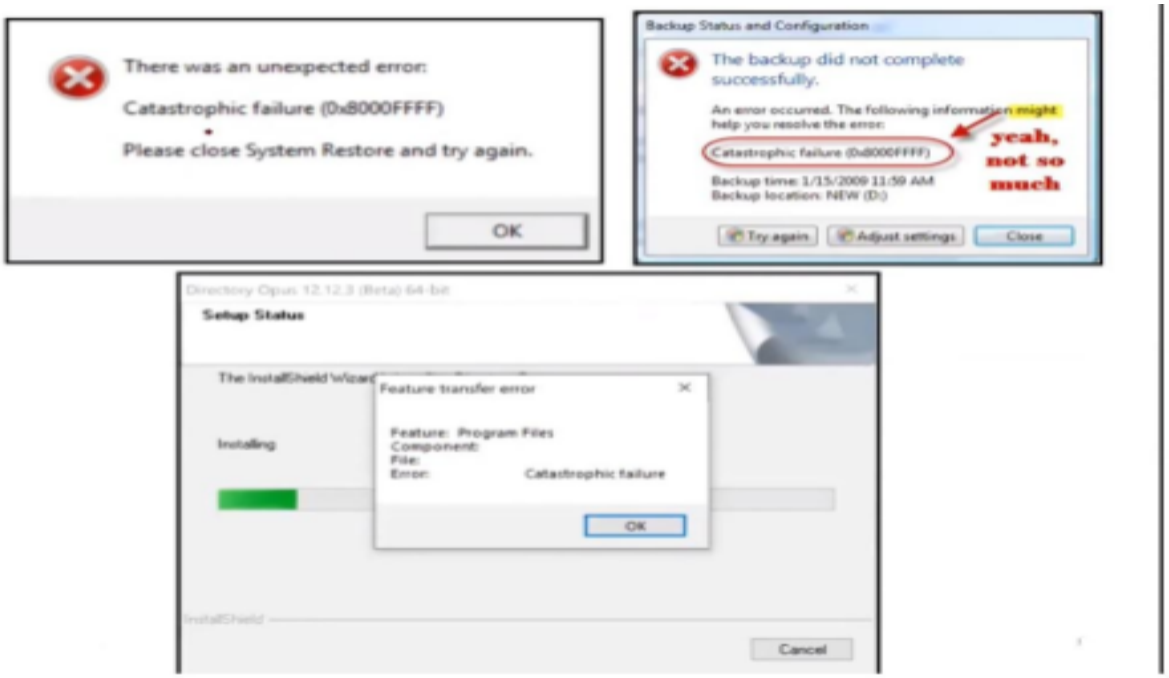
Disaster

Disaster Recovery

Disaster recovery is the practice of making a system capable of surviving unexpected or extraordinary failures. A disaster recovery plan, for example, will help your IT systems survive a fire in your data center that destroys all of the servers in that data center and the systems they support. Every organization should have a documented disaster recovery process and should test that process at least twice each year

1. **Disaster Recovery Planning: Documented process**

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.



