Scaling Up to Your First 10 Million Users

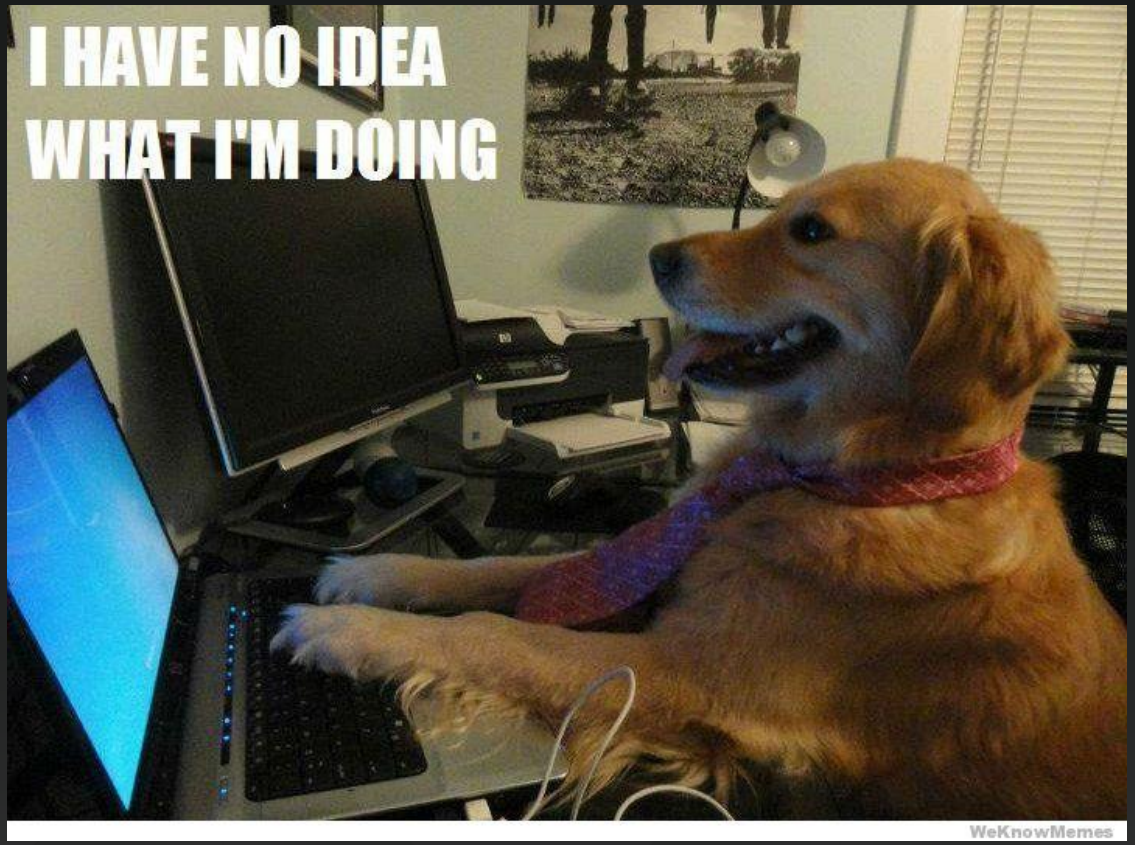
Brett Hollman, Manager, AWS Solutions Architecture

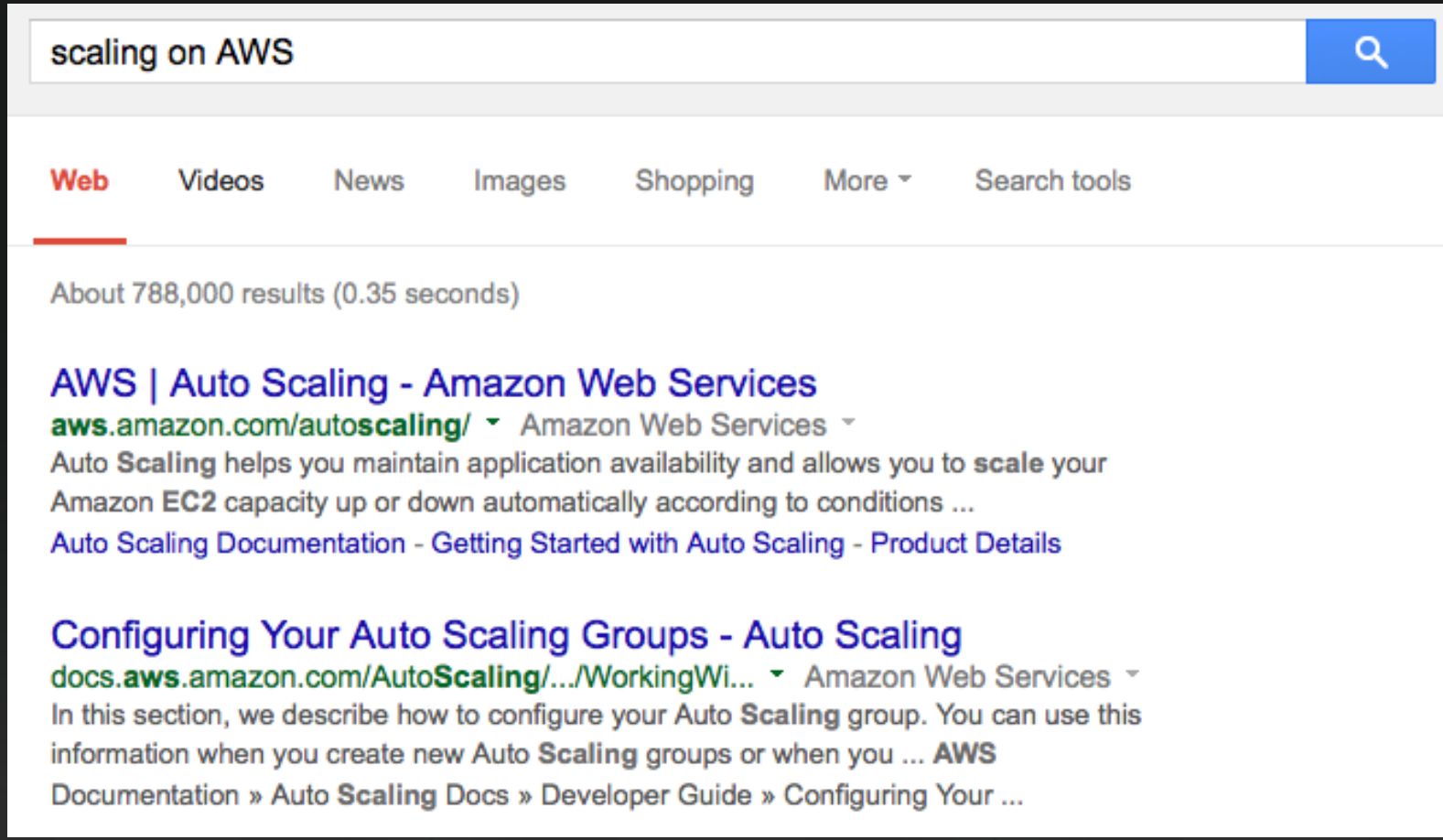
amazon web services

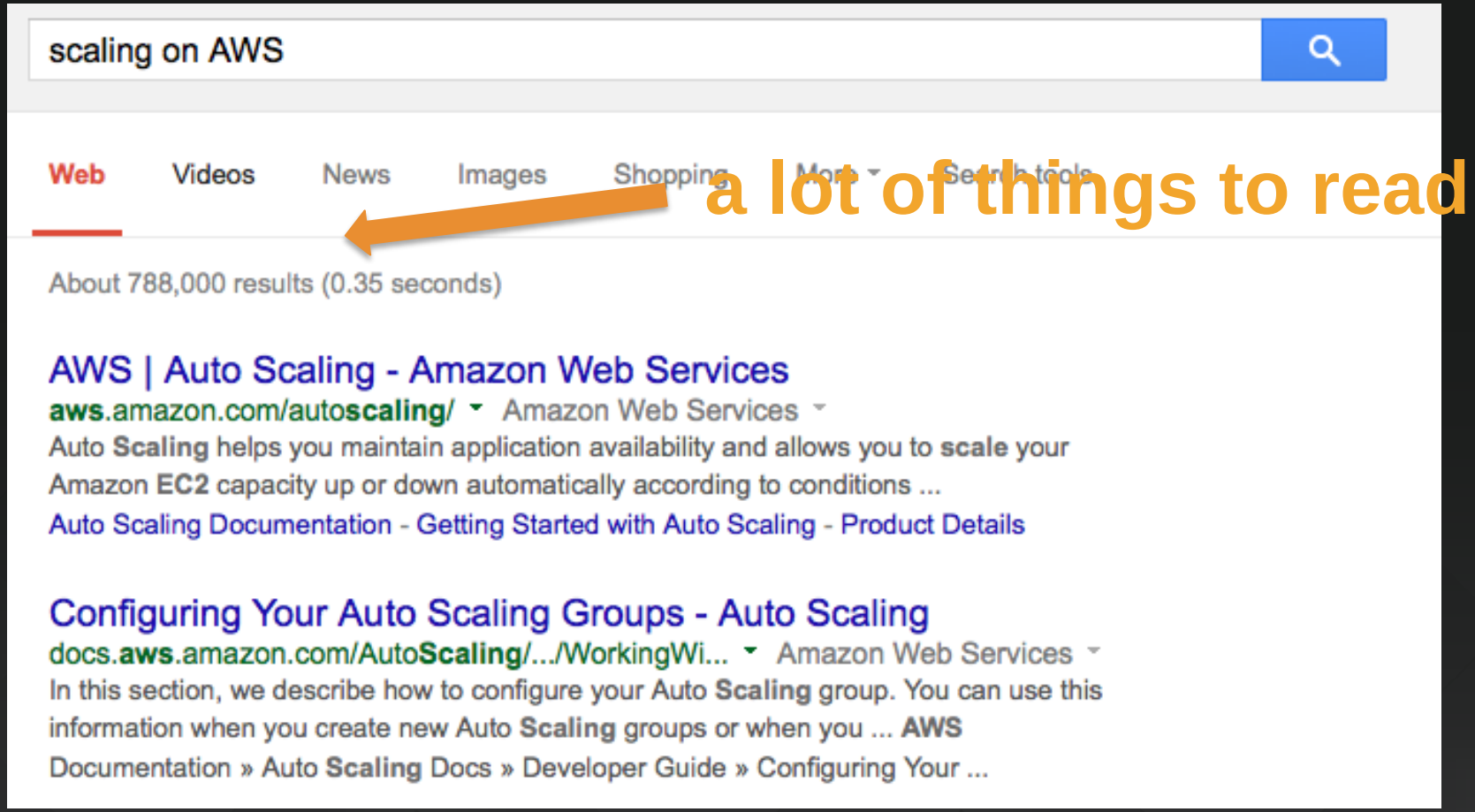
Hello, San Francisco!

- ME: Brett Hollman—Manager, Solutions Architecture, Amazon Web Services
- YOU: Here to learn more about scaling infrastructure on AWS
- TODAY: About best practices and things to think about when building for large scale

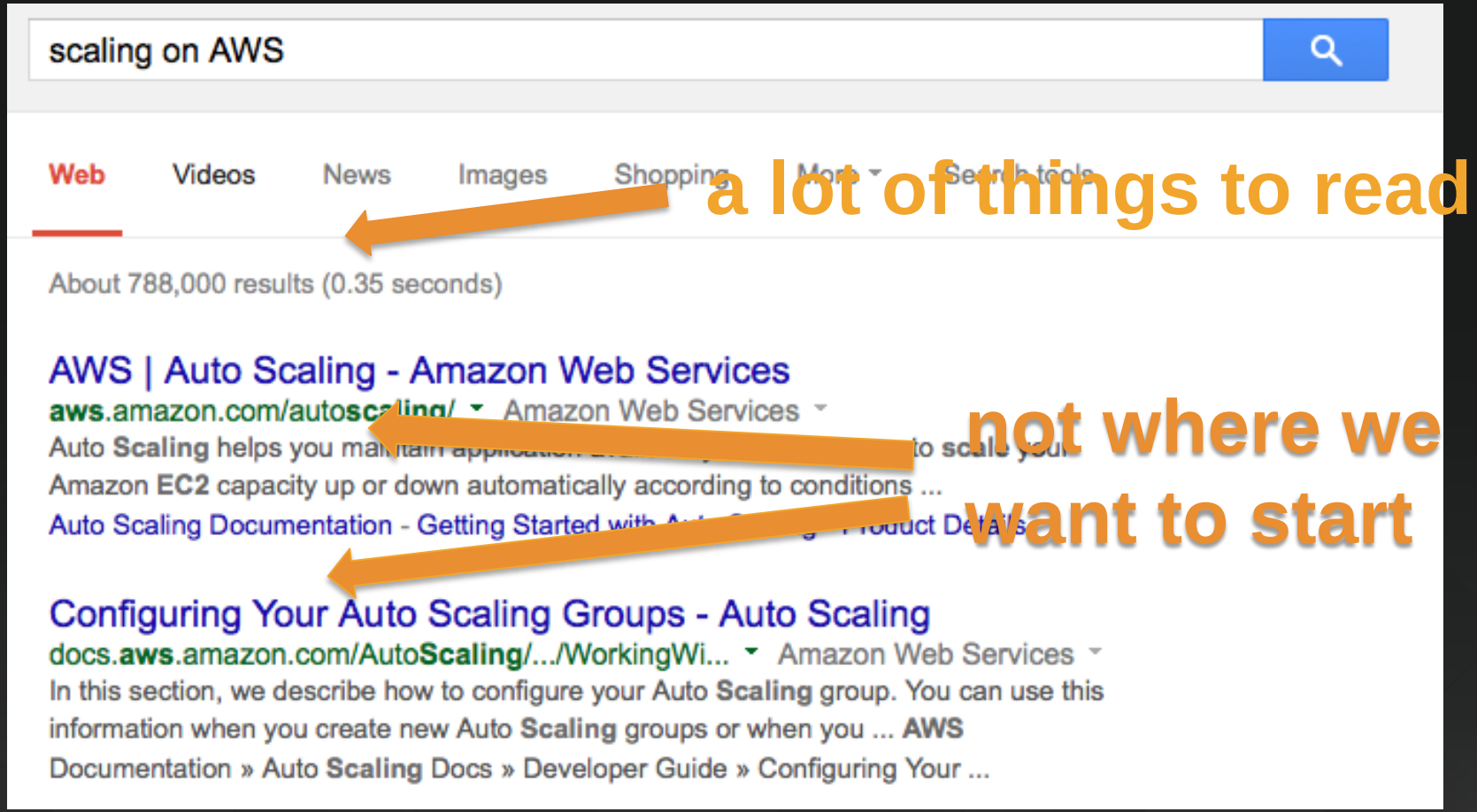
So how do we scale?







