Solutions Architect Professional

AWS Scaling and Resiliency

Cloud Architecture Principles

Goal as an architecture

Deliver Highly Available,
Highly Scalable,
Fault Tolerant,
High Performance,
Secure Cloud Architecture

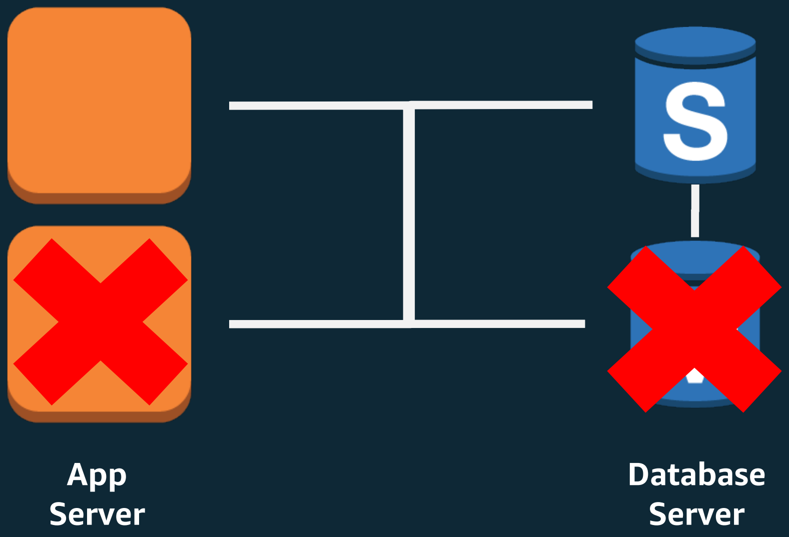
Cloud Architecture Principles

“Everything fails, all the time.”

Werner Vogels, CTO, Amazon.com

Cloud Architecture Principles

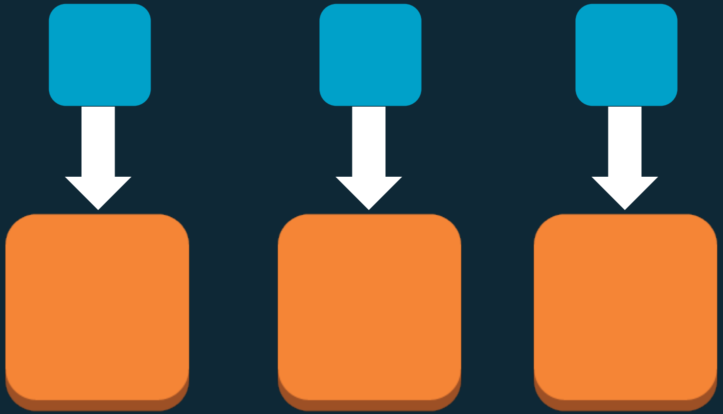
1. Design for failure



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

Cloud Architecture Principles

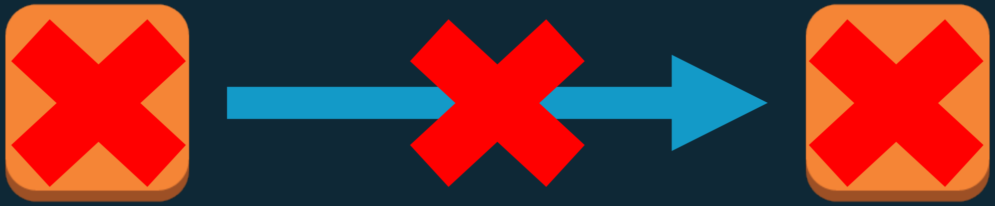
2. Embrace Elasticity & Automate



- Do not assume health, availability or fixed location of components
- Automate installation and configuration of environment
- Favor dynamic configuration

