AWSOME DAY

Architecting for the Cloud – Best Practices

Shanna Chang
Solutions Architect
10/28/2020

Architecting Approaches for AWS

Lift-and-shift

- **Deploy existing apps in AWS with minimal re-design**
- Good strategy if starting out on AWS, or if application can't be re-architected due to cost or resource constraints
- Primarily use core services such as EC2, EBS, VPC

Cloud-optimized

- **Evolve architecture for existing app to leverage AWS services**
- Gain cost and performance benefits from using AWS services such as Auto Scaling Groups, RDS, SQS, and so on

Cloud-native architecture

- **Architect app to be cloud-native from the outset**
- Leverage the full AWS portfolio
- Truly gain all the benefits of AWS (security, scalability, cost, durability, low operational burden, etc)

aws

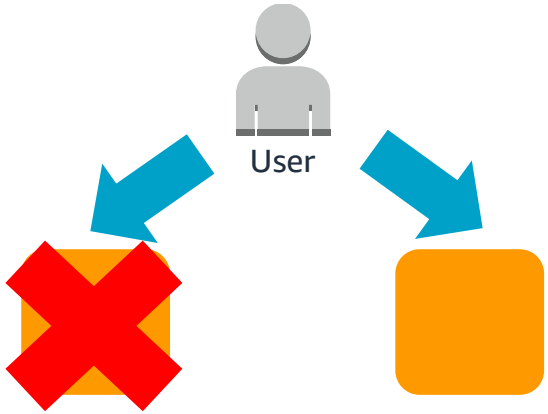
Cloud Architecture Best Practices

1. Design for failure and nothing fails
2. Build security in every layer
3. Leverage different storage options
4. Implement elasticity
5. Think parallel
6. Loose coupling sets you free
7. Don’t fear constraints

Design for Failure and Nothing Fails

Avoid Single Points of Failure

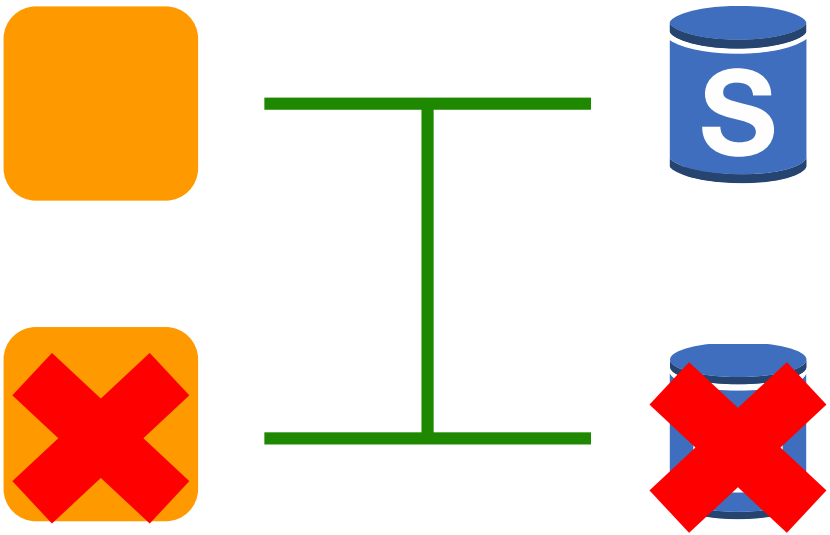
Implement redundancy where possible so that single failures don't bring down an entire system.



If one instance goes down, another is available.

aws

Avoid Single Points of Failure



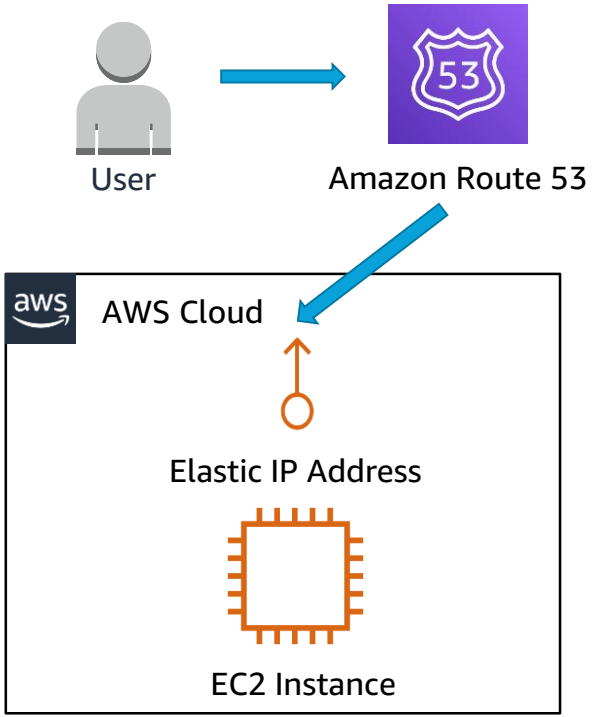
Goal:

Applications should continue to function even if the underlying application component fails, communication is lost or physical hardware fails, is removed/replaced.

aws

Design for Failure

- Amazon Route 53 for DNS
- A single Elastic IP
  - Gives a server a static Public IP address
- A single Elastic Compute Cloud (EC2)
  - Full stack on single host
    - Web application
    - Database
    - Management, etc…

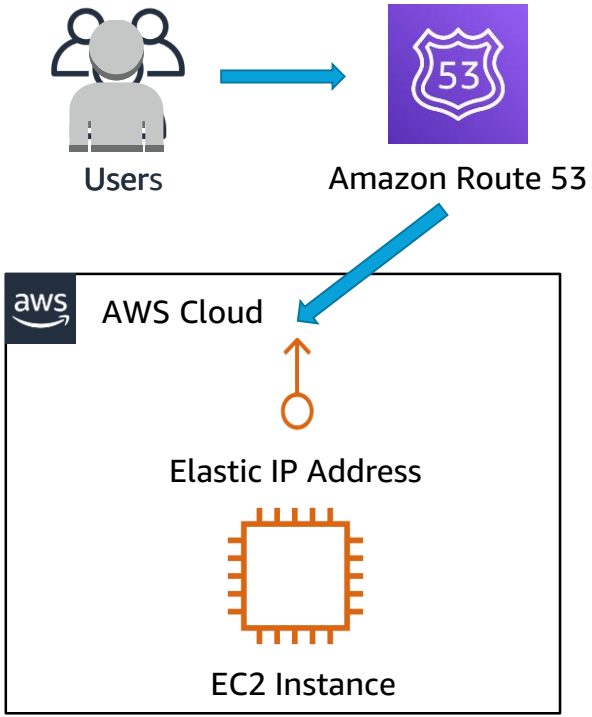


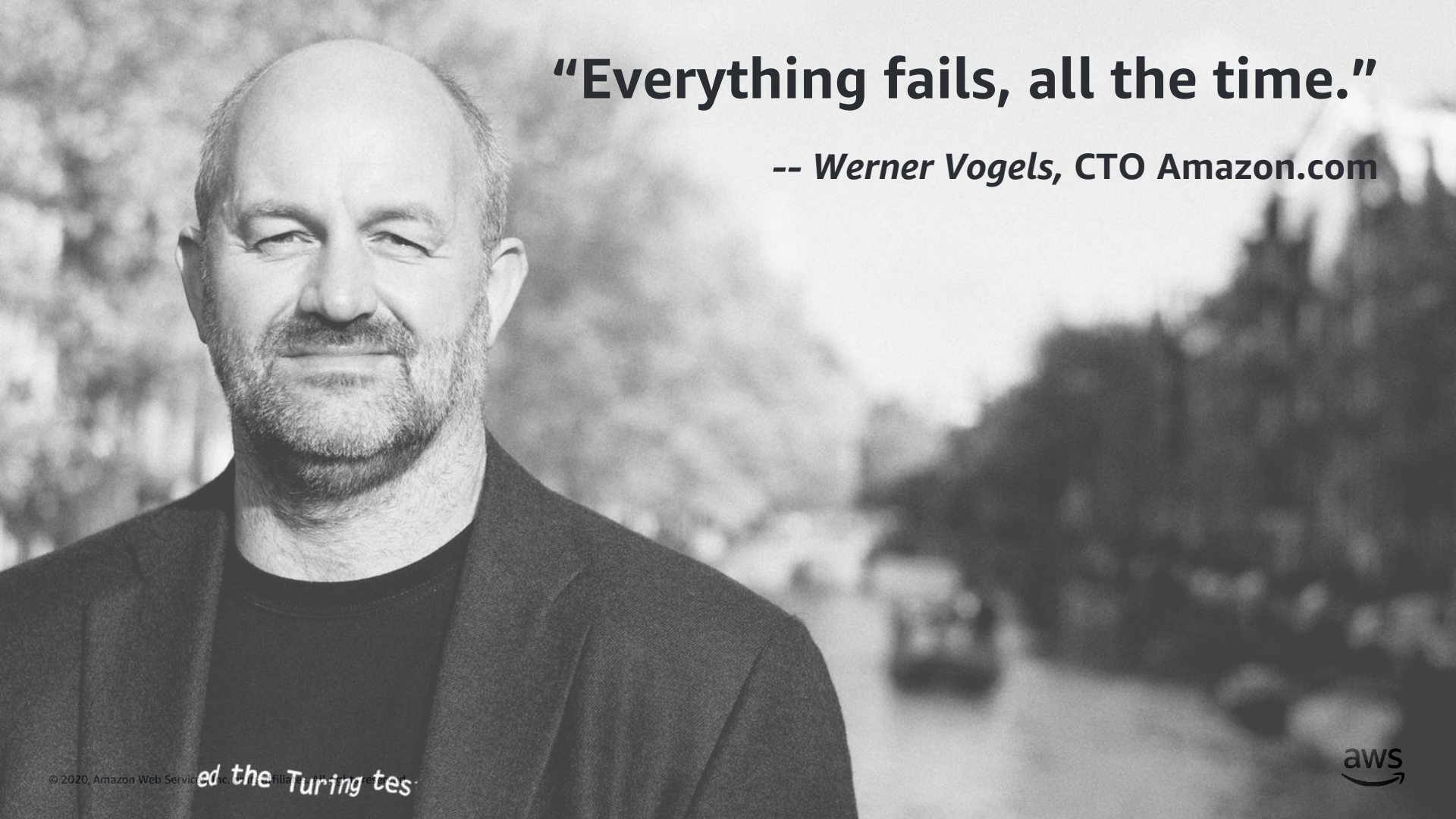
Design for Failure

We could potentially get to a few hundred to a few thousand users depending on application complexity and traffic, but…

There may be difficulty scaling to many more users due to:

- **All eggs in one basket**
- **No failover or redundancy**

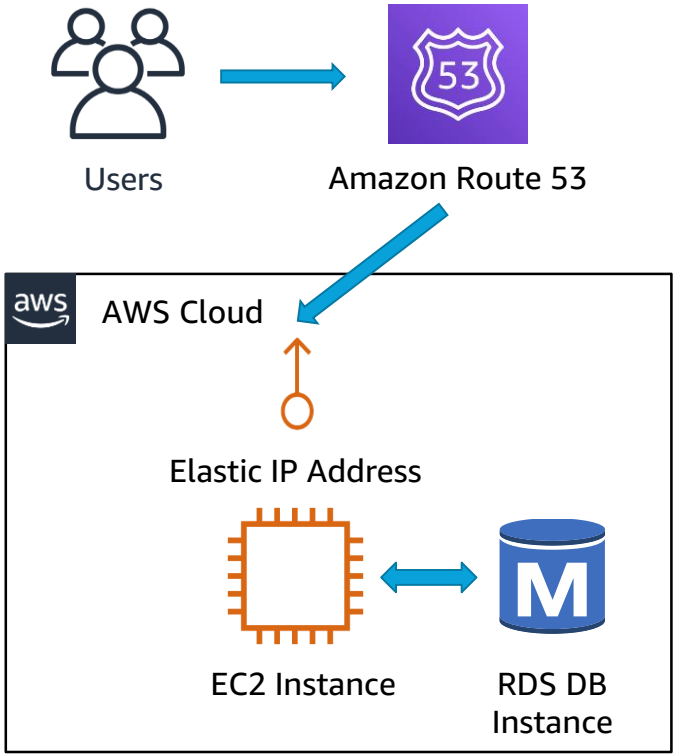




Design for Failure

Separate single EC2 Server into **web** and **database** tiers:

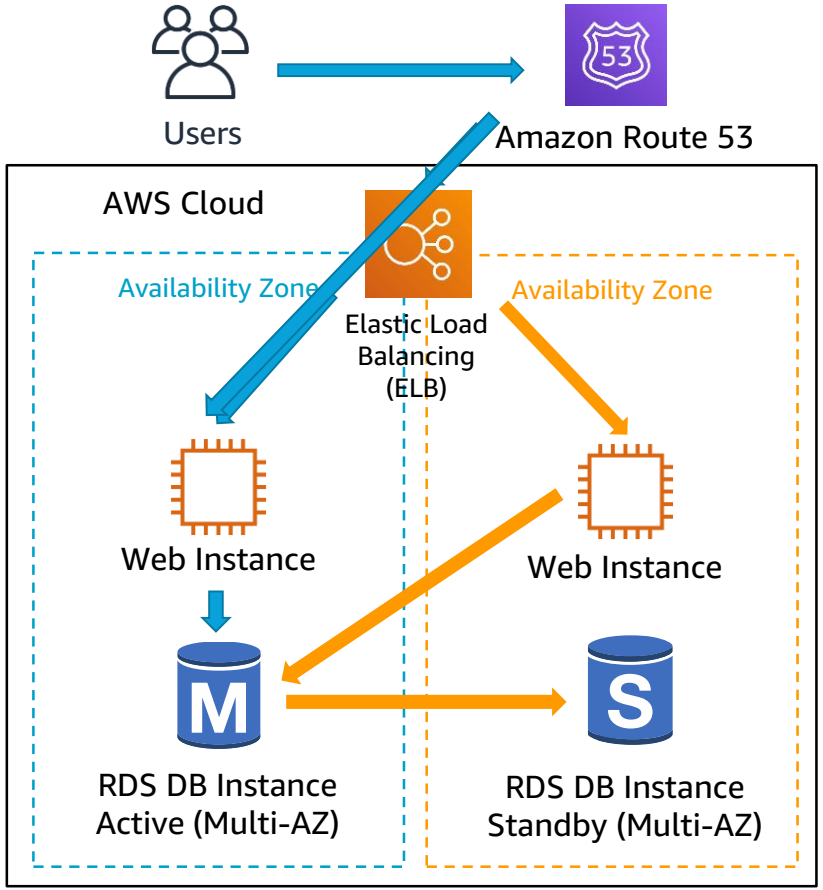
- Web Server on EC2
- Database on EC2 or RDS
  - Amazon Relational Database Service (RDS) can take care of management overhead such as patching, backups, and failure detection



Design for Failure

Leverage multiple Availability Zones for redundancy and high availability.

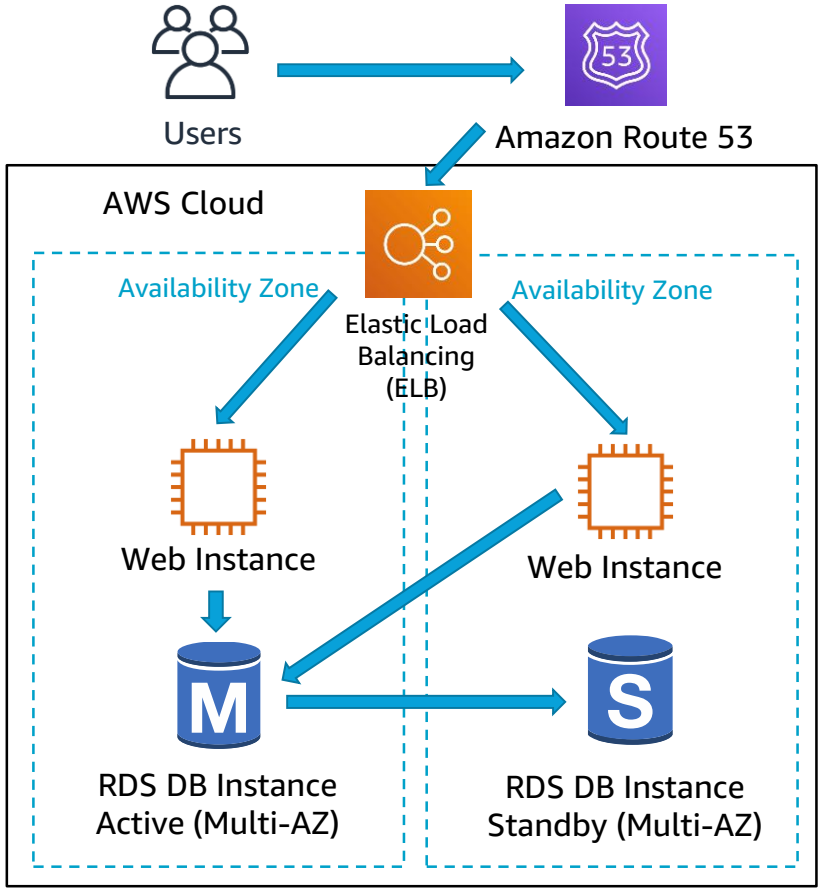
- Use an Elastic Load Balancer (ELB) across AZs for availability and failover
- If using RDS, use the Multi-AZ feature for managed replication and a standby instance
  - If not, use failover and replication features native to your database engine



Design for Failure

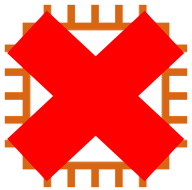
Best Practices:

- Eliminate single points of failure
- Use multiple Availability Zones
- Use Elastic Load Balancing
- Do real-time monitoring with CloudWatch
- Create a database standby across Availability Zones

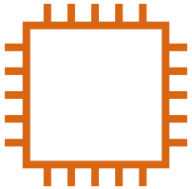


Automate Your Environment

Removing manual processes to improve your system's stability and consistency, and the efficiency of your organization



App server crashes



Replacement automatically launches

aws

Design for Failure

Avoid single points of failure

- Assume everything fails and design backwards
- When, not if, an individual component fails, the application does not fail
  - Think of your servers as cattle, not pets
- Leverage Route 53 DNS Pilot-light or Warm-standby strategies to implement Disaster Recovery
- Auto Scaling groups can be used to detect failures and self-heal, thus protecting against AZ level outages

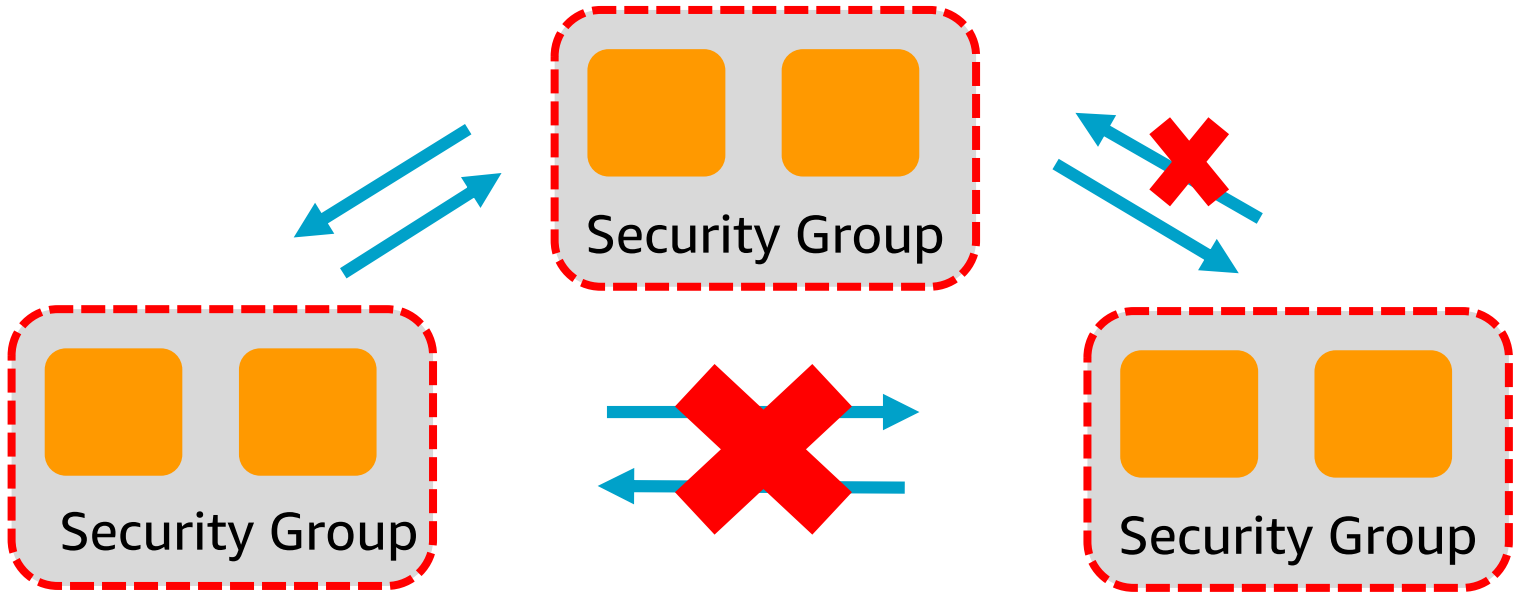
Build Security in Every Layer

AWSOME DAY

aws

Secure Your Infrastructure Everywhere

AWS enables you to implement security both at the perimeter and within/between your resources.