amazon web services

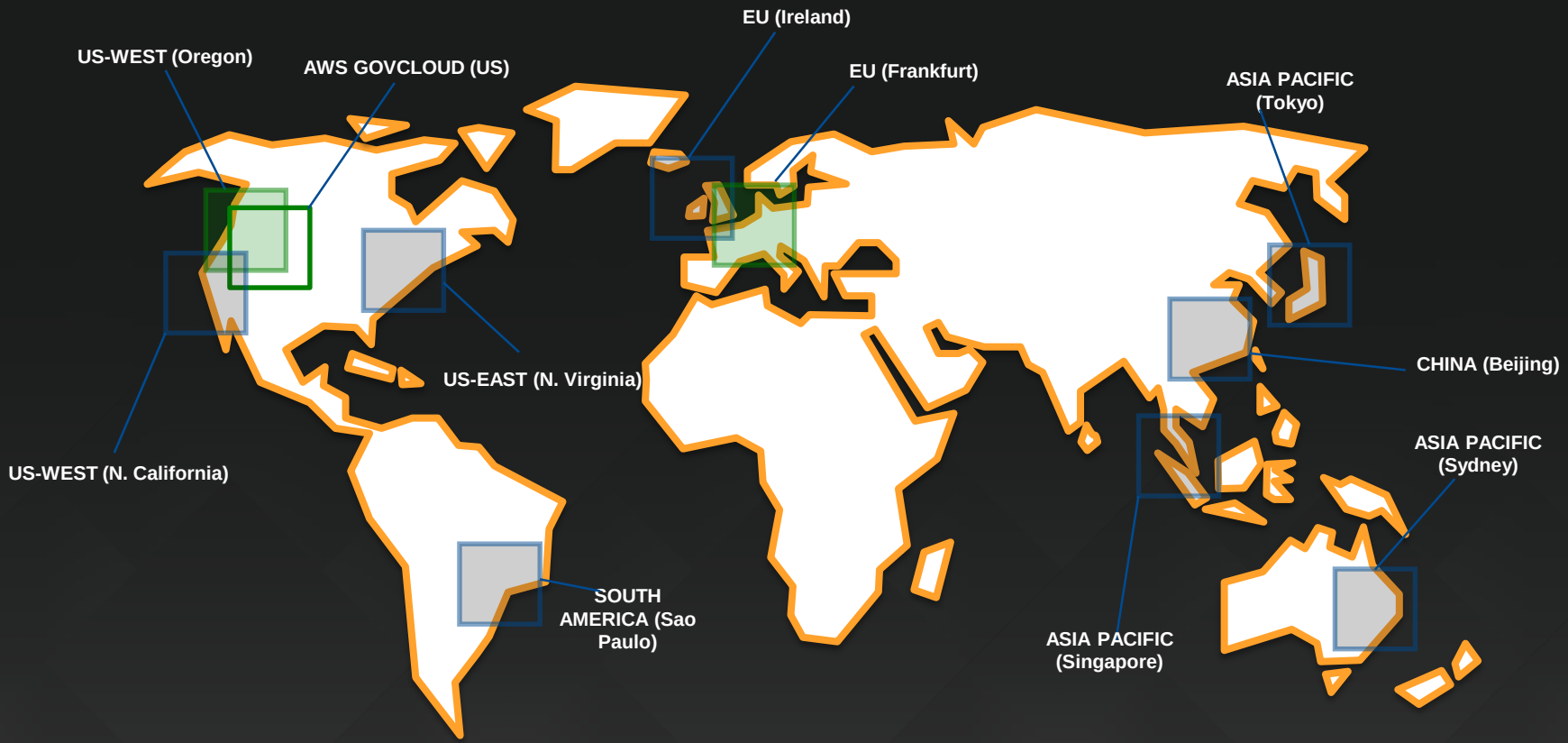


Auto Scaling is a tool and a destination. It's not the single thing that fixes everything.

What do we need first?

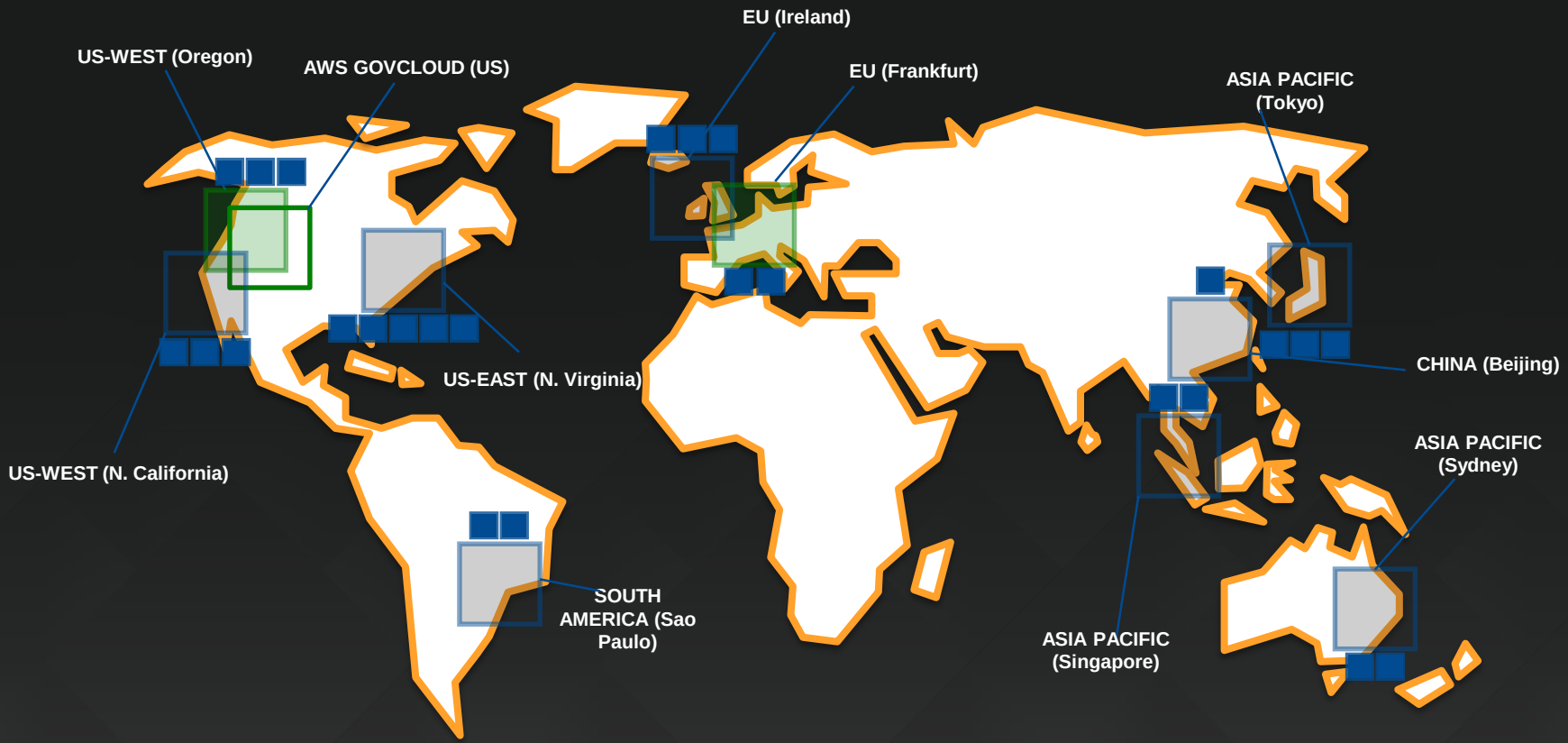
Some basics…

Regions



amazon web services

Availability Zones



amazon web services

Edge locations

amazon web services

Applications

Platform services

Foundation services

Global infrastructure

Applications

Virtual Desktops

Collaboration and Sharing

Platform services

Databases

- Relational
- No SQL
- Caching

Analytics

- Hadoop
- Real-time
- Data Warehouse
- Data Workflows

App Services

- Queuing
- Orchestration
- App Streaming
- Transcoding
- Email
- Search

Deployment & Management

- Containers
- Managed User Directories
- Dev/ops Tools
- Resource Templates
- Usage Tracking
- Monitoring and Logs

Mobile Services

- Identity
- Sync
- Mobile Analytics
- Notifications

Foundation services

Compute
(VMs, Auto Scaling and Load Balancing)

Storage
(Object, Block and Archive)

Security and Access Control

Networking

Infrastructure

Regions

Availability Zones

CDN and Points of Presence

amazon web services

AWS building blocks

Inherently highly available and fault-tolerant services

- Amazon CloudFront
- Amazon Route53
- Amazon S3
- Amazon DynamoDB
- Elastic Load Balancing
- Amazon SQS
- Amazon SNS
- Amazon SES
- Amazon SWF
- …

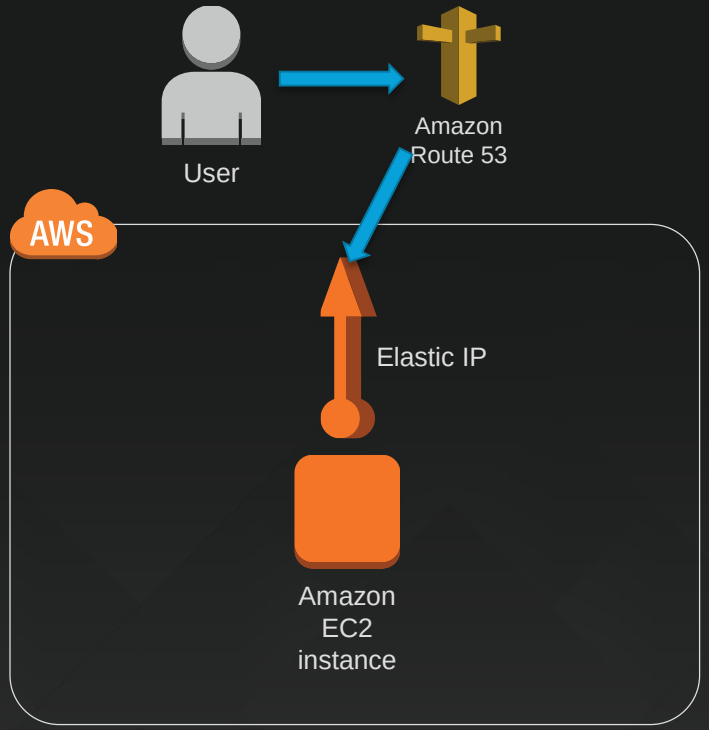
Highly available with the right architecture

- Amazon EC2
- Amazon Elastic Block Store
- Amazon RDS
- Amazon VPC

So let’s start from day 1, user 1 (you)

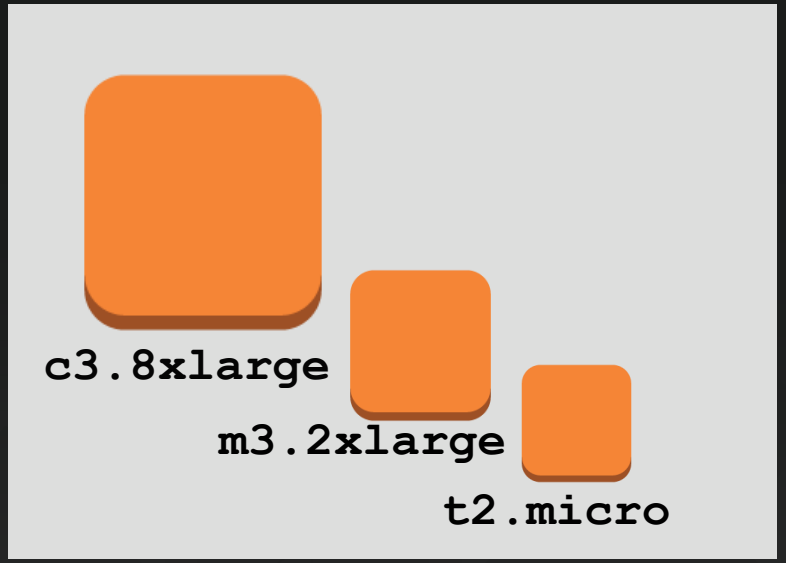
Day 1, user 1

- A single Amazon EC2 instance
  - With full stack on this host
    - Web app
    - Database
    - Management
    - And so on…
- A single Elastic IP
- Amazon Route 53 for DNS



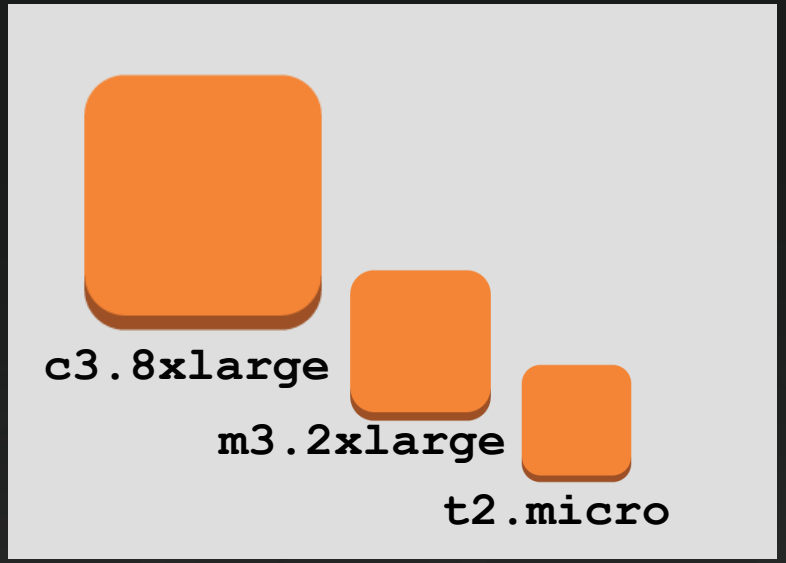
“We’re gonna need a bigger box”