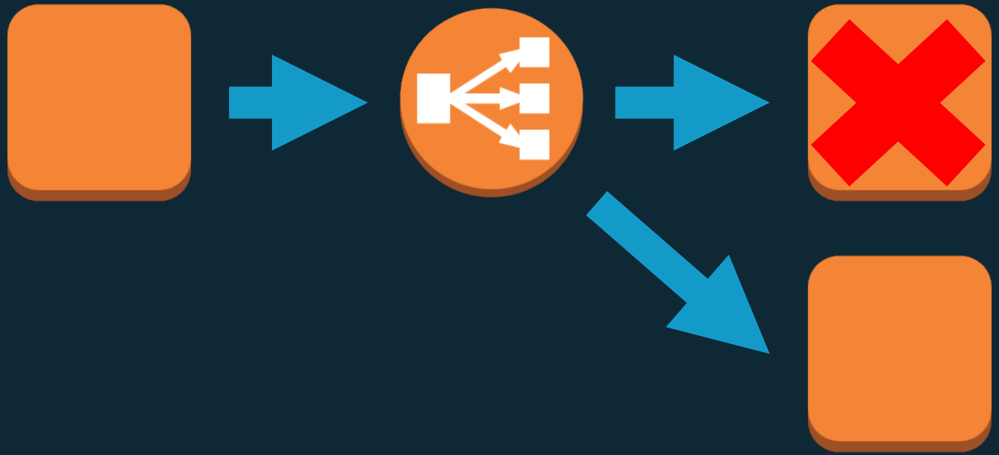
Cloud Architecture Principles

3. Loosely Couple



Cloud Architecture Principles

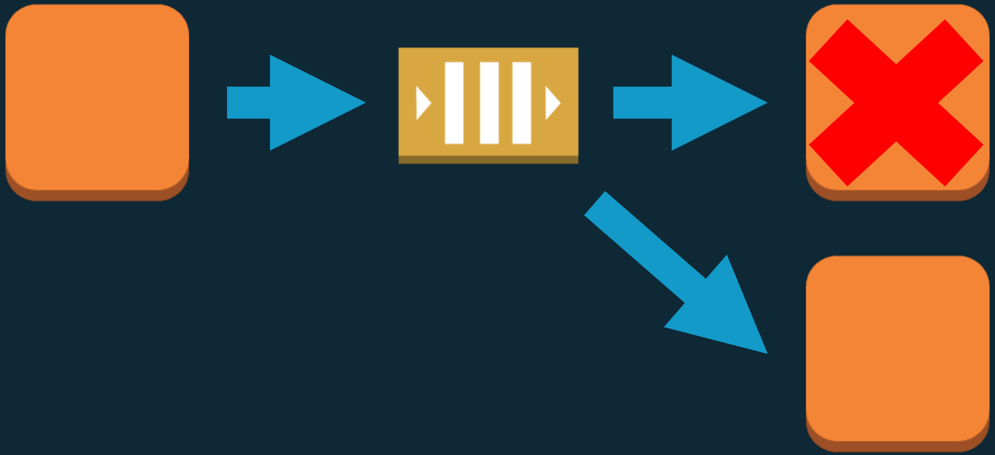
3. Loosely Couple



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

Cloud Architecture Principles

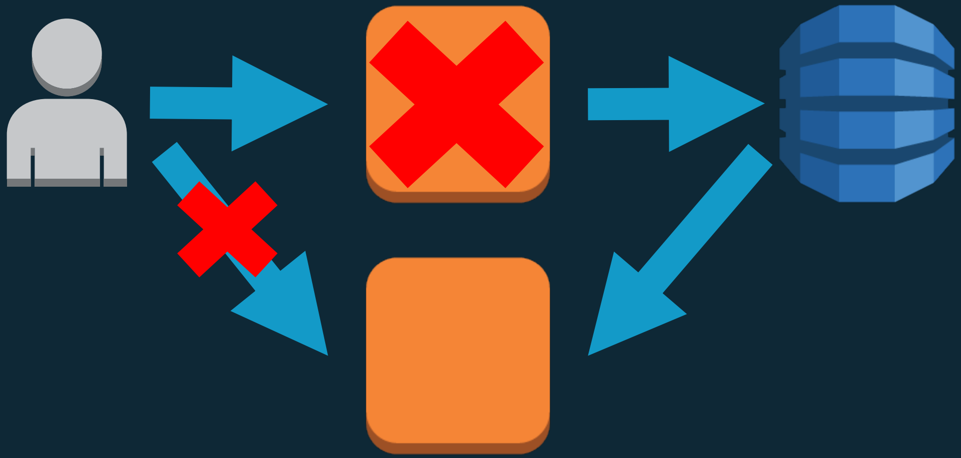
3. Loosely Couple



- Use queues to pass messages between components

Cloud Architecture Principles

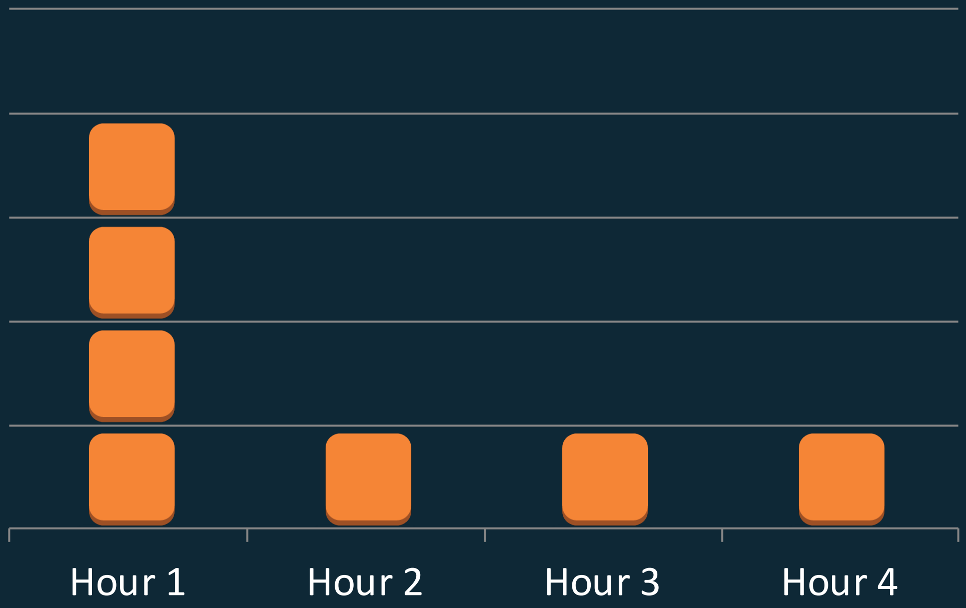
4. Become Stateless



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

Cloud Architecture Principles

5. Leverage Parallelism

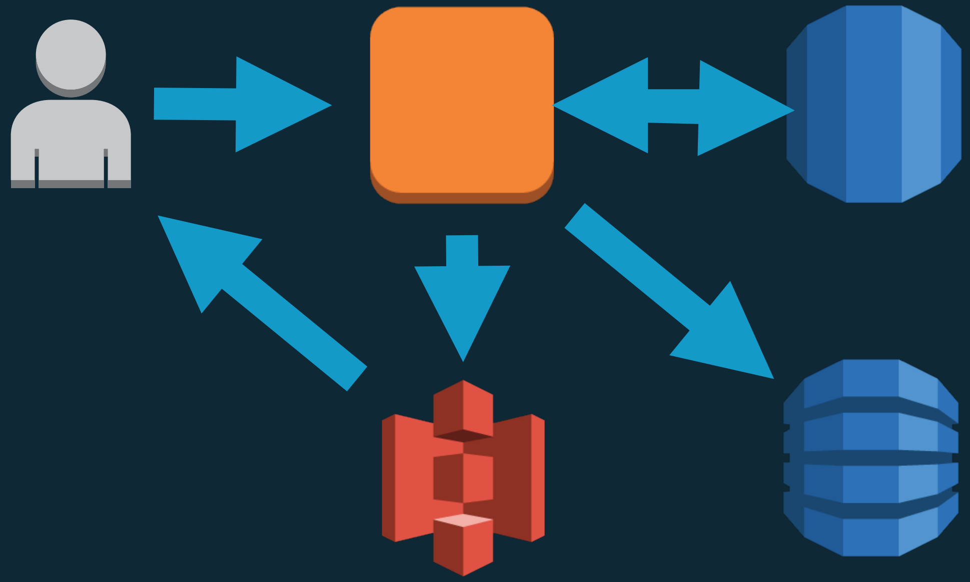


- One Server working for Four hours costs the same as Four servers working for One hour

- Combine with elasticity to increase capacity when you need it most

Cloud Architecture Principles

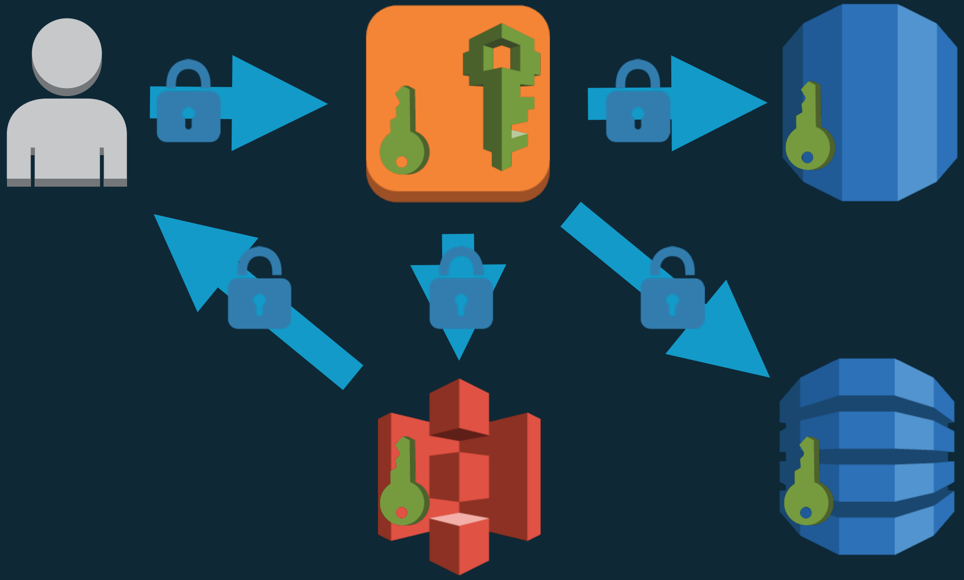
6. Use appropriate storage options



- 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

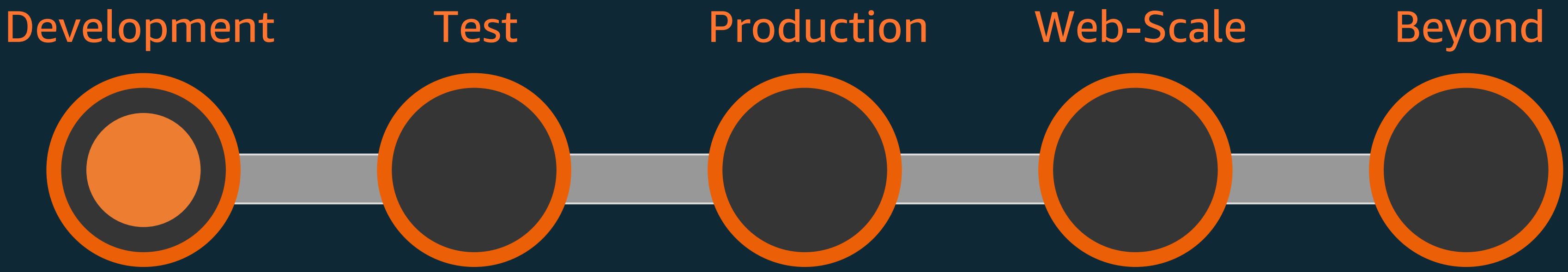
Cloud Architecture Principles

7. Build Security into every layer



- Encrypt data in transit and rest between application tiers
- Enforce principle of least privilege across every service
- Automatically rotate security keys frequently

From Development to Web-Scale



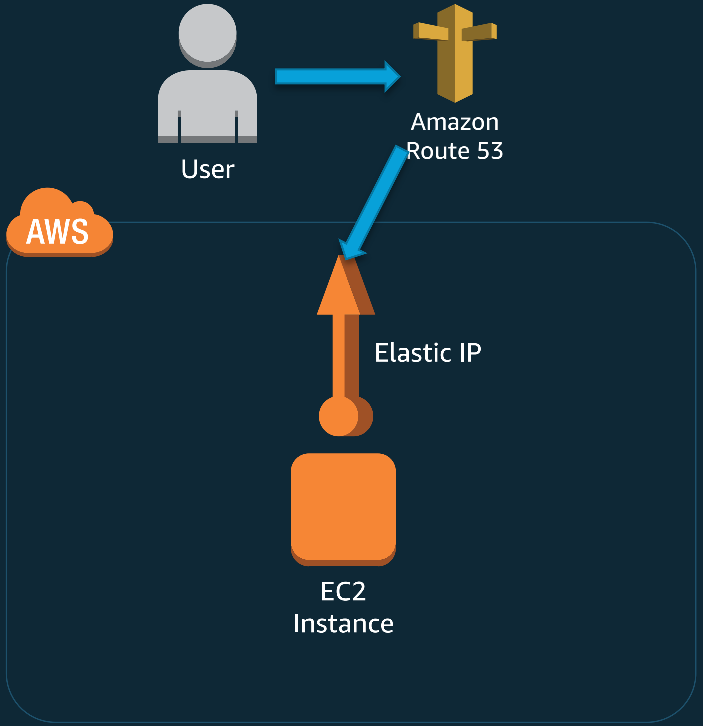
Development

Your own development environment:

- An EC2 Instance, hosting
  - Web, App
  - Database
  - Management
  - Etc.