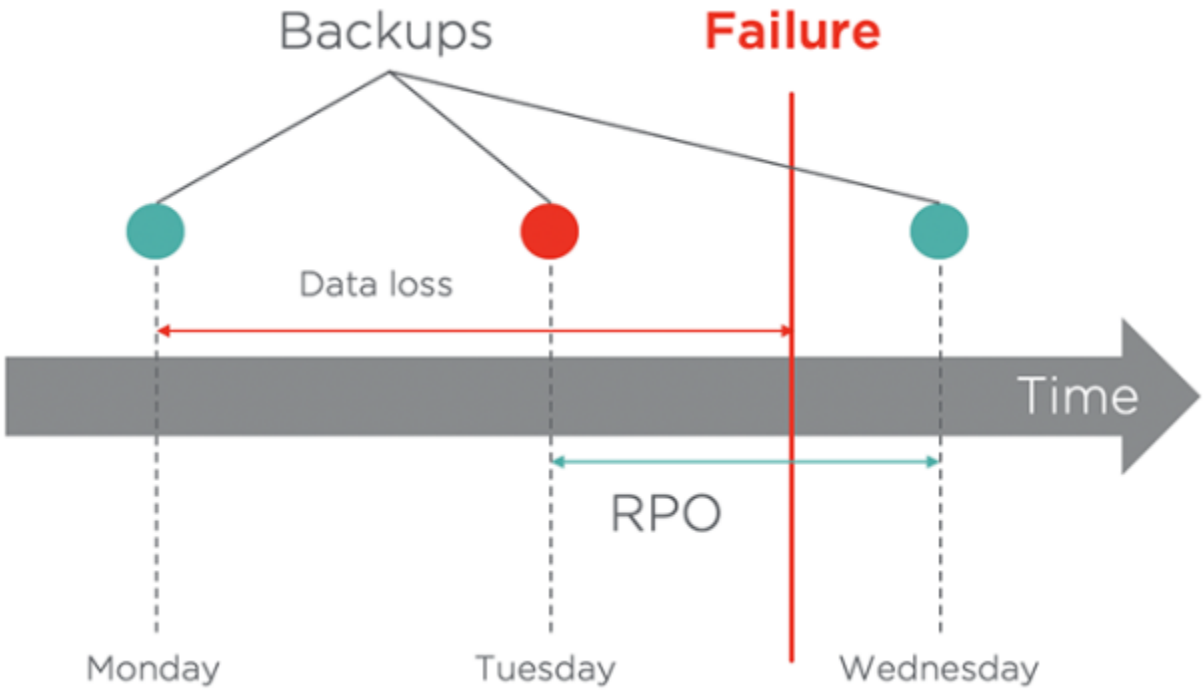
Although each single disaster is unexpected over the lifetime of a system, the possibility of some disaster occurring over time is reasonably nonzero.

"Disaster Recovery Planning is identifies an acceptable recovery state and develops processes and procedures to achieve the recovery state in the event of a disaster."

Defining a disaster recovery plan involves **two key metrics**:

Recovery Point Objective (RPO)

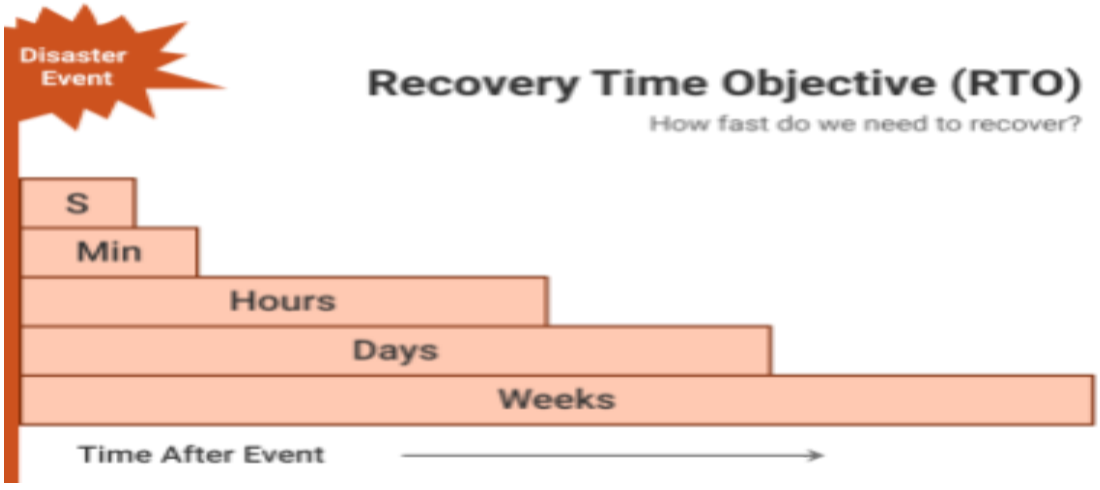
"The recovery point objective identifies how much data you are willing to lose in the event of a disaster**".** This value is typically specified in a number of hours or days of data. **For example**,



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.