- Simplest approach
- Can now leverage PIOPS
- High I/O instances
- High memory instances
- High CPU instances
- High storage instances
- Easy to change instance sizes
- Will hit an endpoint eventually



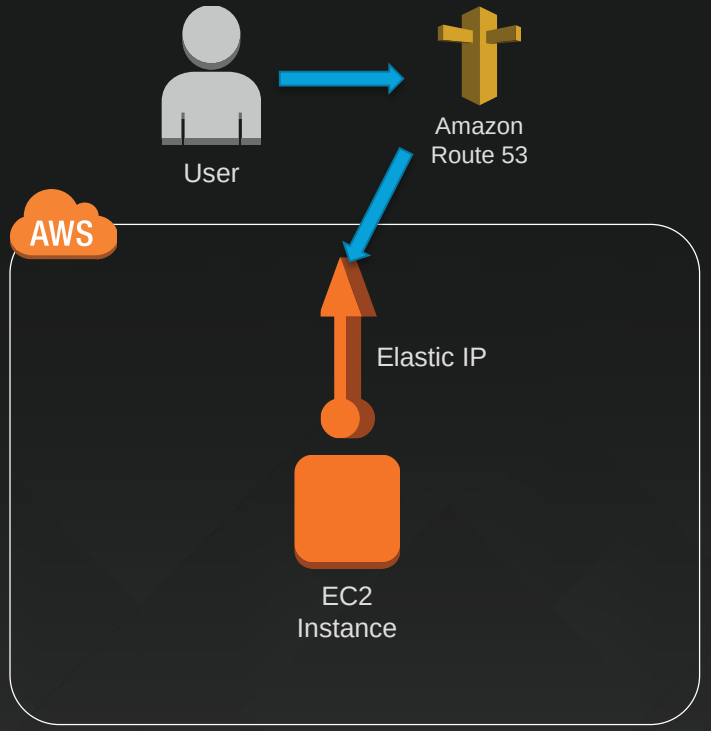
“We’re gonna need a bigger box”

- Simplest approach
- Can now leverage PIOPS
- High I/O instances
- High memory instances
- High CPU instances
- High storage instances
- Easy to change instance sizes
- **Will hit an endpoint eventually**



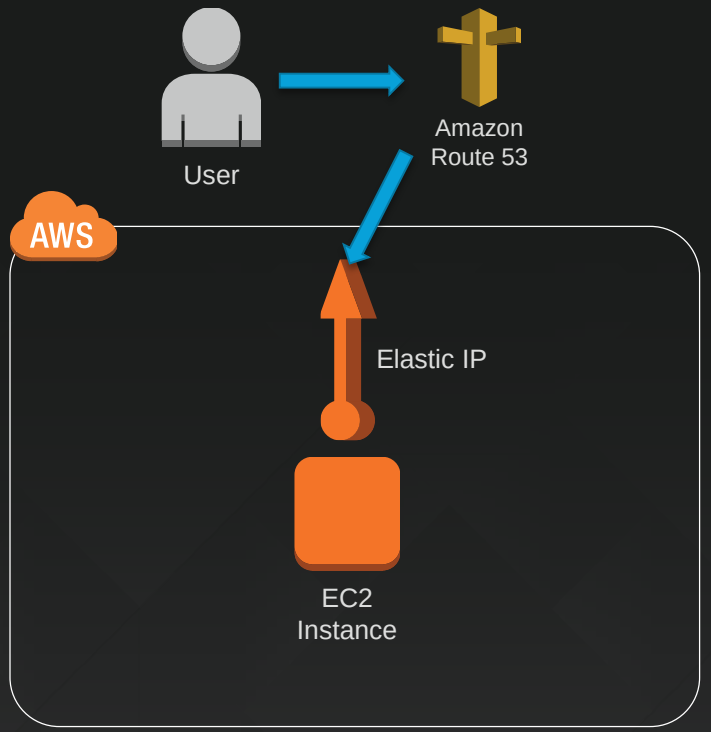
Day 1, user 1

- We could potentially get to a few hundred to a few thousand depending on application complexity and traffic
- No failover
- No redundancy
- Too many eggs in one basket



Day 1, user 1

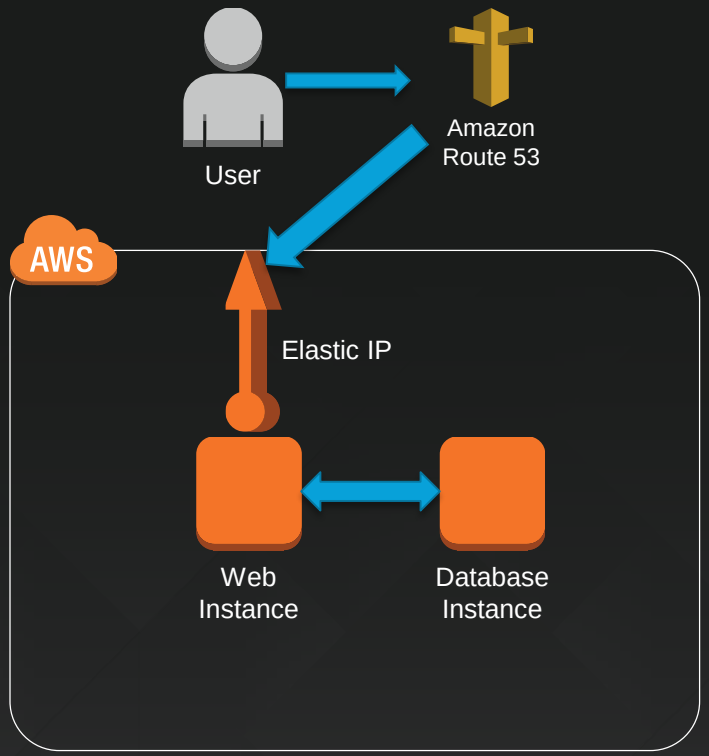
- We could potentially get to a few hundred to a few thousand depending on application complexity and traffic
- **No failover**
- **No redundancy**
- **Too many eggs in one basket**



Day 2, user > 1

First, let's separate out our single host into more than one.

- Web
- Database
  - Make use of a database service?



Database options

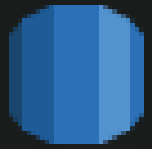
Self-managed

Database server on Amazon EC2

Your choice of database running on Amazon EC2

Bring Your Own License (BYOL)

Fully managed



Amazon RDS

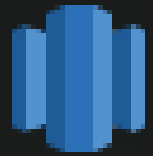
Microsoft SQL Server, Oracle, MySQL, or PostgreSQL as a managed service

Flexible licensing: BYOL or license Included

Amazon DynamoDB

Managed NoSQL database service using SSD storage

Seamless scalability
Zero administration



Amazon Redshift

Massively parallel, petabyte-scale data warehouse service

Fast, powerful, and easy to scale

But how do I choose what DB technology I need? SQL? NoSQL?

Some folks won’t like this.
But…

Start with SQL databases

Why start with SQL?

- Established and well-worn technology.
- Lots of existing code, communities, books, background, tools, and more.
- You aren’t going to break SQL DBs in your first 10 million users. No, really, you won’t.*
- Clear patterns to scalability.

*Unless you are doing something SUPER weird with the data or you have MASSIVE amounts of it, but even then SQL will have a place in your stack.

AH HA! You said “massive amounts,” and I will have massive amounts!

If your usage is such that you will be generating several TB (> 5) of data in the first year OR have an incredibly data intensive workload, then you might need NoSQL

Why else might you need NoSQL?

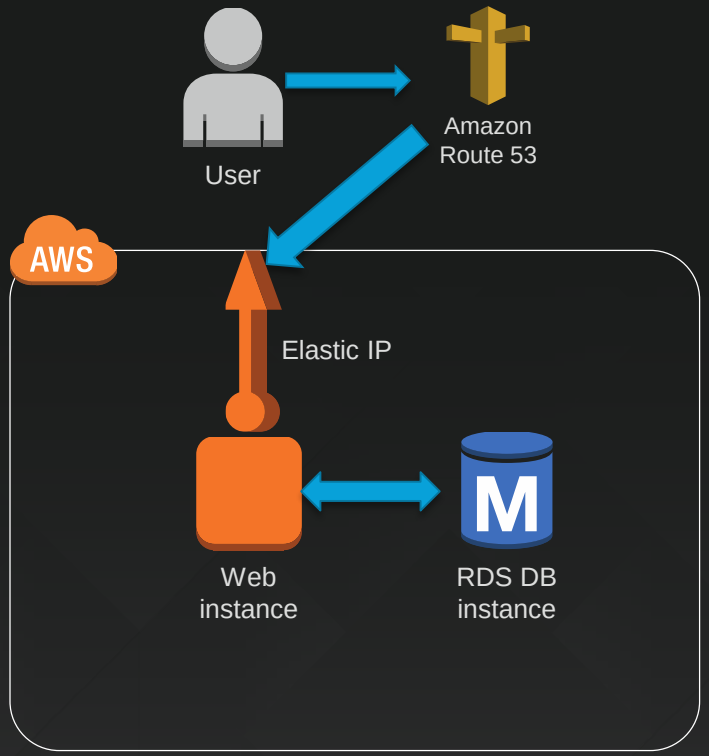
- Super low-latency applications
- Metadata-driven datasets
- Highly nonrelational data
- Need schema-less data constructs*
- Massive amounts of data (again, in the TB range)
- Rapid ingest of data (thousands of records/sec)

*Need != “It’s easier to do dev without schemas”

User > 100

First, let's separate out our single host into more than one:

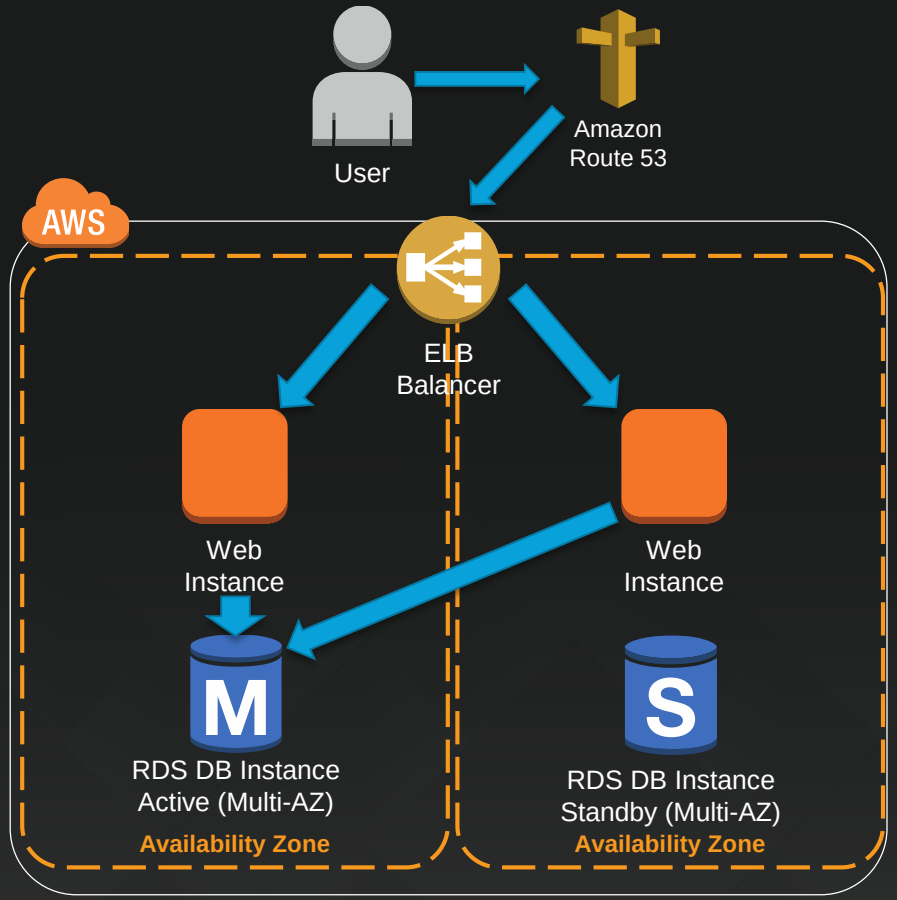
- Web
- Database
  - Use Amazon RDS to make your life easier



User > 1000

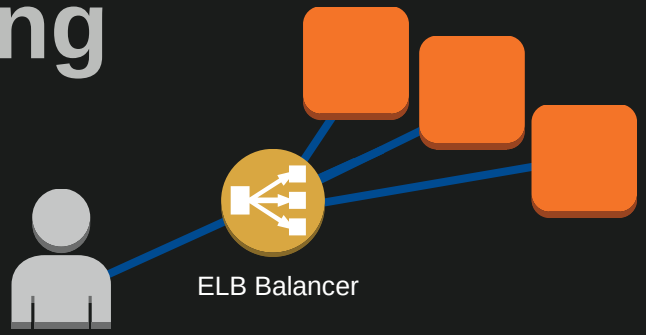
Next, let's address our lack of failover and redundancy issues:

- Elastic Load Balancing (ELB)
- Another web instance
  - In another Availability Zone
- RDS Multi-AZ