A single Elastic IP

Route53 for DNS



“We’re gonna need a bigger box”

Different EC2 instance type

- High memory instances
- High CPU instances
- High I/O instances
- High storage instances

Can now leverage PIOPs

Easy to change instance sizes

Will hit an endpoint eventually

r5.8xlarge
c5.2xlarge
t3.micro

“We’re gonna need a bigger box”

Different EC2 instance type

- High memory instances
- High CPU instances
- High I/O instances
- High storage instances

Can now leverage PIOPs

Easy to change instance sizes

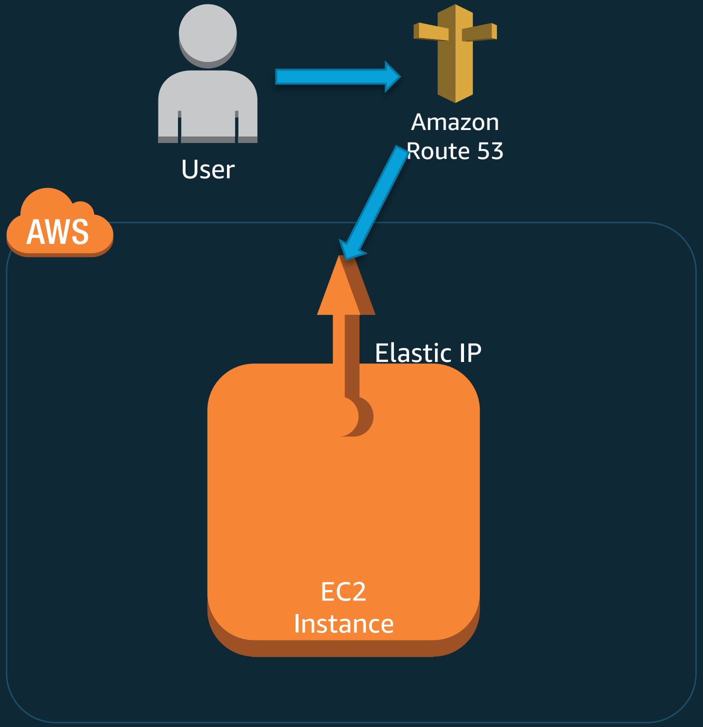
Will hit an endpoint eventually

r5.8xlarge
c5.2xlarge
t3.micro

Shall we use this to test with?

There are some issues

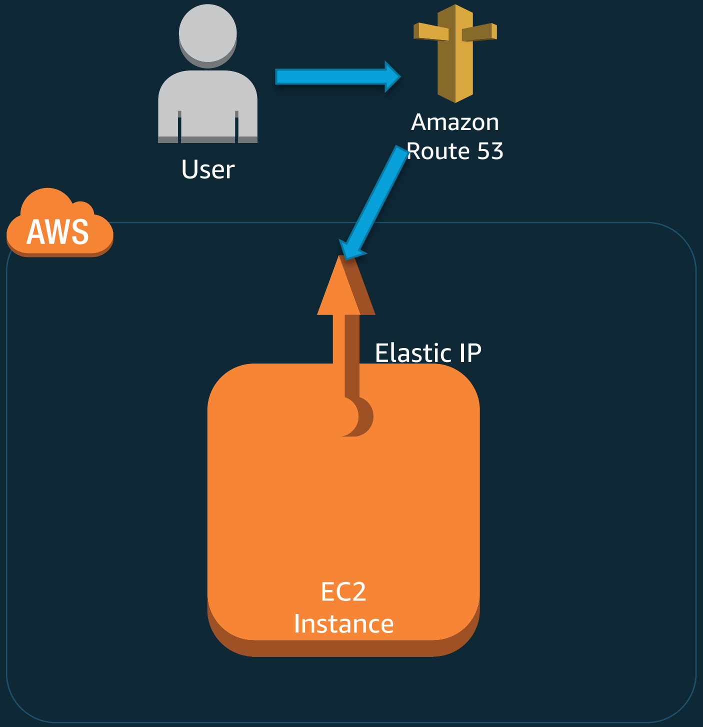
- Constrained by a single environment
- Too many eggs in one basket
- No failover
- No redundancy



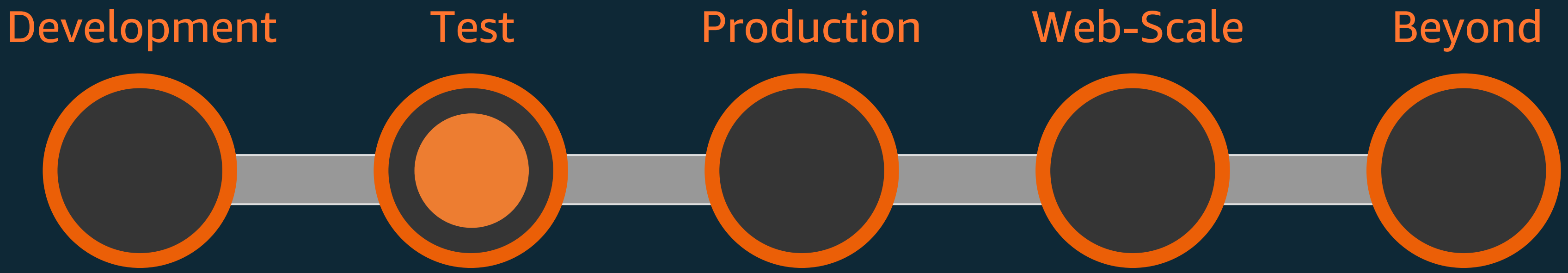
Shall we use this to test with?

There are some issues

- Constrained by a single environment
- Too many eggs in one basket
- No failover
- No redundancy



From Development to Web-Scale



Development

Test

Production

Web-Scale

Beyond

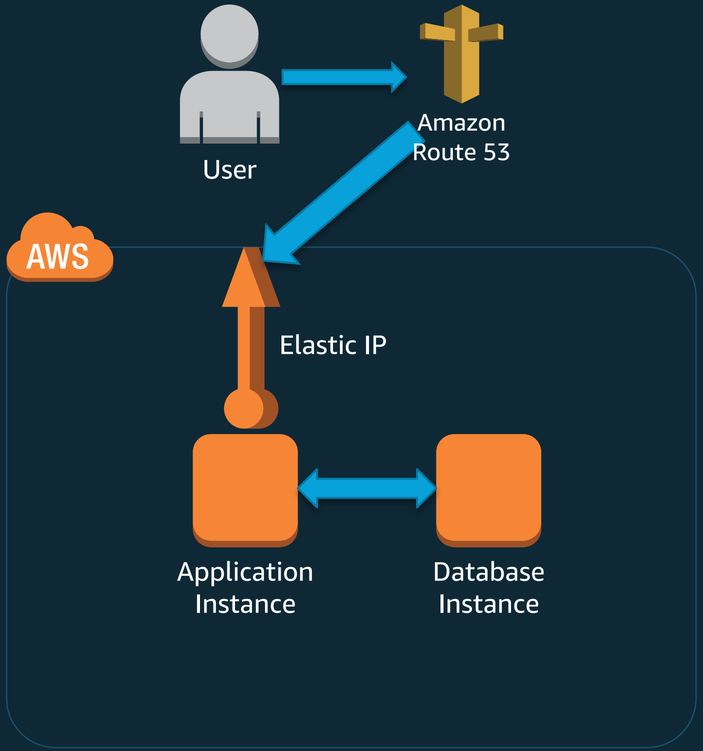
Test

First let’s separate out our single host into different tiers:

Web

Database

Should we make use of a database service?



AWS 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, Oracle PostgresSQL or MySQL 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

Why start with SQL?

Established and well worn technology

Lots of existing code, communities, books, background, tools, etc

You aren’t going to break SQL DBs until you’re really big

- But you might break parts of it (hence blended approach)

Clear patterns to scalability

- But, can suffer vertical scaling constraints ultimately

Why else might you need NoSQL?

Super low latency applications

Metadata driven datasets

Highly un-relational data

Need schema-less data constructs*

Massive amounts of data (again, in the TB range)

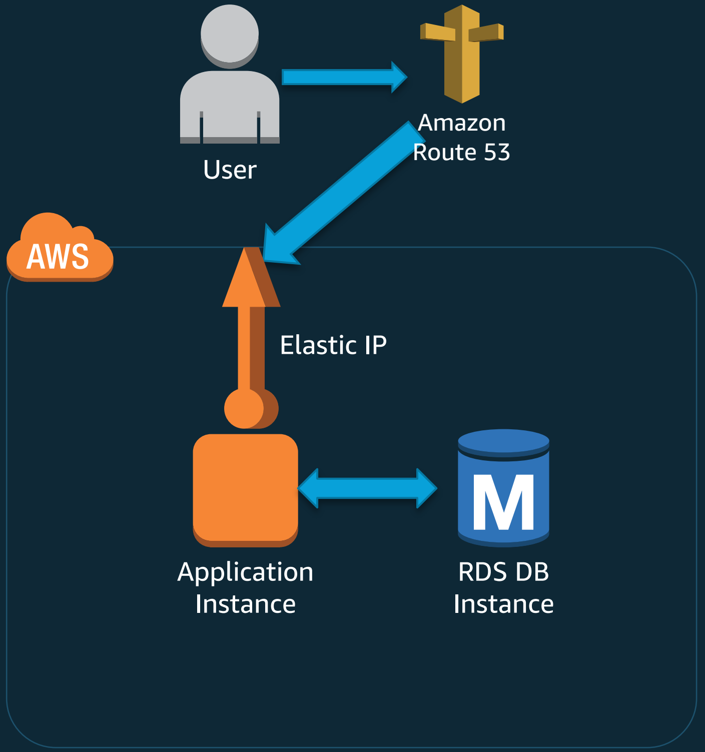
Rapid ingest of data (thousands of records/sec)

*Need != “its easier to do dev without schemas”