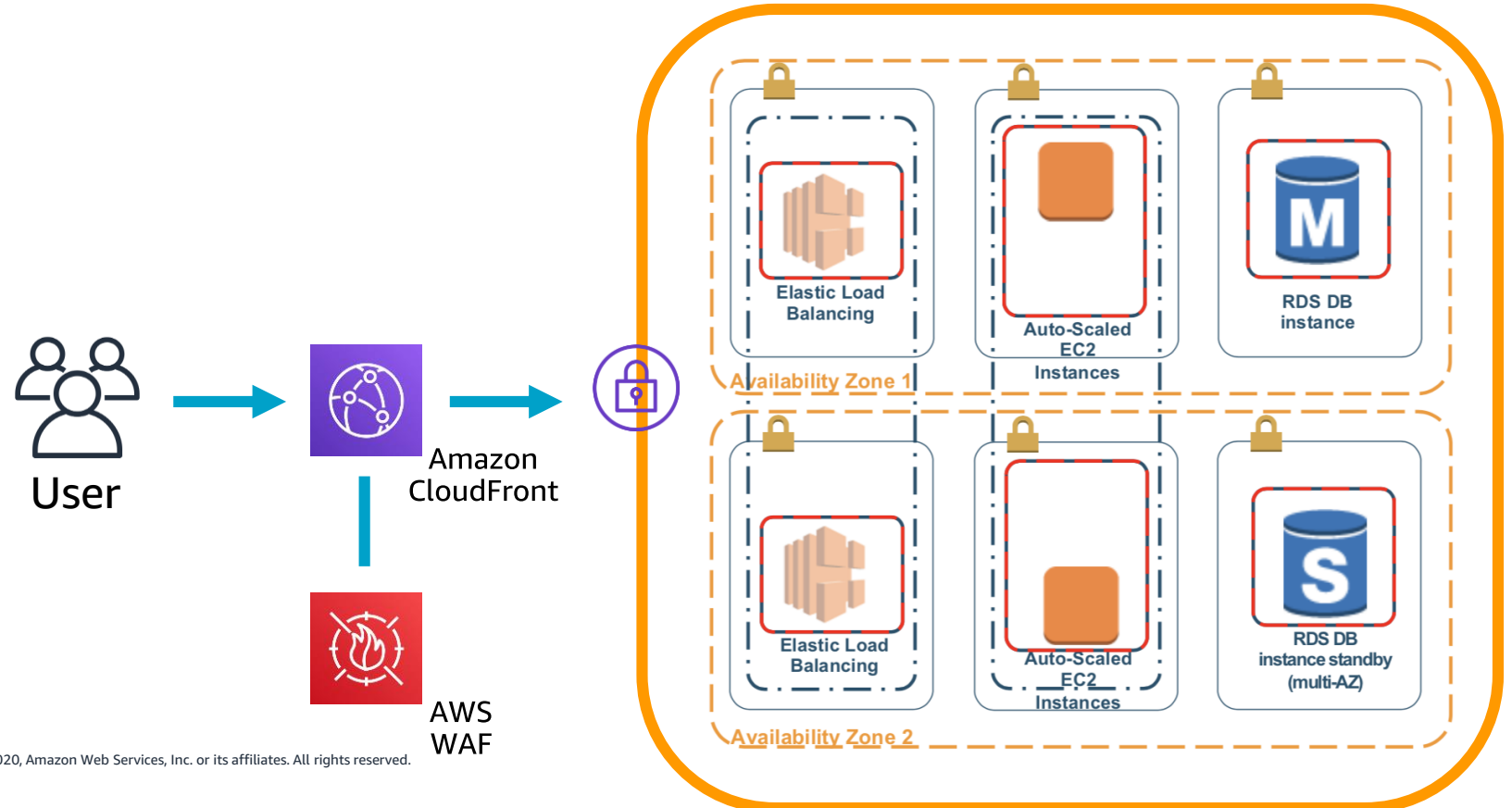


Cloud Architecture Principles

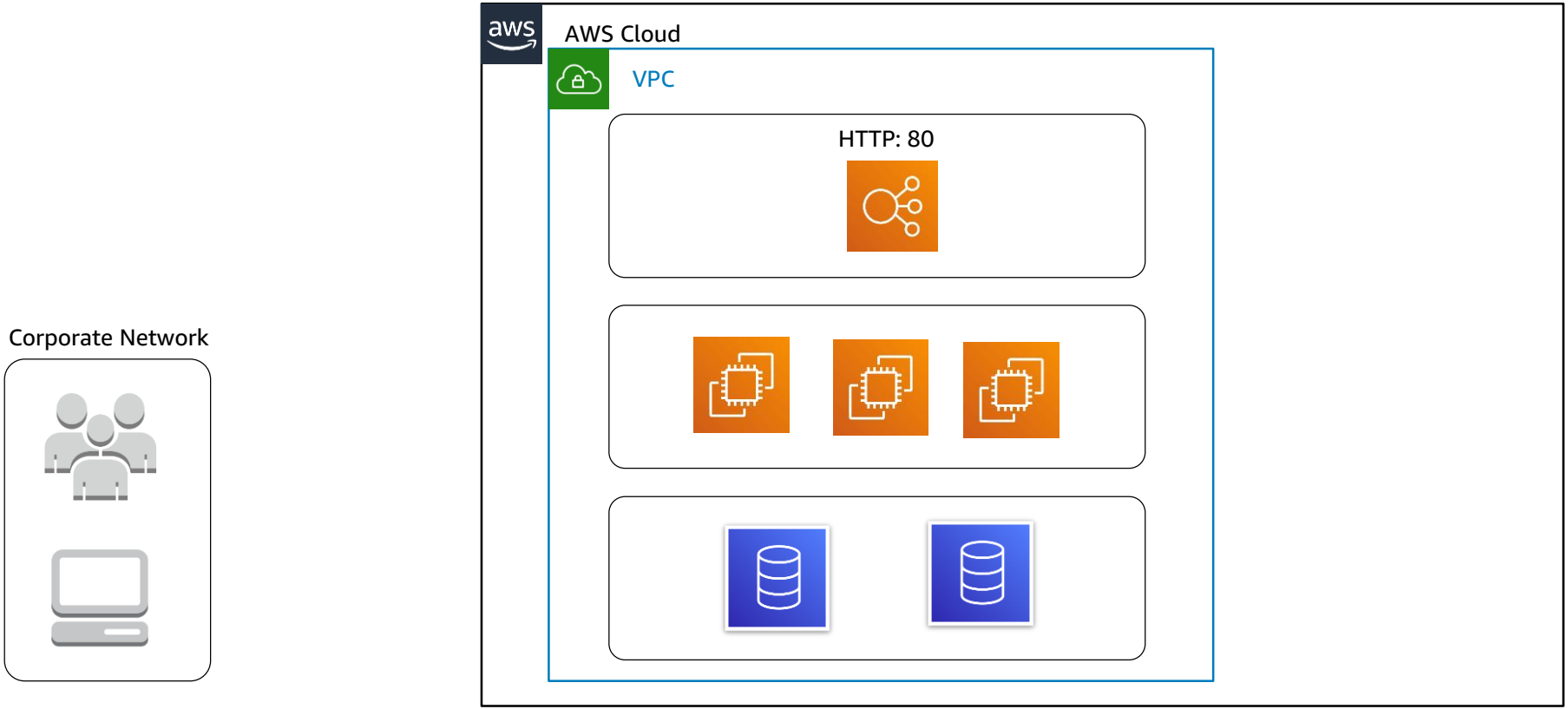
Build Security into every layer

- Encrypt data **in transit** and **at rest** application tiers
- Enforce principle of **least privilege**
- Automatically rotate security keys frequently

