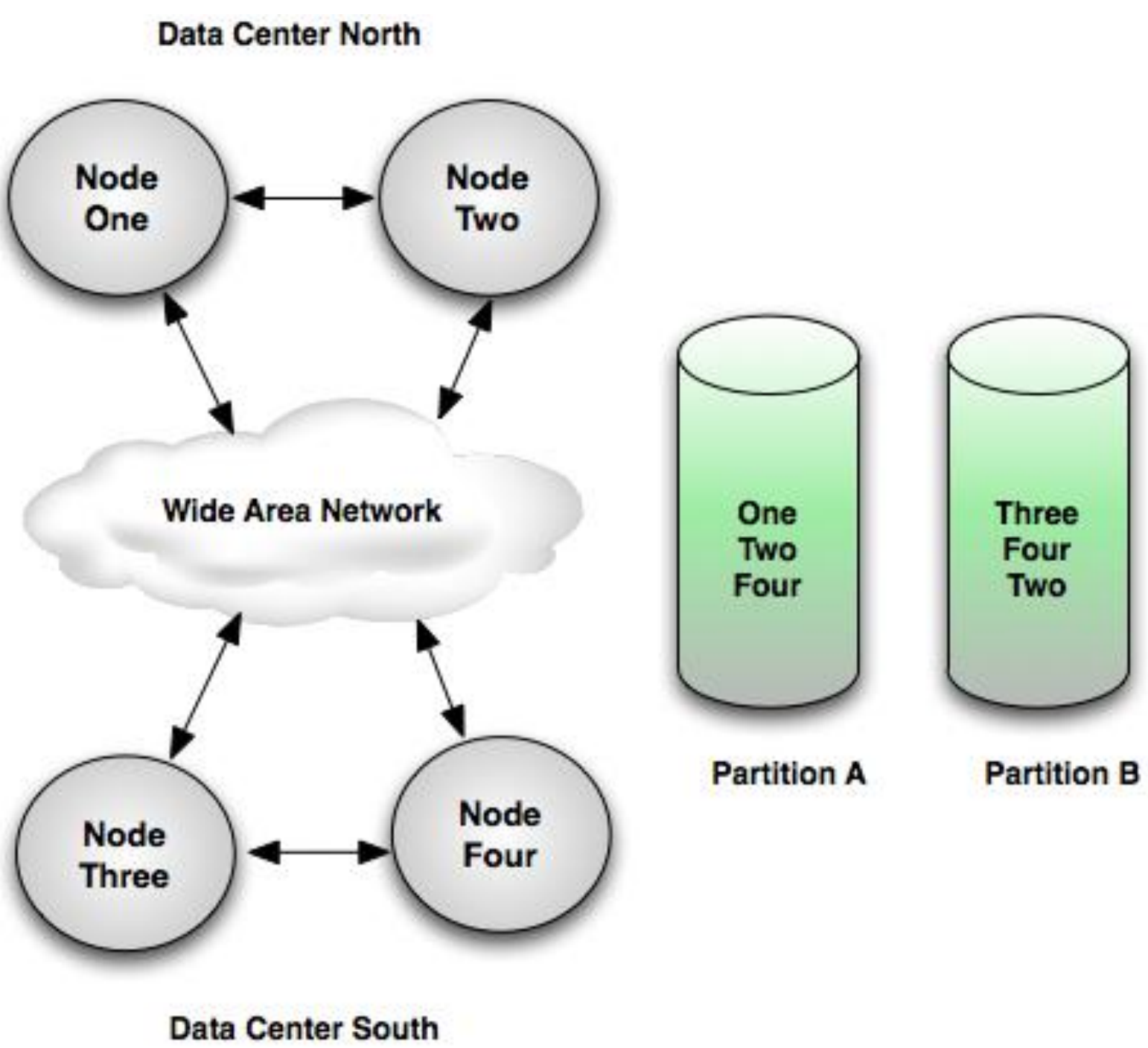
- Turning now to your Recovery Time Objective, the key is redundancy in infrastructure.
- If you can develop geographical redundancy, you can survive just about any physical disaster that might happen. With a physical infrastructure, geographical redundancy is expensive. In the cloud, however, it is relatively cheap.