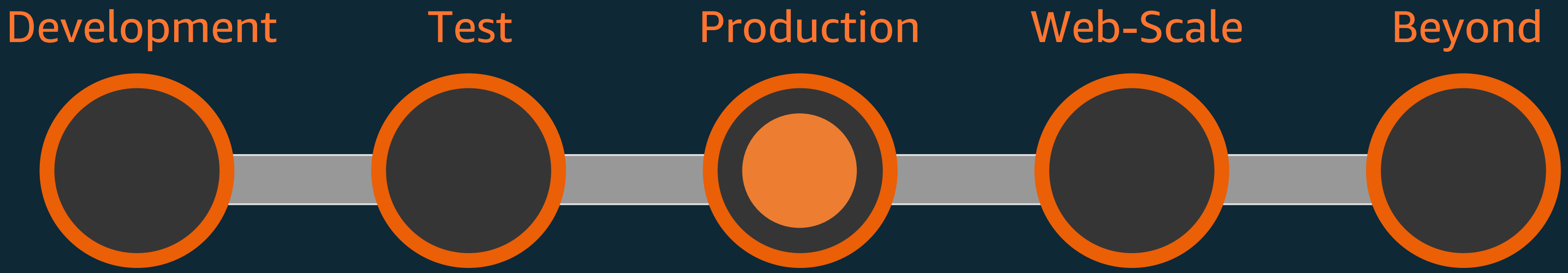
Test

Lets leverage RDS for our database tier minimize the infrastructure management

Now our test environment is available as a separate set of infrastructure



From Development to Web-Scale



Production V1

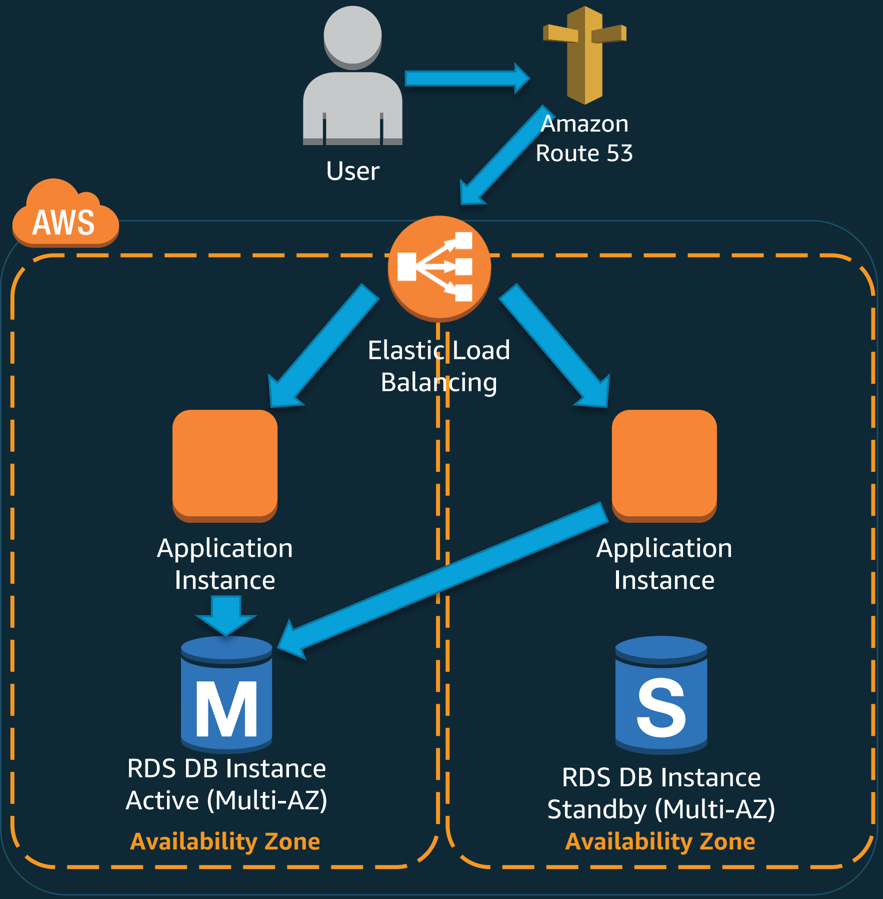
Now we're ready to go to production, we need to address our lack of failover and redundancy issues

Another Application Instance

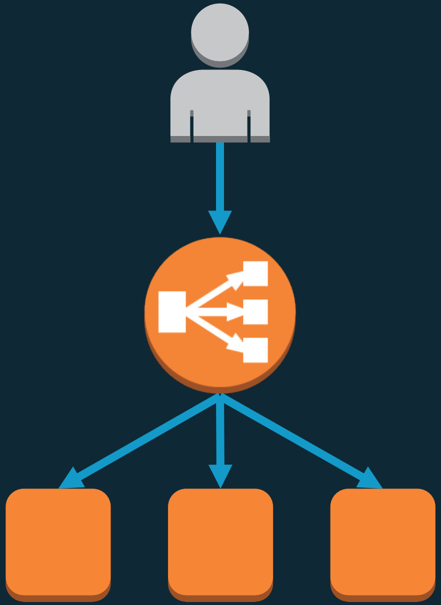
- In another Availability Zone

Elastic Load Balancing

Enable Amazon RDS multi-AZ



Elastic Load Balancing

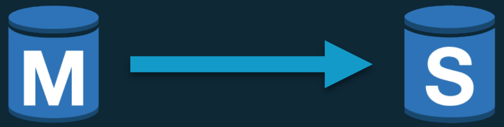


- Load Balancing as a service
- Automatically distributes incoming application traffic across multiple Amazon EC2 instances
- Enables you to achieve greater levels of fault tolerance in your applications
- Built-in application health detection, serve traffic only to operational application instances
- Provides SSL offload
- Seamlessly provides required amount of capacity needed to distribute application traffic

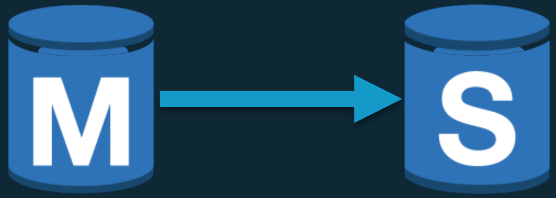
RDS Availability & Scaling Options

Multi-AZ Replication across availability zones of master DB

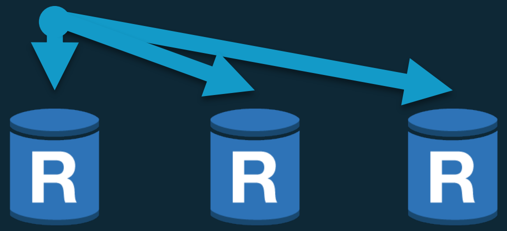
Automated failover from master to slave



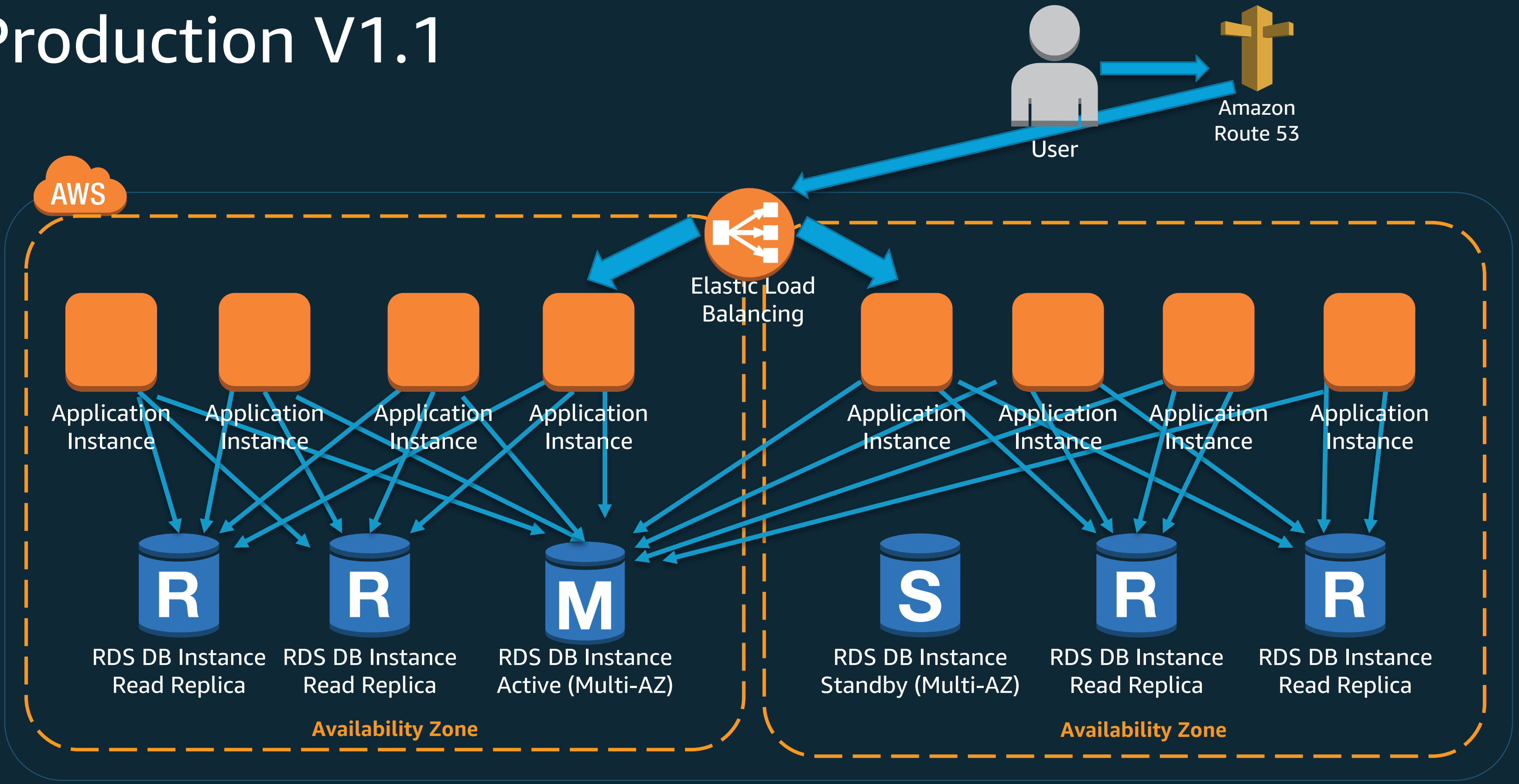
Vertically scale RDS by changing instance size



