Highly Available Distributed Systems in AWS

FRANK WALSH
RICHARD FRISBY

Agenda

- Design for Failure
  - Redundancy
  - Multiple AZs
- Building for Scale
  - Scale up/out
  - Elasticity
- SOA and Loose Coupling
  - Decoupling components
- Case Study: Transcoding

Design for Failure

Design for Failure: Failure is Inevitable

"Did you try turning it off and on again?"

Roy - The IT Crowd

"Everything fails over time"

Werner Vogels – CTO Amazon

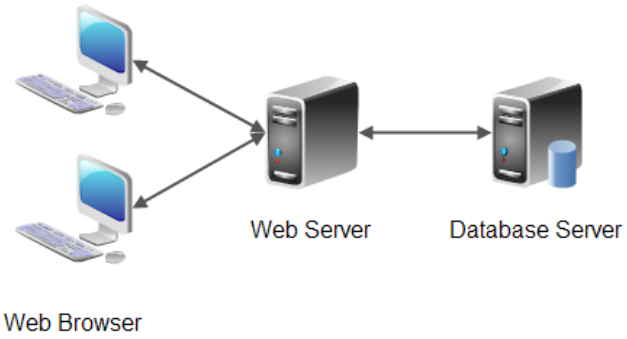
"I'm not a real programmer. I throw together things until it works then I move on. The real programmers will say 'yeah it works but you're leaking memory everywhere. Perhaps we should fix that.' I'll just restart Apache every 10 requests."

Rasmus Lerdorf - creator of PHP

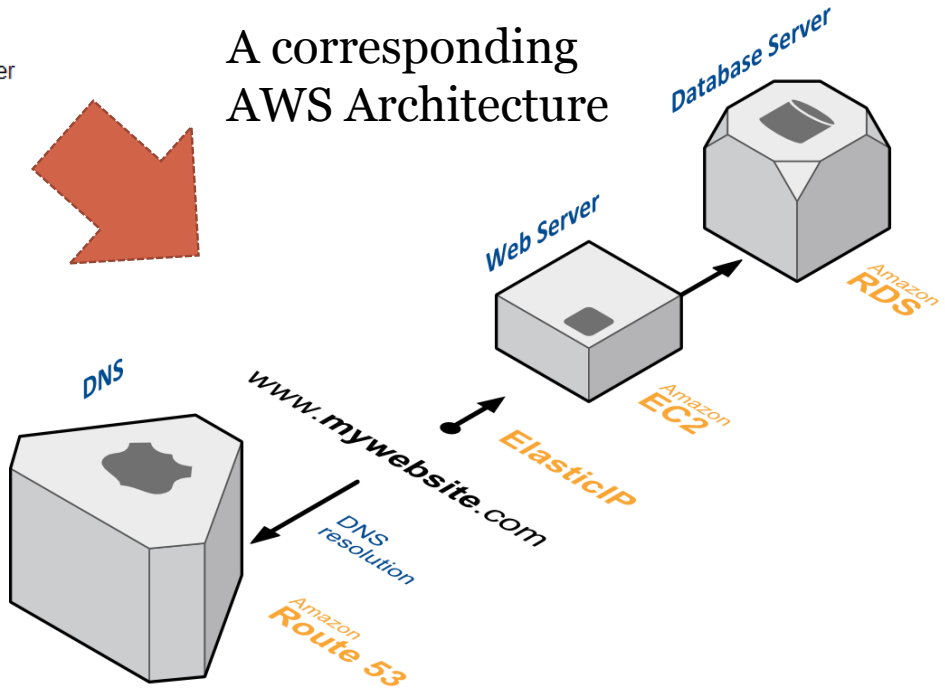
- Most software systems will degrade over time
  - Memory leaks, file fragmentation, hardware failure…
- Would be nice if applications could continue to function even if the underlying physical hardware fails, is removed or replaced.
  - should be impervious to reboots
  - Avoid a single point of failure(SPOF)
- Since everything can go wrong, the path to success is designing for failure.

Design for Failure: Traditional Client Server DS

Simple, generic 3-teir architecture



A corresponding AWS Architecture

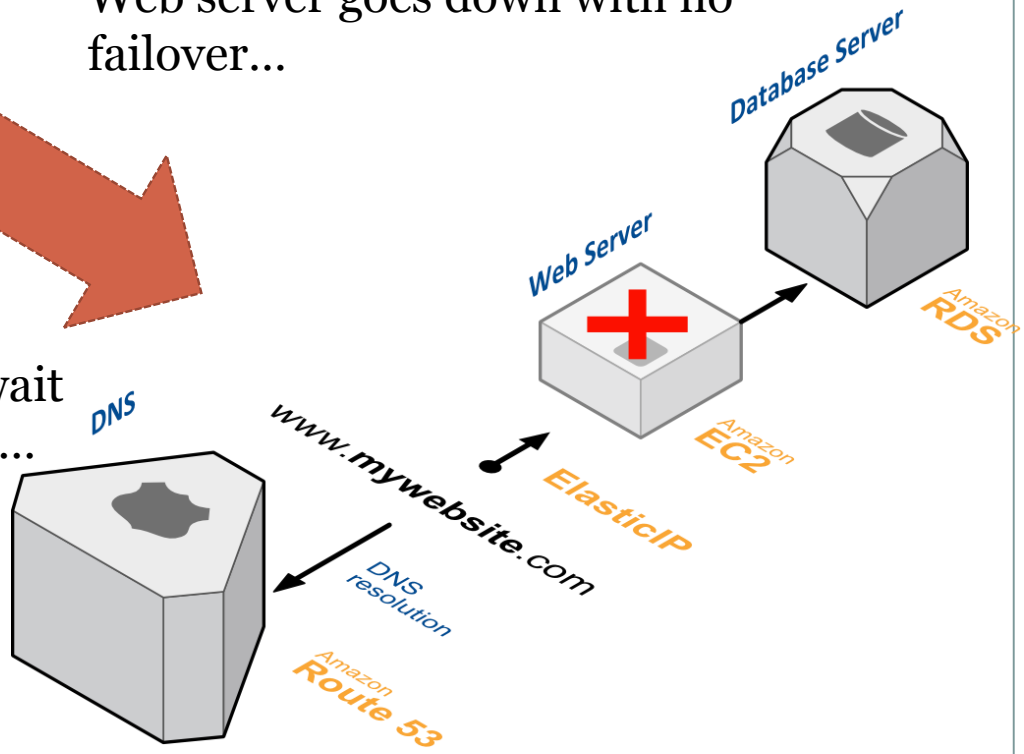


Design for Failure: What could possibly go wrong…

- Single point of failure.

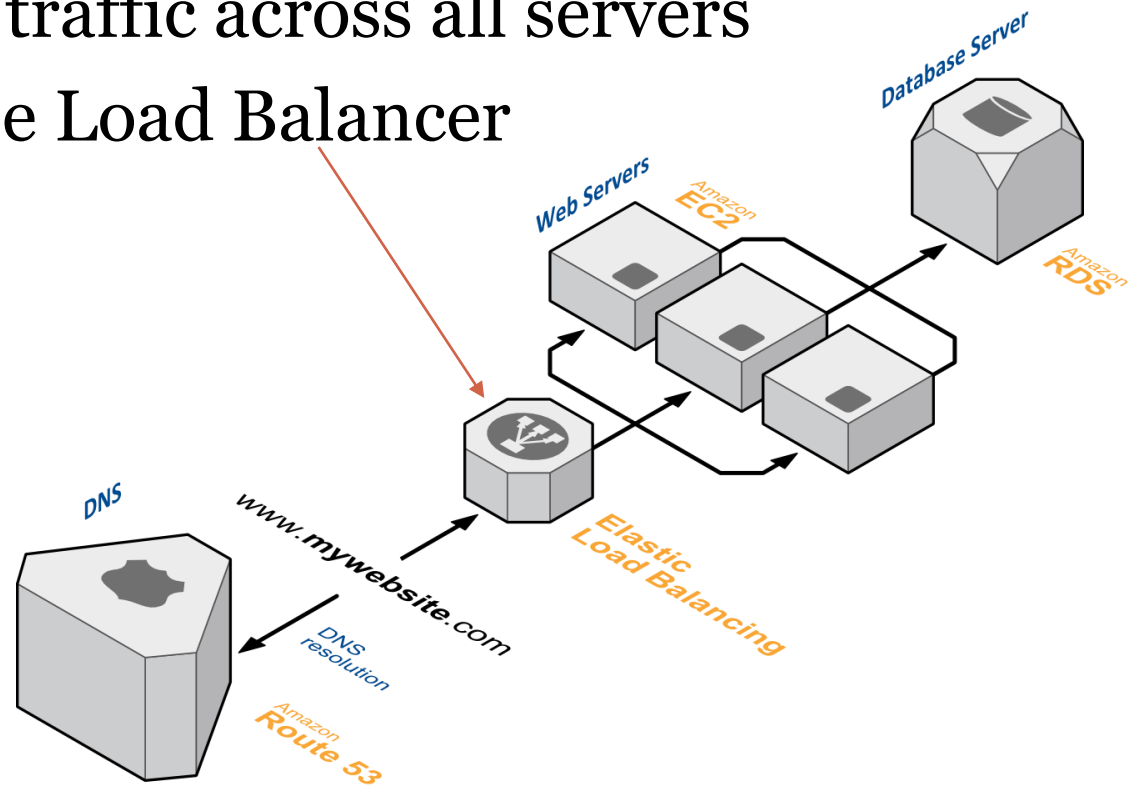
Web server goes down with no failover…

For web apps, not good enough to wait until somebody notices and reboots…



Design for Failure: Redundancy

- Include more than one web server
- Distribute traffic across all servers
- In AWS use Load Balancer



Design for Failure: Health Checks

- Want traffic to be directed to “healthy servers” – use health check.