Cloud Architecture Principles

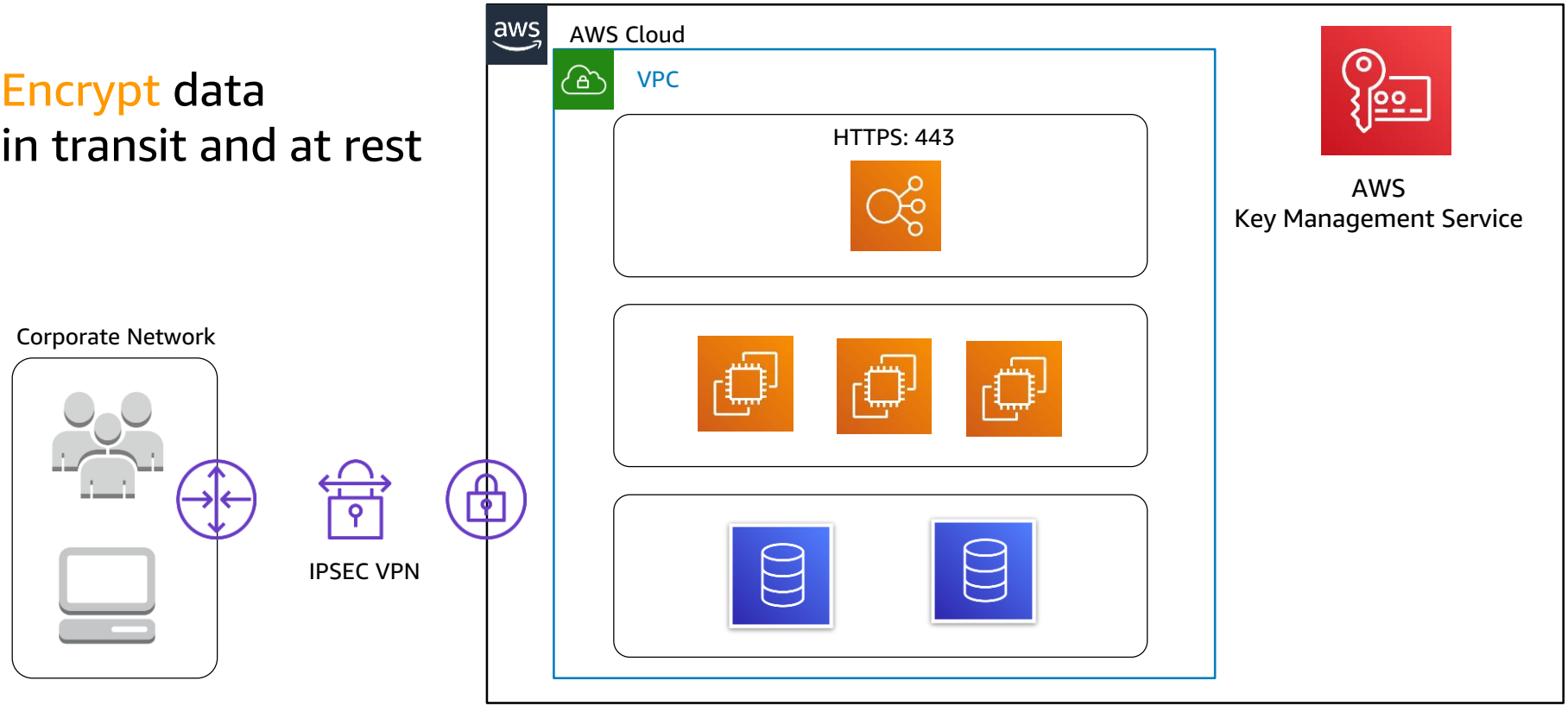


Build Security in Every Layer



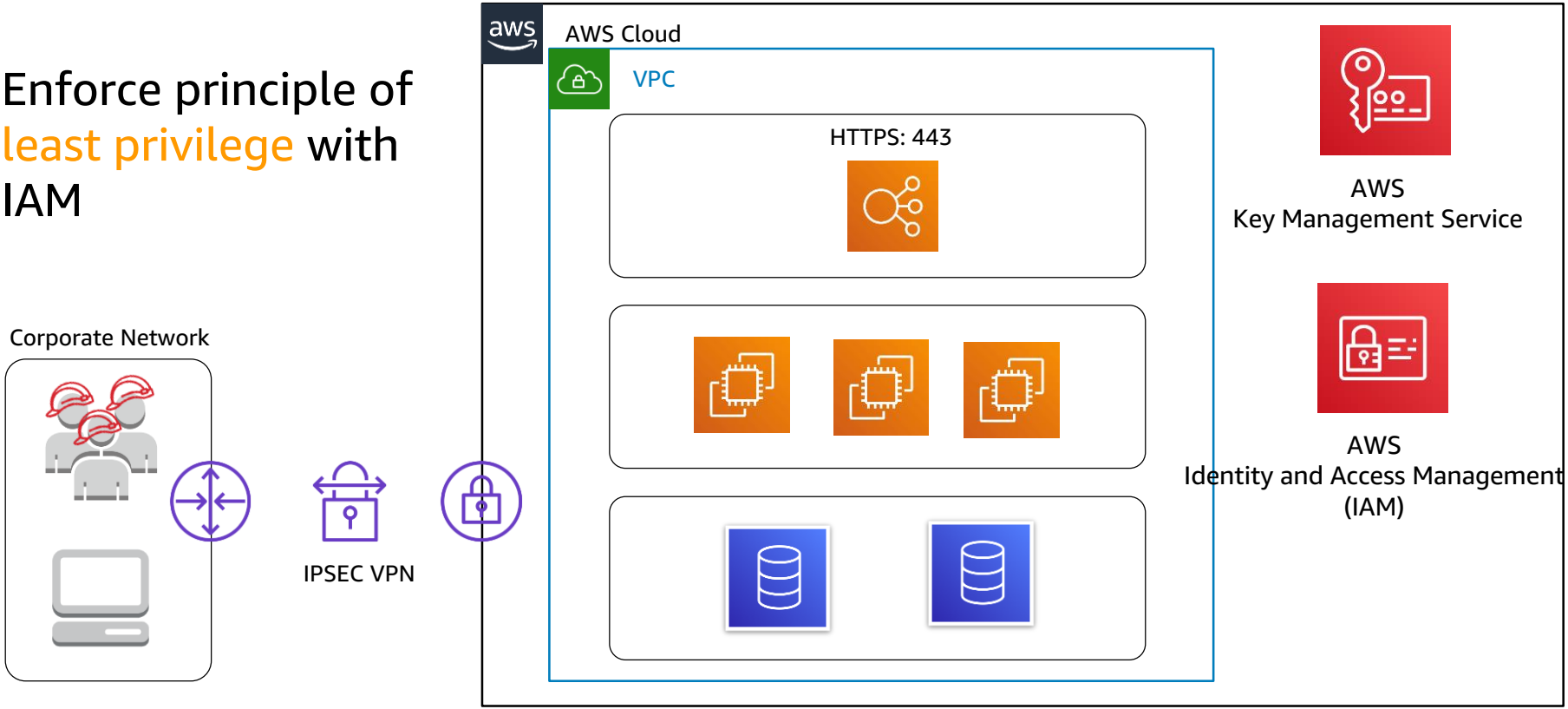
Build Security in Every Layer

Encrypt data
in transit and at rest



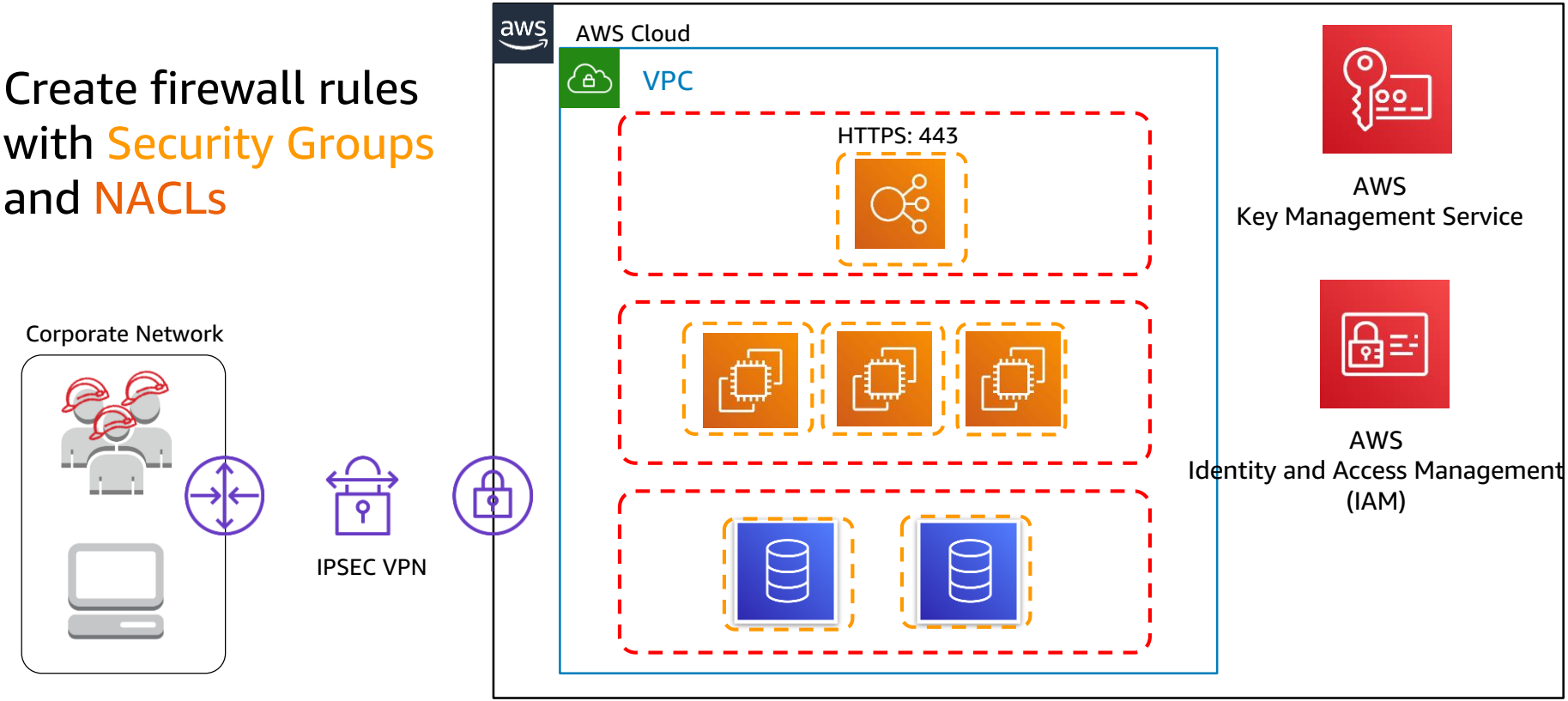
Build Security in Every Layer

Enforce principle of least privilege with IAM



Build Security in Every Layer

Create firewall rules with Security Groups and NACLs



Build Security in Every Layer

You have the control to implement in important layers:

- Encrypt data in transit and at rest
  - Key Management Service (KMS)
- Enforce principle of least privilege with IAM
  - Identity and Access Management (IAM)
- Create firewall rules with Security Groups and NACLs

Leverage Many Storage Options

AWSOME DAY

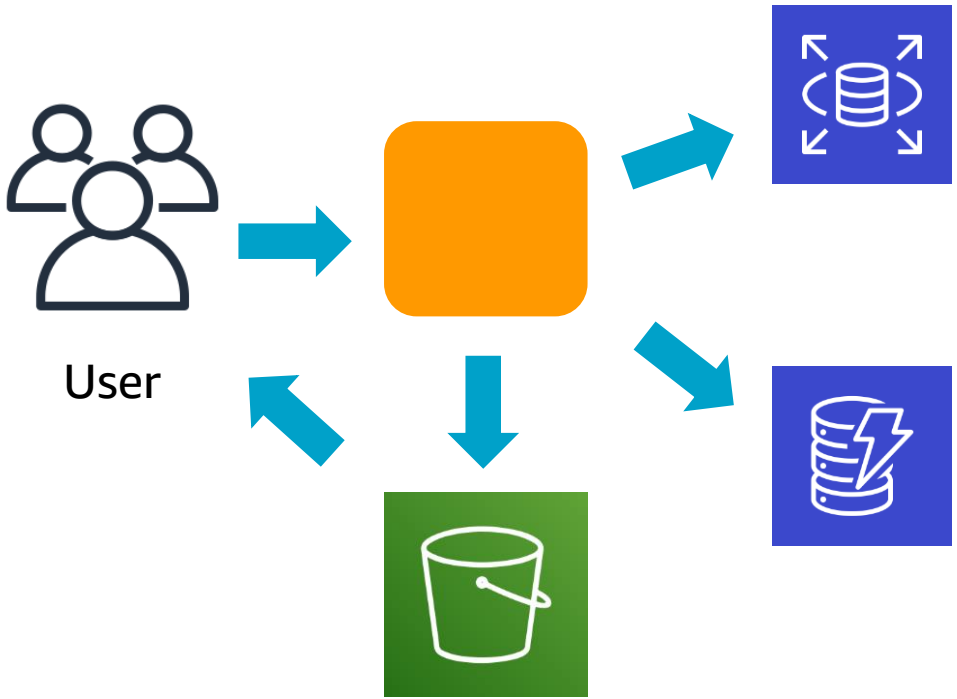
aws

Leverage Many Storage Options

One size does NOT fit all

- Amazon S3 – object/blob store, good for large objects
- Amazon Glacier – Long-term, object storage for data archiving
- Amazon CloudFront – CDN
- Amazon DynamoDB – non-relational data (key-value)
- Amazon EC2 Ephemeral Storage – transient data
- Amazon Elastic Block Storage (EBS) – persistent block storage
- Amazon EFS – shared storage service
- Amazon RDS – managed relational database
- Amazon Redshift – Data warehouse

Choose the Right Database Solutions

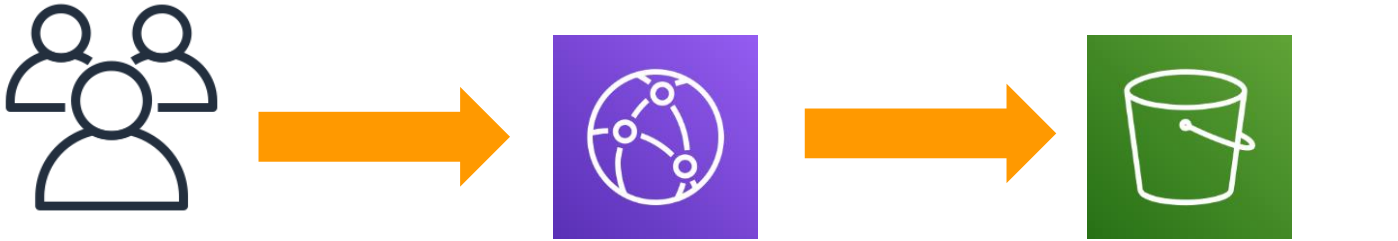


- Don’t **log** clicks to RDBMS, use **NoSQL** data store
- Don’t store **images** in RDBMS, use **object store**
- Offload log **files** to scalable **object storage**