Horizontally scale RDS with Read-Replicas



Production V1.1



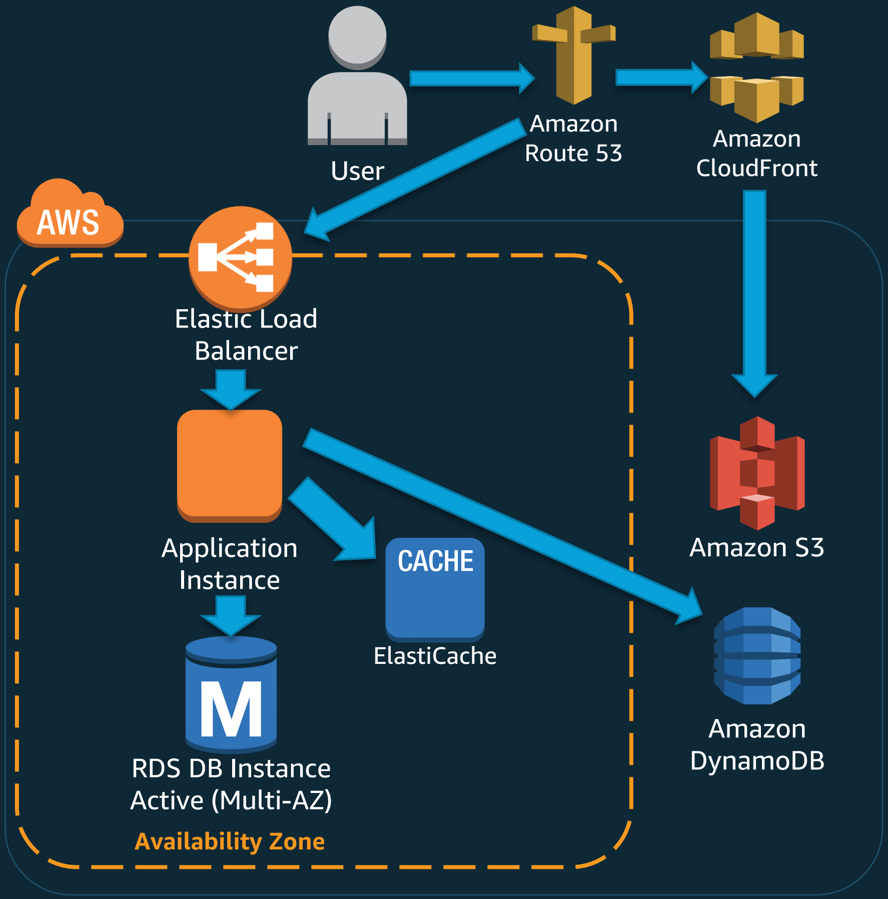
Shift Some Load Around

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

Move static content from the Application Instance to Amazon S3 and CloudFront

Move dynamic content from the Elastic Load Balancing to CloudFront

Move session/state and DB caching to ElastiCache or DynamoDB



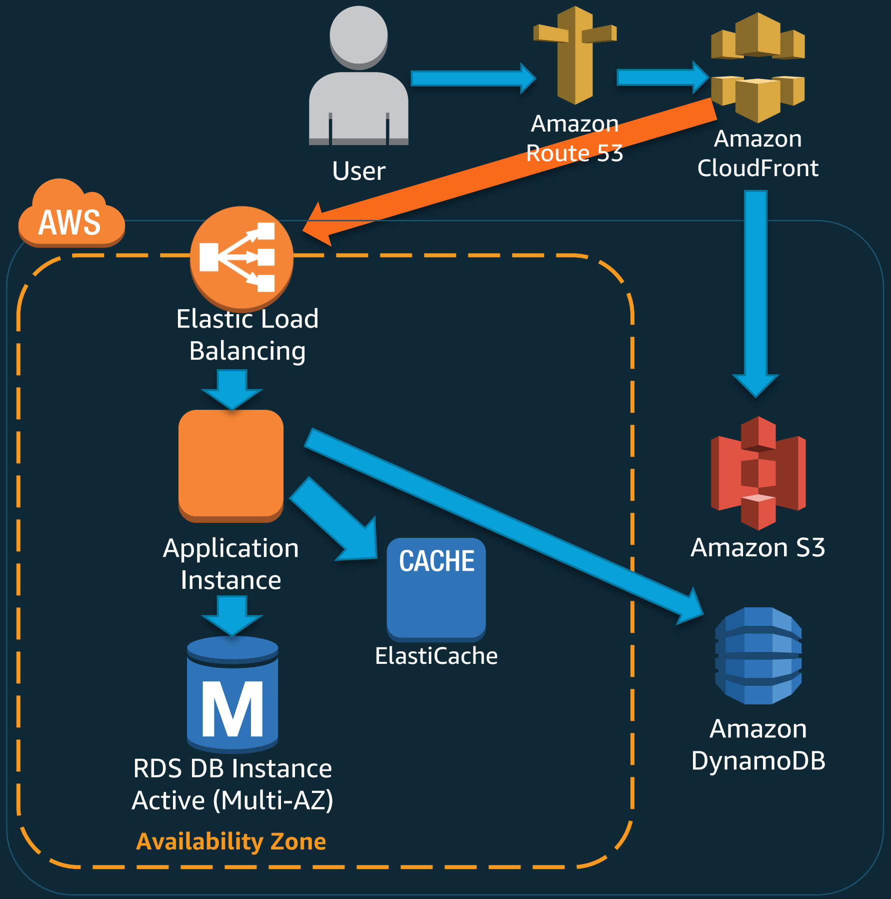
Shift Some Load Around

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

Move static content from the Application Instance to Amazon S3 and CloudFront

Move dynamic content from the Elastic Load Balancing to CloudFront

Move session/state and DB caching to ElastiCache or DynamoDB



Amazon CloudFront

Amazon CloudFront is a web service for scalable content delivery.

Cache content at the edge locations around the world for faster delivery

Helps lower load on origin infrastructure

Dynamic and static content

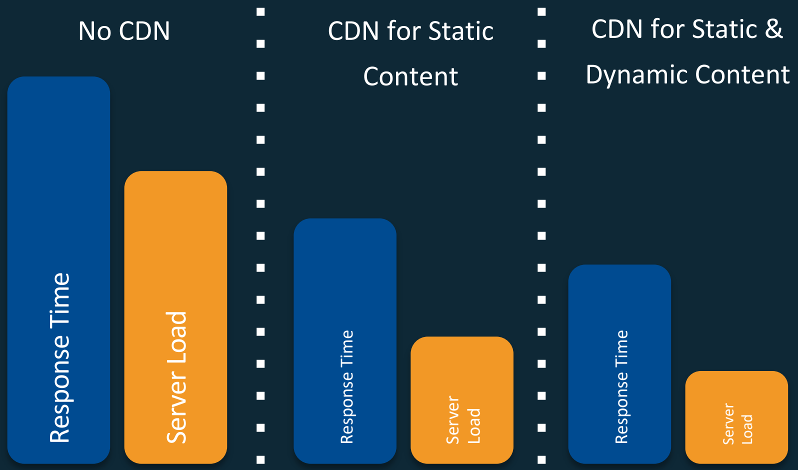
Streaming video

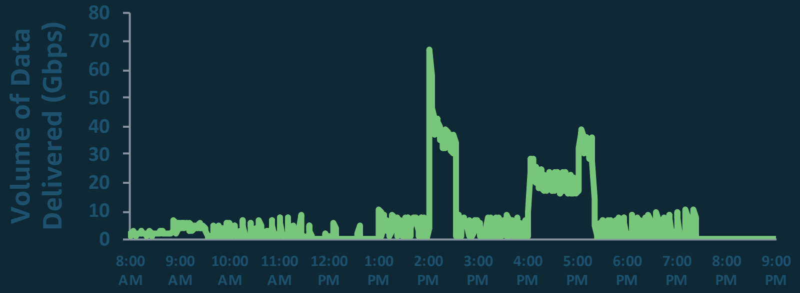
Zone apex support

Custom SSL certificates

Low TTLs (as short as 0 seconds)