Geographic Redundancy

- You don’t necessarily need to have your application running actively in all locations, but you need the ability to bring your application up from the redundant location in a state that meets your Recovery Point Objective within a timeframe that meets your Recovery Time Objective.
- If you have a 2-hour RTO with a 24-hour RPO, geographical redundancy means that your second location can be operational within two hours of the complete loss of your primary location using data that is no older than 24 hours.

Geographic Redundancy

- Amazon provides built-in geographic redundancy in the form of regions and availability zones.
- If you have your instances running in a given availability zone, you can get them started back up in another availability zone in the same region without any effort. If you have specific requirements around what constitutes geographic redundancy,‡ Amazon's availability zones may not be enough—you may have to span regions.


Data Center North
Node One
Node Two
Wide Area Network
Node Three
Node Four
Data Center South
One Two Four
Three Four Two
Partition A
Partition B

Spanning availability zones

- Just about everything in your Amazon infrastructure except block storage devices is available across all availability zones in a given region.
- Although there is a charge for network traffic that crosses availability zones, that charge is generally well worth the price for the leveraging ability to create redundancy across availability zones.

Spanning availability zones

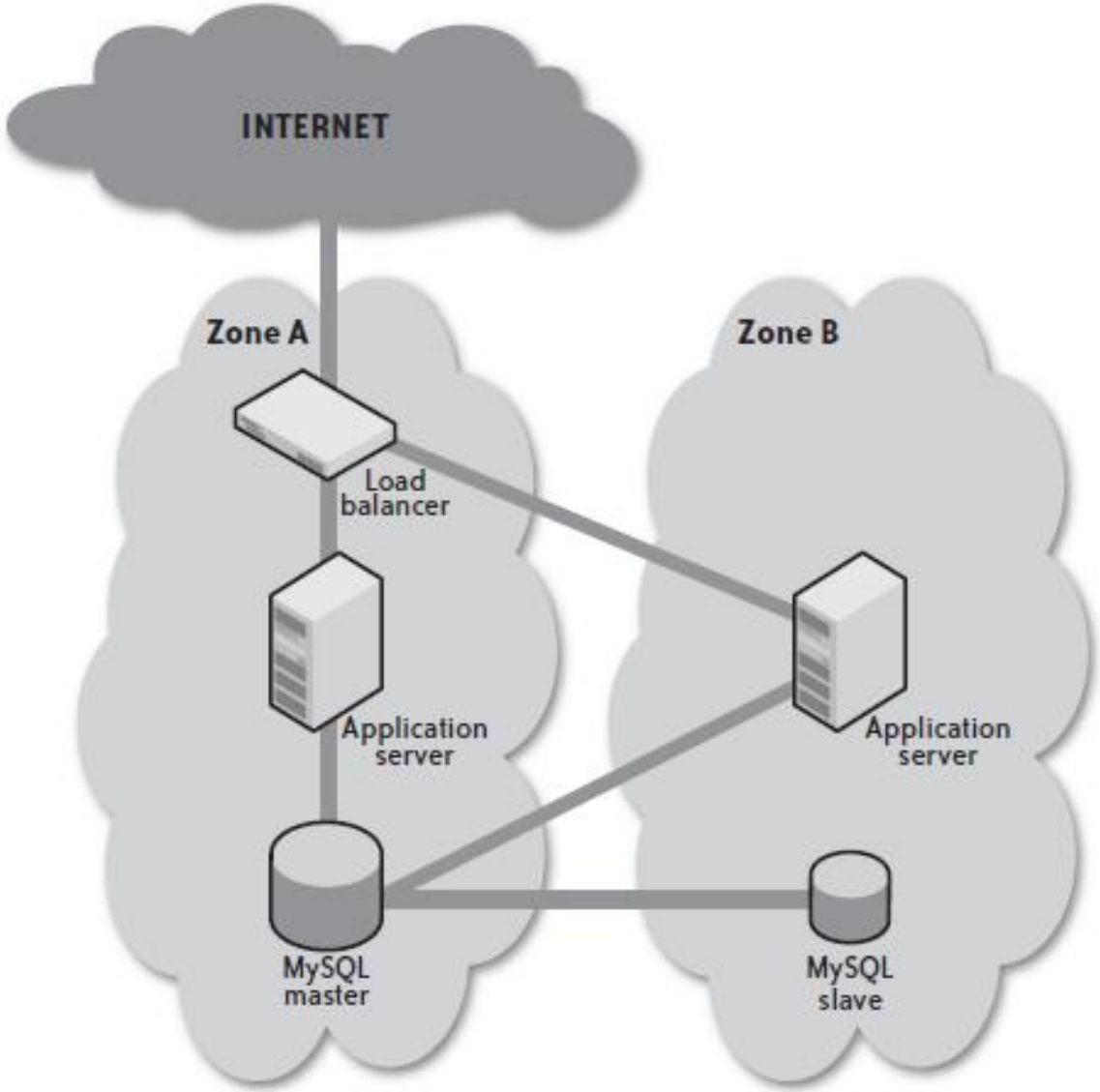


FIGURE: By spanning multiple availability zones, you can achieve geographic redundancy

Spanning availability zones

- If you lose the entire availability zone B, nothing happens. The application continues to operate normally, although perhaps with degraded performance levels.
- If you lose availability zone A, you will need to bring up a new load balancer in availability zone B and promote the slave in that availability zone to master. The system can return to operation in a few minutes with little or no data loss. If the database server were clustered and you had a spare load balancer running silently in the background, you could reassign the IP address from the old load balancer to the spare and see only a few seconds of downtime with no data loss.

Spanning availability zones

- The Amazon SLA provides for a 99.95% uptime of at least two availability zones in each region. If you span multiple availability zones, you can actually exceed the Amazon SLA in regions that have more than two availability zones. The U.S. East Coast, for example, has three availability zones. As a result, you have only a 33% chance of any given failure of two availability zones being exactly the two zones you are using.