Accomplishing that level of redundancy is expensive. It would also come with a nontrivial performance penalty. The cold reality for most businesses is likely that the cost of losing 24 hours of data is less than the cost of maintaining a zero downtime/zero loss of data infrastructure.

Determining an appropriate RPO and RTO is ultimately a financial calculation:

at what point does the cost of data loss and downtime exceed the cost of a backup strategy that will prevent that level of data loss and downtime? The right answer is radically different for different businesses.

The final element of disaster recovery planning understands the catastrophic scenario. A good disaster recovery plan can describe that scenario so that all stakeholders can understand and accept the risk.

1.1 Recovery Point Objective (RPO)

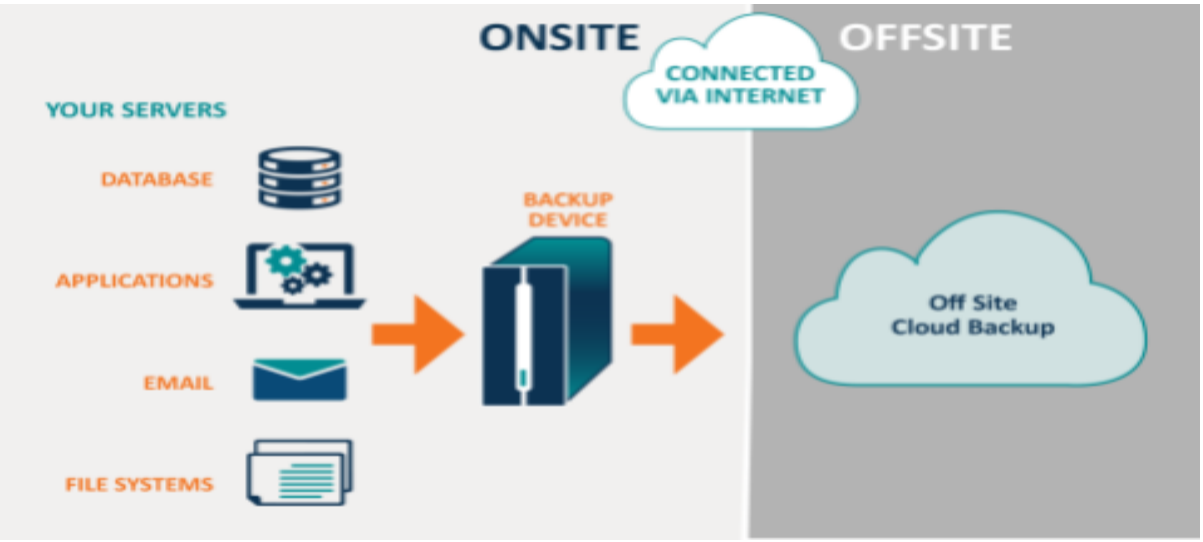
Any software system should be able to attain an RPO between 24 hours for a simple disaster to one week for a significant disaster without incurring absurd costs. Losing 24 hours of banking transactions would never be acceptable, much less one week.

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 (Network Attached System) /SAN (Storage Area Network) 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.



1.2 The Recovery Time Objective(RTO)

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

2.Disasters in the Cloud

Assuming unlimited budget and capabilities, three key things in disaster recovery planning:

1. Backups and data retention
2. Geographic redundancy
3. Organizational redundancy

It can take care of those three items, it's nearly certain meet most RPO and RTO nee

In addition, if you're hosting provider is a less-proven organizati organizational redundancy may be more important than geographic redundar Fortunately, the structure of the Amazon cloud makes it very easy to take care of

first and second items. In addition, cloud computing in general makes the third it much easier.

Backup Management

Now it's time to take a step back from the technical details and examine kinds of data you are planning to back up and how it all fits into your overall disas recovery plan.