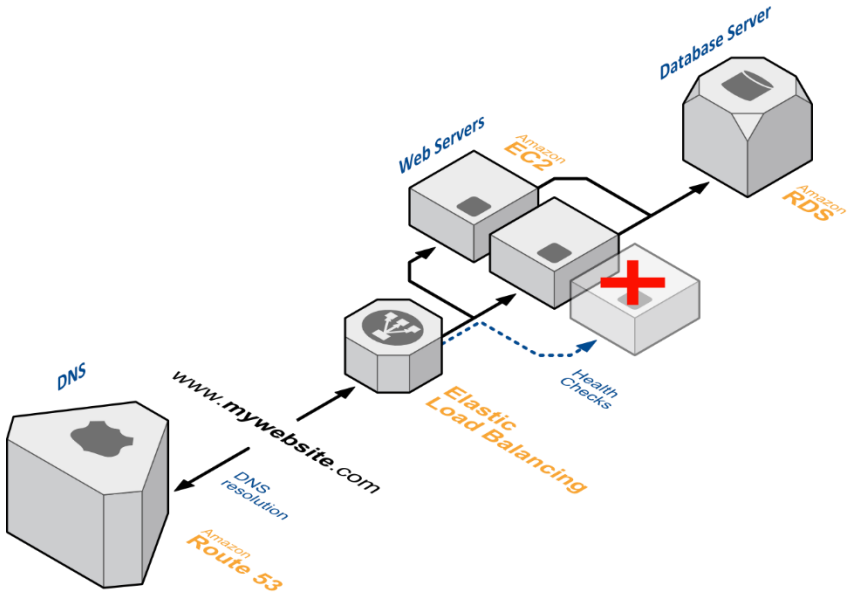
Configure Health Check

Your load balancer will automatically perform health checks on your EC
check. If an instance fails the health check, it is automatically removed
specific needs.

Ping Protocol	HTTP
Ping Port	80
Ping Path	/index.html

Advanced Details

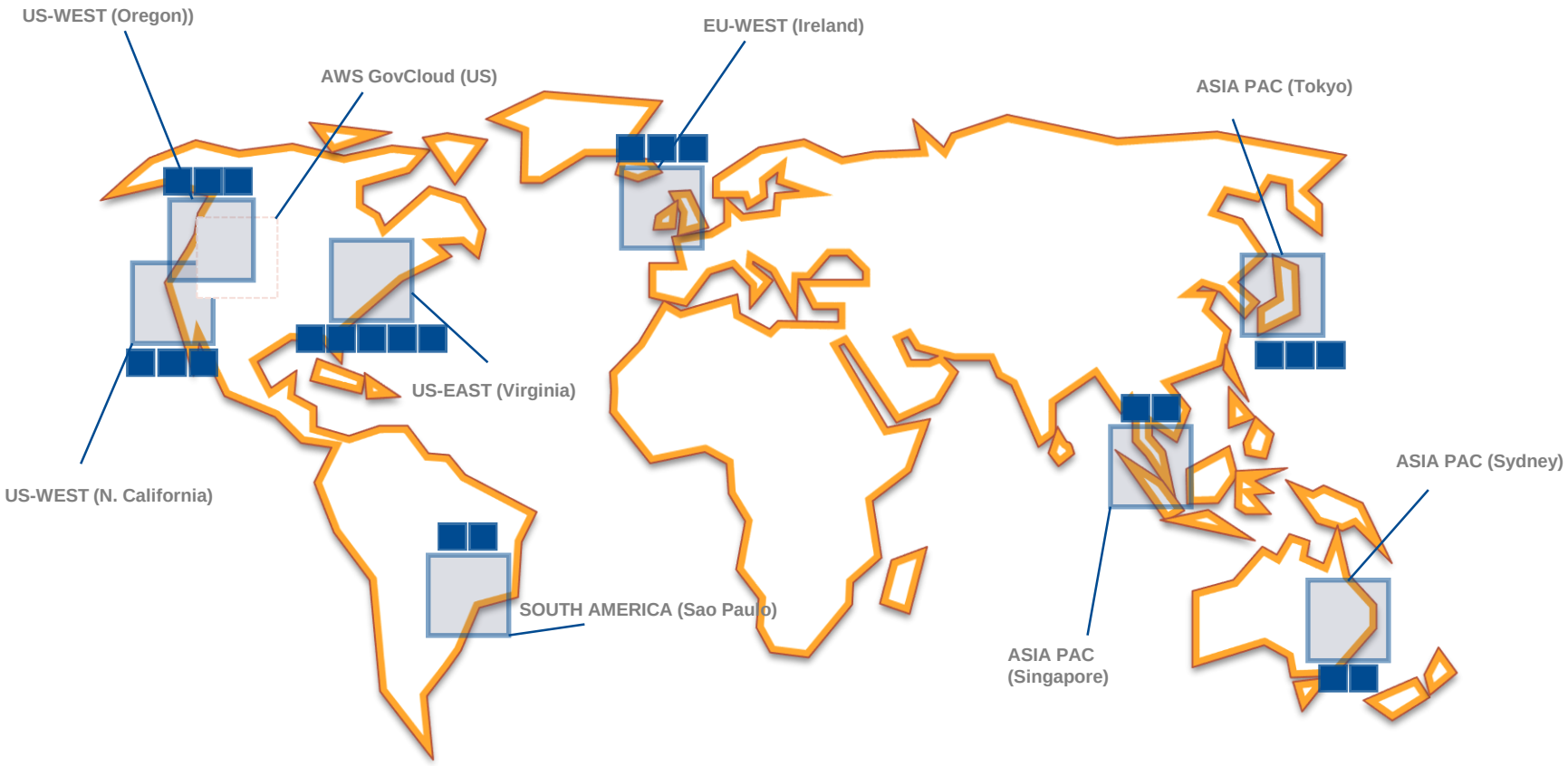
Response Timeout	5	seconds
Health Check Interval	30	seconds
Unhealthy Threshold	2	
Healthy Threshold	10	



Design for Failure: Availability Zones(AZs)

- Don’t put all your eggs in the one basket.
- Don’t have all your servers deployed on same physical infrastructure
- In AWS, can use multiple AZs to distribute servers

Design for Failure: Amazon AZs



Design for Failure: Redundancy with Multiple AZs

Add Instances to Load Balancer

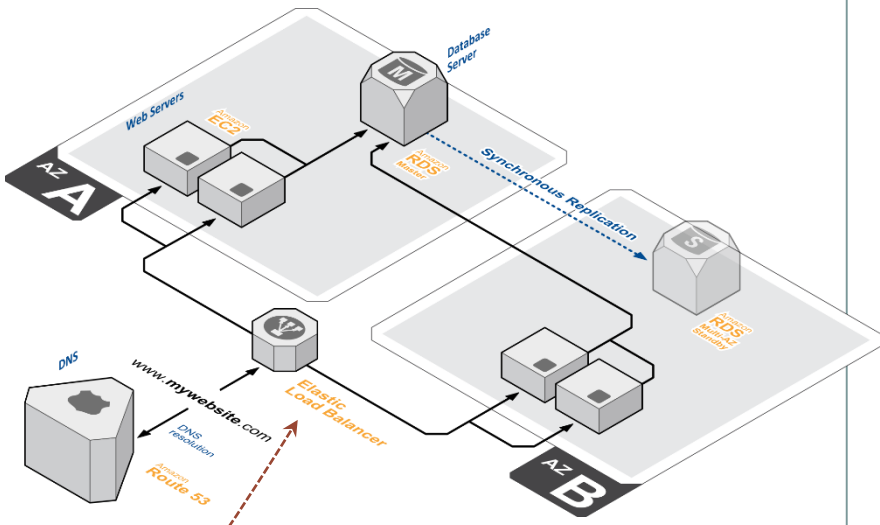
The table below lists all your running EC2 Instances. Check the

VPC vpc-a68d75c3 (10.250.0.0/16) | FX_VPC

	Instance	Name	State
☑	i-fd6825be	FX WebServer	stopped
☐	i-70642933	FX DBServer	stopped
☐	i-c1511c82	FX NATServer	stopped
☐	i-4b185608	FX Linux Node	stopped

Availability Zone Distribution
1 instance in eu-west-1a

☑ Enable Cross-Zone Load Balancing ⓘ



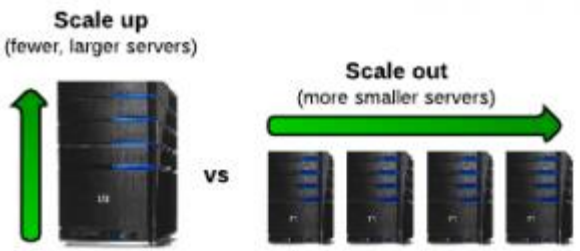
Design for Failure: Strategies

- Have a coherent backup and restore strategy for your data and automate it
- Build process threads that resume on reboot
- Allow the state of the system to re-sync by reloading messages from queues
- Keep pre-configured and pre-optimized virtual images to support (2) and (3) on launch/boot
- Avoid in-memory sessions or stateful user context, move that to data stores.

Building For Scale

"We're gonna need a bigger ~~boat~~ server..."