- Even in the event that you are unfortunate enough to be operating in exactly the two zones that fail, you can still exceed Amazon's SLA as long as the region you are operating in has more than two availability zones. The trick is to execute your disaster recovery procedures and bring your infrastructure back up in the remaining availability zone. As a result, you can be operational again while the other two availability zones are still down.

Operating across regions

- Amazon supports **two regions**: **us-east-1** (Eastern United States) and **eu-west-1** (Western Europe). These regions share little or no meaningful infrastructure.
- The **advantage** of this structure is that your application can basically **survive** a **nuclear attack on the U.S**. or EU (but not on both!) if you operate across regions.
- On the other hand, **the lack of common infrastructure** makes the task of replicating your environments across regions more difficult.

Operating across regions

- Each region has its own associated Amazon S3 region. Therefore, you cannot launch EC2 instances in the EU using AMIs from the U.S., and you cannot use IP addresses formerly associated with a load balancer in the EU with a replacement in the U.S.
- How you manage operations across regions depends on the nature of your web application and your redundancy needs. It's entirely likely that just having the capability to rapidly launch in another region is good enough, without actually developing an infrastructure that simultaneously operates in both regions.
- The issues you need to consider for simultaneous operation include:

DNS management

- DNS management software is computer software that **controls** Domain Name System (**DNS**) **server clusters**. DNS data is typically deployed on multiple physical servers.
- The main purposes of DNS management software are: **to reduce human error** when editing complex and repetitive DNS data.
- You can use **round-robin DNS** to work around the fact that **IP addresses are not portable across regions**, but you will end up sending European visitors to the U.S. and vice versa (very inefficient network traffic management) and lose half your traffic when one of the regions goes down.

DNS Management

Point your domain to a web site by pointing to an IP Address, or forward to another site, or point to a temporary page (known as Parking), and more. These records are also known as sub-domains.

Host Name	Record Type	Address	Priority
*	A (Address)	xxx.xxx.xxx.xxx	N/A
mail	MXE (Mail Easy)	xxx.xxx.xxx.xxx	N/A
2	A (Address)	2	N/A
	A (Address)		