aws

Use Caching

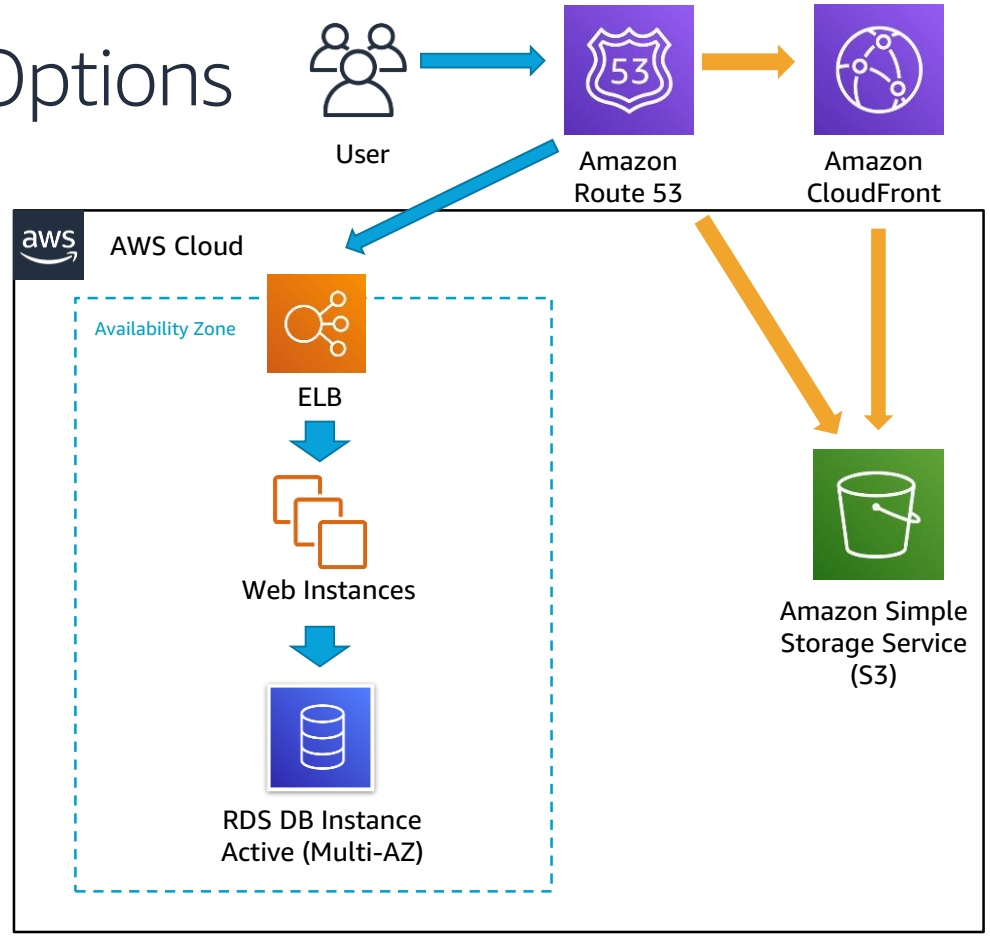
Use caching to minimize redundant data retrieval operations.



Leverage Many Storage Options

We can shift some load around…

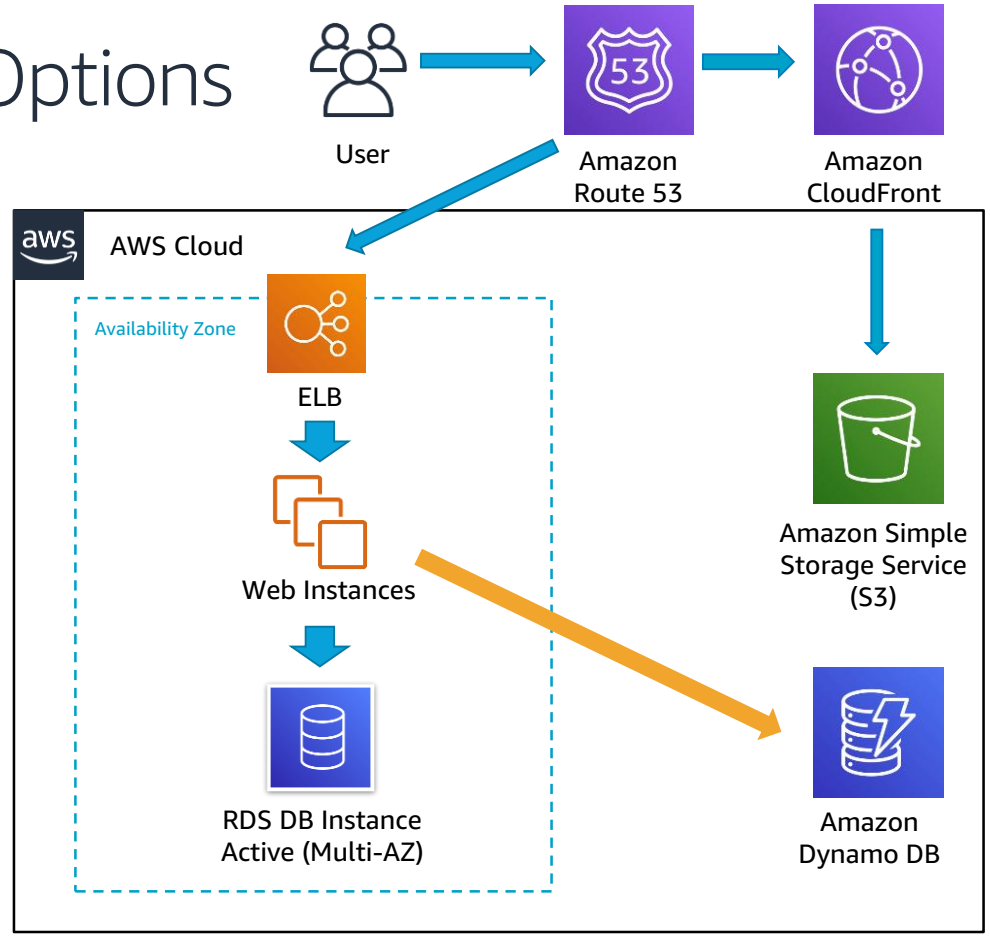
- Static content to Amazon S3 and Amazon CloudFront



Leverage Many Storage Options

We can shift some load around…

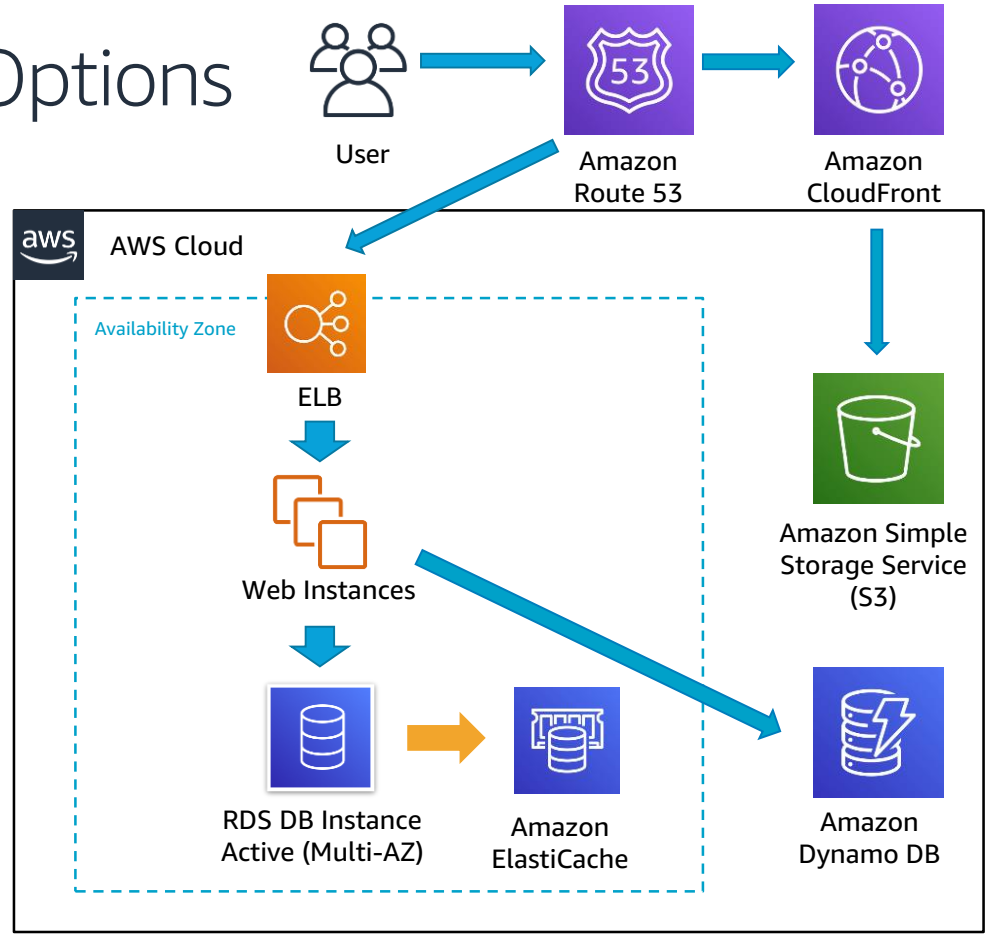
- Static content to Amazon S3 and Amazon CloudFront
- Session/state to Amazon DynamoDB



Leverage Many Storage Options

We can shift some load around…

- Static content to Amazon S3 and Amazon CloudFront
- Session/state to Amazon DynamoDB
- DB caching to Amazon ElastiCache