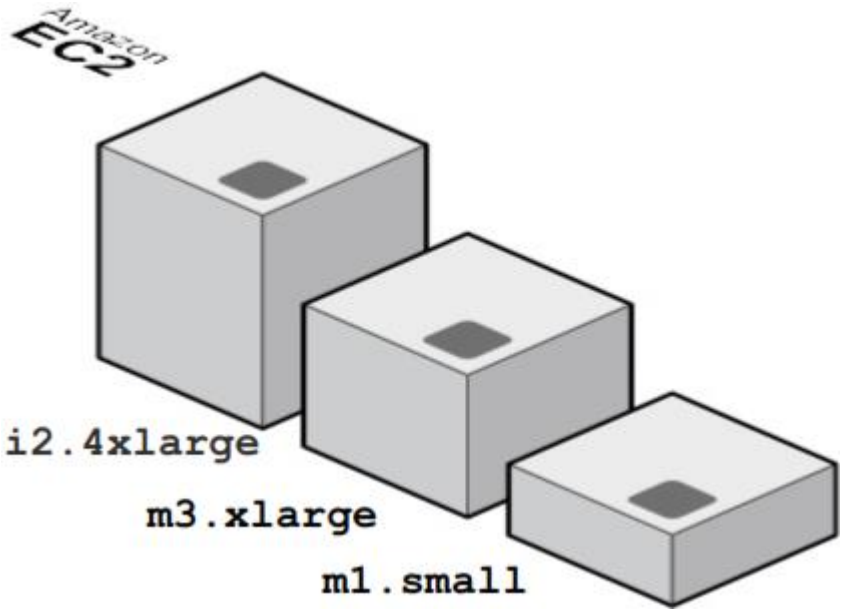
Building For Scale:Reactive Scaling



- Approaches to scaling applications
  - ***Scale-up approach:***
    - *not worrying about the scalable application architecture and investing heavily in larger and more powerful computers (vertical scaling) to accommodate the demand. This approach usually works to a point, but could either cost a fortune or the demand could out-grow capacity before the new "big iron" is deployed.*
  - ***Scale –out approach:***
    - *creating an architecture that scales horizontally and investing in infrastructure in small chunks.*
    - *often more effective than a scale up approach.*
    - *must predict the demand at regular intervals and then deploying infrastructure in chunks to meet the demand.*
    - *may lead to excess capacity ("burning cash") and constant manual monitoring ("burning human cycles").*

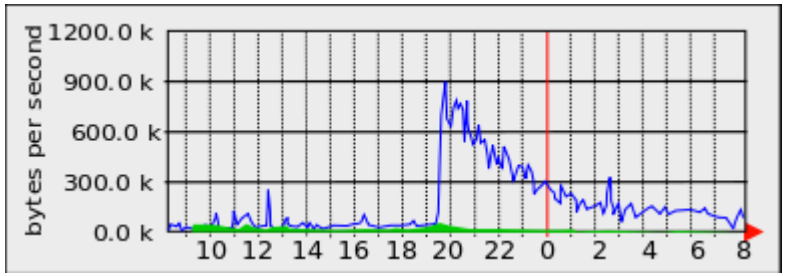
Building For Scale: Scale up in AWS

- Simple approach.
- High memory/ IO/ CPU/ Storage.
- Easy to change instance size.
- Will ultimately hit limit.



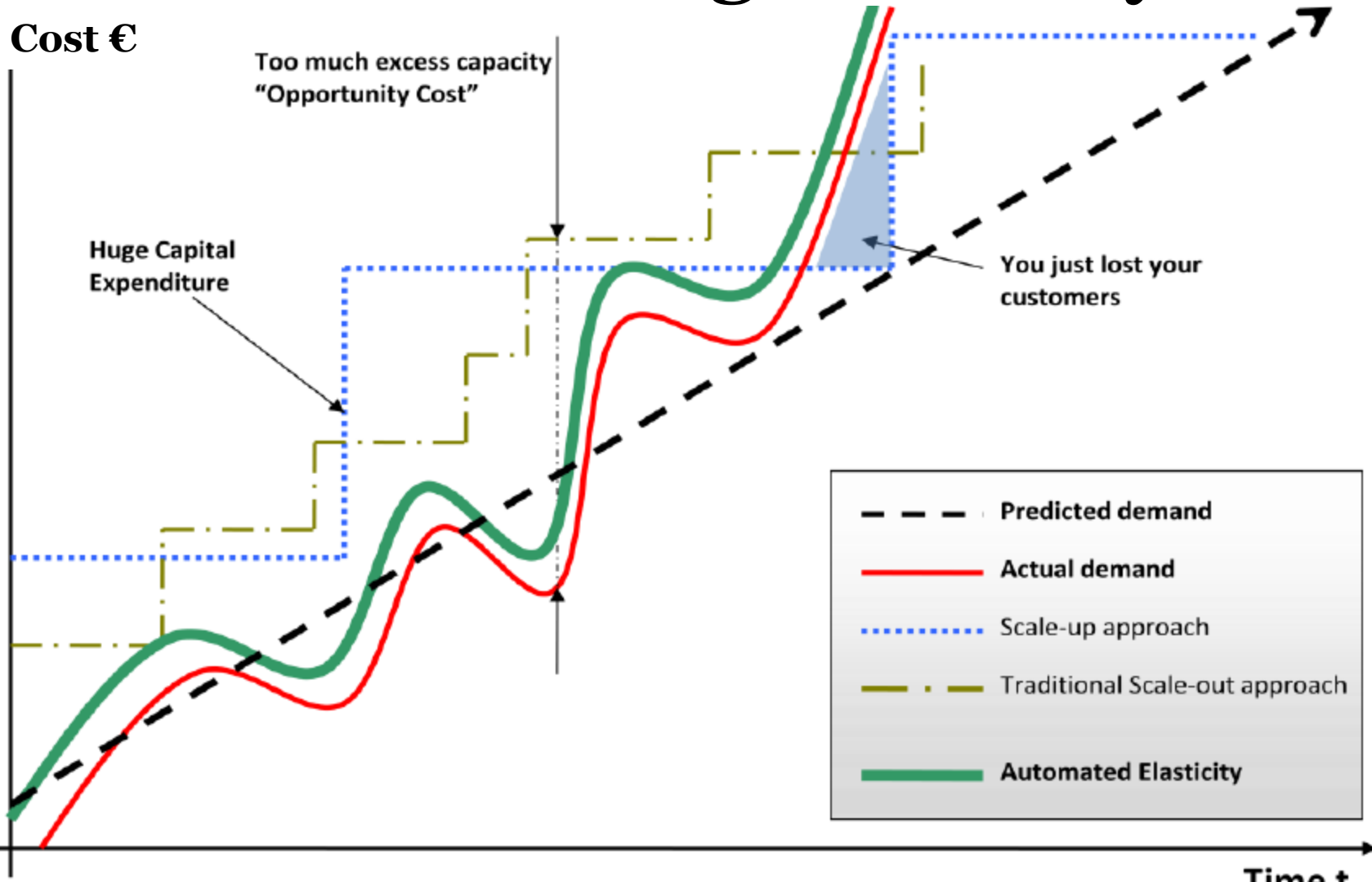
Building For Scale: What about Elasticity

- Unlike conventional Enterprise systems, Web Apps usage is unpredictable.
  - E.g. flash crowds from Slashdot effect in early 00s
- Cloud architecture so far can handle failure but what about sharp increase of traffic



- Would be cool if infrastructure could scale up and scale down to match demand - Elasticity

Building For Scale: Understanding Elasticity…



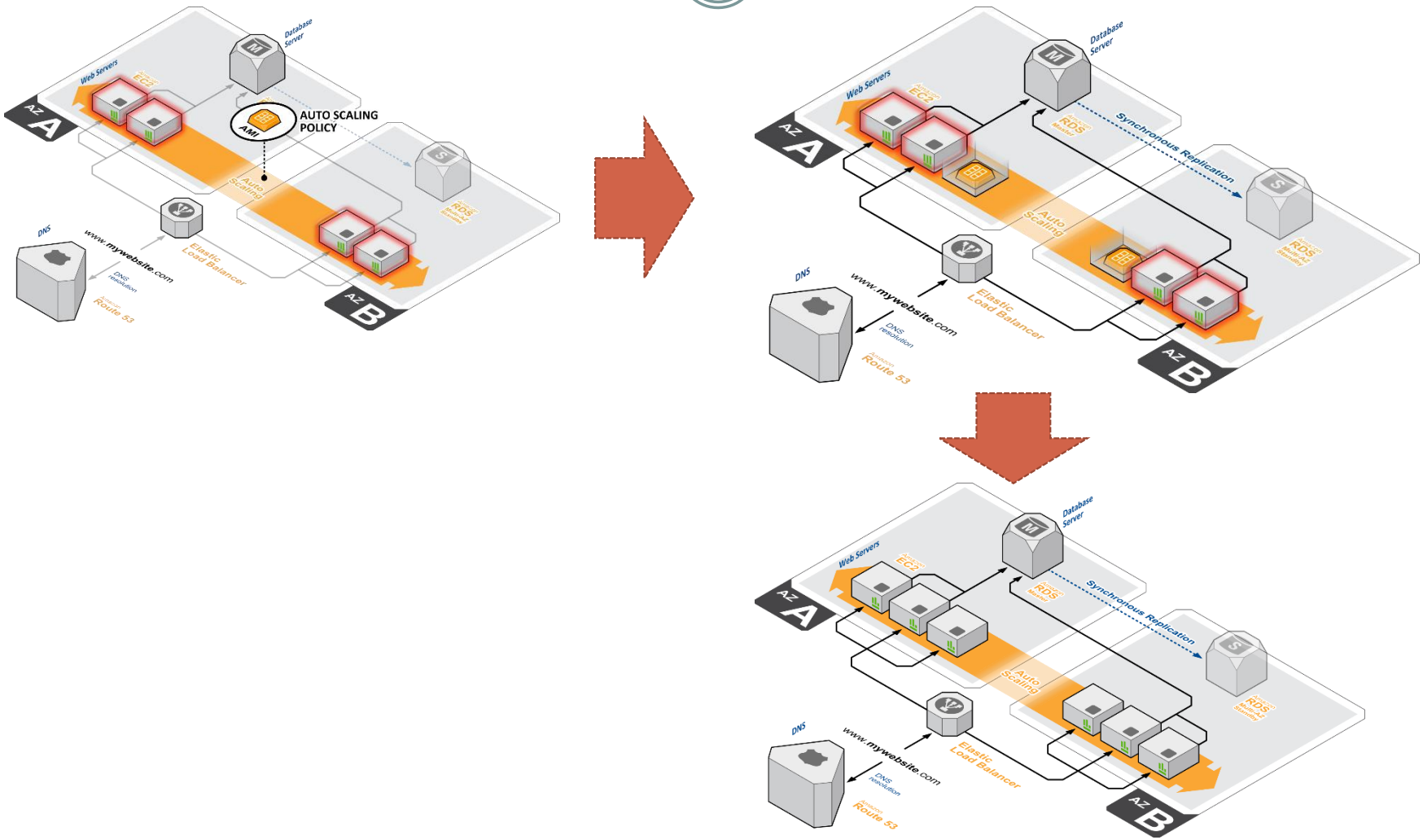
Building For Scale: Understanding Elasticity…

- Elasticity is one of the fundamental properties of distributed applications in the cloud
- Examples:

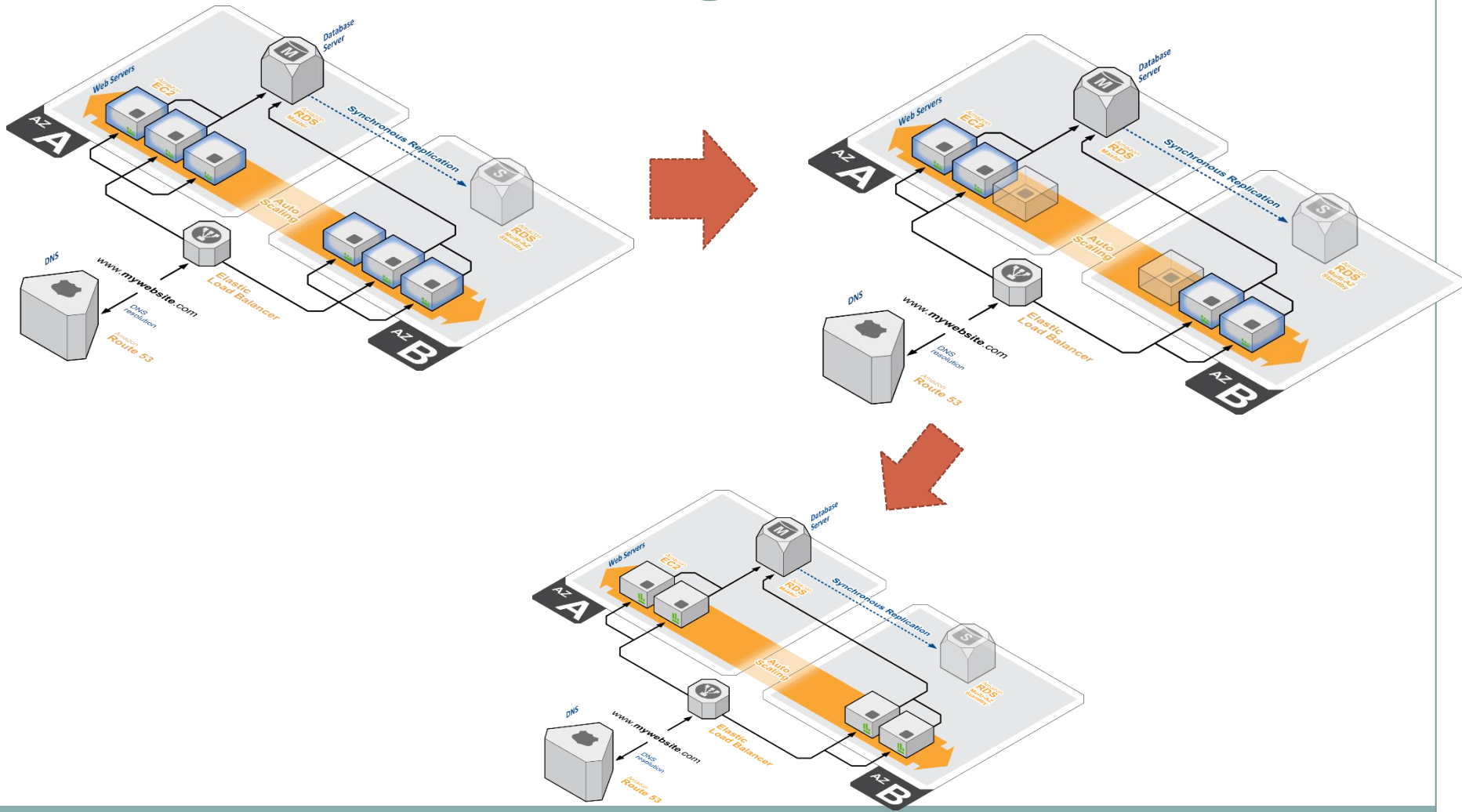
  infrastructure that used to run daily nightly builds and perform regression and unit tests every night at 2:00 AM for two hours (often termed as the "QA/Build box") was sitting idle for rest of the day. Now, with elastic infrastructure, one can run nightly builds on boxes that are "alive" and being paid for only for 2 hours in the night.

  An internal trouble ticketing web application that always used to run on peak capacity (5 servers 24x7x365) to meet the demand during the day can now be provisioned to run on-demand (5 servers from 9AM to 5 PM and 2 servers for 5 PM to 9 AM) based on the traffic pattern.

Building For Scale: Auto-scaling in AWS – Scale up



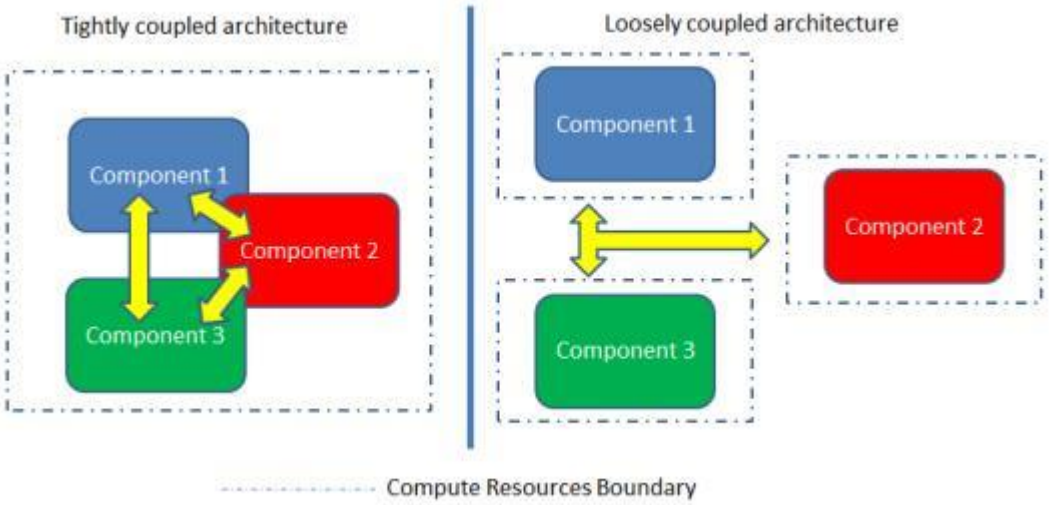
Building For Scale: Auto-scaling in AWS – Scale down



AWS: Best Practice

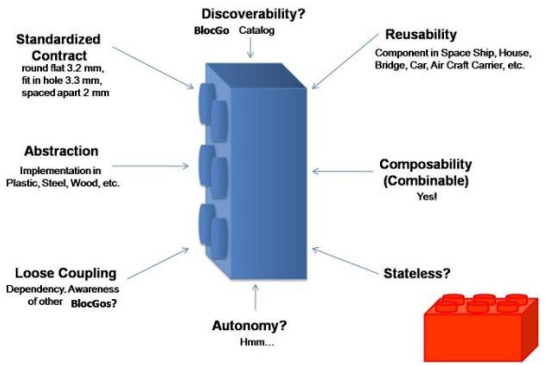
- Failover gracefully using Elastic IPs: Elastic IP is a static IP that is dynamically re-mappable. You can quickly remap and failover to another set of servers so that your traffic is routed to the new servers. It works great when you want to upgrade from old to new versions or in case of hardware failures
- Utilize multiple Availability Zones: Availability Zones are conceptually like logical datacenters. By deploying your architecture to multiple availability zones, you can ensure highly availability. Utilize Amazon RDS Multi-AZ [21] deployment functionality to automatically replicate database updates across multiple Availability Zones.
- Maintain an Amazon Machine Image so that you can restore and clone environments very easily in a different Availability Zone; Maintain multiple Database slaves across Availability Zones and setup hot replication.
- Utilize Amazon CloudWatch (or various real-time open source monitoring tools) to get more visibility and take appropriate actions in case of hardware failure or performance degradation.
- Setup an Auto scaling group to maintain a fixed fleet size so that it replaces unhealthy Amazon EC2 instances by new ones.
- Utilize Amazon EBS and set up cron jobs so that incremental snapshots are automatically uploaded to Amazon S3 and data is persisted independent of your instances.
- Utilize Amazon RDS and set the retention period for backups, so that it can perform automated
- backups.

Loose Coupling



Loose Coupling: The Looser the better…

- Loose coupling refers to minimising dependencies between services
- Promotes interface programming (separating interface from implementation)
- Trend towards REST and generic interfaces(More later…)
- Variable communication patterns