Elastic Load Balancing

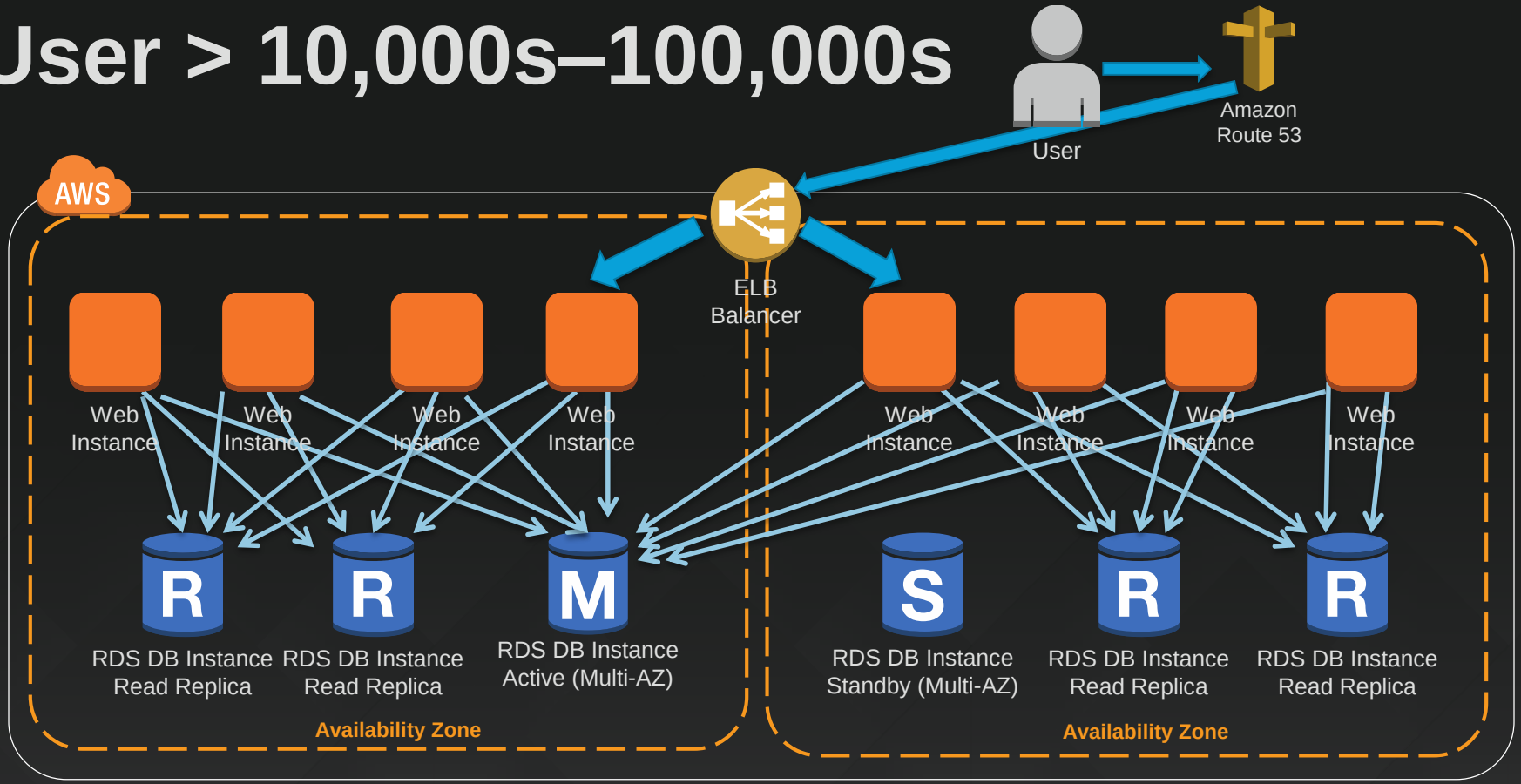
- Create highly scalable applications
- Distribute load across EC2 instances in multiple Availability Zones



Feature	Details
Available	Load balances across instances in multiple Availability Zones
Health checks	Automatically checks health of instances and takes them in or out of service
Session stickiness	Routes requests to the same instance
Secure sockets layer	Supports SSL offload from web and application servers with flexible cipher support
Monitoring	Publishes metrics to Amazon CloudWatch and can get logs of requests processed

Scaling this horizontally and vertically will get us pretty far (tens to hundreds of thousands)

User > 10,000s–100,000s



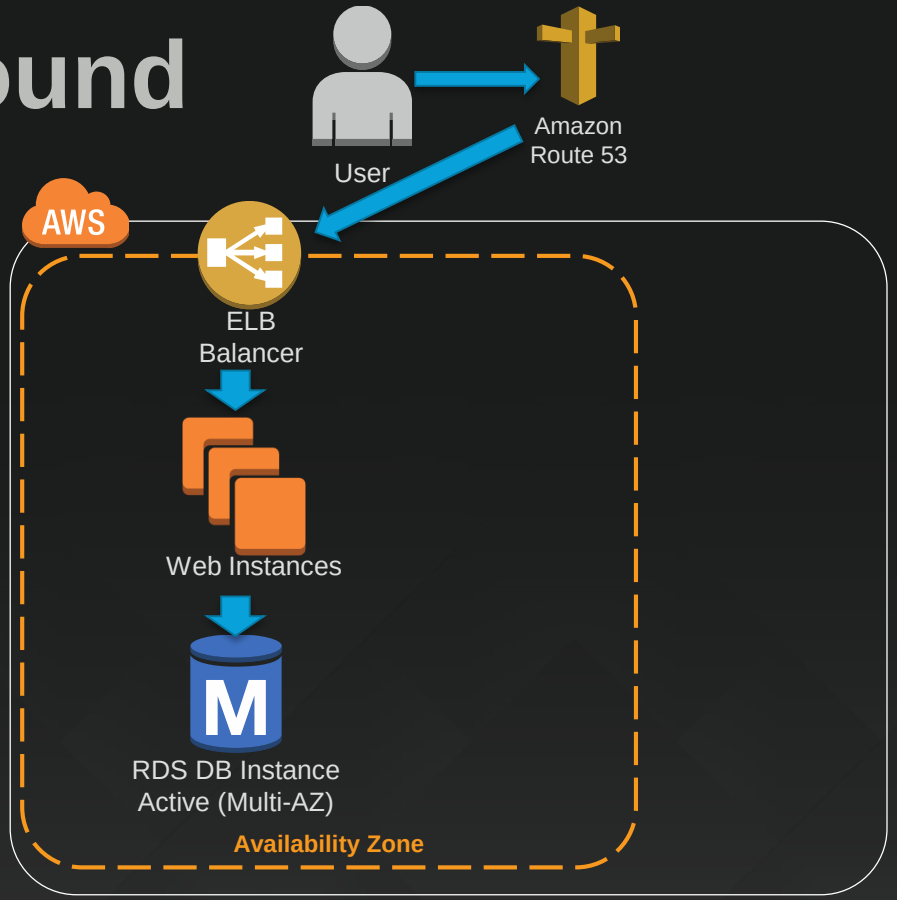
amazon web services

This will take us pretty far, but we care about *performance* and *efficiency*, so let's improve further

Shift some load around

Let's lighten the load on our web and database instances:

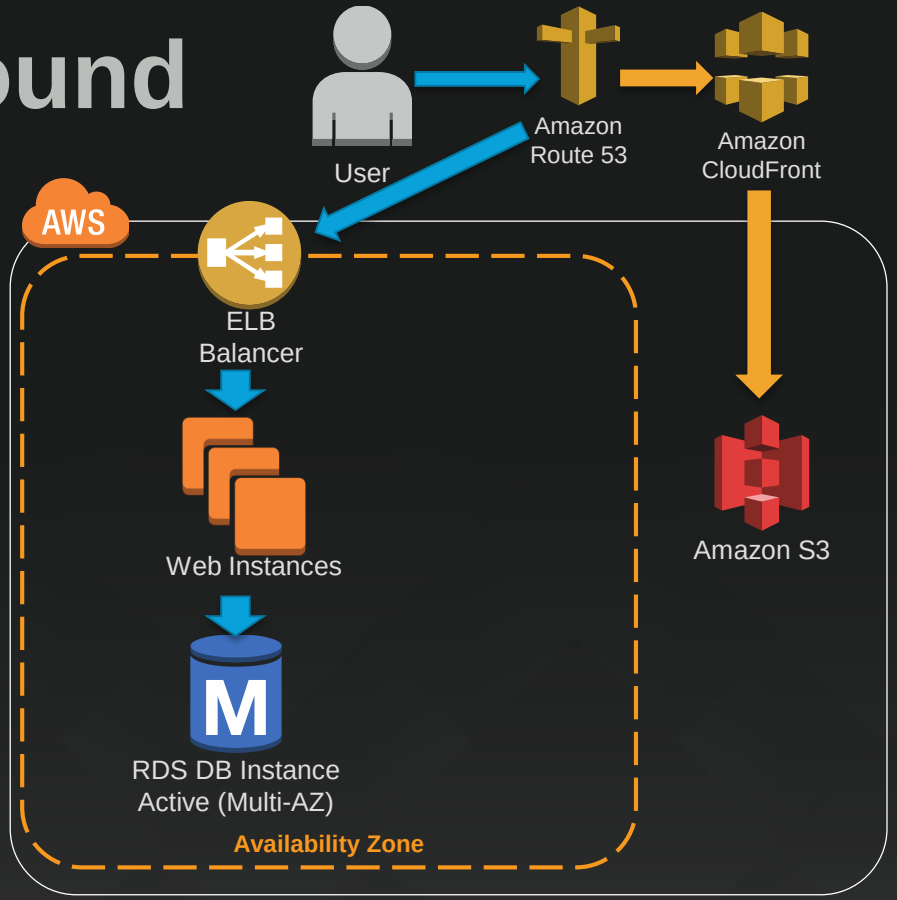
- Move static content from the web instance to Amazon S3 and Amazon CloudFront
- Move session/state and DB caching to Amazon ElastiCache or Amazon DynamoDB



Shift some load around

Let's lighten the load on our web and database instances:

- **Move static content from the web instance to Amazon S3 and Amazon CloudFront**
- Move session/state and DB caching to Amazon ElastiCache or Amazon DynamoDB



amazon web services

Amazon S3

Amazon S3 is cloud storage for the Internet:

- Object-based storage
- 11 9s of durability
- Good for things like the following:
  - Static assets (CSS, JS, images, videos)
  - Backups
  - Logs
  - Ingest of files for processing
- “Infinitely scalable”
- Objects up to 5 TB in size

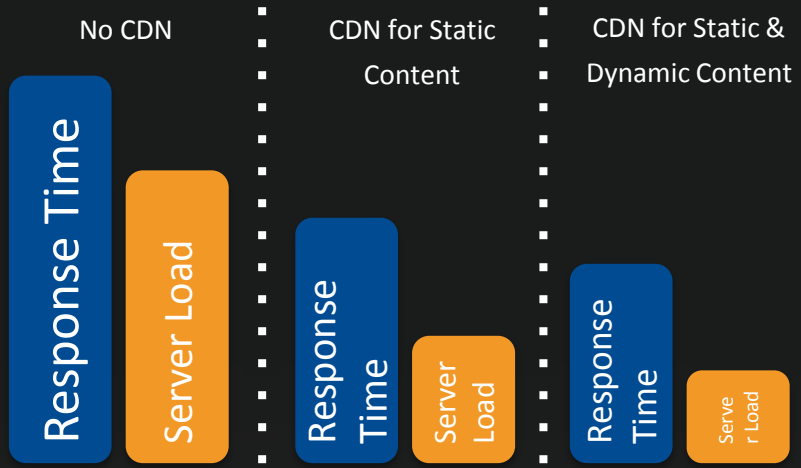
Amazon S3

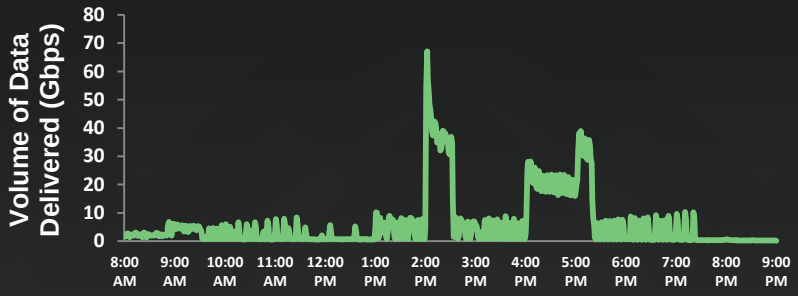
- Can host static websites
- Supports fine-grained permission control
- Ties in well with Amazon CloudFront
- Ties in with Amazon EMR
- Acts as a logging endpoint for S3, CloudFront, Billing, ELB, AWS CloudTrail, and more
- Supports encryption at transit and at rest
- Reduced redundancy is 1/3 cheaper
- Amazon Glacier for super long-term storage at 1/3 the cost of S3

Amazon CloudFront

Amazon CloudFront is a web service for scalable content delivery:

- Cache static content at the edge for faster delivery
- Helps lower load on origin infrastructure
- Dynamic and static content
- Streaming video
- Zone apex support
- Custom SSL certificates
- Low TTLs (as short as 0 seconds)
- Lower costs for origin fetches (between Amazon S3 / Amazon EC2 and Amazon CloudFront)
- Optimized to work with Amazon EC2, Amazon S3, Elastic Load Balancing, and Amazon Route 53