Optimized to work with EC2, Amazon S3, Elastic Load Balancing, and Route53





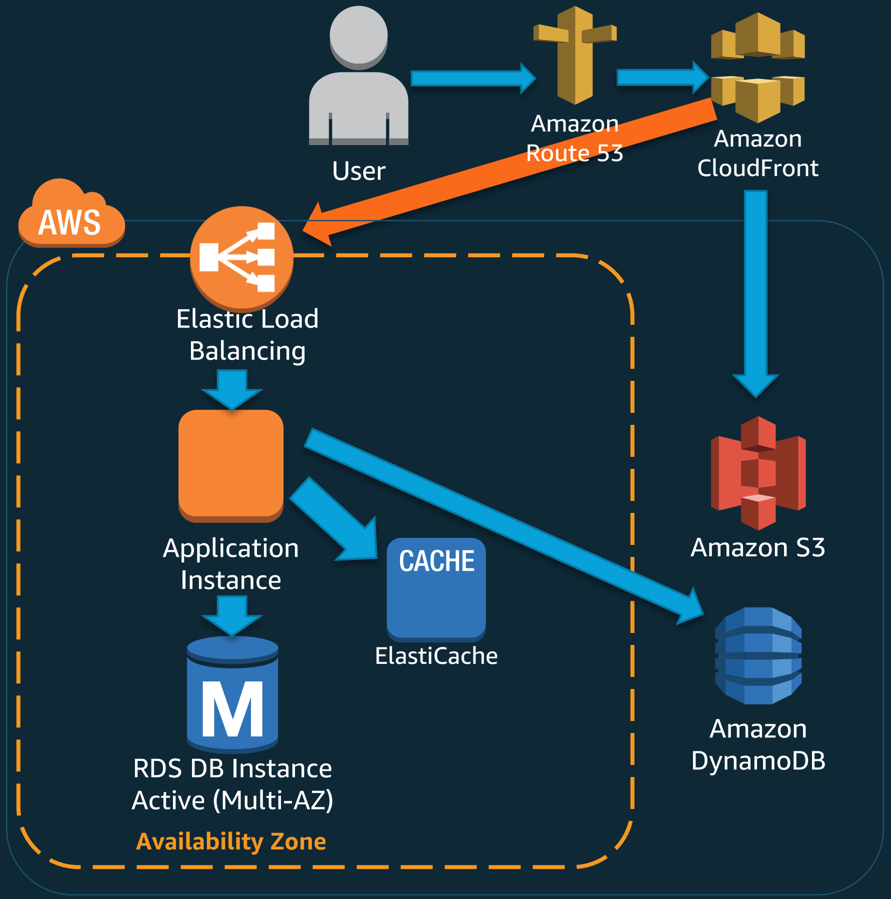
Shift Some Load Around

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

Move static content from the Application Instance to Amazon S3 and CloudFront

Move dynamic content from the Elastic Load Balancing to CloudFront

Move session/state and DB caching to ElastiCache or Amazon DynamoDB



ElastiCache

Hosted Memcached & 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)

Local to a single AZ for Memcache, with no persistence or replication

With Redis can put a replica in a different AZ with persistence

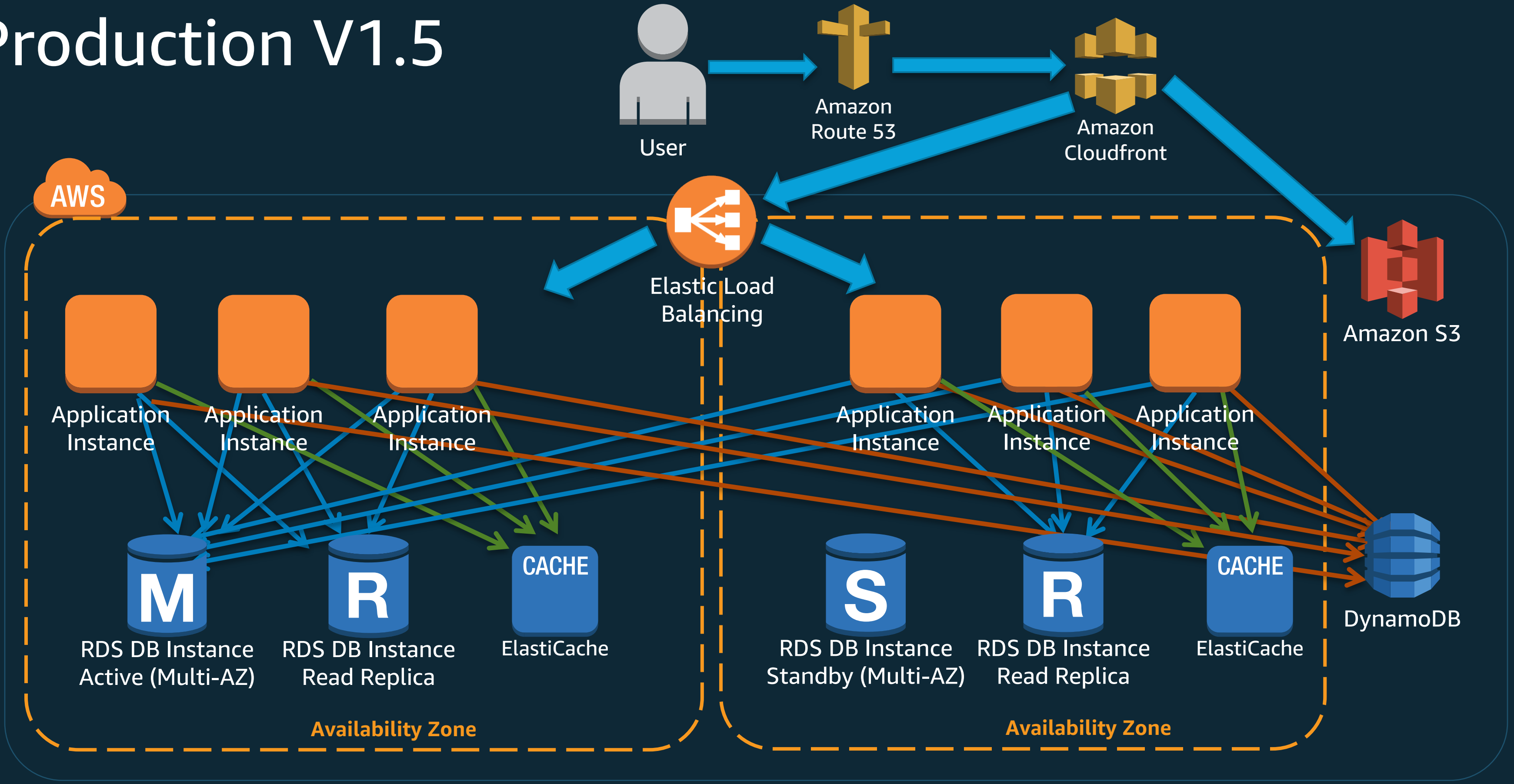
Use AWS’s Auto Discovery client to simplify clusters growing and shrinking, without affecting your application

Amazon DynamoDB

- Fully managed, provisioned throughput NoSQL database
- Fast, predictable performance
- Fully distributed, auto-partitioning, fault-tolerant architecture
- Consider for non-uniform data

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 to CloudWatch
Secure	Integrates with AWS Identity and Access Management (IAM)