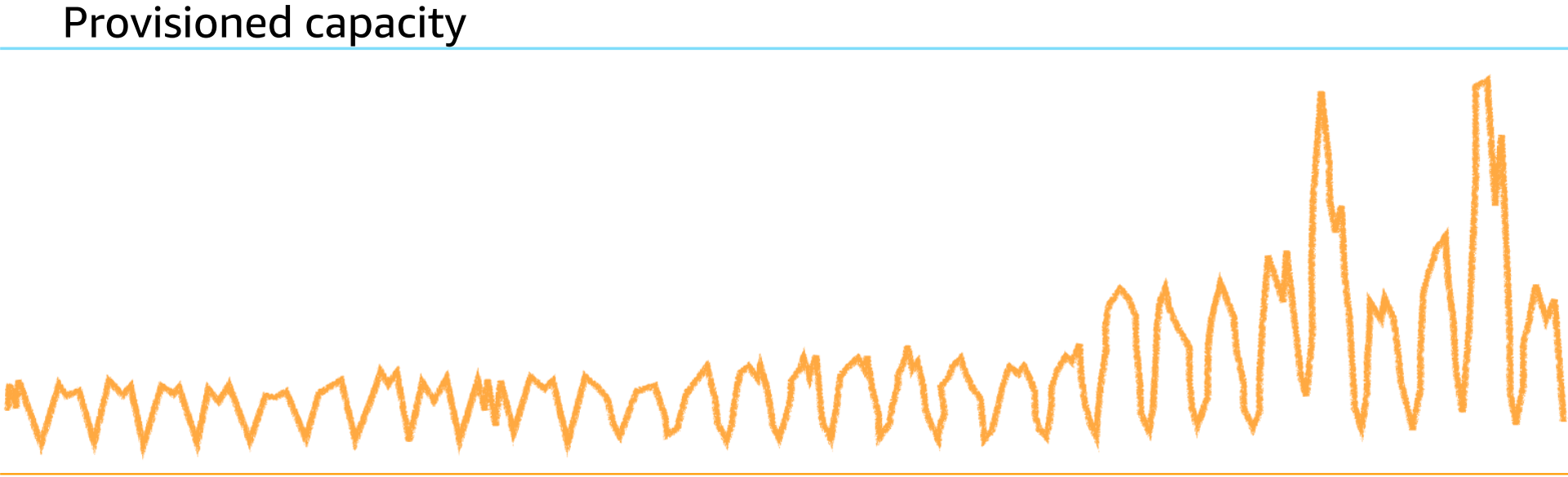


Implement Elasticity

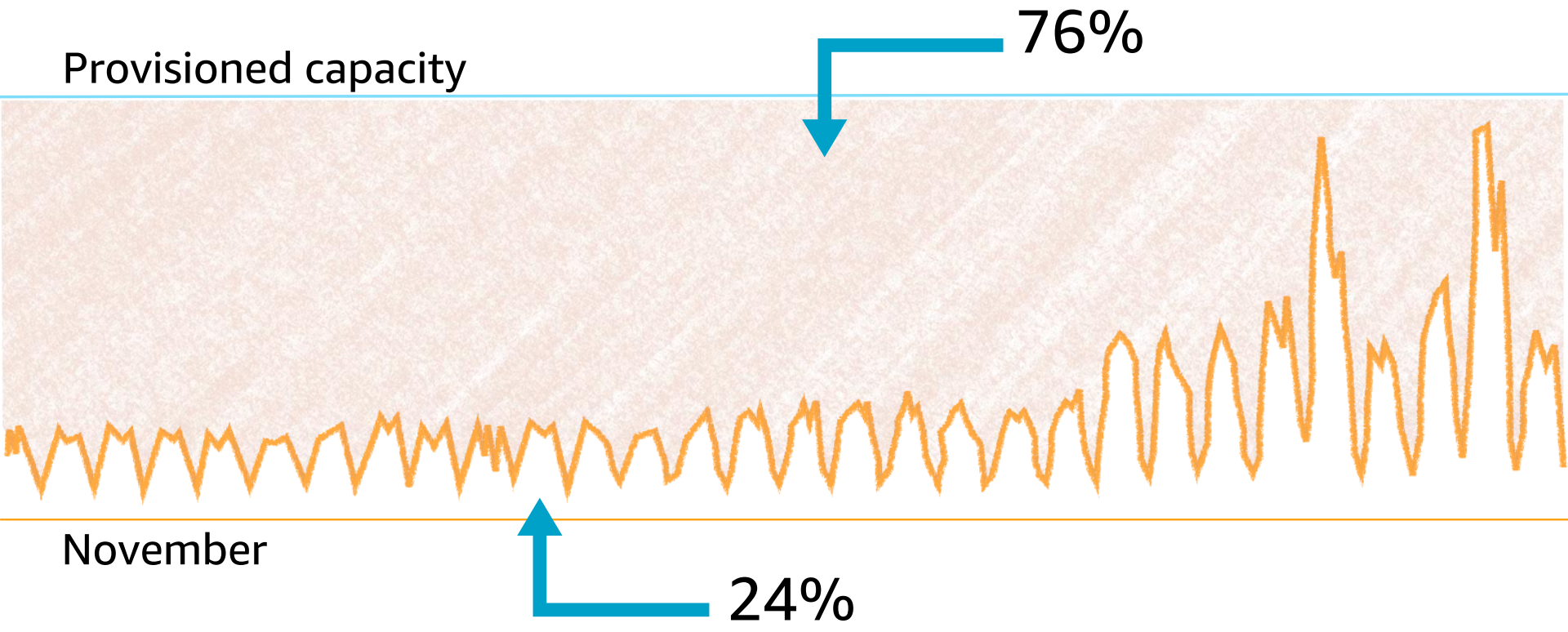
AWSOME DAY

aws

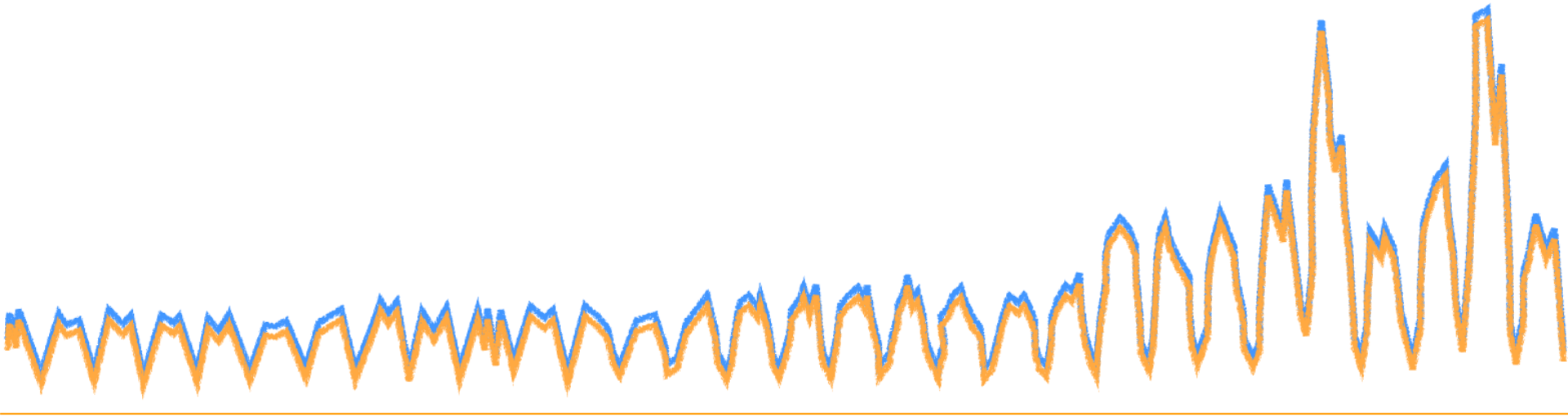
November traffic to Amazon.com



November traffic to Amazon.com



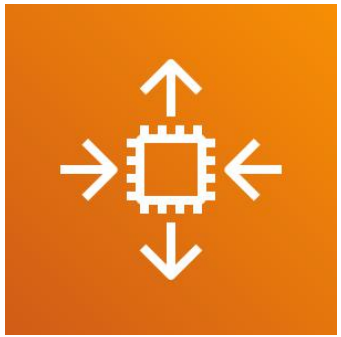
November traffic to Amazon.com

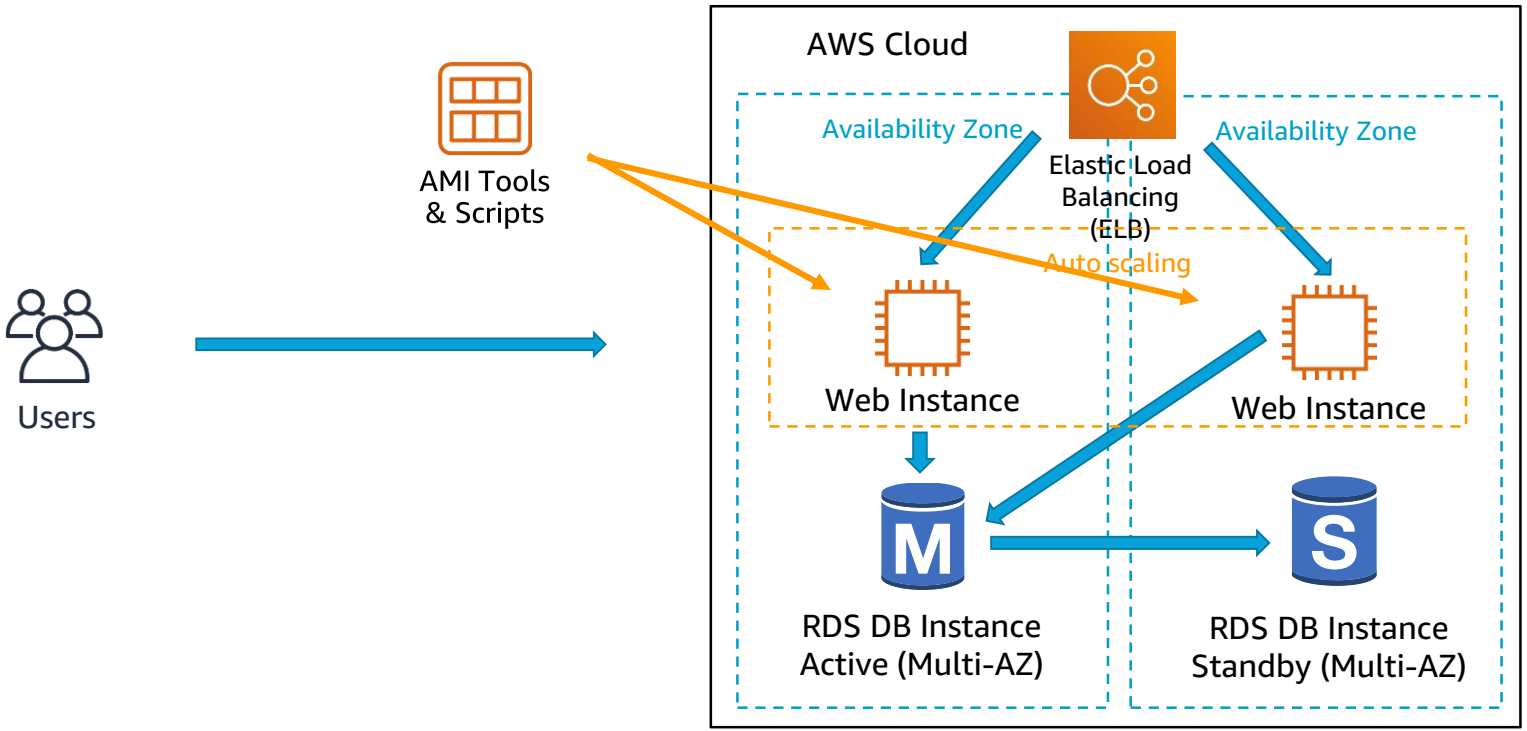


November

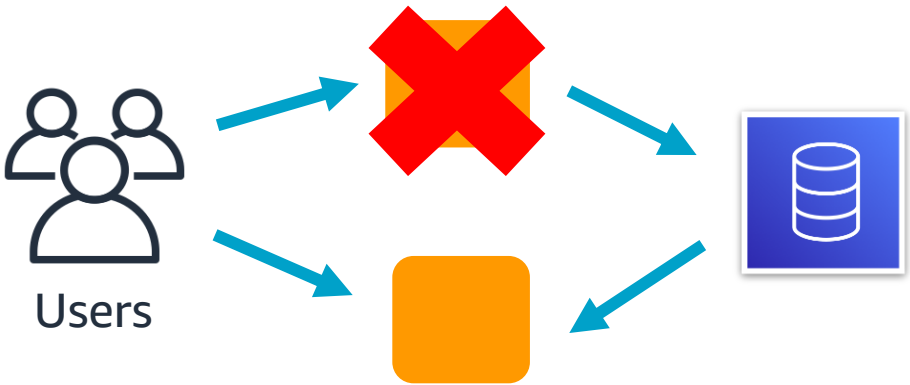
Enable Scalability

Ensure that your architecture can handle changes in demand.