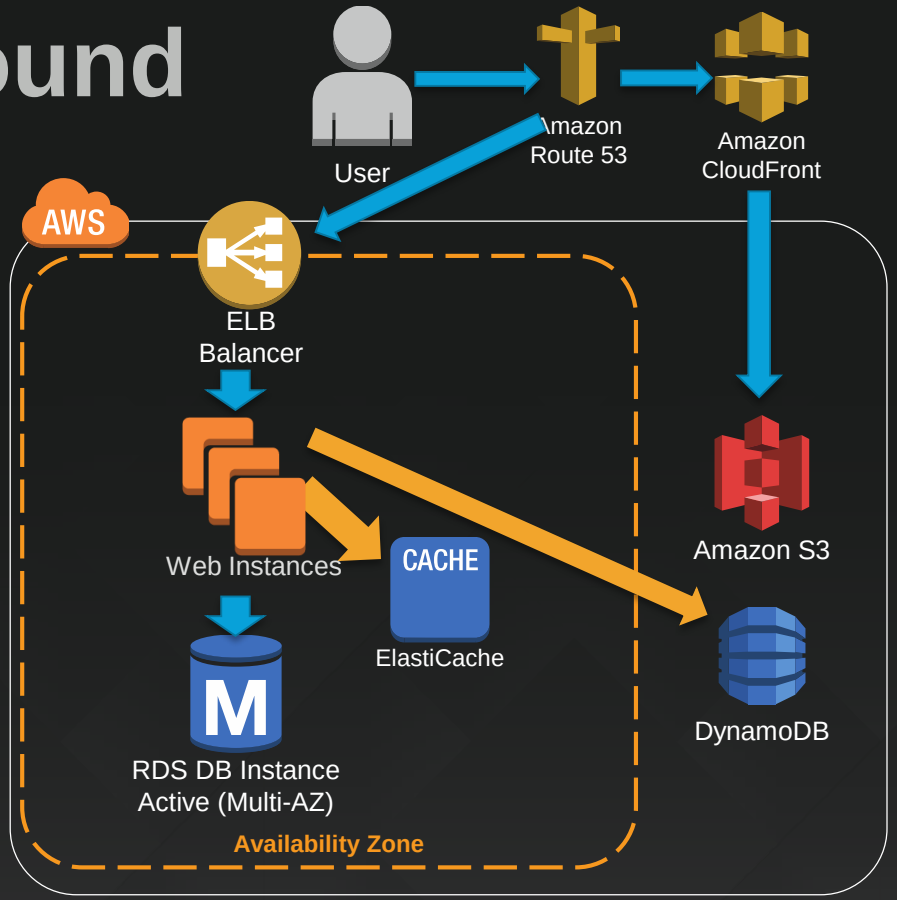


amazon web services

Shift some load around

Let's lighten the load on our web and database instances:

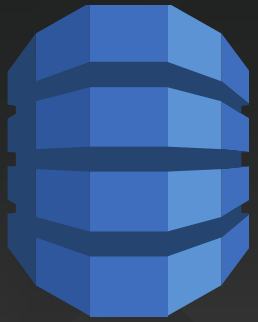
- Move static content from the web instance to Amazon S3 and Amazon CloudFront
- **Move session/state and DB caching to Amazon ElastiCache or Amazon DynamoDB**



amazon web services

Amazon DynamoDB

- Managed, provisioned throughput NoSQL database
- Fast, predictable performance
- Fully distributed, fault tolerant architecture
- JSON support (NEW)
- Items up to 400 KB (NEW)



Feature	Details
Provisioned throughput	Dial up or down provisioned read/write capacity
Predictable performance	Average single digit millisecond latencies from SSD-backed infrastructure
Strong consistency	Be sure you are reading the most up to date values
Fault tolerant	Data replicated across Availability Zones
Monitoring	Integrated with Amazon CloudWatch
Secure	Integrates with AWS Identity and Access Management (IAM)
Amazon EMR	Integrates with Amazon EMR for complex analytics on large datasets

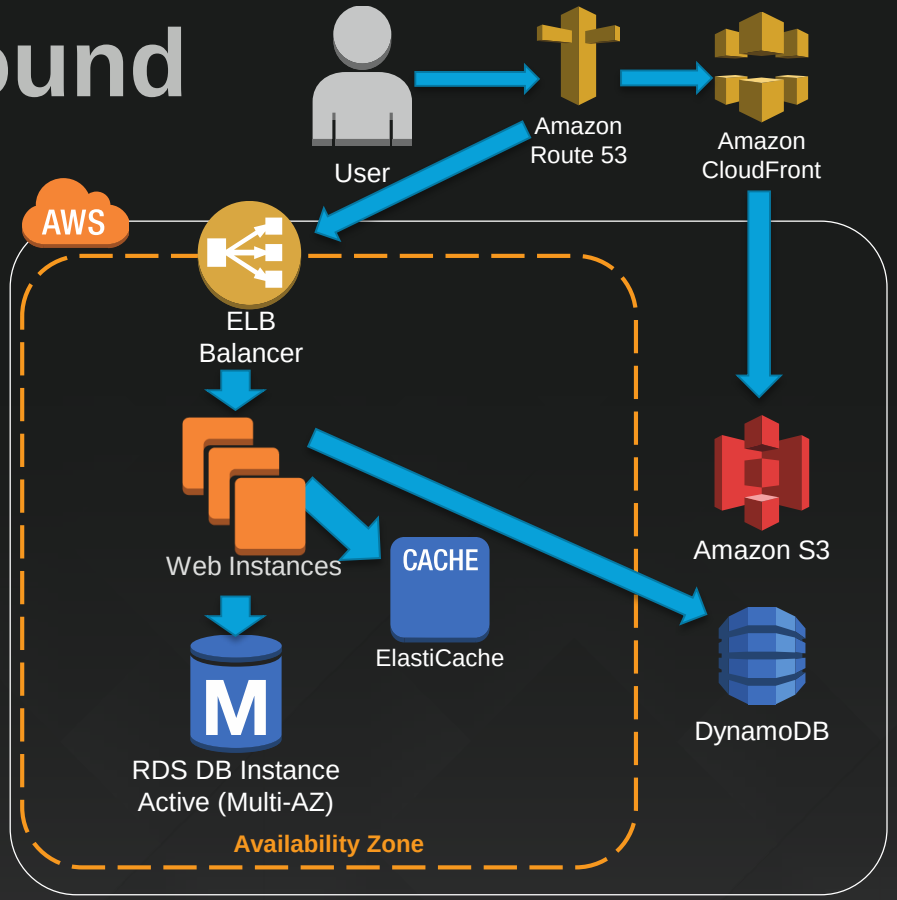
Amazon ElastiCache

- Hosted Memcached and Redis
  - Speaks same API as traditional open source Memcached and Redis
- Scale from one to many nodes
- Self-healing (replaces dead instance)
- Very fast (single digit ms speeds usually (or less))
- Local to a single AZ for Memcache, with no persistence or replication
- With Redis, can put a replica in a different AZ with persistence
- Use the AWS Auto Discovery client to simplify clusters growing and shrinking without affecting your application

Shift some load around

Let's lighten the load on our web and database instances:

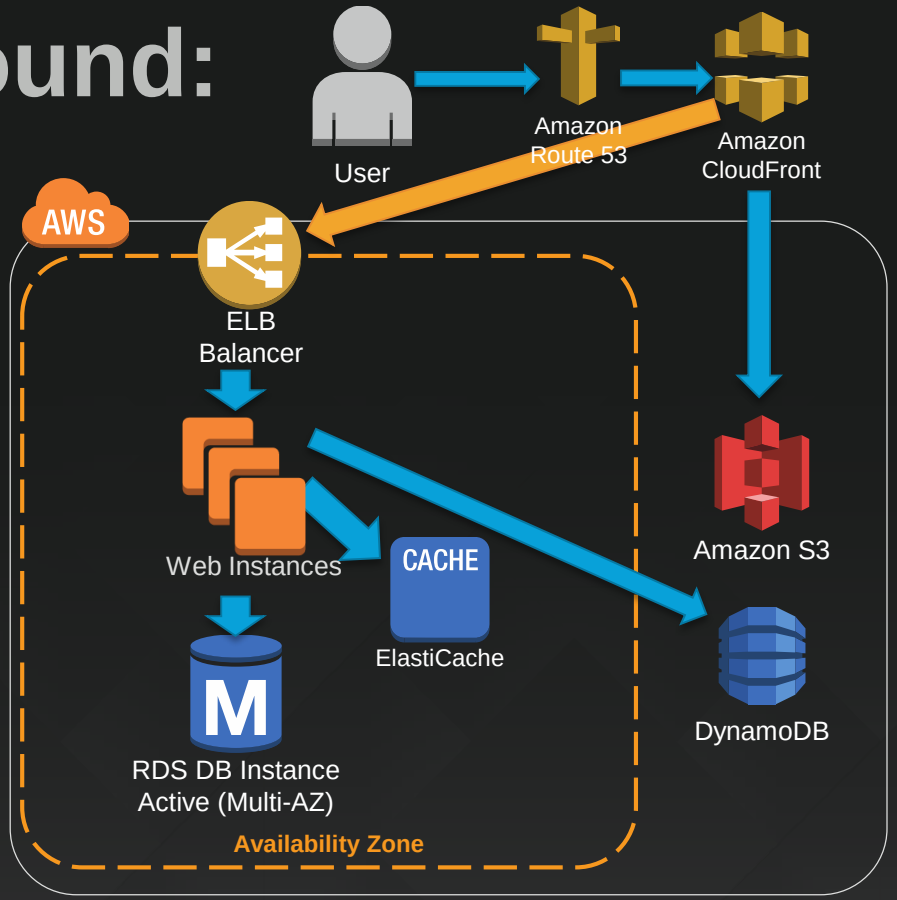
- Move static content from the web instance to Amazon S3 and Amazon CloudFront
- Move session/state and DB caching to ElastiCache or DynamoDB
- Move dynamic content from the ELB balancer to Amazon CloudFront



Shift some load around:

Let's lighten the load on our web and database instances:

- Move static content from the web instance to Amazon S3 and Amazon CloudFront
- Move session/state and DB caching to ElastiCache or DynamoDB
- **Move dynamic content from the ELB balancer to Amazon CloudFront**



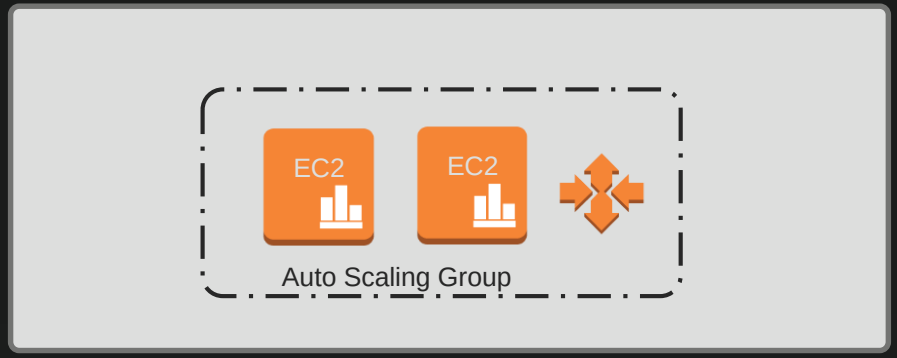
amazon web services

Now that our web tier is much more lightweight, we can revisit the beginning of our talk…

Auto Scaling!

Auto Scaling

Automatic resizing of compute clusters based on demand

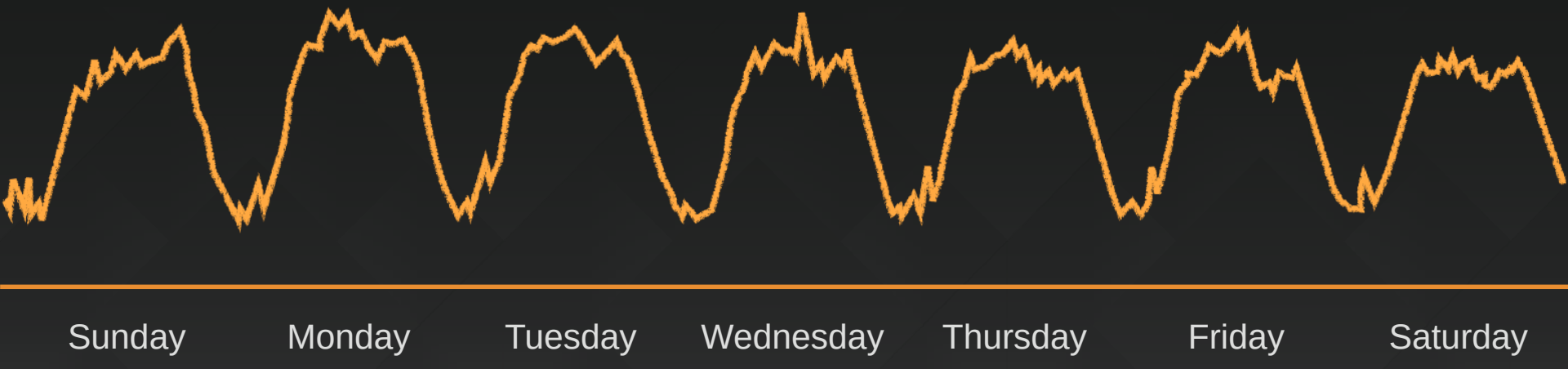


Feature	Details
Control	Define minimum and maximum instance pool sizes and when scaling and cool down occurs.
Integrated with Amazon CloudWatch	Use metrics gathered by CloudWatch to drive scaling.
Instance types	Run Auto Scaling for on-demand and Spot Instances. Compatible with VPC.

aws autoscaling create-auto-scaling-group
--auto-scaling-group-name **MyGroup**
--launch-configuration-name **MyConfig**
--min-size **4**
--max-size **200**
--availability-zones **us-west-2c, us-west-2b**