Table 6-1 illustrates the different kinds of data that web applications typically manag

Table: Backup requirements by data type

Kind of data	Description
Fixed data	Fixed data, such as your operating system and common utilities in your AMI. In the cloud, you **don't back up** your AMI, becau no value beyond the cloud.
Transient data	File caches and other data that can be lost completely impacting the integrity of the system. Because your application not dependent on this data, **don't back it up**.
Configuration data	Runtime configuration data necessary to make the system properly in a specific context. This data is not transient, since survive machine restarts. On the other hand, it should b reconfigured from a clean application install. **This data sh backed up semi-regularly.**
Persistent data	Your application state, including critical customer data **purchase orders**. It changes constantly and a database engi best tool for managing it. Your database engine should store its a block device, and you should be performing constant Clustering and/or replication are also critical tools in mana database.

In disaster recovery, persistent data is generally the data of greatest concern. We always rebuild the operating system, install all the software, and reconfigure it, but have no way of manually rebuilding the persistent data.

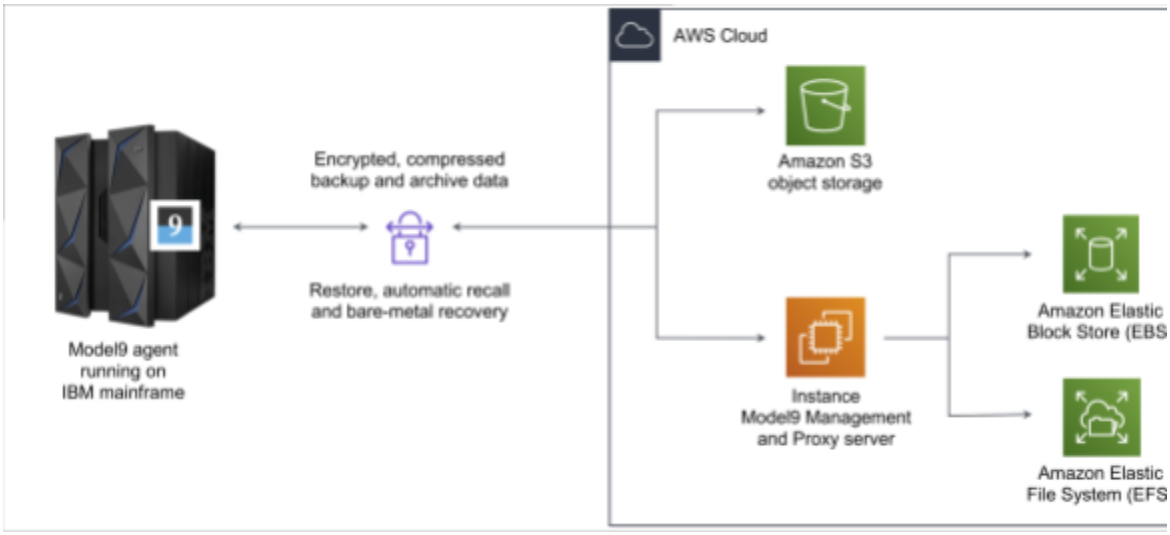
Fixed data strategy

If you are fixated on the idea of backing up your machine images, you download the images out of S3 and store them outside of the Amazon cloud. If S3 w to go down and incur data loss or corruption that had an impact on your AMIs, y would be able to upload the images from your off-site backups and reregister them

Configuration data strategy

A good backup strategy for configuration information comprises two levels. The f level can be either a regular file system dump to your cloud storage or a filesyst snapshot. For most applications, you can back up your configuration data once a day even once a week and be fine. You should, however, think back to your Recovery Po Objective. If your configuration data changes twice a day and you have a two-h RPO, you will need to back up your configuration data twice a day. If configurat data changes irregularly, it may be necessary to make hourly backups or specifically your backups to changes in application configuration.

An alternate approach is to check your application configuration into a source c repository outside of the cloud and leverage that repository for recovery from e minor losses. Whether you perform filesystem snapshots or simply zip up filesystem that data will hibernate inside S3. Snapshots tend to be the most effici and least intrusive mechanism for performing backups, but they are also the le portable



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 I 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 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)

Relational database to store customer information and other persistent d After all, the purpose of a relational database is to maintain the consistency of comp transactional data. MySQL, like all database engines, provides several convenient tc for backups, but you must use them carefully to avoid data corruption.