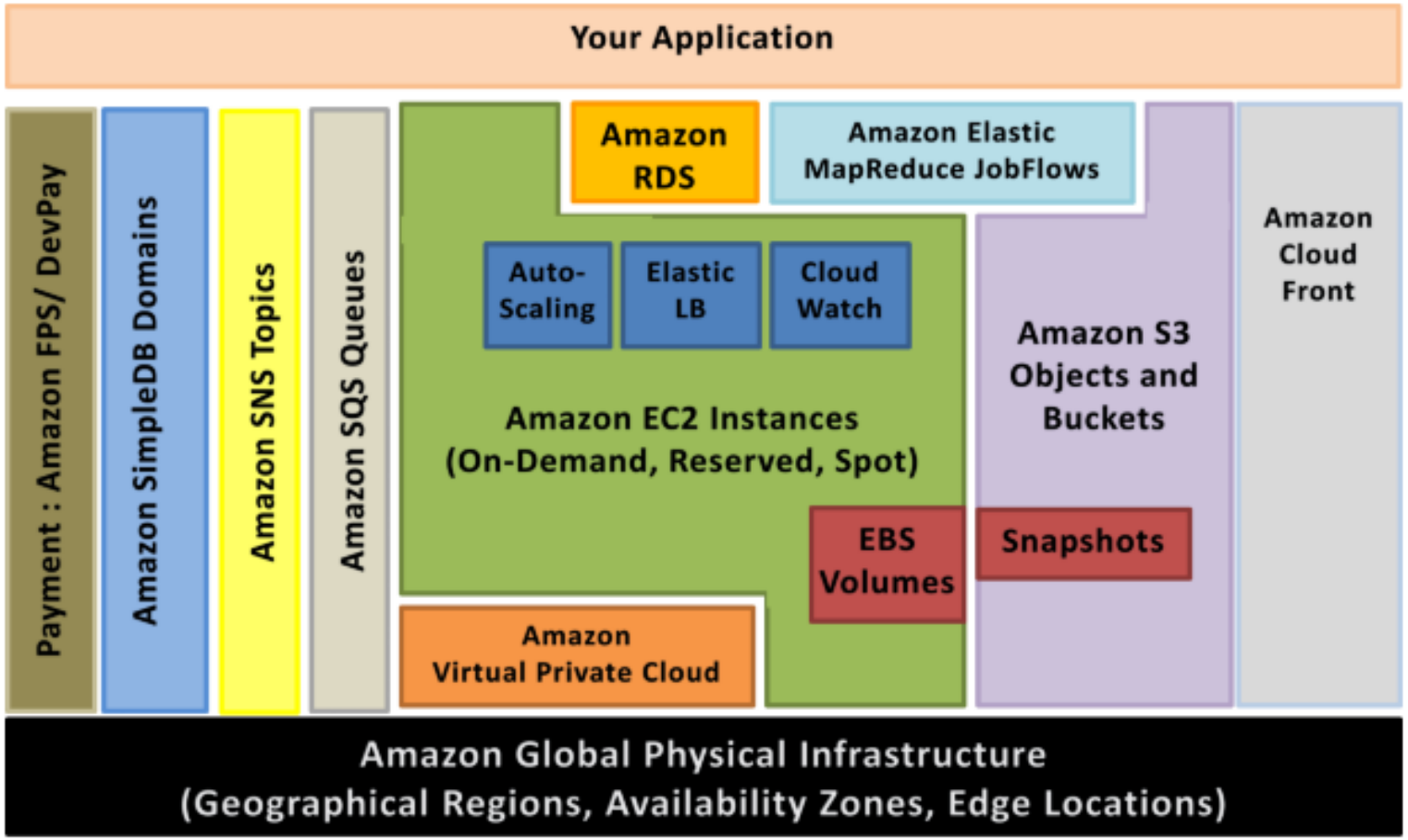
Loose Coupling: In AWS…

- Independent components
- Everything is “Black boxed” – just care about interface
- Decouple interactions
- Can use ‘off the shelf’ services that have redundancy built-in. – Nice! E.g. SQS

Loose Coupling: Fault tolerant Services…

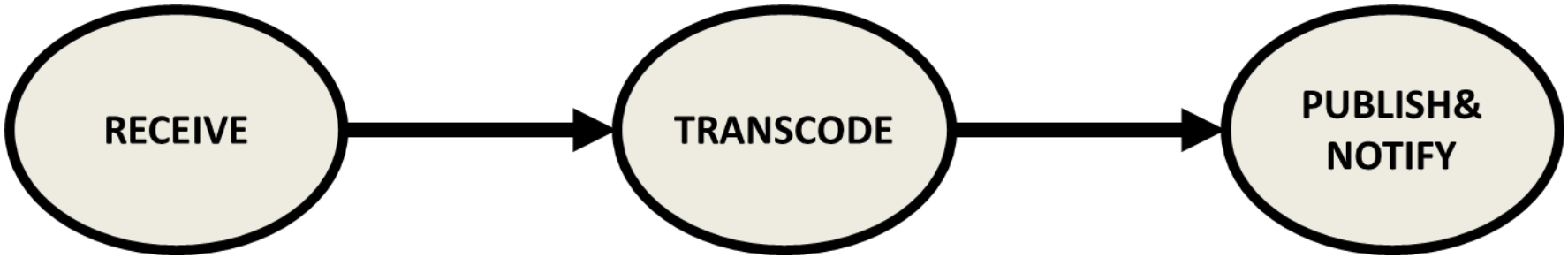
- Amazon S3
- Amazon SimpleDB
- Amazon DynamoDB
- Amazon CloudFront
- Amazon SWF
- Amazon SQS
- Amazon SNS
- Amazon SES
- Amazon Route53
- Elastic Load Balancing
- AWS IAM
- AWS Elastic Beanstalk
- Amazon ElastiCache
- Amazon EMR
- Amazon CloudSearch