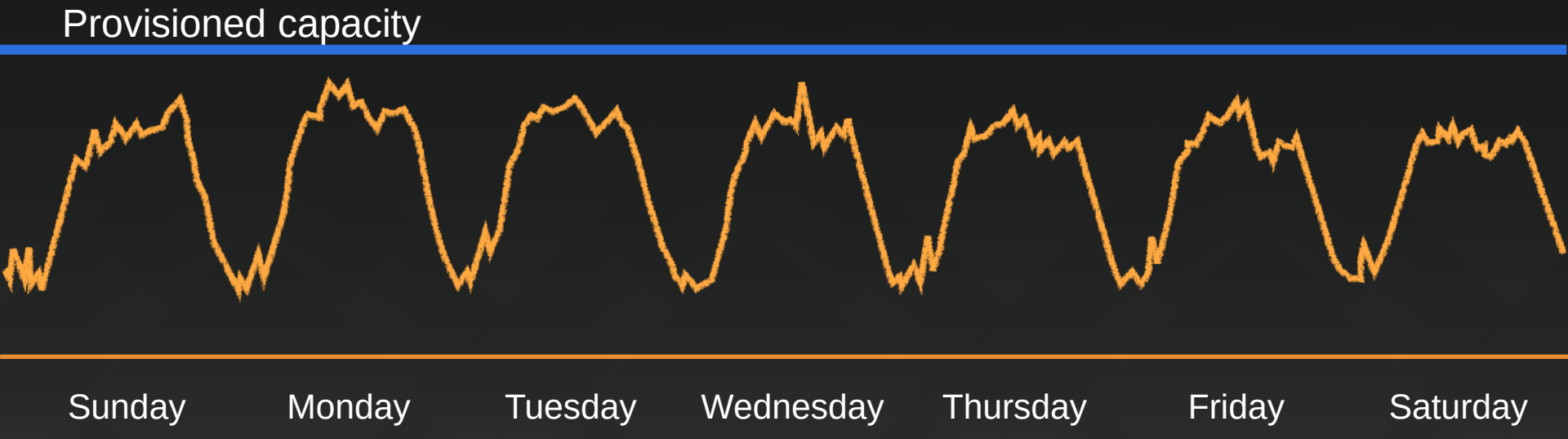
amazon web services

Typical weekly traffic to Amazon.com

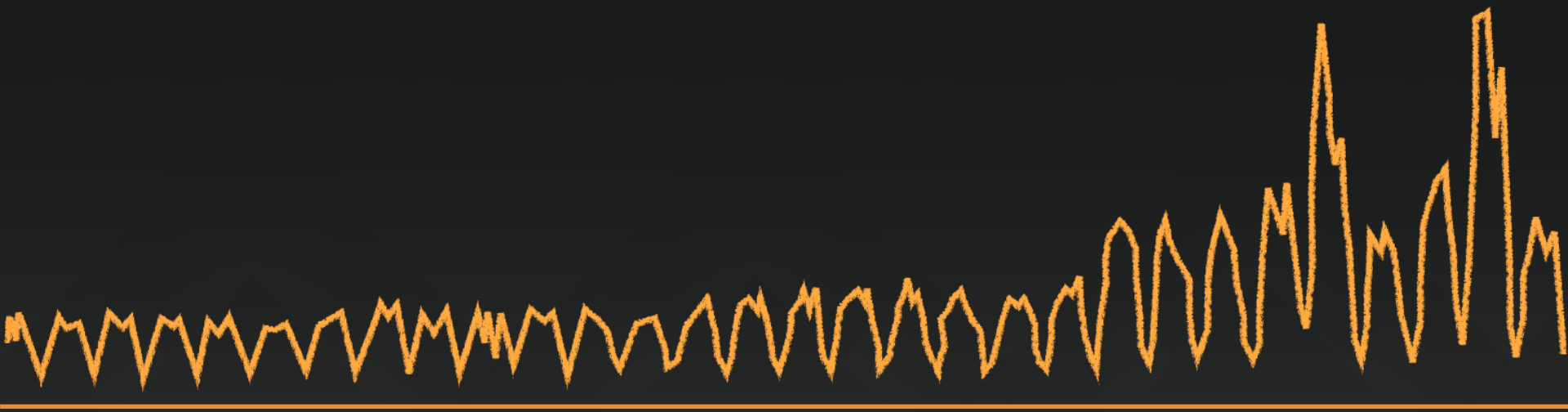


amazon web services

Typical weekly traffic to Amazon.com

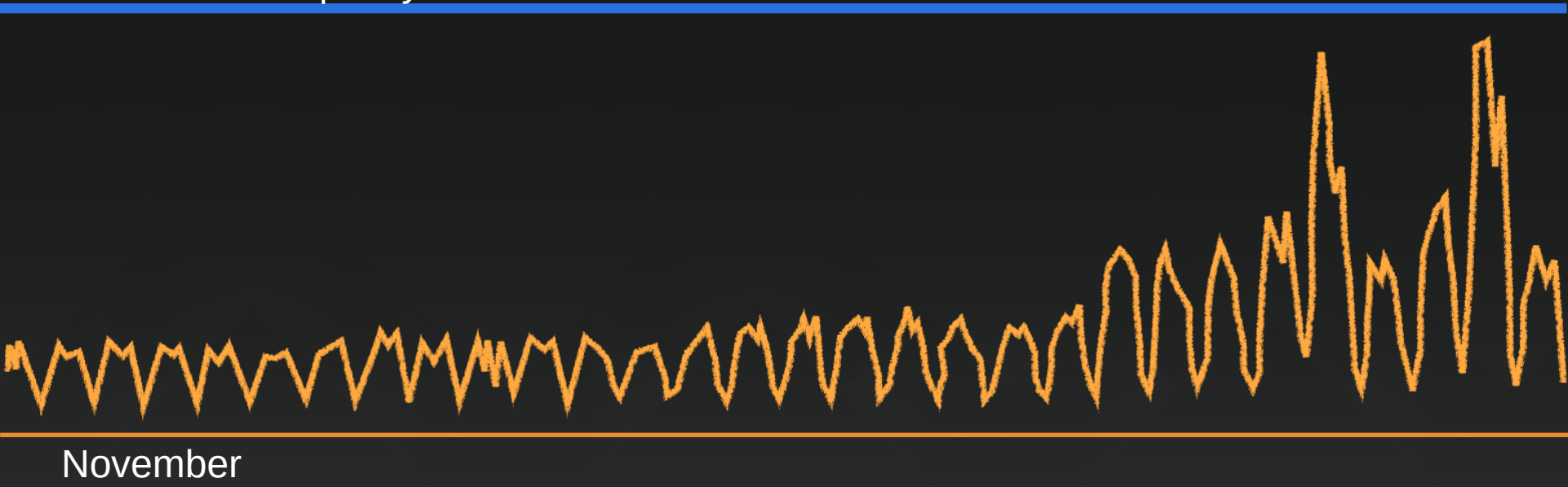


November traffic to Amazon.com

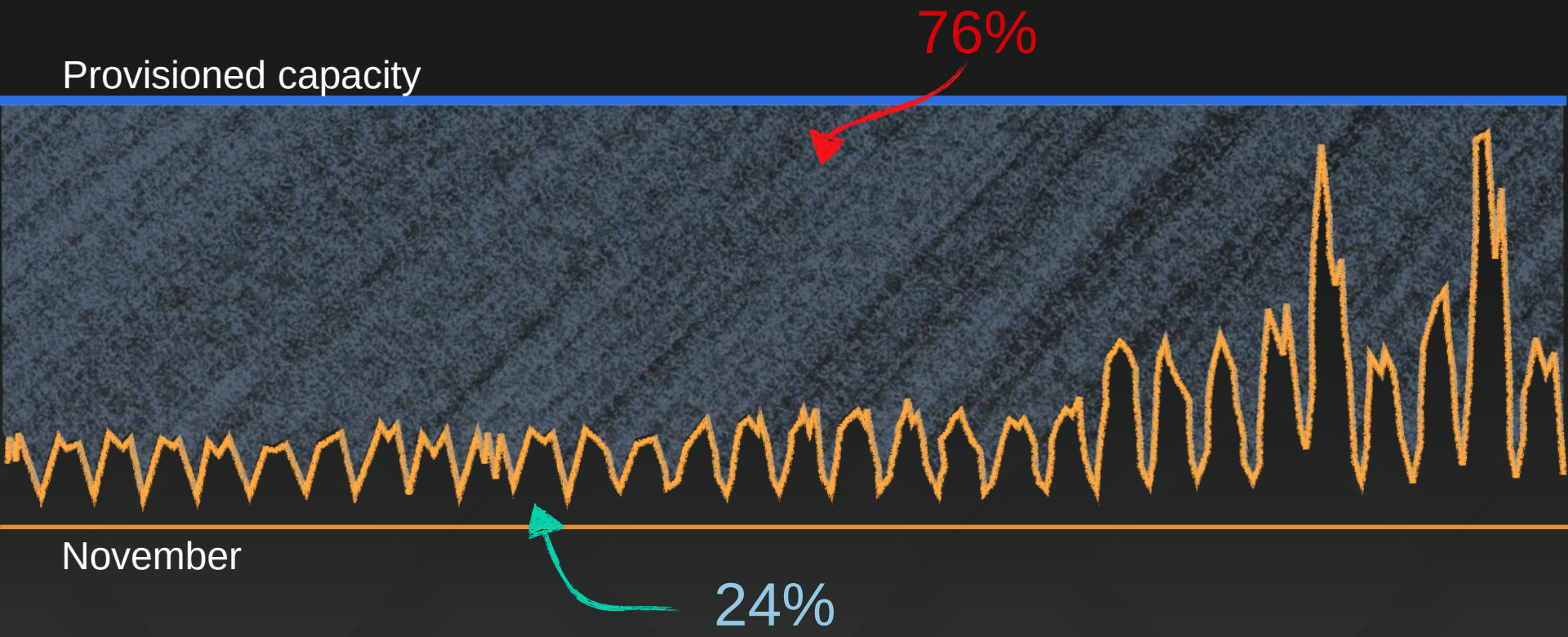


November

November traffic to Amazon.com

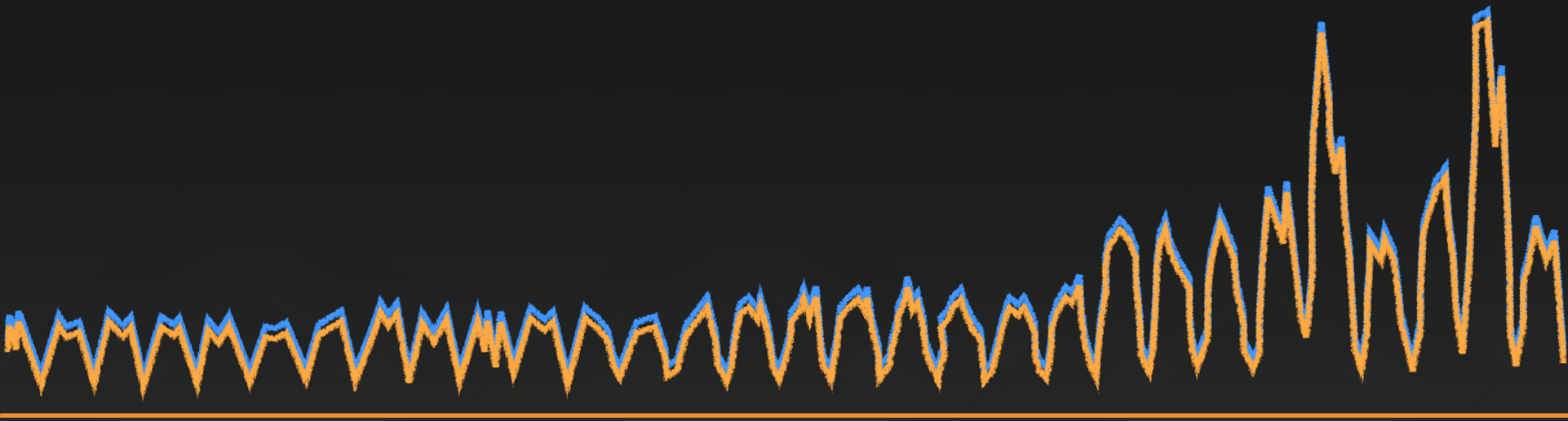


November traffic to Amazon.com



amazon web services

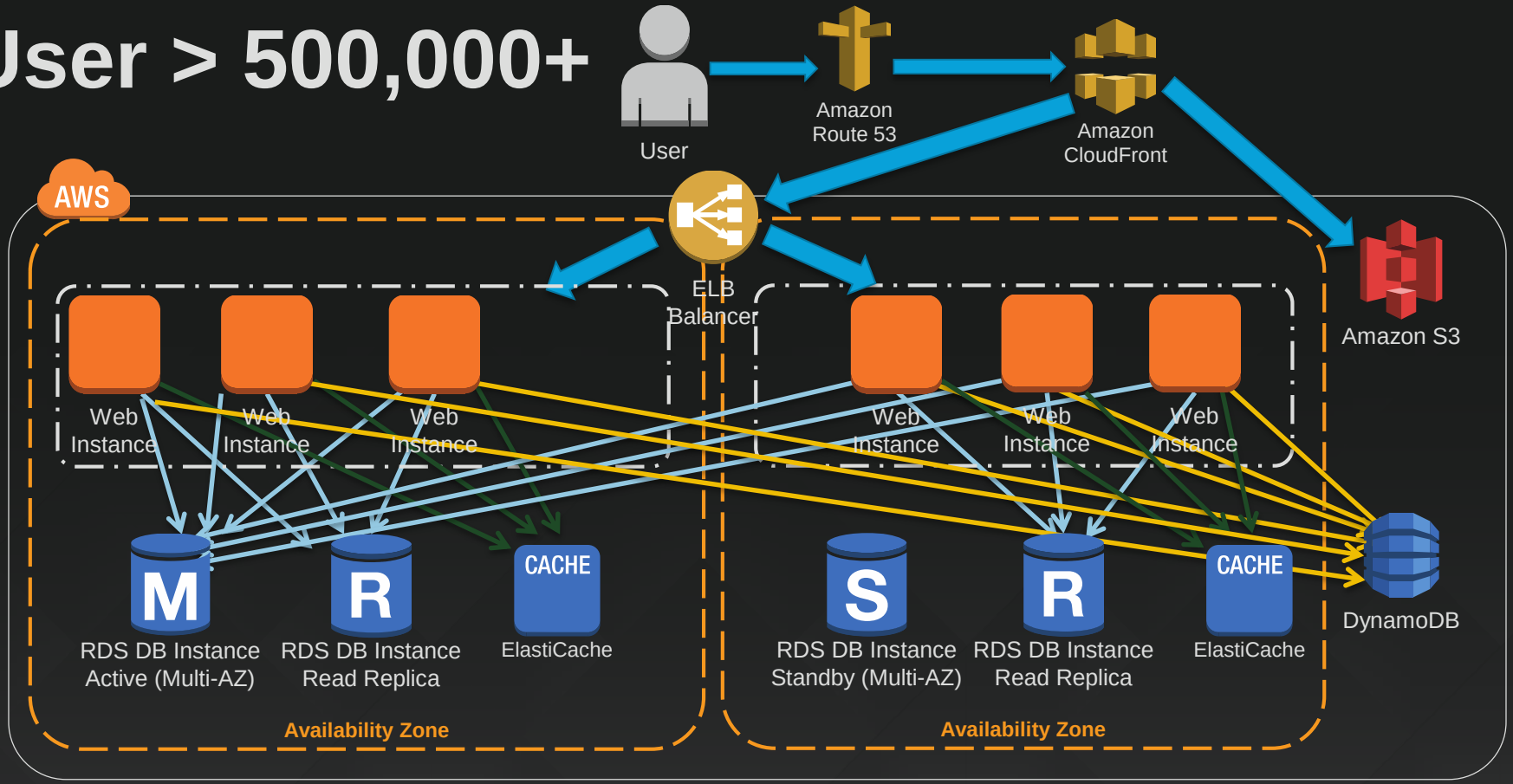
November traffic to Amazon.com



November

Auto Scaling
lets you do this!