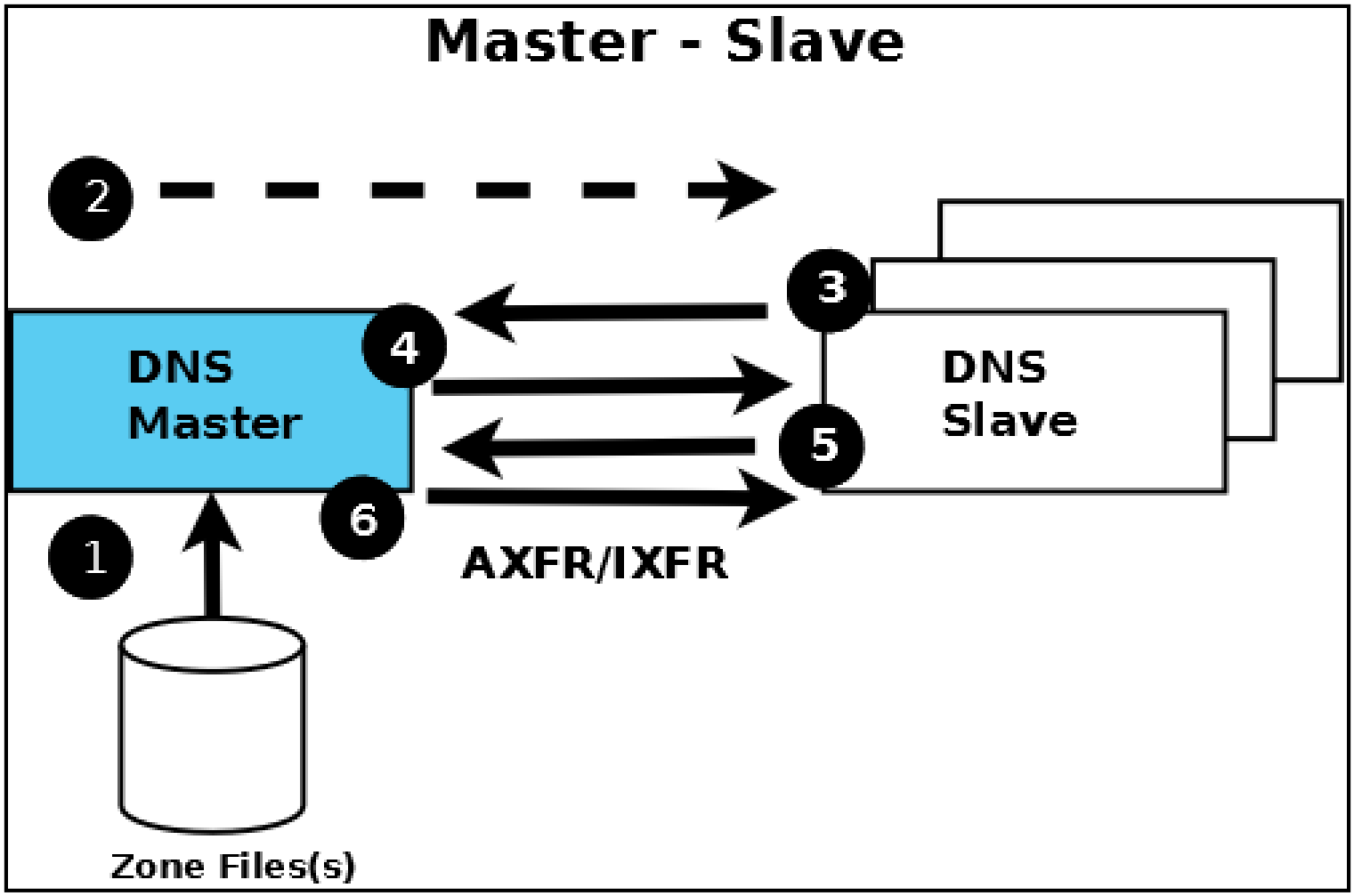
* Priority Record for MX Only

Save Changes | Cancel Changes

Database management

- Clustering across regions is likely not practical (but you can try it). You can also **set up a master in one region with a slave in the other.** Then you perform **write operations** against the master, but read against the slave for traffic from the region with the slave.
- Another option is to segment your database so that the European region has “European data” and the U.S. region has “American data.” Each region also has a slave in the other region to act as a recovery point from the full loss of a region.

Database management



Regulatory issues

- The EU does not allow the storage of certain data outside of the EU. As a result, legally you may not be allowed to operate across regions, no matter what clever technical solutions you devise.
- In reality, an Amazon+GoGrid or Amazon+Rackspace approach to redundancy may be more effective than trying to use Amazon's two cross-jurisdictional regions.

Regulatory issues

Organizational Redundancy

- Physical disasters are a relatively rare thing, but companies go out of business everywhere every day—even big companies like Amazon and Rackspace.
- Even if a company goes into bankruptcy restructuring, there's no telling what will happen to the hardware assets that run their cloud infrastructure.
- Your disaster recovery plan should therefore have contingencies that assume your cloud provider simply disappears from the face of the earth.