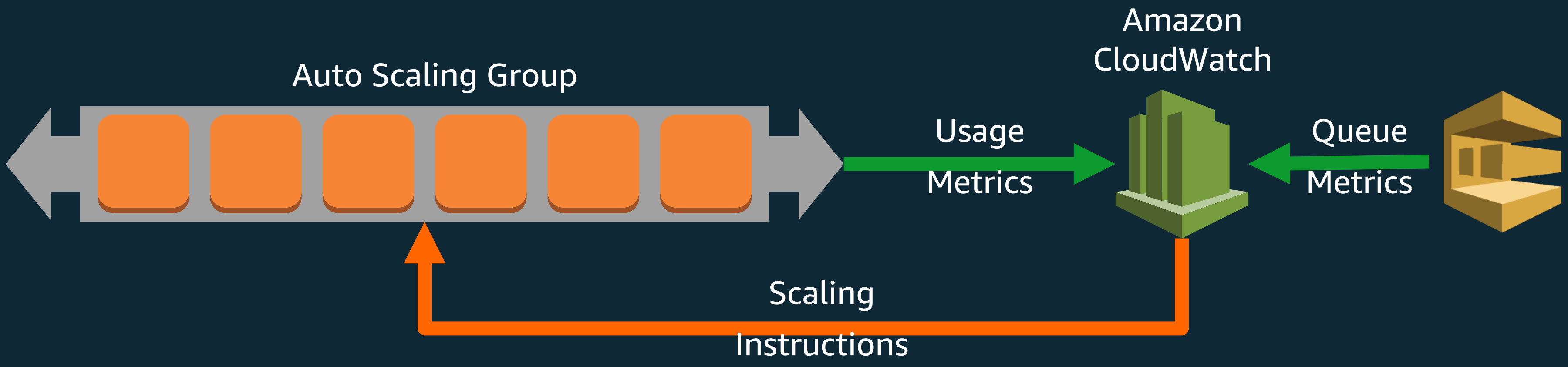
Production V1.5



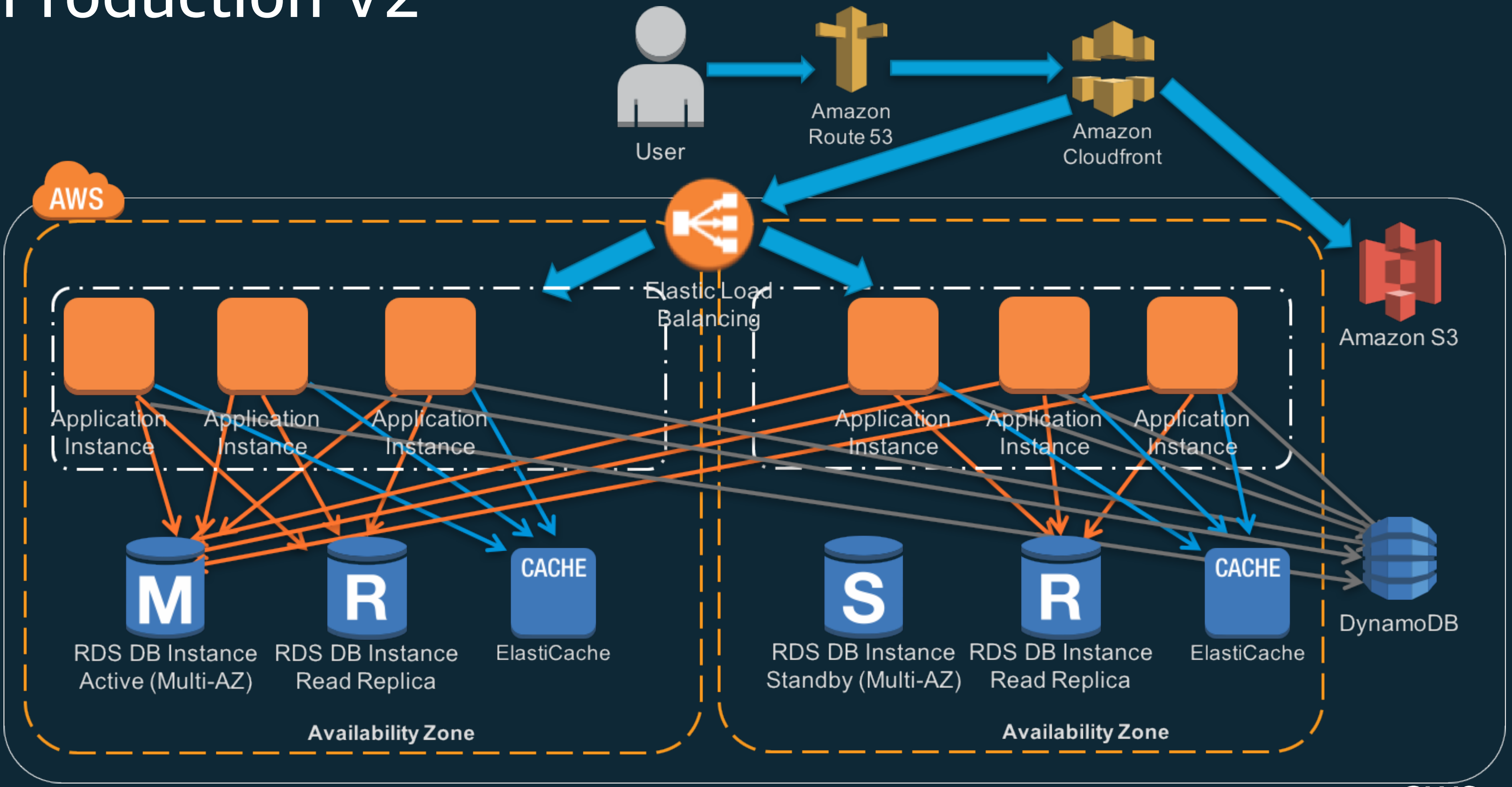
Elasticity with Auto-Scaling

Auto Scaling

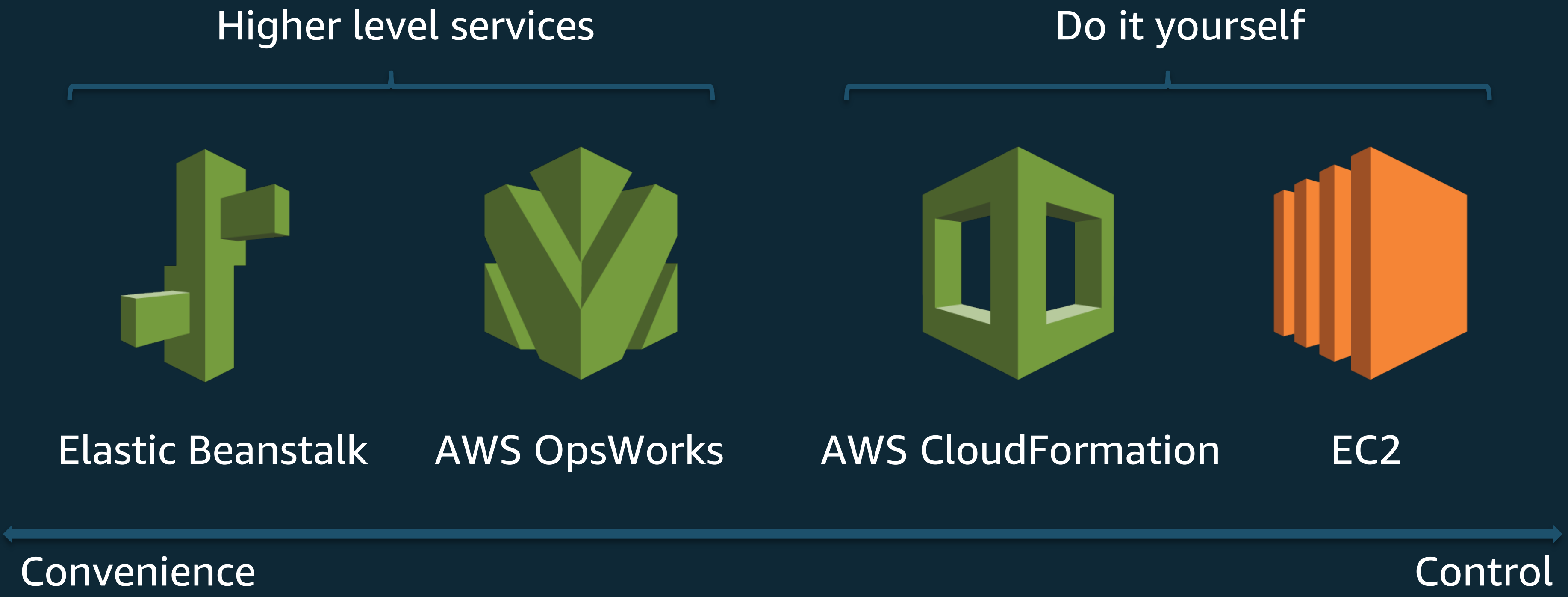
Automatic resizing of compute clusters based on demand
Define minimum and maximum number of instances
Define when scaling up and down occurs
Use metrics collected in Amazon CloudWatch to drive scaling
Run Auto Scaling for On-Demand and Spot instance types



Production V2



AWS Application Management Solutions



Higher level services

Do it yourself

Elastic Beanstalk

AWS OpsWorks

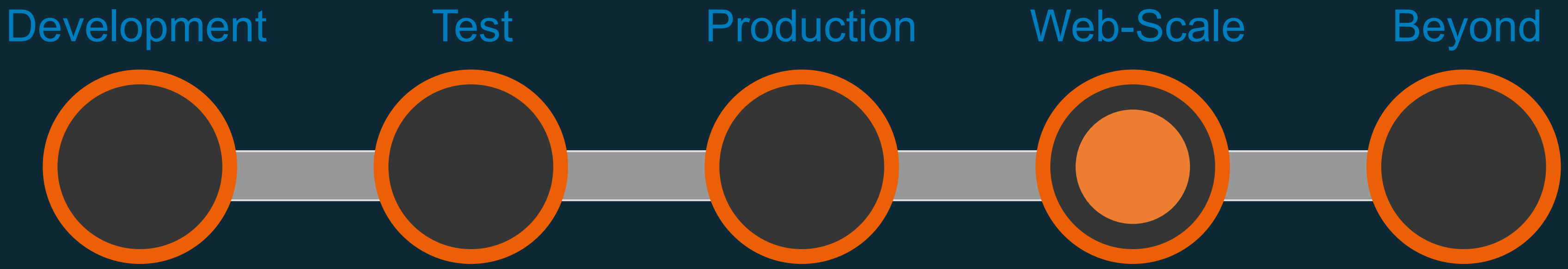
AWS CloudFormation

EC2

Convenience

Control

From Development to Web-Scale



Service

Oriented

Architecture

SOA

Service

Oriented

Architecture

- Move services into their own tiers or modules
- Treat each of these as 100% separate pieces of your architecture
- Scale them independently