User > 500,000+



amazon web services

Use automation

Managing your infrastructure will become an ever increasing important part of your time. Use tools to automate repetitive tasks:

- Tools to manage AWS resources
- Tools to manage software and configuration on your instances
- Automated data analysis of logs and user actions

AWS application management solutions

Higher-level services

Do it yourself

AWS
Elastic Beanstalk

AWS
OpsWorks

AWS
CloudFormation

Amazon EC2

Convenience

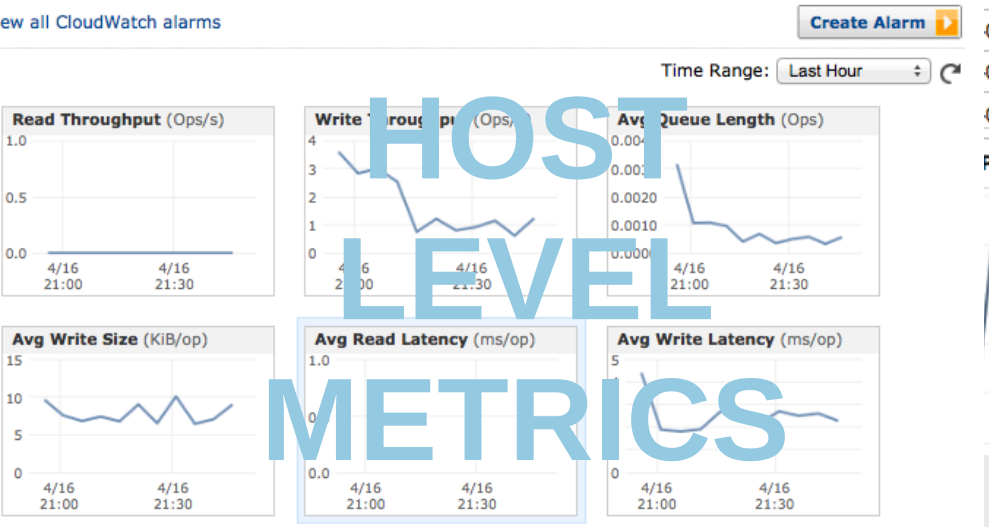
Control

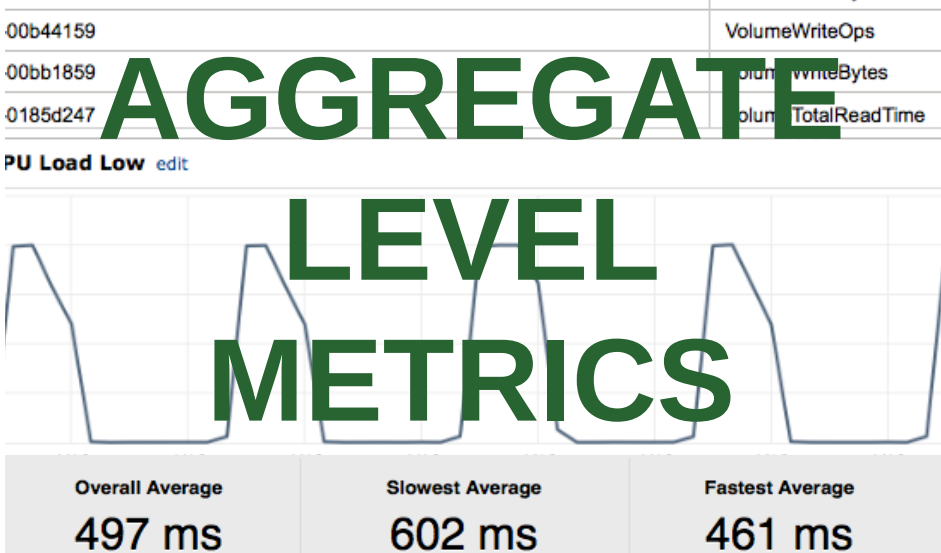
amazon web services

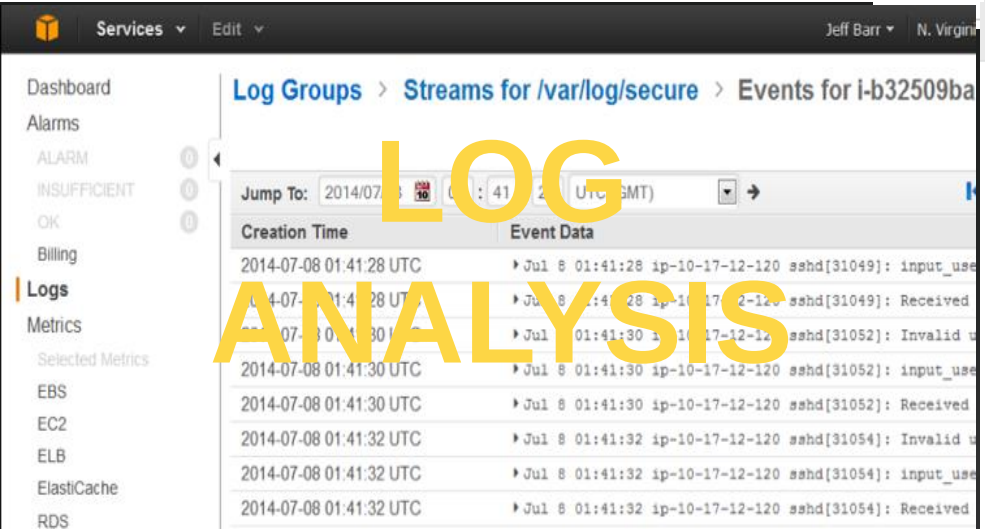
User > 500,000+

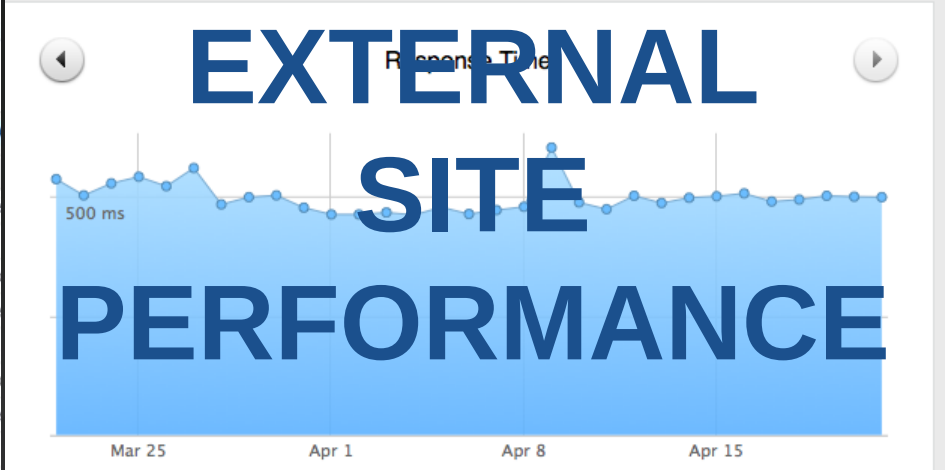
You'll potentially start to run into issues with speed and performance of your applications:

- Make sure you have monitoring, metrics, and logging in place
  - If you can't build it internally, outsource it! (Third-party SaaS)
- Pay attention to what customers are saying works well vs. what doesn't, and use this direction
- Try to squeeze as much performance out of each service/component


ew all CloudWatch alarms
Create Alarm
Time Range: Last Hour
Read Throughput (Ops/s)
Avg Queue Length (Ops)
Avg Write Size (KiB/op)
Avg Read Latency (ms/op)
Avg Write Latency (ms/op)
HOST LEVEL METRICS


00b44159
VolumeWriteOps
00bb1859
TotalReadTime
0185d247
PU Load Low edit
AGGREGATE LEVEL METRICS
Overall Average
497 ms
Slowest Average
602 ms
Fastest Average
461 ms


Services
Edit
Jeff Barr
Dashboard
Alarms
Billing
Logs
Metrics
EBS
EC2
ELB
ElastiCache
RDS
Log Groups
Streams for /var/log/secure
Events for i-b32509ba
Jump To:
Creation Time
Event Data
2014-07-08 01:41:28 UTC
2014-07-08 01:41:30 UTC
2014-07-08 01:41:32 UTC
LOG ANALYSIS


500 ms
Mar 25
Apr 1
Apr 8
Apr 15
EXTERNAL SITE PERFORMANCE

There are further improvements to be made in breaking apart our web/app layer

SOA…what does this mean?

amazon web services

SOAing

- Move services into their own tiers/modules. Treat each of these as 100% separate pieces of your infrastructure, and scale them independently.

- Amazon.com and AWS do this extensively! It offers flexibility and greater understanding of each component.

Loose coupling + SOA = winning

In the early days, if someone has a service for it already, opt to use that instead of building it yourself.

DON’T REINVENT THE WHEEL.

Examples:

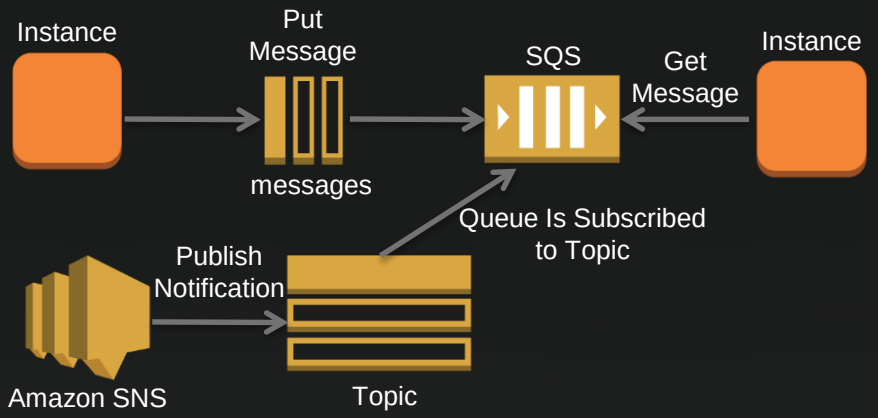
- Email
- Queuing
- Transcoding
- Search
- Databases
- Monitoring
- Metrics
- Logging
- Compute

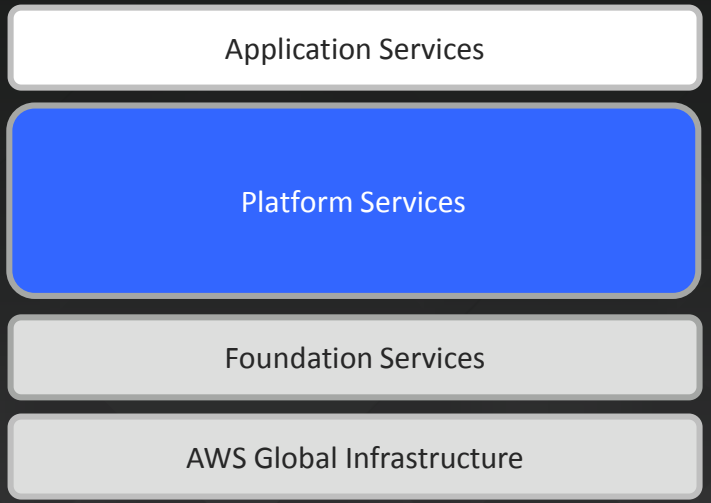
amazon web services

Application Services

Amazon SQS

Reliable, highly scalable queue service for storing messages as they travel between instances