Become Stateless



- Don’t store state in server
- Leverage services to hold state information
- Application functions regardless of which application node processes the request

aws

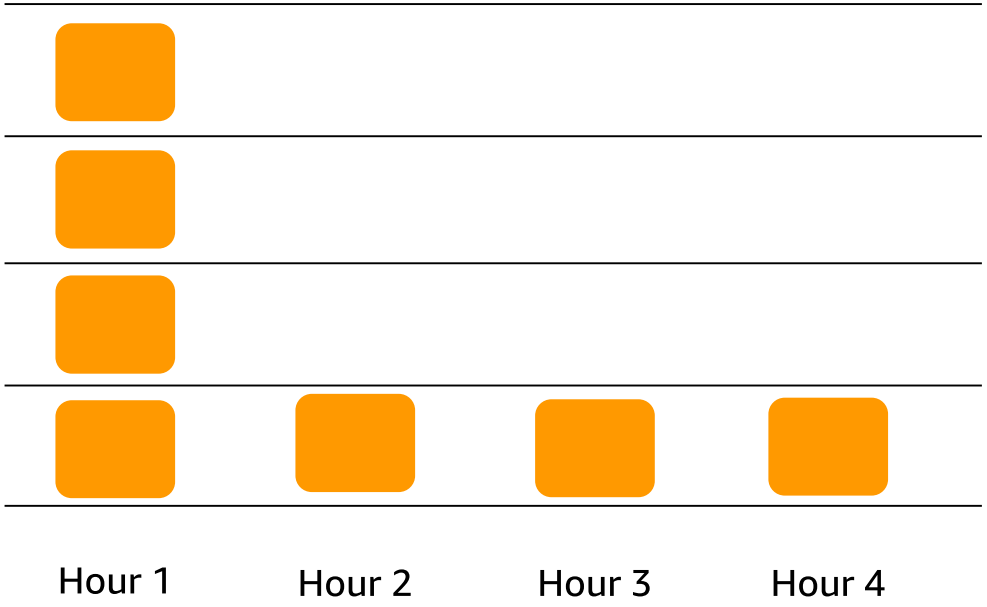
Implement Elasticity

How To Guide

- Write Auto Scaling policies with your specific application access patterns in mind
- Prepare your application to be flexible: don't assume the health, availability, or fixed location of components
- Architect resiliency to reboot and relaunch
  - When an instance launches, it should ask "*Who am I and what is my role?*"
- Leverage highly scalable, managed services such as S3 and DynamoDB

Think Parallel

Think Parallel



- One Server working for Four hours costs the same as Four servers working for One hour
- Combine with elasticity to increase capacity when you need it most
- The beauty of the cloud shines when you combine **elasticity** and **parallelization**

aws

Think Parallel

Scale Horizontally, Not Vertically

- Decouple compute from state/session data
- Use ELBs to distribute load
- Break up big data into pieces for distributed processing
  - AWS Elastic Map Reduce (EMR) – managed Hadoop

Think Parallel

Faster doesn’t need to mean more expensive

- With EC2 On Demand, the following will cost the same:
  - 12 hours of work using 4 vCPUs
  - 1 hour of work using 48 vCPUs
- Right Size your infrastructure to your workload to get the best balance between cost and performance

Think Parallel

Parallelize using native managed services

- Get the best performance out of S3 with parallelized reads/writes
  - Multi-part uploads (API) and byte-range GETs (HTTP)
- Take on high concurrency with Lambda
  - Initial soft limit: 1000 concurrent requests per region

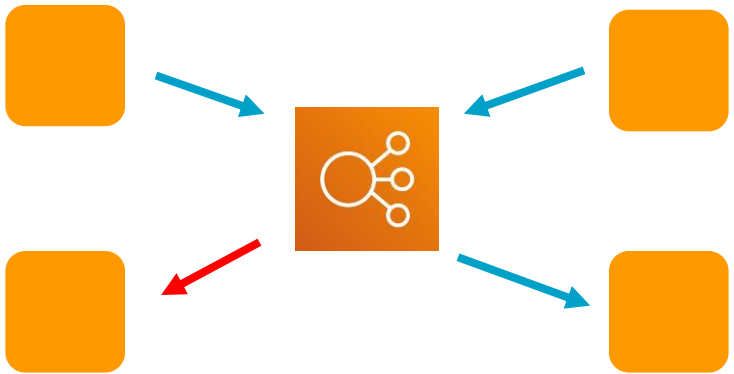
Loose Coupling Sets You Free

AWSOME DAY

aws

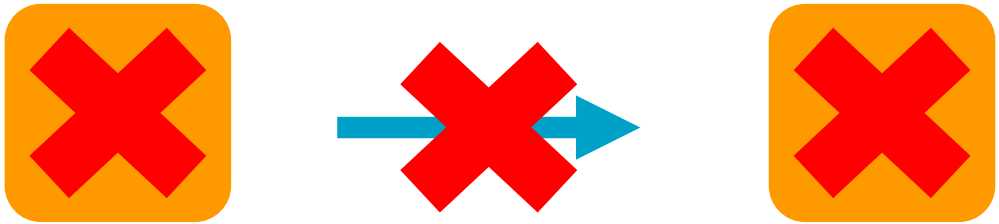
Loosely Couple Your Components

Reduce interdependencies so that the change or failure of one component does not affect other components.



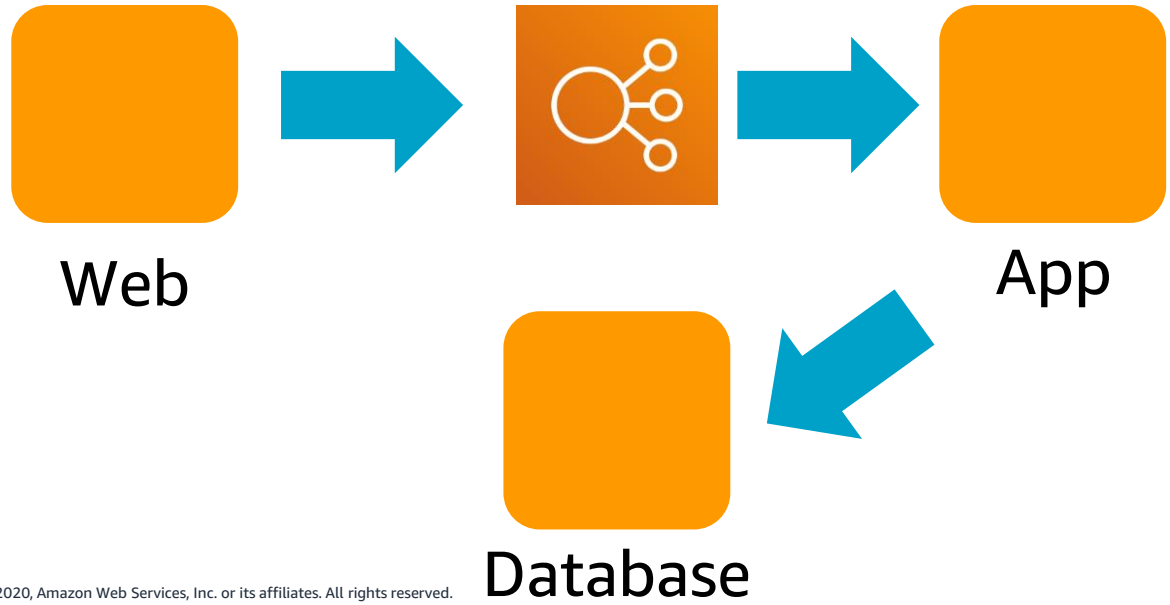
Cloud Architecture Principles

Loose Coupling sets you free



Cloud Architecture Principles

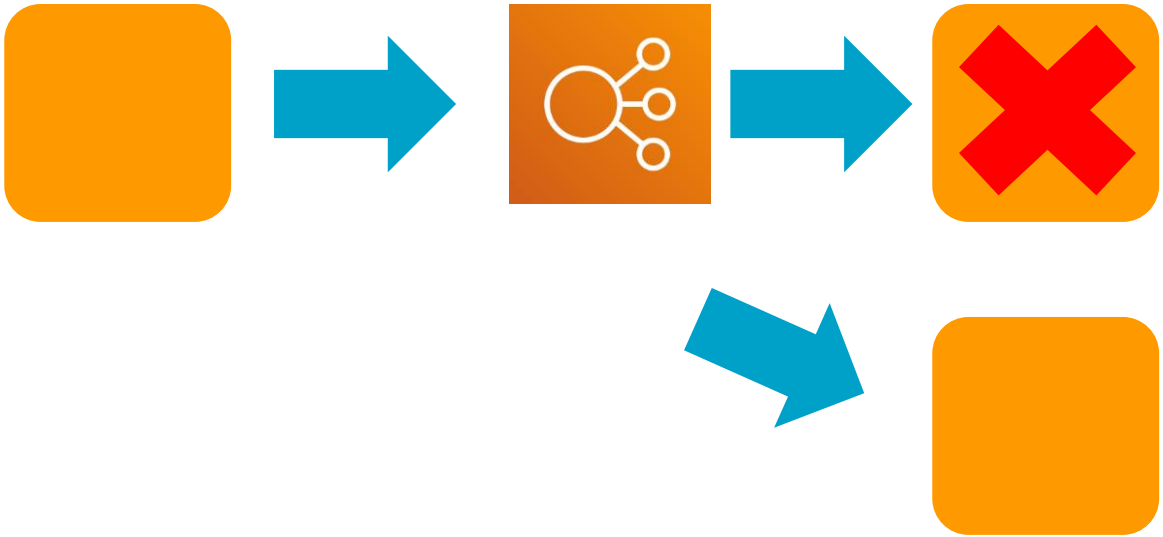
Loose Coupling sets you free



- Separate application into independent tiers

Cloud Architecture Principles

Loose Coupling sets you free

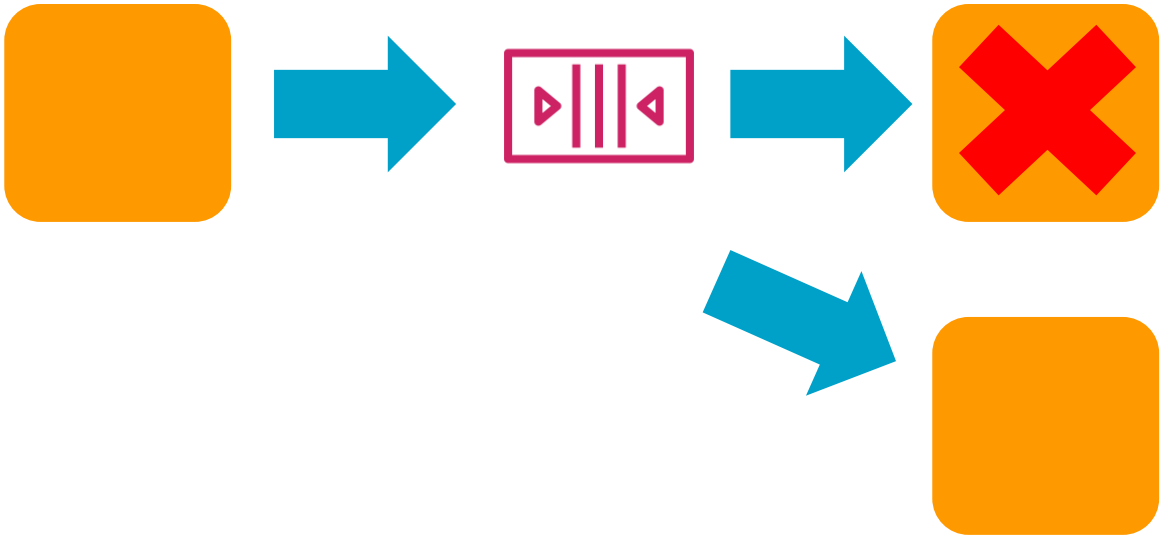


- Design architectures with independent components
- Design every component as a black box
- Load balance clusters