AWS Services

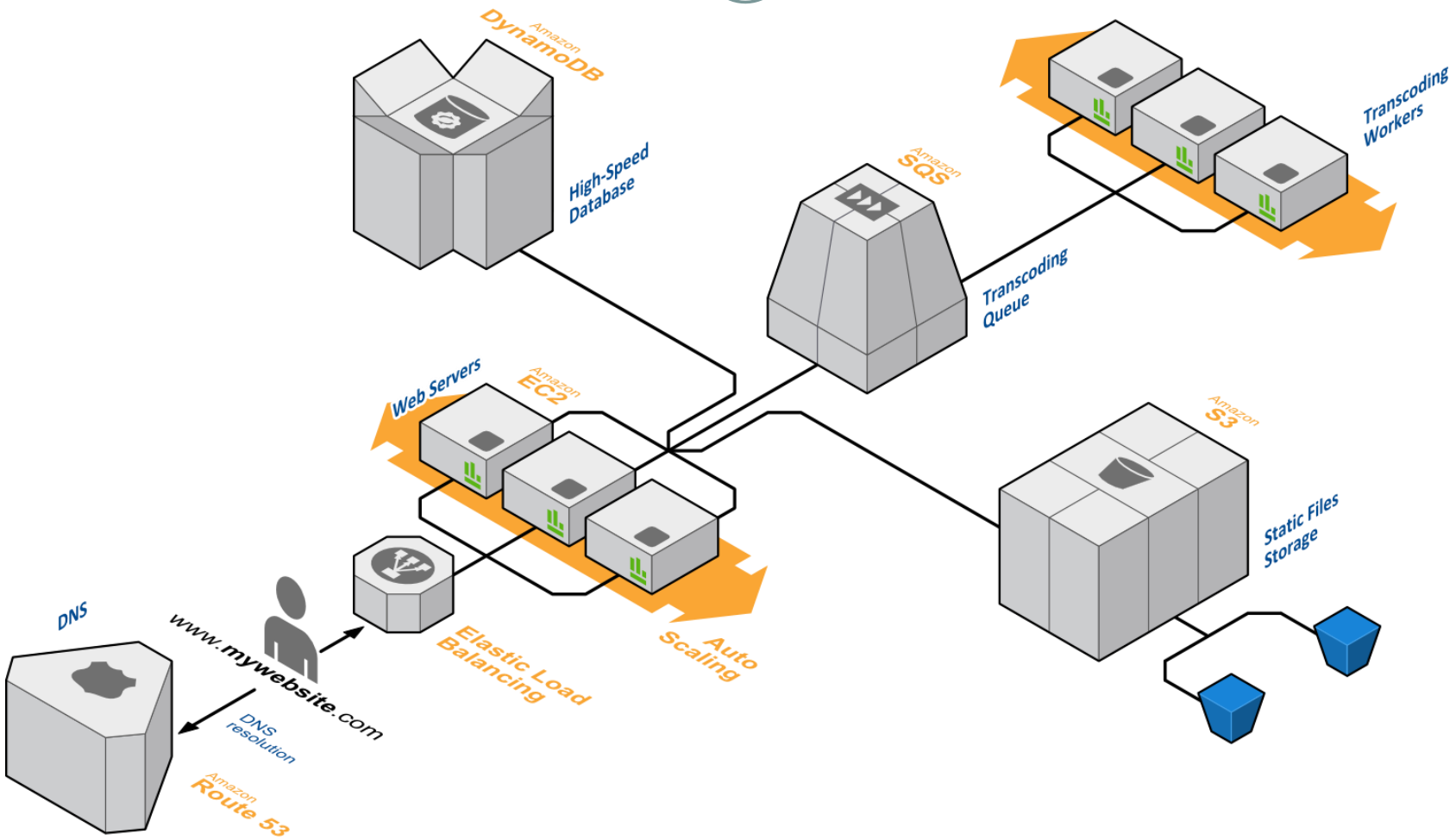


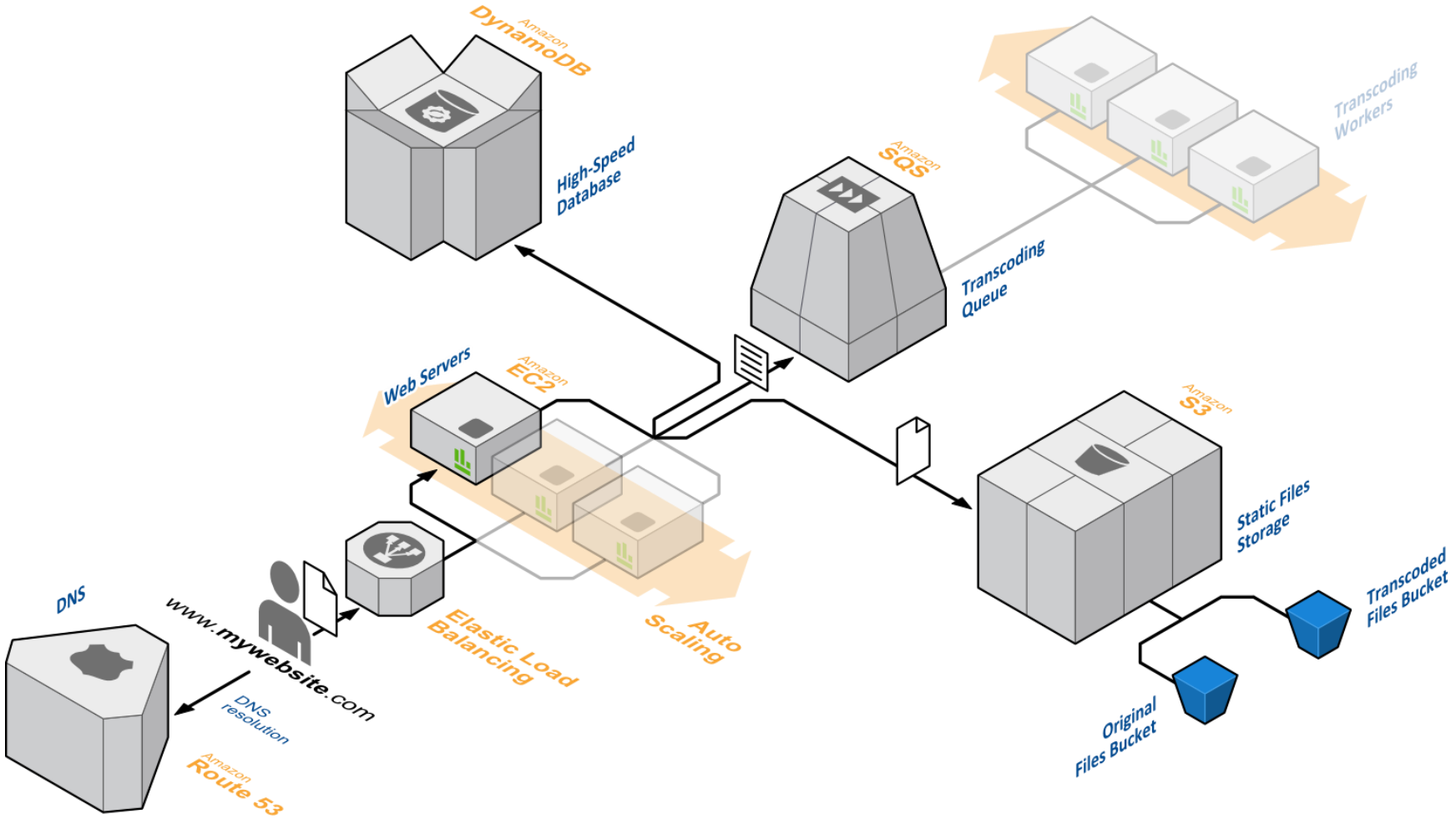
Our Case Study - Transcoding

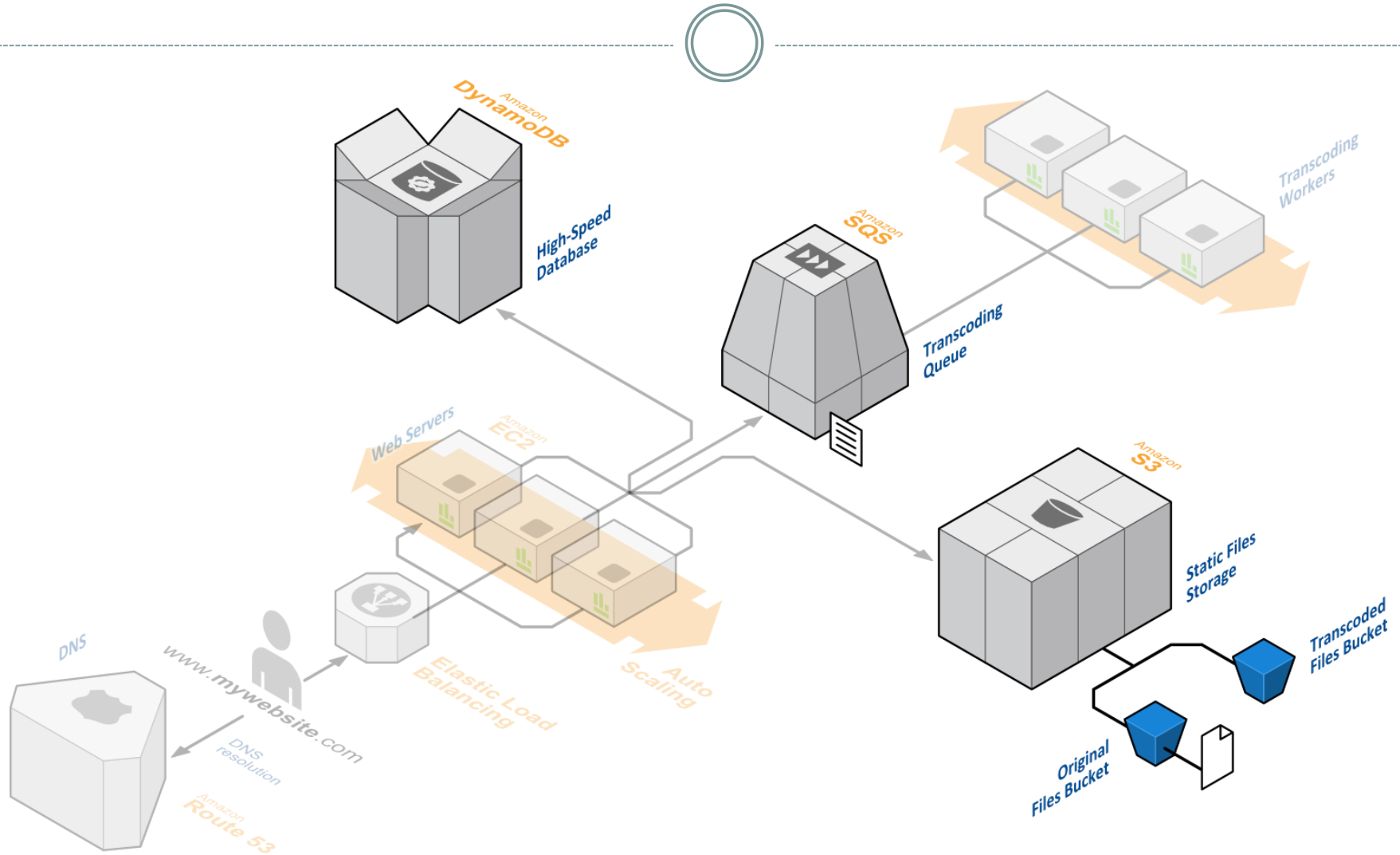
Example: Business wants to convert (or “transcode”) customers media files from their source format into versions that will playback on devices like smartphones, tablets and PCs…