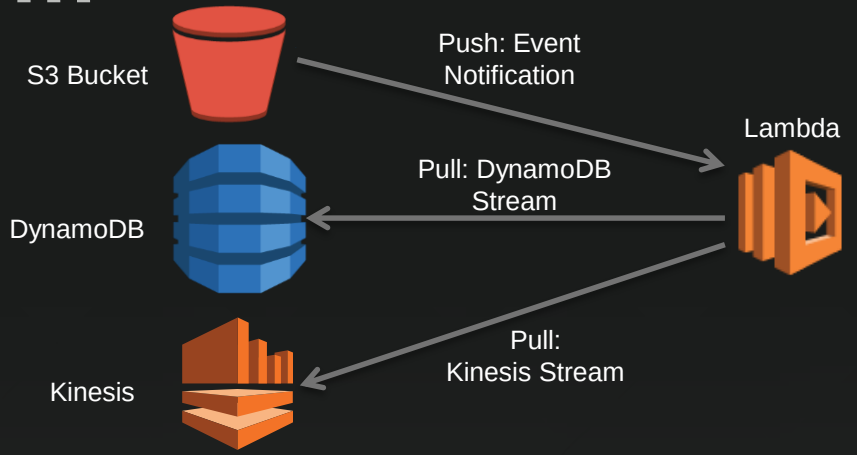
Feature	Details
Reliable	Messages stored redundantly across multiple availability zones
Simple	Simple APIs to send and receive messages
Scalable	Unlimited number of messages
Secure	Authentication of queues to ensure controlled access

amazon web services

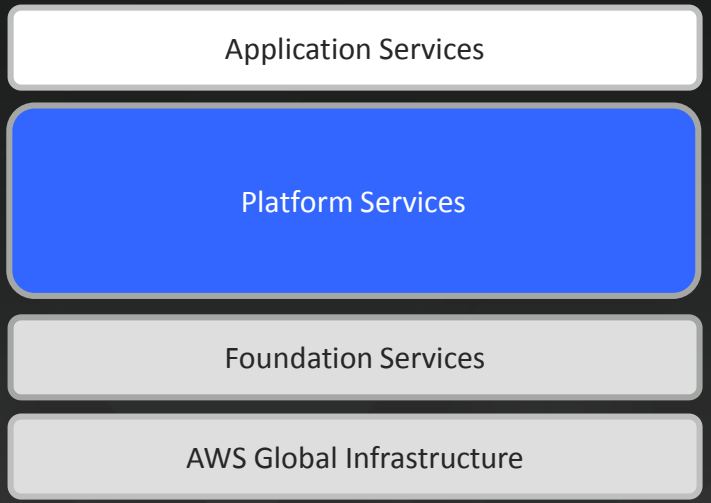
Compute / Platform

AWS Lambda

Event-driven compute, connective tissue for AWS services



Feature	Details
Stateless	Request-driven code called Lambda functions triggered by events
Easy	Fixed OS and language—JavaScript
Management	AWS owns and manages the infrastructure
Scaling	Implicit scaling; just make requests

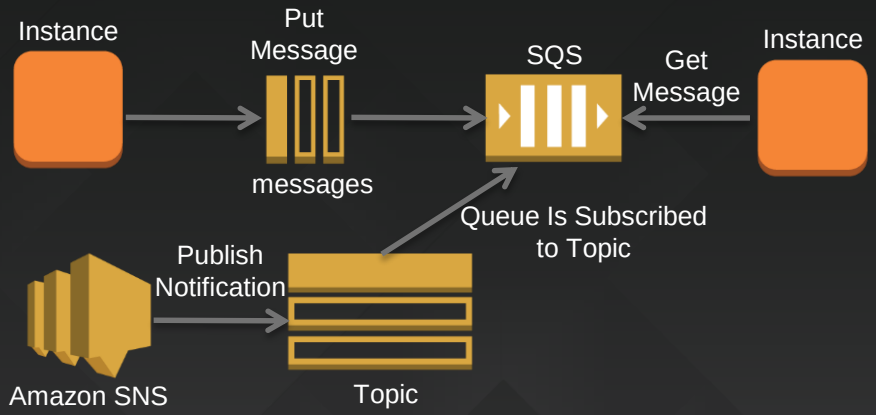


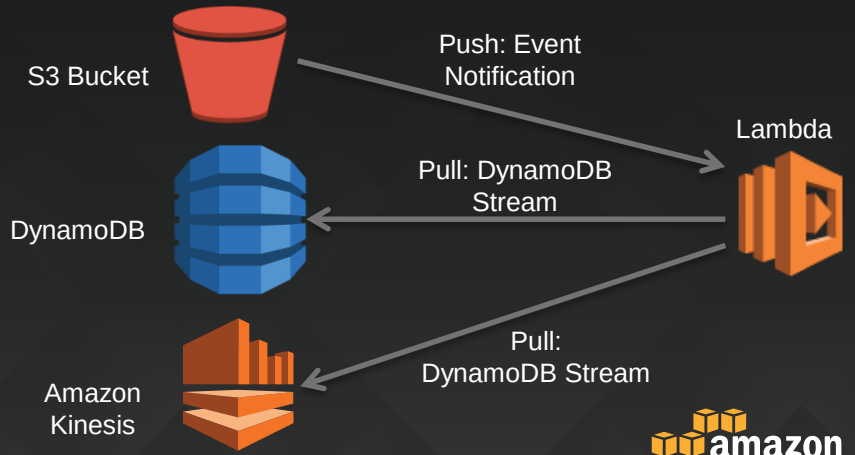
amazon web services

Loose coupling sets you free!

The looser they're coupled, the bigger they scale

- Independent components
- Design everything as a black box
- Decouple interactions
- Favor services with built-in redundancy and scalability rather than building your own





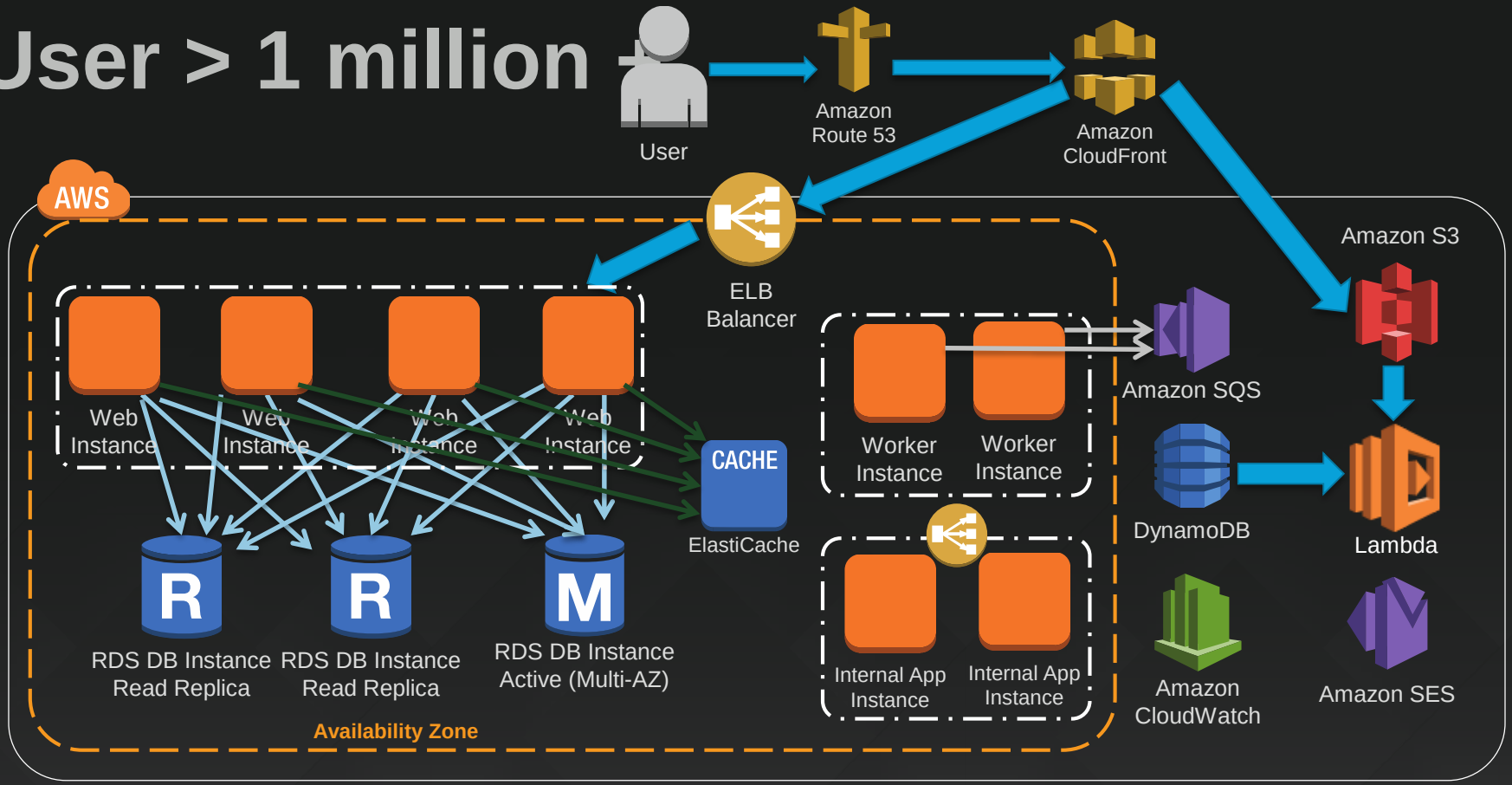
amazon web services

User > 1 million +

Reaching a million and above is going to require some bit of all the previous things:

- Multi-AZ
- Elastic Load Balancing between tiers
- Auto Scaling
- Service Oriented Architecture
- Serving content smartly (Amazon S3/CloudFront)
- Caching off DB
- Moving state off tiers that auto scale

User > 1 million



amazon web services

The next big steps

User > 5 million–10 million

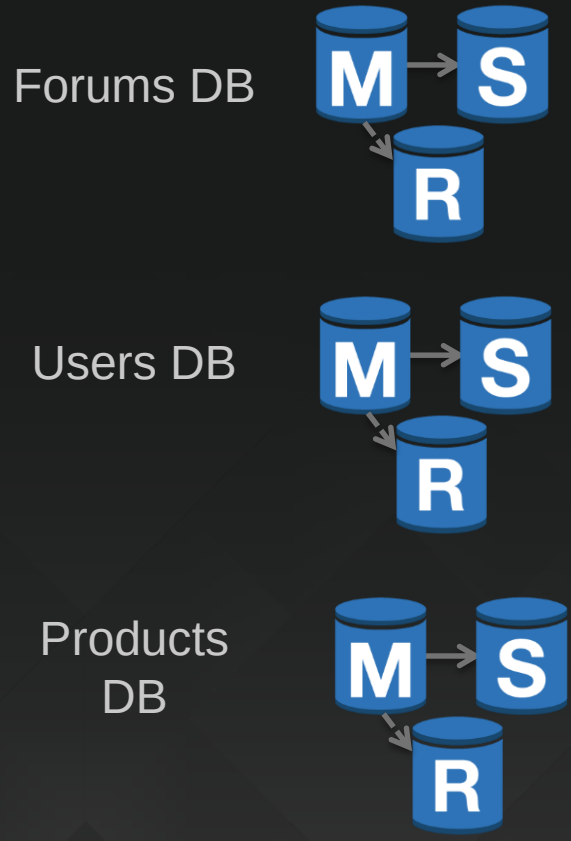
You'll potentially start to run into issues with your database around contention on the write master.

How can you solve it?

- Federation—splitting into multiple DBs based on function
- Sharding—splitting one data set up across multiple hosts
- Moving some functionality to other types of DBs (NoSQL, Graph)

Database federation

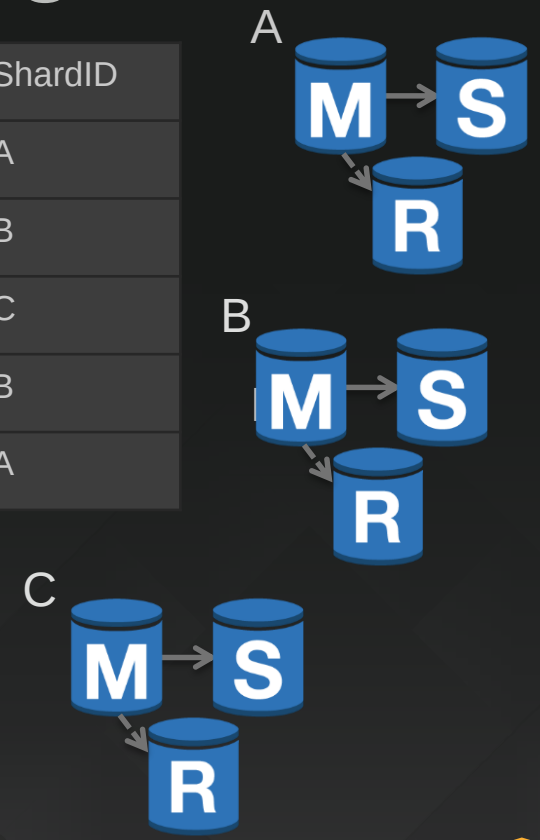
- Split up databases by function/purpose
- Harder to do cross-function queries
- Essentially delaying the need for something like sharding/NoSQL until much further down the line
- Won't help with single huge functions/tables



Sharded horizontal scaling

- More complex at the application layer
- ORM support can help
- No practical limit on scalability
- Operation complexity/sophistication
- Shard by function or key space
- RDBMS or NoSQL

User	ShardID
002345	A
002346	B
002347	C
002348	B
002349	A



Shifting functionality to NoSQL

- Similar in a sense to federation
- Again, think about the earlier points for when you need NoSQL vs SQL
- Leverage hosted services like DynamoDB
- Some use cases:
  - Leaderboards/scoring
  - Rapid ingest of clickstream/log data
  - Temporary data needs (cart data)
  - “Hot” tables
  - Metadata/lookup tables

DynamoDB

A quick review

A quick review

- Multi-AZ your infrastructure.
- Make use of self-scaling services—ELB, Amazon S3, Amazon SNS, Amazon SQS, Amazon SWF, Amazon SES, and more.
- Build in redundancy at every level.
- Start with SQL. Seriously.
- Cache data both inside and outside your infrastructure.
- Use automation tools in your infrastructure.

A quick review continued

- Make sure you have good metrics/monitoring/logging tools in place
- Split tiers into individual services (SOA)
- Use Auto Scaling once you’re ready for it
- Don’t reinvent the wheel
- Move to NoSQL if and when it makes sense

Putting all this together means we should now easily be able to handle 10+ million users!

To infinity...

User > 10 million

Iterating on top of the patterns seen here will get you up and over 100 million users

User > 10 million

- More fine-tuning of your application
- More SOA of features/functionality
- Going from Multi-AZ to multi-region
- Possibly start to build custom solutions
- Deep analysis of your entire stack

Next steps?

READ!

- aws.amazon.com/documentation
- aws.amazon.com/architecture
- aws.amazon.com/start-ups

START USING AWS

- aws.amazon.com/free/

Next steps?

ASK FOR HELP!

- forums.aws.amazon.com
- aws.amazon.com/premiumsupport/
- Your Account Manager
- A Solutions Architect

THANKS FOR LISTENING!

Brett Hollman

amazon web services

Thank You

SAN FRANCISCO

amazon web services