With the configuration data, it is highly unlikely you will be making a bacl in between the writing of two different files that must remain consistent or in middle of writing out a file to the filesystem. With database storage, it is a r certainty that every time you try to copy those files, the database will be in the mid of doing something with them. As a result, you need to get clever with your datab backup strategy.

The first line of defense is either multimaster replication or clustering multimaster database is one in which two master servers execute write transacti independently and replicate the transactions to the other master. A clustered datab environment contains multiple servers that act as a single logical server. Under b scenarios, when one goes down, the system remains operational and consistent.

Instead, you can perform master-slave replication. Master-slave replicat involves setting up a master server that handles your write operations and replicat transactions over to a slave server. Each time something happens on the maste replicates to the slave. If a master can crash after a transaction has completed before it has had time to replicate to the slave. To get around this problem, I 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 Amazon cloud.

Actually taking snapshots or creating database dumps for some datab engines is actually very tricky in a runtime environment, especially if you want to d hourly or even more frequently. The challenge in creating your backups for th database engines is the need to stop processing all transactions while the backu

taking place. To complicate the situation, database dumps can take a long time complete. As a result, your applications will grind to a halt while you make database dumps.

You need to freeze the database only for an instant to create your snapshot. process follows these steps:

1. Lock the database.
2. Sync the filesystem (this procedure is filesystem-dependent).
3. Take a snapshot.
4. Unlock the database.

The whole process should take about one second.

On Amazon EC2, you will store your snapshots directly onto block stora Unfortunately, the snapshots are not portable, so you can't use them for off-site stora You therefore will need to do database dumps, no matter how much you would rat avoid doing them. Because of this need, I run my backups against a database slave. slave can afford to be locked for a period of time while a database dump completes.

The steps for creating the database dump are:

1. Execute the database dump.
2. When complete, encrypt the dump and break it into small, manageable chunks.
3. Move the dump over to S3.

Amazon S3 limits your file size to 5 GB. As a result, you probably need to br your database into chunks, and you should definitely encrypt it and anything else send into Amazon S3. Now that you have a portable backup of your database ser you can copy that backup out of the Amazon cloud and be protected from the loss your S3 backups.

Backup security

The harder part is securing your portable backups as you store them in S3 and move them off site. I 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.

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.

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.

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

The virtualization technologies behind the cloud simply make it a lot easier to automate those processes and have a relatively inexpensive mechanism for off-site backups.

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.

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.

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.

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.