Organizational Redundancy

- You probably won't run concurrent environments across multiple clouds unless it provides some level of geographic advantage.
- Even in that case, your environments are not likely to be redundant so much as segmented for the geographies they are serving.
- Instead, the best approach to organizational redundancy is to identify another cloud provider and establish a backup environment with that provider in the event your first provider fails.

Organizational Redundancy

- The issues associated with organizational redundancy are similar to the issues I discussed earlier around operating across Amazon EC2 regions. In particular, you must consider all of the following concerns:
  - Storing your portable backups at your secondary cloud provider.
  - Creating machine images that can operate your applications in the secondary provider's virtualized environment.
  - Keeping the machine images up to date with respect to their counterparts with the primary provider.
  - Not all cloud providers and managed service providers support the same operation systems or filesystems. If your application is dependent on either, you need to make sure you select a cloud provider that can support your needs.

Disaster Management

- Monitoring
- Load Balancer Recovery
- Application Server Recovery
- Database Recovery

Disaster Management

- You are performing your backups and have an infrastructure in place with all of the appropriate redundancies.
- **To complete the disaster recovery scenario, you need to recognize when a disaster has happened and have the tools and processes in place to execute your recovery plan.**
- One of the coolest things about the cloud is that all of this can be automated. You can recover from the loss of Amazon's U.S. data centers while you sleep.

Disaster Management

Monitoring

- Monitoring your cloud infrastructure is extremely important. You cannot replace a failing server or execute your disaster recovery plan if you don't know that there has been a failure.
- The trick is that your monitoring systems cannot live in either your primary or secondary cloud provider's infrastructure.
- They must be independent of your clouds. If you want to enable automated disaster recovery, they also need the ability to manage your EC2 infrastructure from the monitoring site.

Monitoring

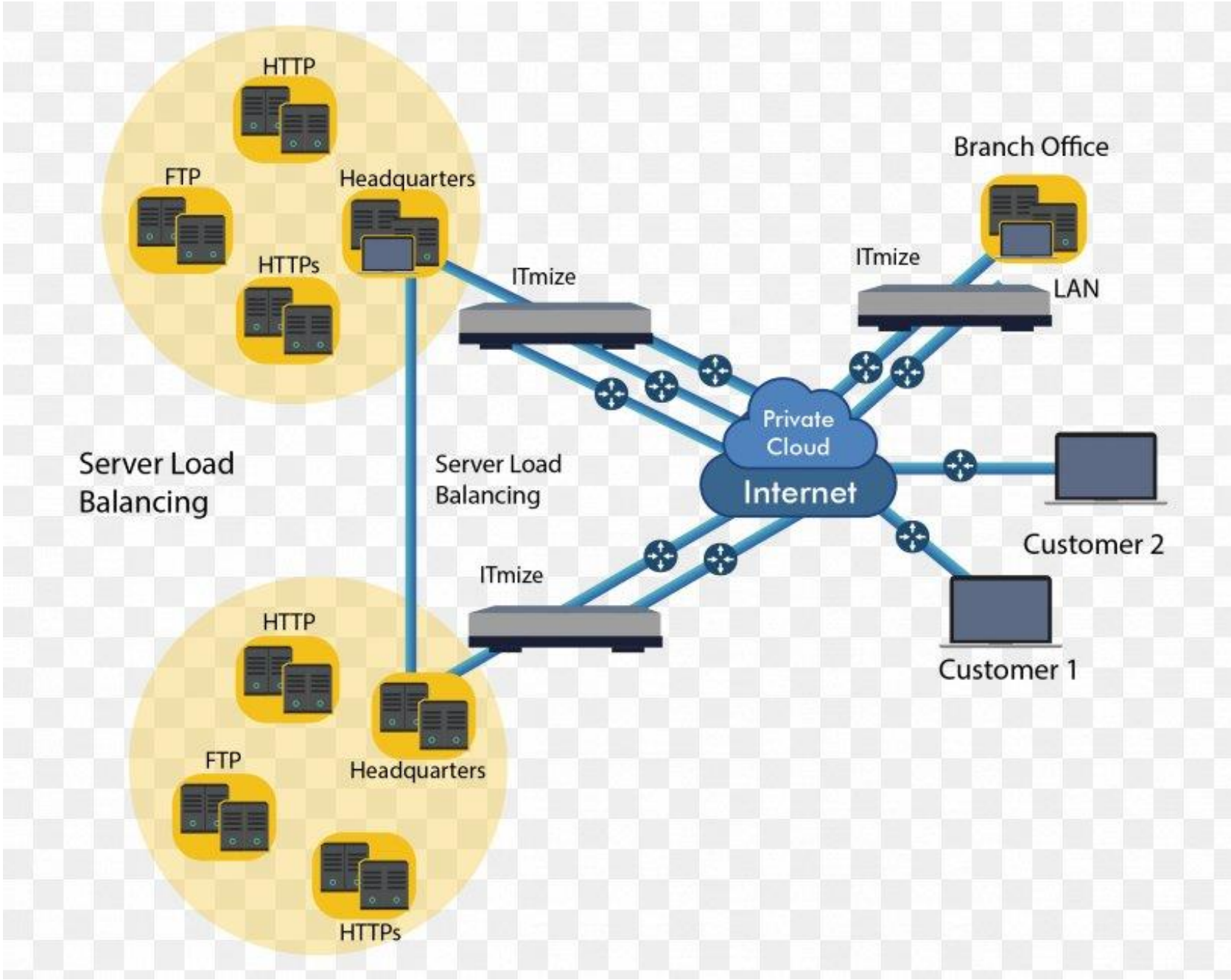
- There are many other more common things that you should check on in a regular environment. In particular, you should be checking capacity issues such as disk usage, RAM, and CPU.
- In the end you will need to monitor for failure at three levels:
  - Through the provisioning API (for Amazon, the EC2 web services API)
  - Through your own instance state monitoring tools
  - Through your application health monitoring tools