aws

Cloud Architecture Principles

Loose Coupling sets you free

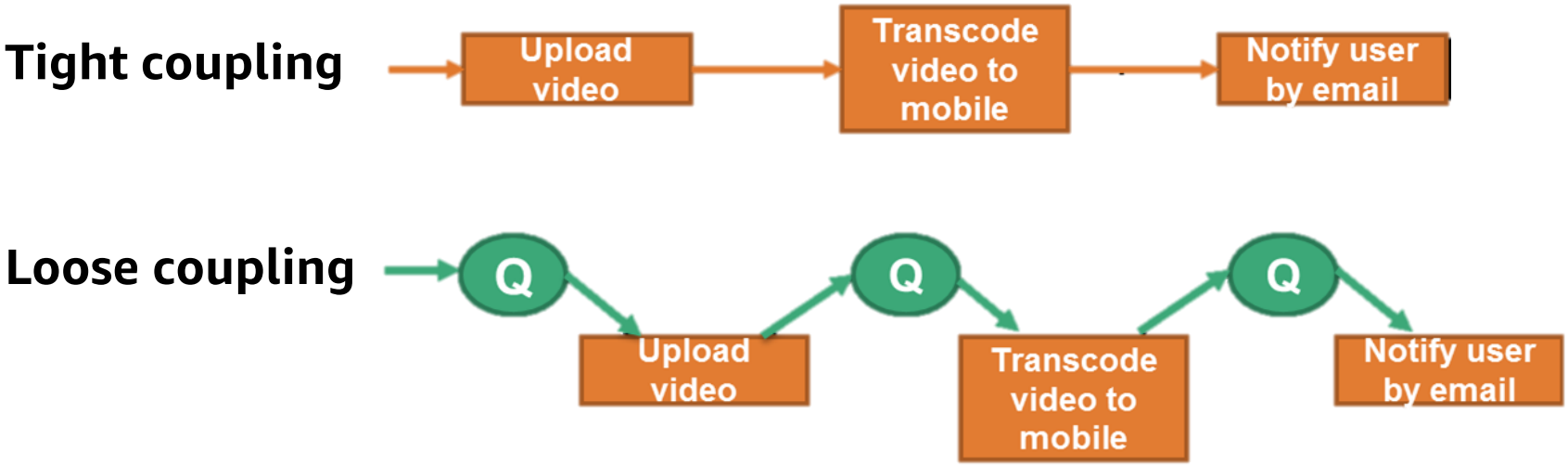


- Use queues to pass message between components

aws

Loose Coupling Sets You Free: Queueing

Use Amazon Simple Queue Service (SQS) to pass messages between loosely coupled components

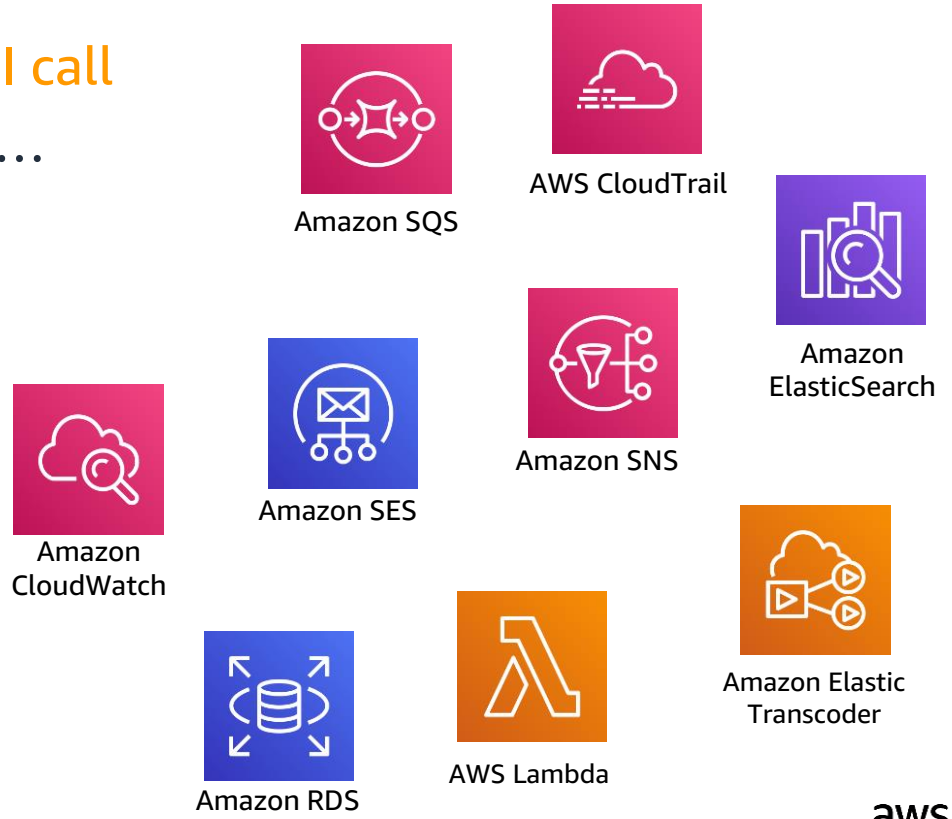


Loose Coupling Sets You Free

Nearly everything in AWS is an API call

Leverage AWS Native Services for…

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

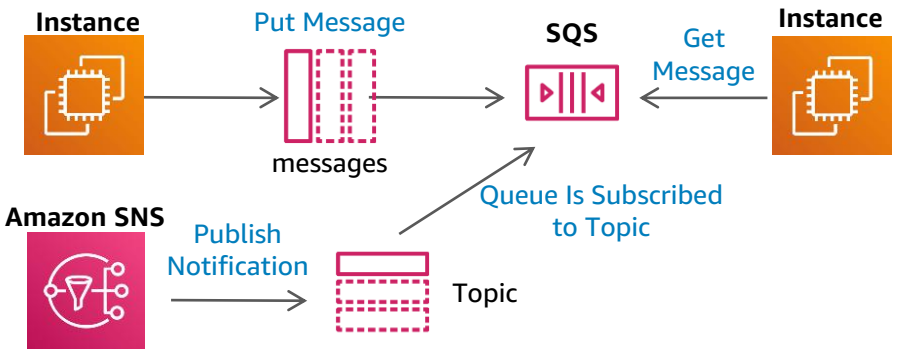


Loose Coupling Sets You Free

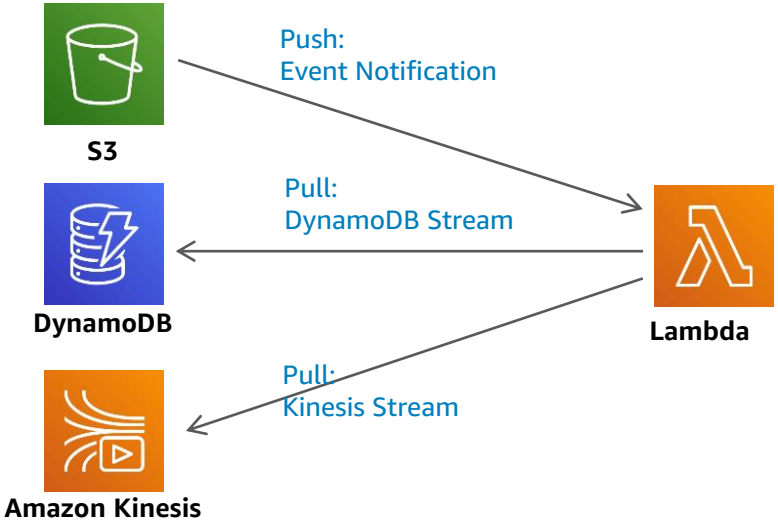
- Design everything as a black box
- Build separate services instead of something that is tightly interacting with something else
- Uses common interfaces or common APIs between the components
- Favor services with built-in redundancy and scalability rather than building your own

Loose Coupling Sets You Free

Using SNS and SQS to asynchronously scale:



Using Lambda triggers to decouple actions:



aws

Don’t Fear Constraints

Don’t Fear Constraints

Rethink traditional architectural constraints

Need more RAM?

- Don’t: vertically scale
- Do: distribute load across machines or a shared cache

Need better IOPS for database?

- Don’t: rework schema/indexes or vertically scale
- Do: create read replicas, implement sharding, add a caching layer

Don’t Fear Constraints

Rethink traditional architectural constraints

Hardware failed or config got corrupted?

- Don’t: waste production time diagnosing the problem
- Do: “Rip and replace” – stop/terminate old instance and relaunch

Need a Cost Effective Disaster Recovery (DR) strategy?

- Don’t: double your infrastructure costs when you don’t need to
- Do: implement Pilot Light or Warm Standby DR stacks

Takeaway

Cloud Architecture Best Practices

1. Design for failure and nothing fails
2. Build security in every layer
3. Leverage different storage options
4. Implement elasticity
5. Think parallel
6. Loose coupling sets you free
7. Don’t fear constraints

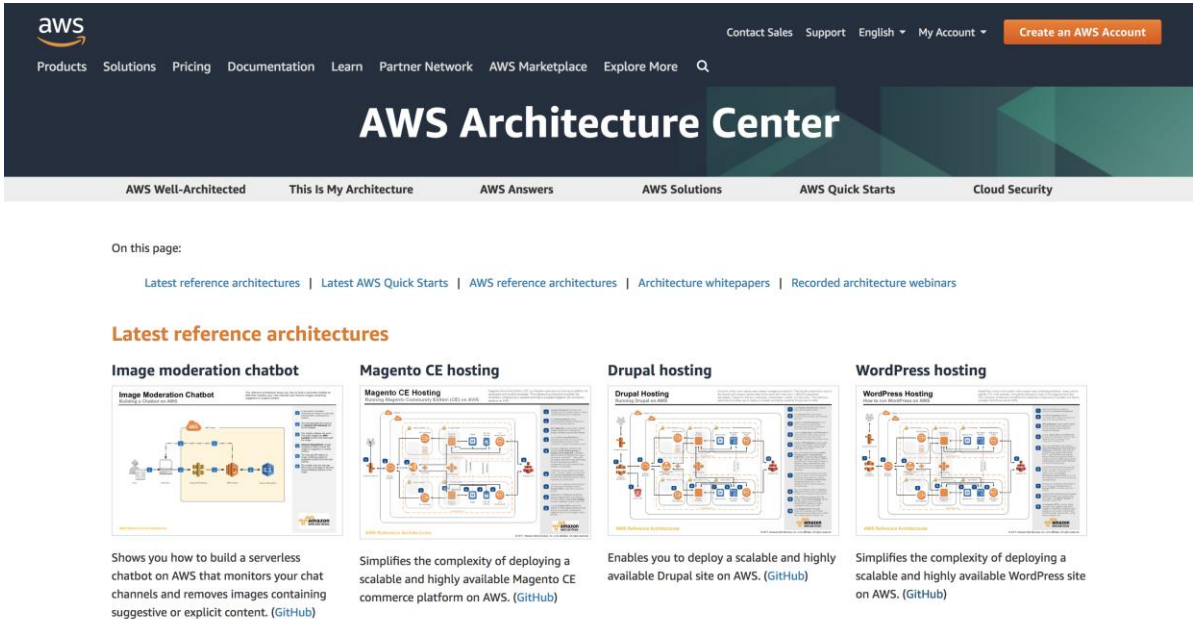
Want to know more about architecting on AWS?

AWSOME DAY

aws

AWS Architecture Center

https://aws.amazon.com/architecture/



Thank you!

Shanna Chang
Solutions Architect