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

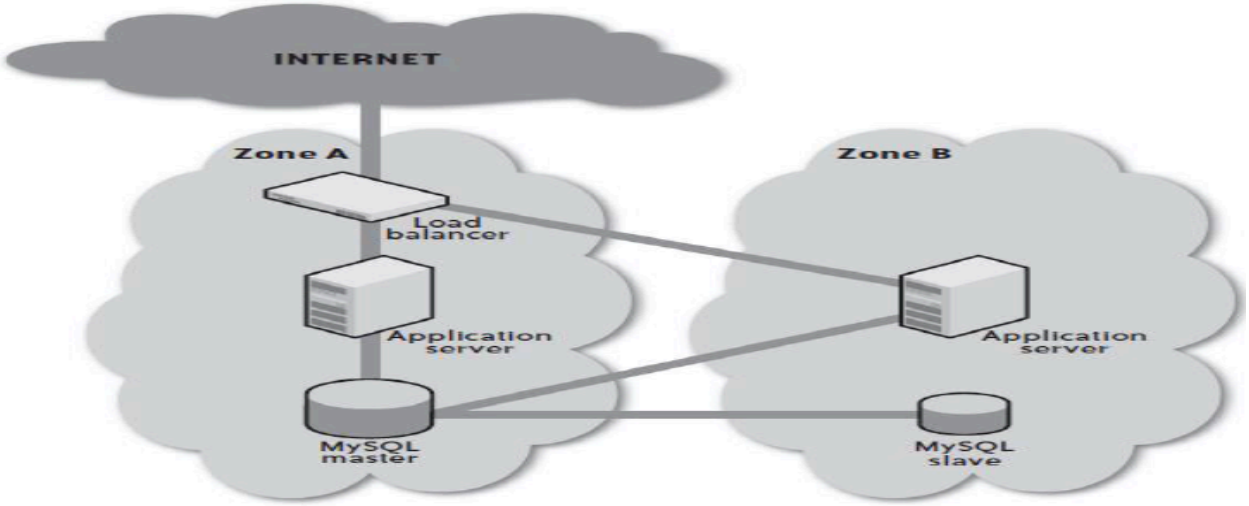
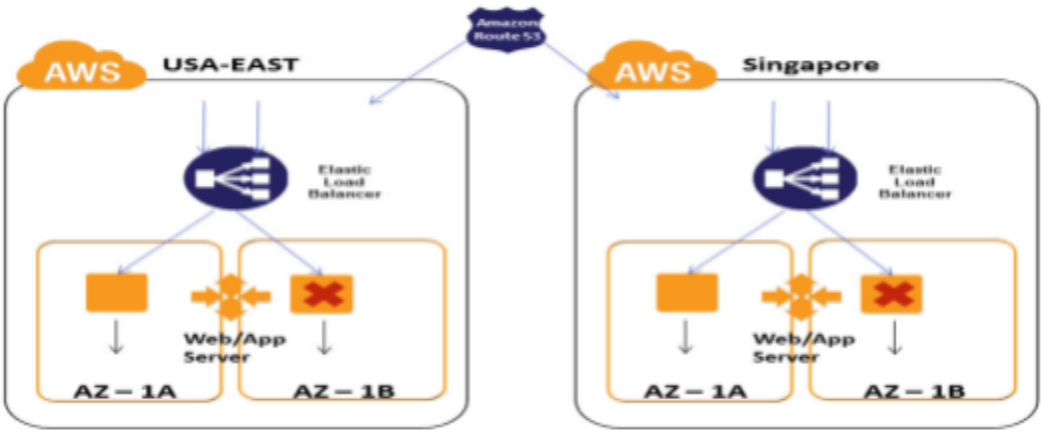


FIGURE . By spanning multiple availability zones, you can achieve geographic redundancy

.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.

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.
How you manage operations across regions depends on the nature of your web application and your redundancy needs.

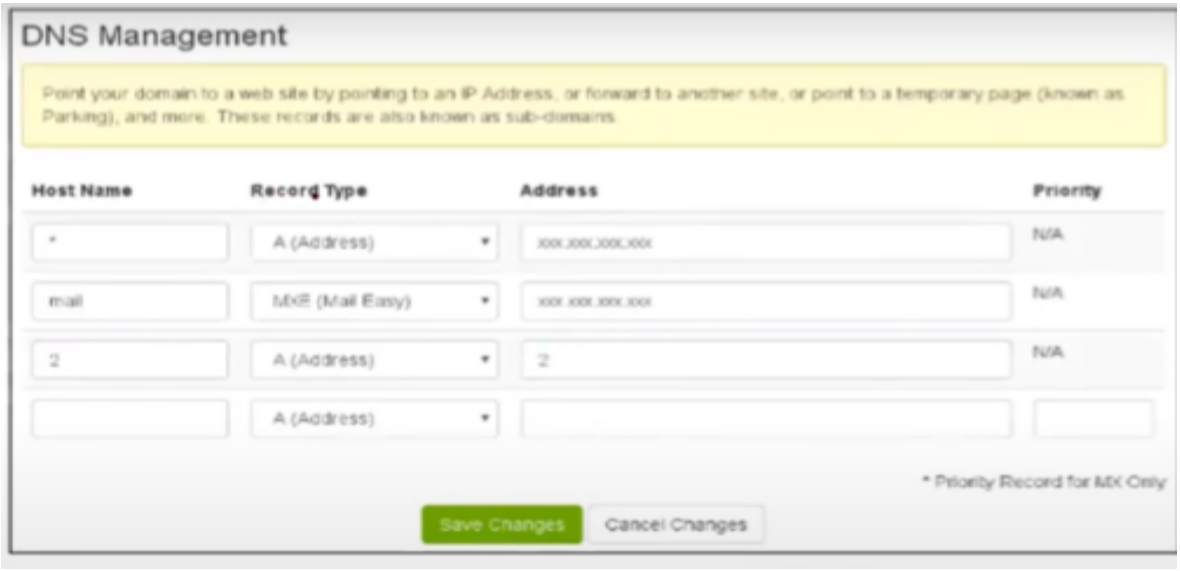


The issues you need to consider for simultaneous operation include:

DNS management

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

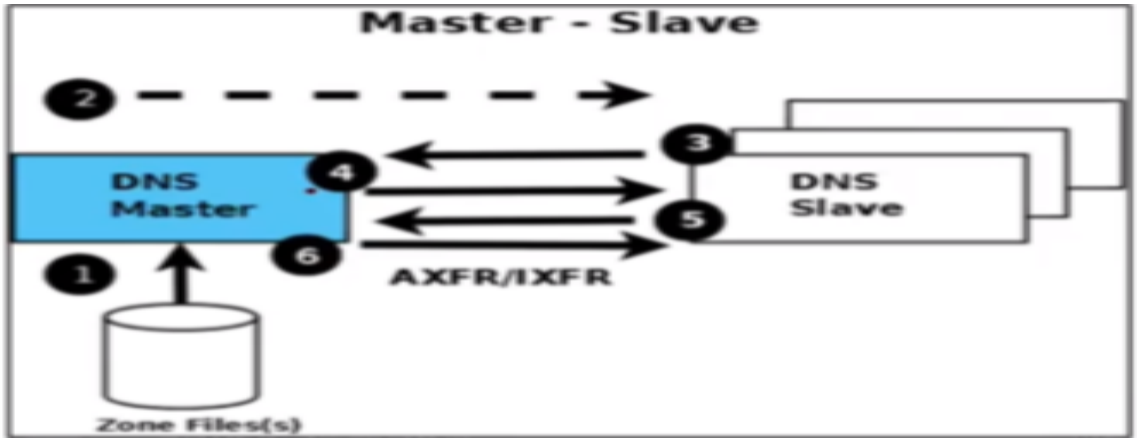
and lose half your traffic when one of the regions goes down. You can leverage a dynamic DNS system such as UltraDNS that will offer up the right DNS resolution based on source and availability.



Database management

Clustering across regions is likely not practical. 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.



Regulatory issues

The EU does not allow the storage of certain data outside of the EU. As a result, legally may not be allowed to operate across regions, no matter what clever technical solutions you devise

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.

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.

The issues associated with organizational redundancy are similar to 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 file systems. If your application is dependent on either, you need to make sure you select a cloud provider that can support your needs

3. Disaster Management

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.

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, however, is that your monitoring systems cannot live in either your primary or secondary cloud provider's infrastructure. They must be independent of your clouds. The primary monitoring objective should be to figure out what is going to fail before it actually fails.

There are many other more mundane 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, however, 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

Your cloud provider's provisioning API will tell you about the health of your instances, any volumes they are mounting, and the data centers in which they are operating. When you detect a failure at this level, it likely means something has gone

wrong with the cloud itself. Before engaging in any disaster recovery, you will need to determine whether the outage is limited to one server or affects indeterminate servers, impacting an entire availability zone or an entire region.

Monitoring is not simply about checking for disasters; mostly it is checking on the mundane. With enStratus, I put a Python service on each server that checks for a variety of server health indicators—mostly related to capacity management. The service will notify the monitoring system if there is a problem with the server or its configuration and allow the monitoring system to take appropriate action. It also checks for the health of the applications running on the instance.

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. Recovering a load balancer in the cloud, however, is lightning fast. As a result, the downside of a failure in your cloud-based load balancer is minor.

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.

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.

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

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.

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. 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 could be unavailable.
- You could find yourself unable to launch new instances in the volume's availability zone.

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 snapshots 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.

Step 4 typically represents your worst-case scenario. If you get to 5, there is something wrong with the way you are doing backups.