Load Balancer Recovery

- One of the reasons companies pay absurd amounts of money for physical load balancers is to greatly reduce the likelihood of load balancer failure.
- With cloud vendors such as GoGrid—and in the future, Amazon—you can realize the benefits of hardware load balancers without incurring the costs.
- Under the current AWS offering, you have to use less-reliable EC2instances. Recovering a load balancer in the cloud is lightning fast.
- As a result, the downside of a failure in your cloud-based load balancer is minor.

Load Balancer Recovery

- Recovering a load balancer is simply a matter of **launching a new load balancer instance** from the AMI and notifying it of the IP addresses of its application servers.
- You can further reduce any downtime by keeping a load balancer running in an alternative availability zone and then remapping your static IP address upon the failure of the main load balancer.


HTTP
FTP
Headquarters
HTTPs
Branch Office
ITmize
ITmize
LAN
Server Load
Balancing
Server Load
Balancing
Private
Cloud
Internet
Customer 2
ITmize
HTTP
FTP
Headquarters
Customer 1
HTTPs

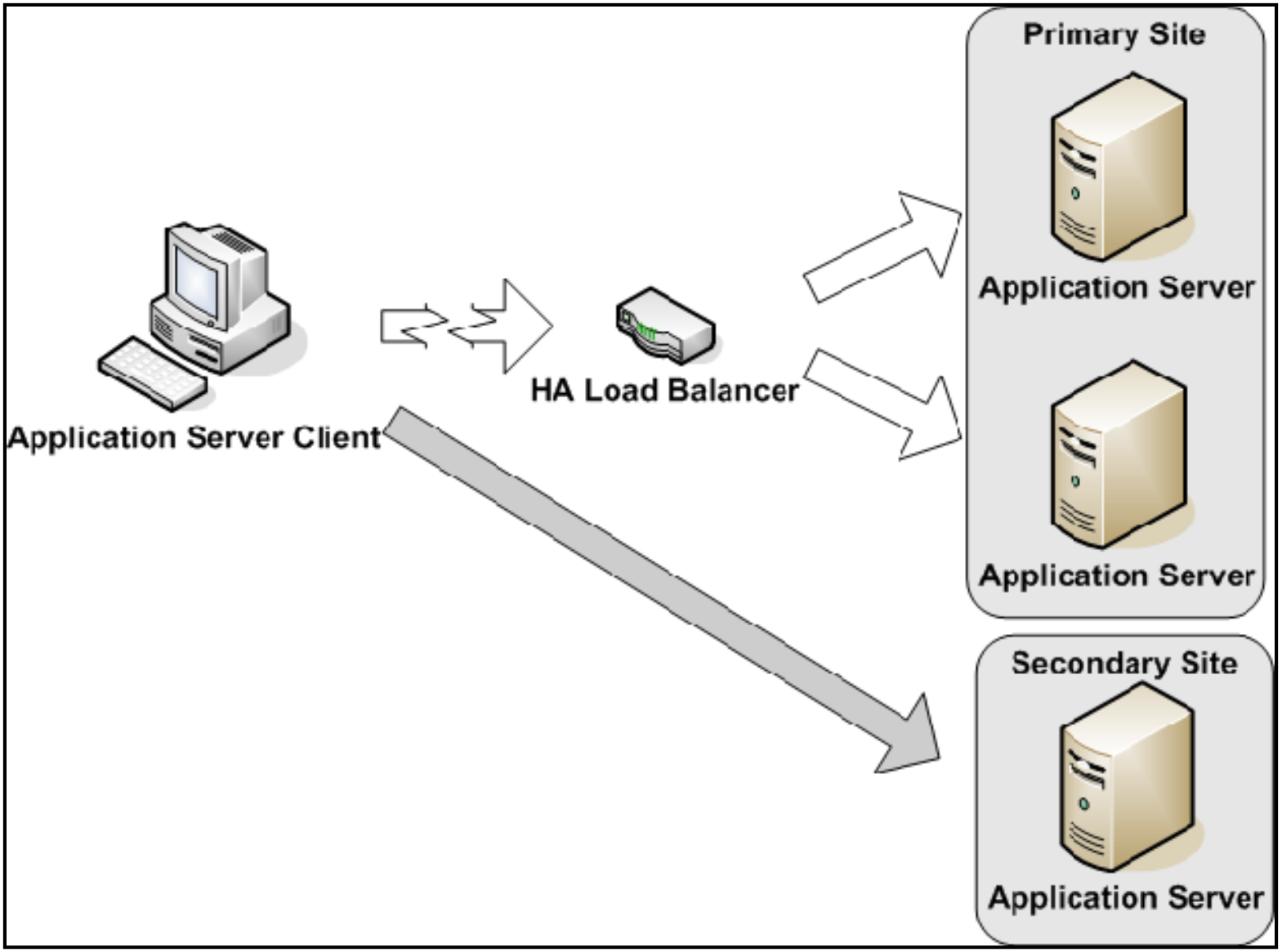
Application Server Recovery

- If you are operating multiple application servers in multiple availability zones, your system as a whole will survive the failure of any one instance—or even an entire availability zone.
- You will still need to recover that server so that future failures don’t affect your infrastructure.

Application Server Recovery

- The recovery of a **failed application server** is only slightly **more complex** than the recovery of a **failed load balancer**.
- Like the failed load balancer, you start up a new instance from the application server machine image. You then pass it configuration information, including where the database is. Once the server is operational, you must notify the load balancer of the existence of the new server (as well as deactivate its knowledge of the old one) so that the new server enters the load-balancing rotation.