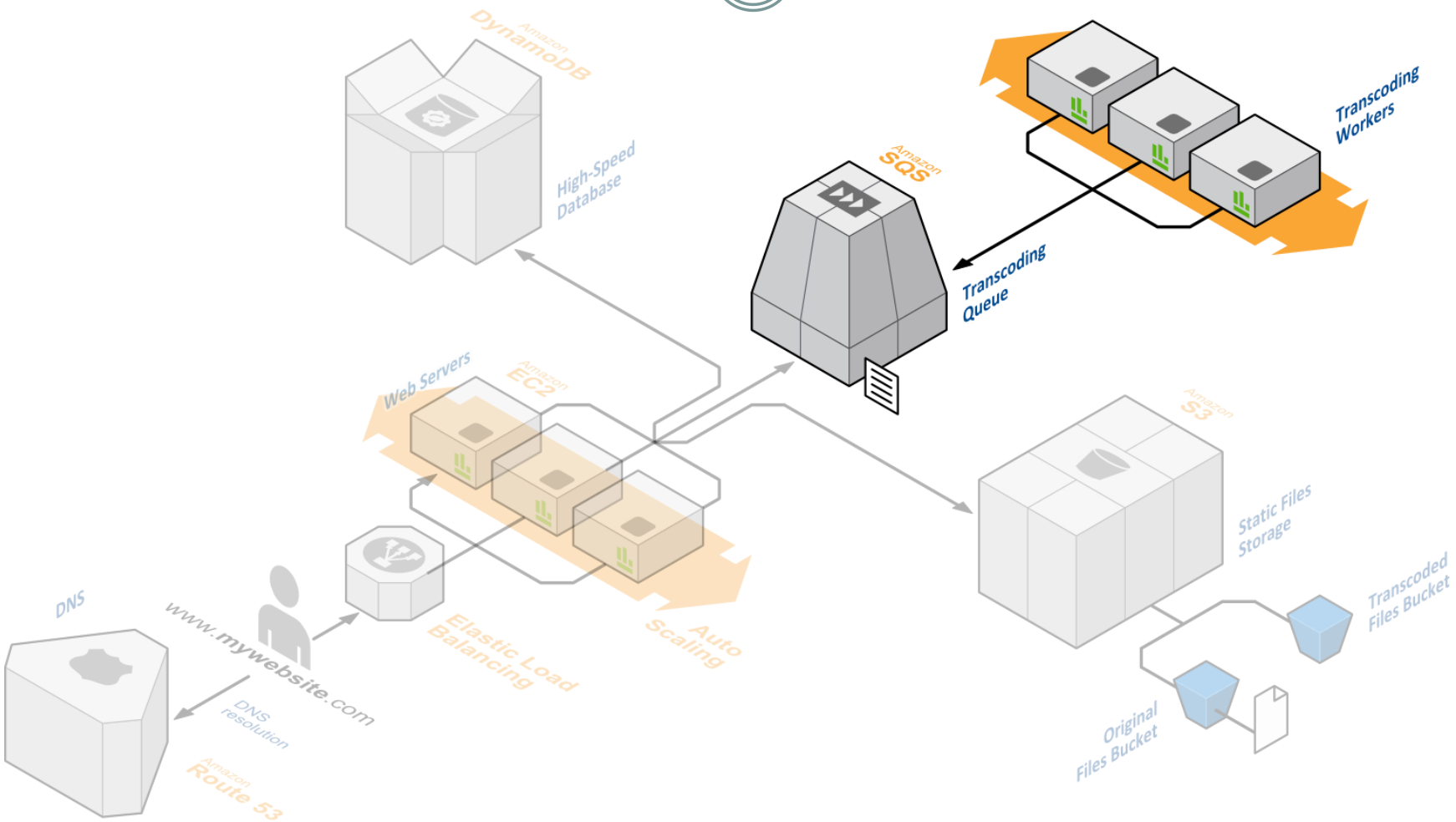


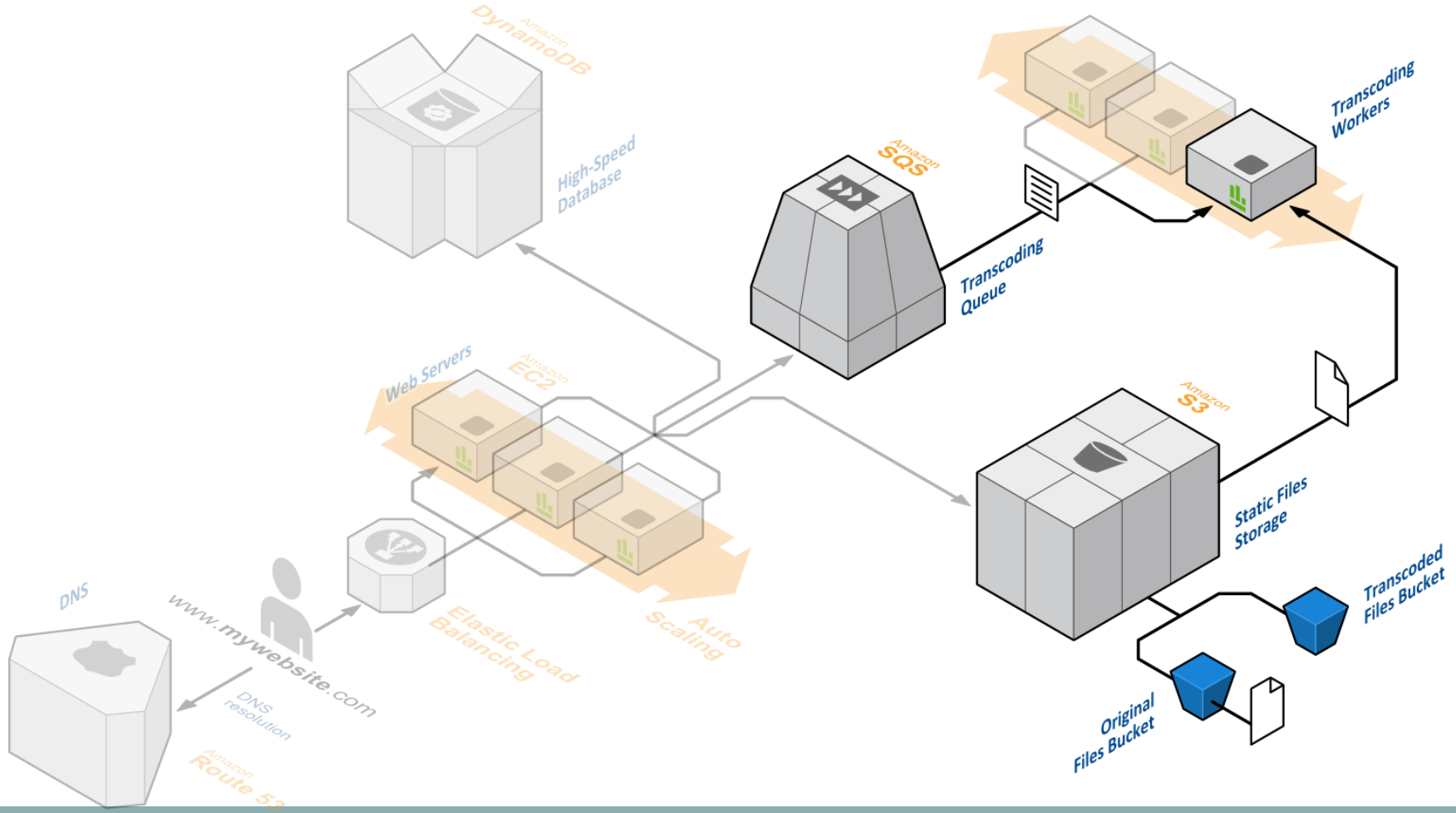
AWS Trancoding Architecture

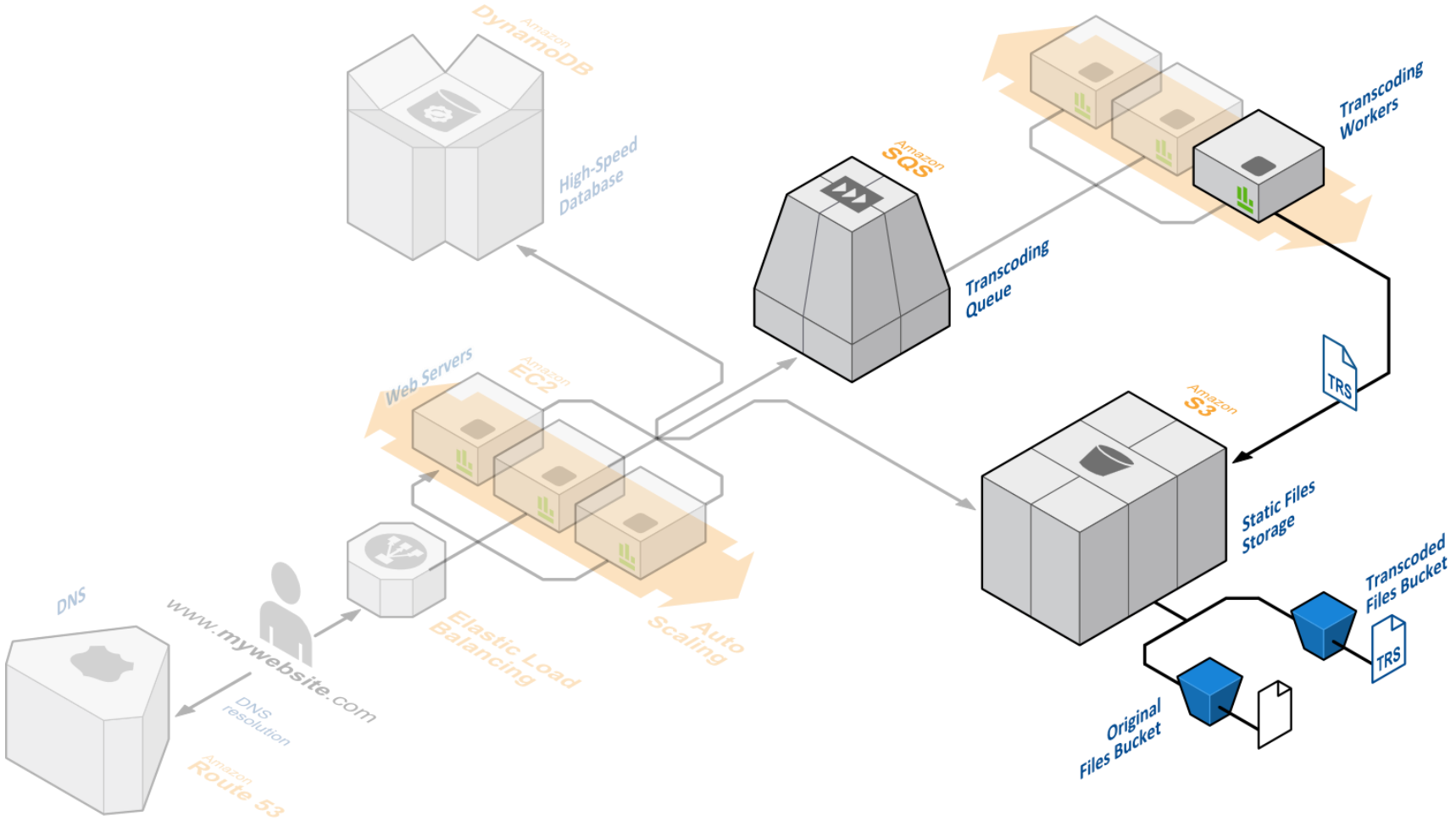



Amazon
DynamoDB
High-Speed
Database
Amazon
SQS
Transcoding
Queue
Transcoding
Workers
Web Servers
Amazon
EC2
Amazon
S3
Static Files
Storage
Transcoded
Files Bucket
Original
Files Bucket
DNS
www.mywebsite.com
DNS
resolution
Amazon
Route 53
Elastic Load
Balancing
Auto
Scaling


Amazon
DynamoDB
High-Speed
Database
Amazon
SQS
Transcoding
Queue
Transcoding
Workers
Web Servers
Amazon
EC2
Elastic Load
Balancing
Auto
Scaling
Amazon
S3
Static Files
Storage
Transcoded
Files Bucket
Original
Files Bucket
DNS
www.mywebsite.com
DNS
resolution
Amazon
Route 53

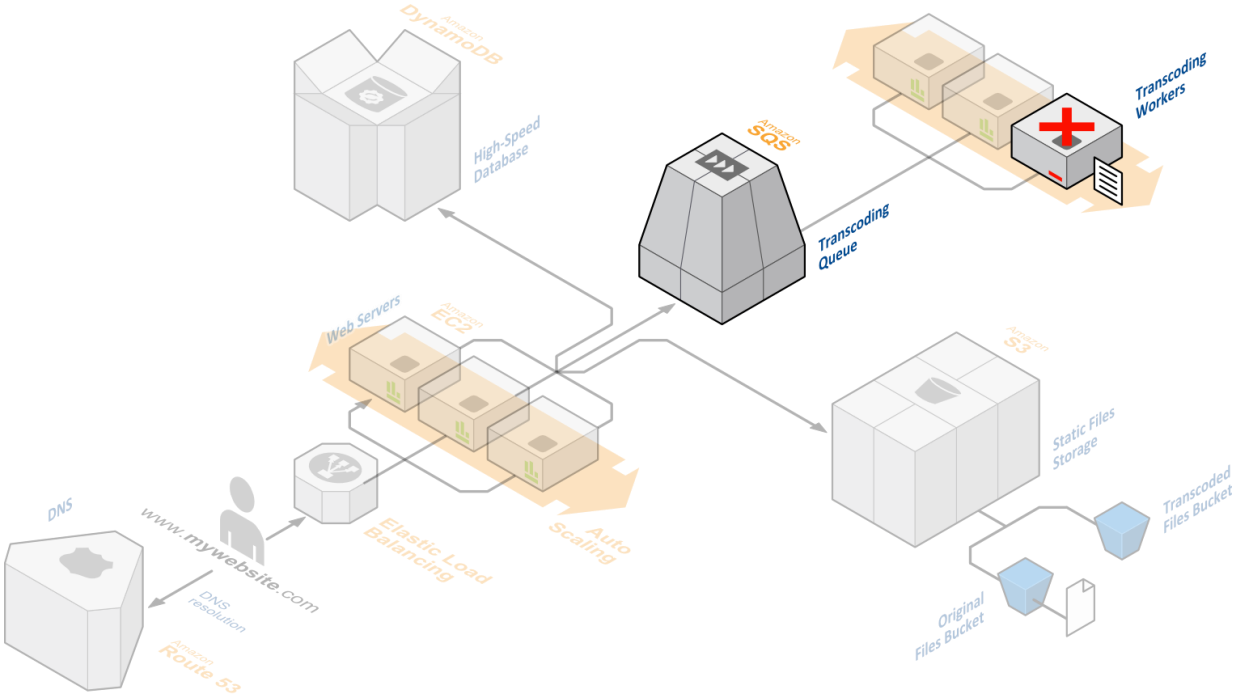

Amazon DynamoDB
High-Speed Database
Amazon SQS
Transcoding Queue
Transcoding Workers
Web Servers
Amazon EC2
Elastic Load Balancing
Auto Scaling
Amazon S3
Static Files Storage
Transcoded Files Bucket
Original Files Bucket
DNS
www.mywebsite.com
DNS resolution
Amazon Route 53


Amazon DynamoDB
High-Speed Database
Amazon SQS
Transcoding Queue
Transcoding Workers
Web Servers
Amazon EC2
Auto Scaling
Elastic Load Balancing
Amazon S3
Static Files Storage
Transcoded Files Bucket
Original Files Bucket
DNS
www.mywebsite.com
DNS resolution
Amazon Route 53


Amazon DynamoDB
High-Speed Database
Amazon SQS
Transcoding Queue
Transcoding Workers
TRS
Amazon S3
Static Files Storage
Transcoded Files Bucket
TRS
Original Files Bucket
Web Servers
Amazon EC2
Elastic Load Balancing
Auto Scaling
DNS
www.mywebsite.com
DNS resolution
Amazon Route 53

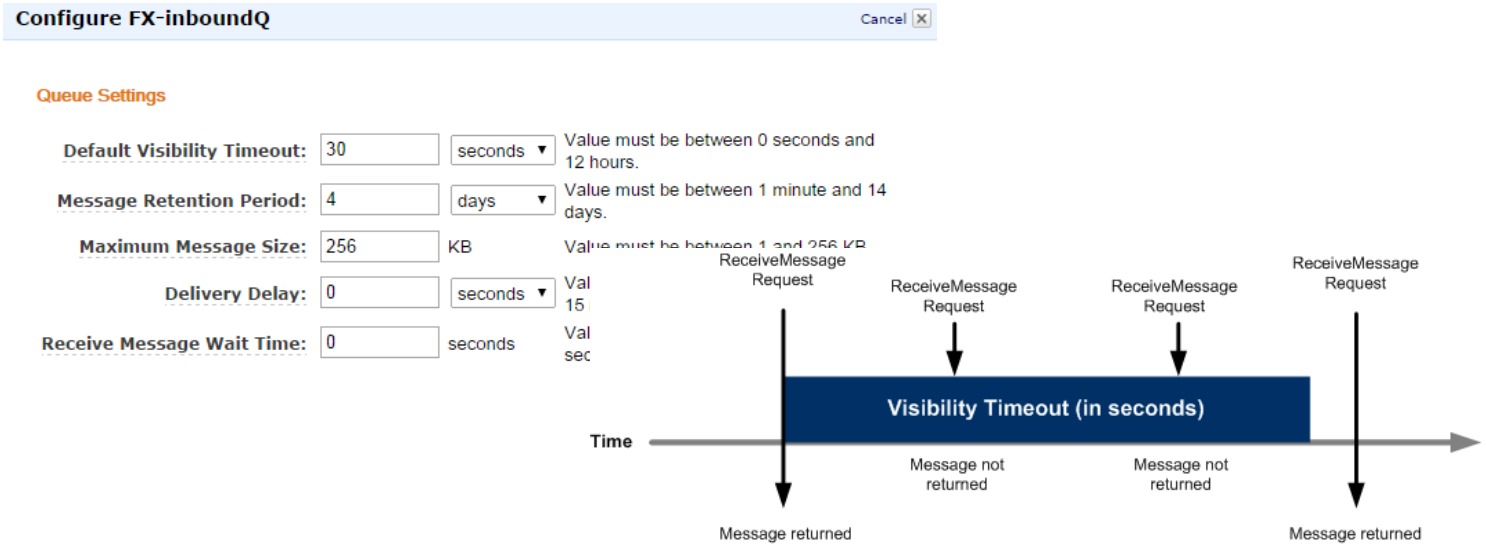
Visibility Timeout

- What if a worker takes a message and fails to complete transcoding…



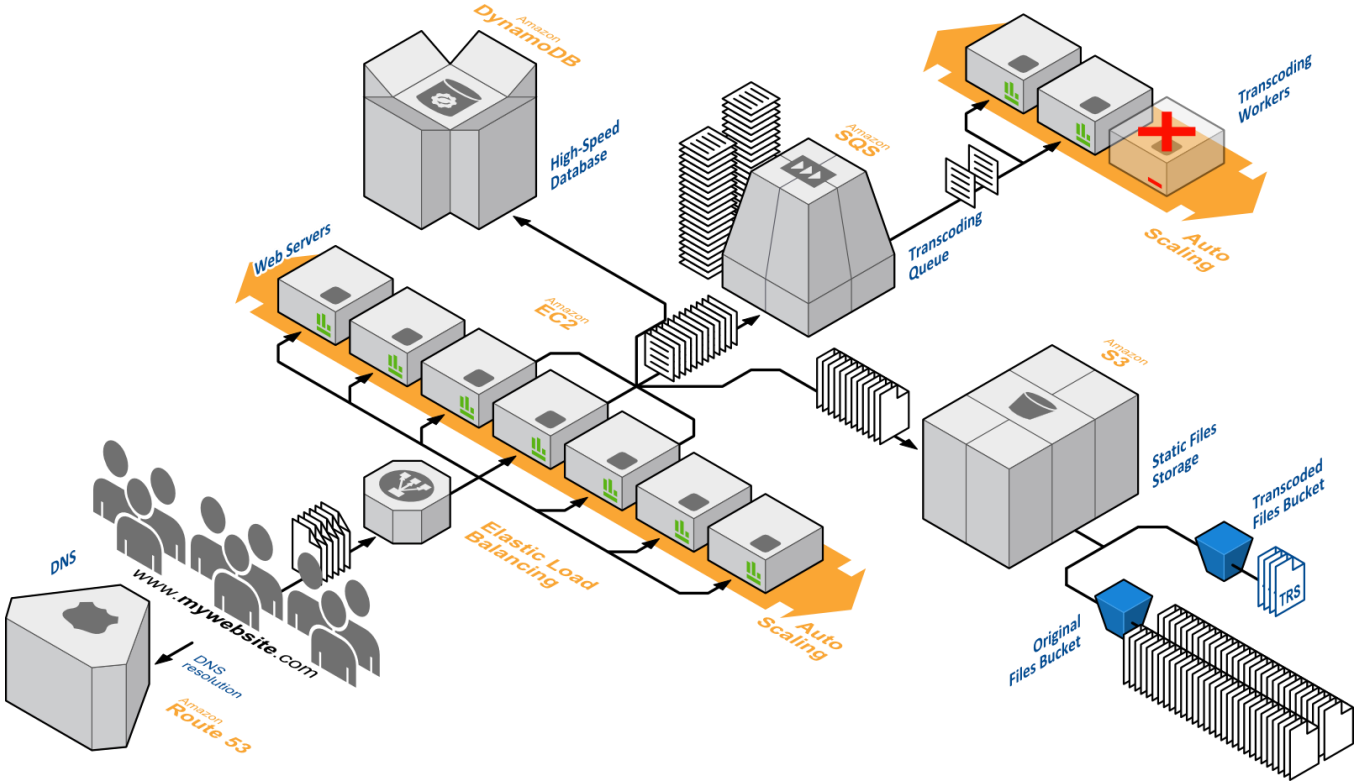
Default Visibility Timeout

- It's a distributed system, so there's no guarantee that the worker will actually receive the message
  - connection could break, worker could fail, component could fail.
- SQS does not delete the message, and instead, the worker process deletes the message from the queue after receiving and processing it.
- If message not removed, will become "visible" after timeout…



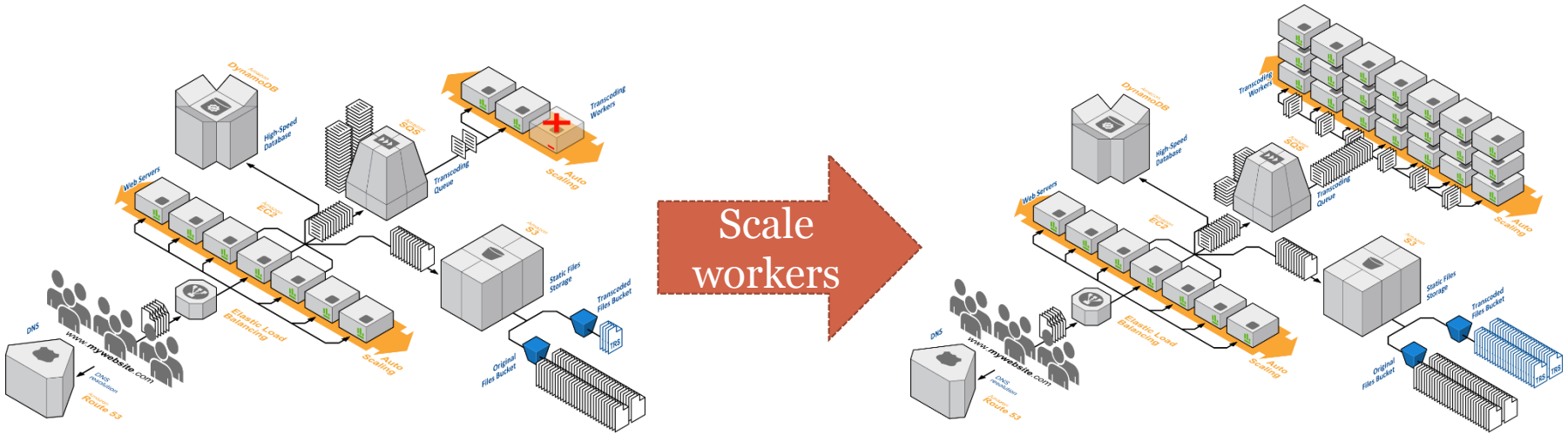
Buffering using Qs

- Can use queue to buffer requests



Using Cloudwatch and Q metrics to Autoscale

- Web Servers autoscale based on traffic in.
- Build up on SQS can be used to spin up worker processes to deal with it



References

- http://www.allthingsdistributed.com/2012/11/efficient-queueing-sqs.html
- http://www.slideshare.net/AmazonWebServices/t1architecting-highly-available-applications-on-aws
- http://www.informit.com/articles/article.aspx?p=349749&seqNum=5