Application Server Recovery



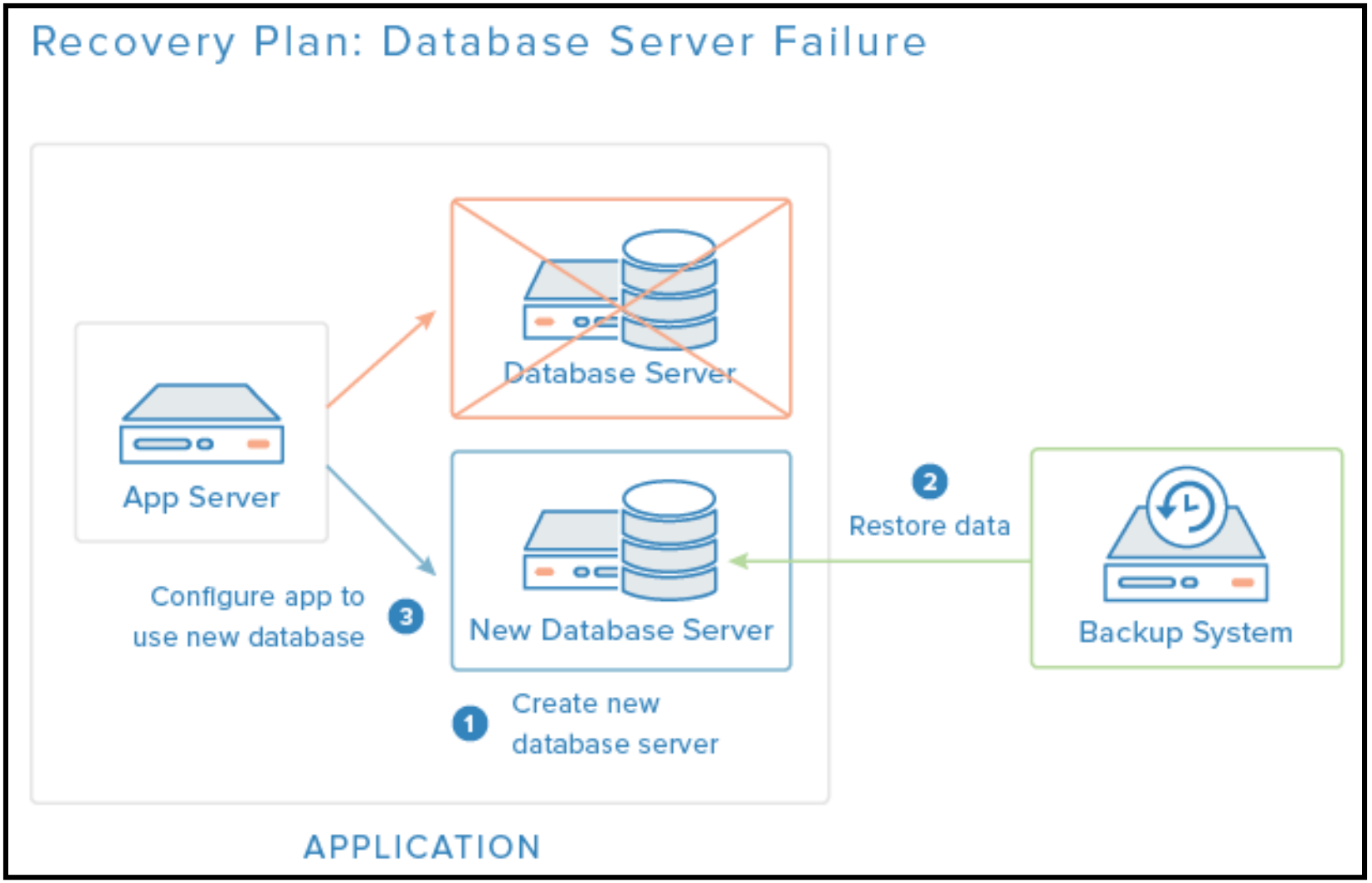
Database Recovery

- Database recovery is the hardest part of disaster recovery in the cloud. Your disaster recovery algorithm has to identify where an uncorrupted copy of the database exists.
- This process may involve promoting slaves into masters, rearranging your backup management, and reconfiguring application servers.

Database Recovery

- The best solution is a clustered database that can survive the loss of an individual database server without the need to execute a complex recovery procedure.
- Absent clustering, the best recovery plan is one that simply launches a new database instance and mounts the still functional EC2 volume formerly in use by the failed instance.

Database Recovery



Database Recovery

- When an instance goes down, however, any number of related issues may also have an impact on that strategy:
  - The database could be irreparably corrupted by whatever caused the instance to crash
  - The volume could have gone down with the instance.
  - The instance's availability zone (and thus the volume as well) could be unavailable
  - You could find yourself unable to launch new instances in the volume's availability zone

Database Recovery

- On the face of it, it might seem that the likelihood of both things going wrong is small, but it happens. As a result, you need a fallback plan for your recovery plan. The following process will typically cover all levels of database failure:
  1. Launch a replacement instance in the old instance's availability zone and mount its old volume.
  2. If the launch fails but the volume is still running, snapshot the volume and launch a new instance in any zone, and then create a volume in that zone based on the snapshot.
  3. If the volume from step 1 or the snapshot from step 2 are corrupt, you need to fall back to the replication slave and promote it to database master.
  4. If the database slave is not running or is somehow corrupted, the next step is to launch a replacement volume from the most recent database snapshot.
  5. If the snapshot is corrupt, go further back in time until you find a backup that is not corrupt.