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

Cloud SOA

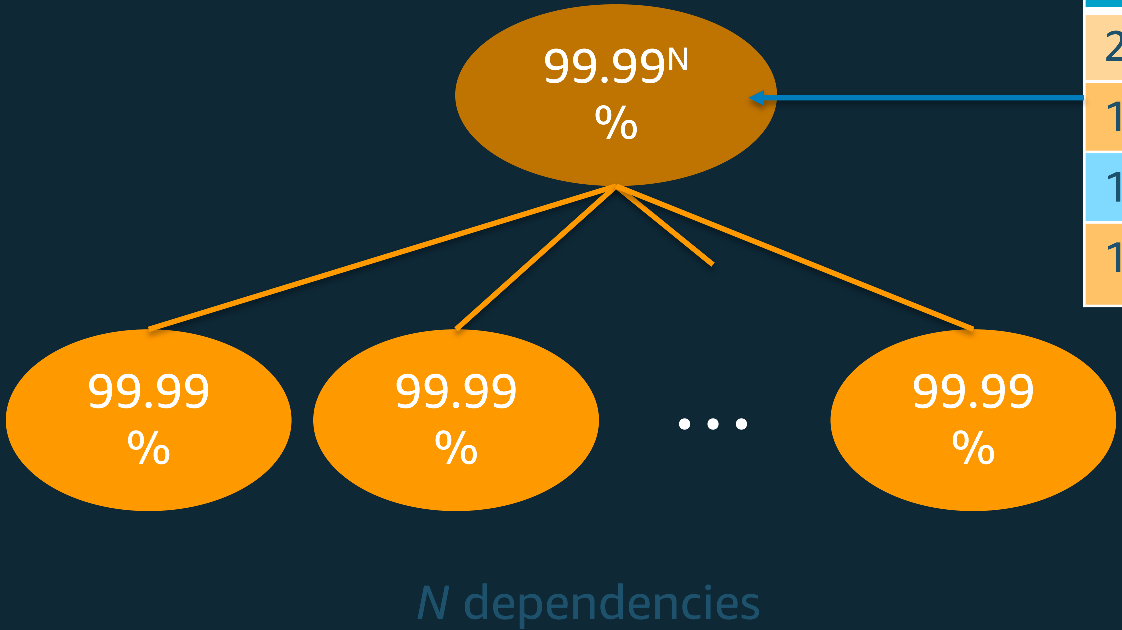
Service

Oriented

Architecture

- Same SOA principles, but leveraging Cloud services
- Allows you to architect for
  - Automation
  - Scale
  - Cost
  - Availability
- Applications are
  - Always-on
  - Self-healing
- Cloud-native New Applications

Availability Conundrum



99.99^N %

99.99 %

99.99 %

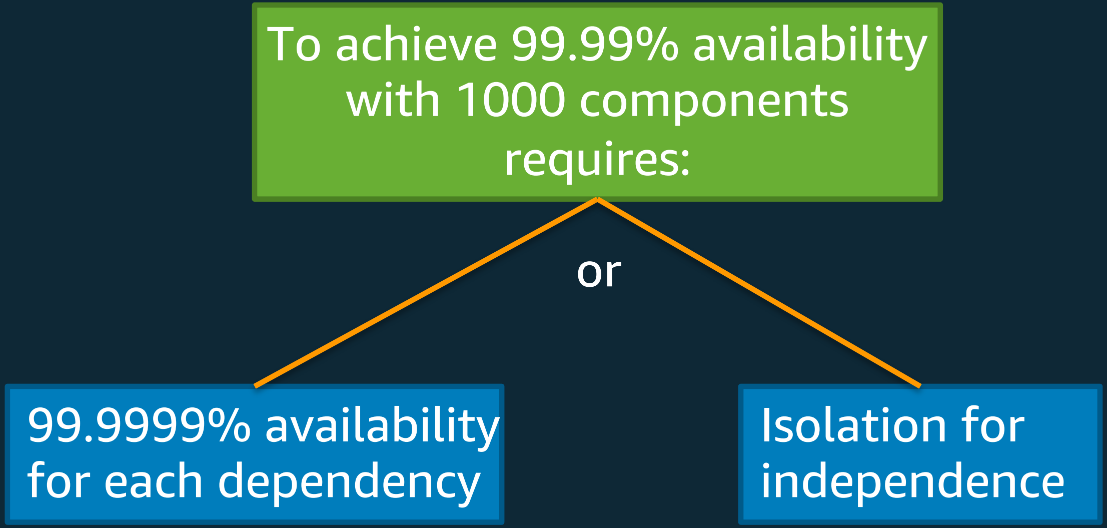
. . .

99.99 %

N dependencies

N **Service Dependencies**	**Availability**
2	99.98%
10	99.9%
100	99%
1000	90%

Availabilities Compound



To achieve 99.99% availability with 1000 components requires:

or

99.9999% availability for each dependency

Component failure leads to system failure

Isolation for independence

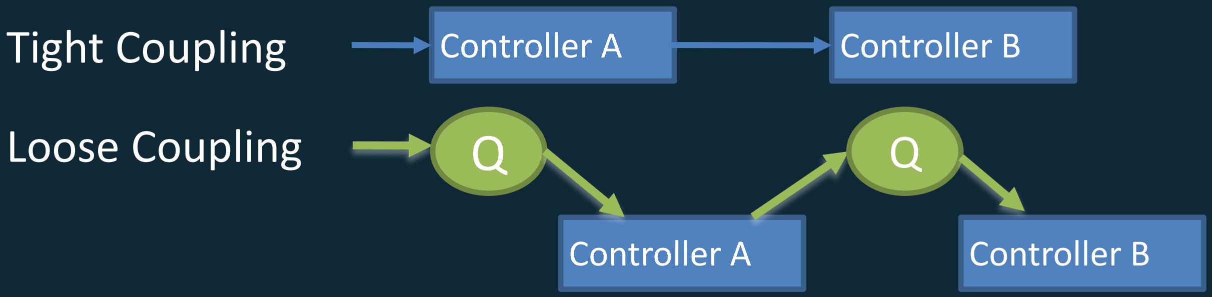
Component failure leads to *degradation* rather than system failure

Loose Coupling

The looser they're coupled, the bigger they scale

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

Use Amazon SQS as Buffers



Loose Coupling + Cloud SOA = Winning

Examples:

Email

Queuing

Transcoding

Search

Databases

Monitoring

Metrics

Logging

Amazon SNS

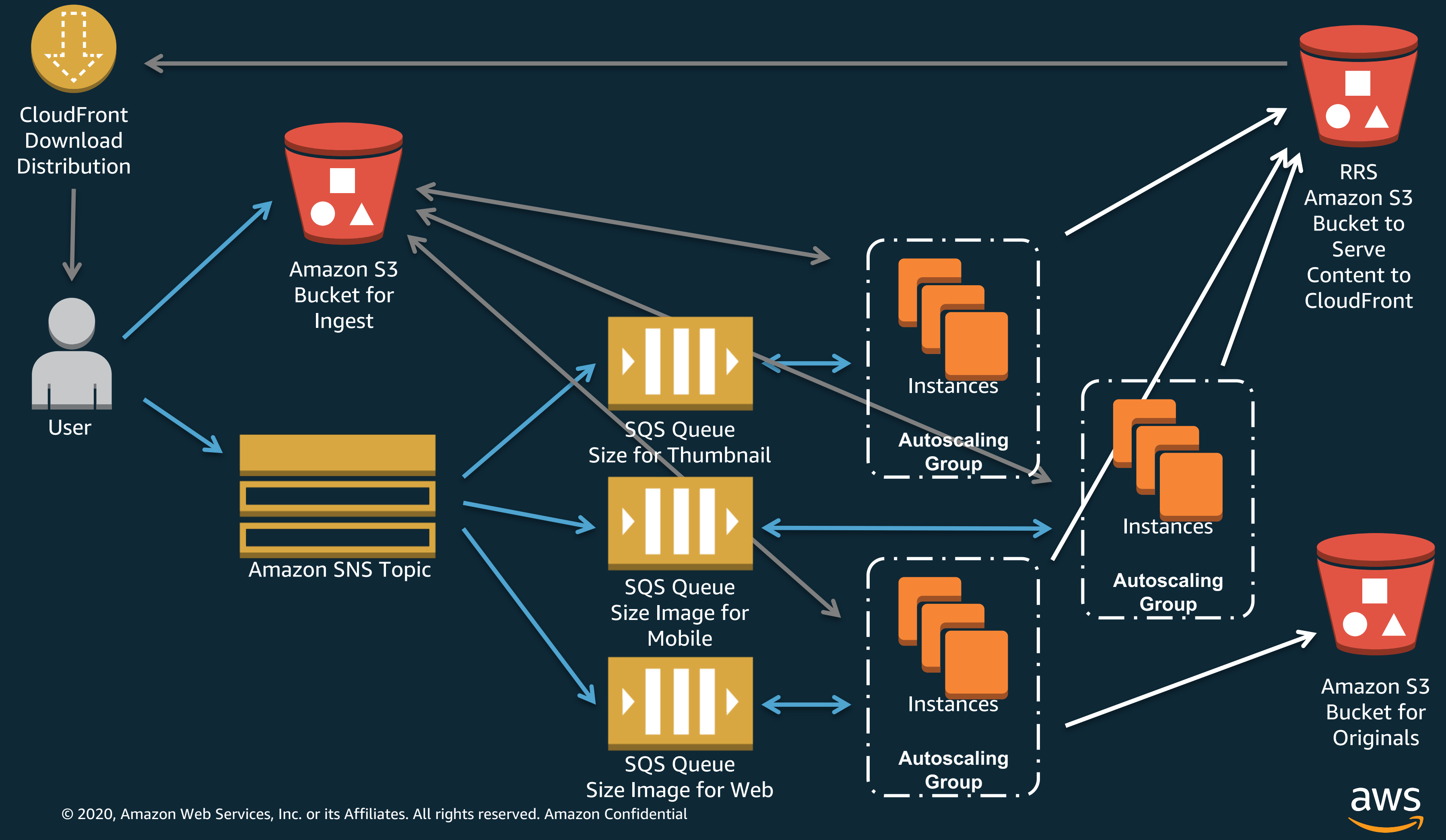
Amazon ElasticSearch

Amazon SQS

Amazon SES

Amazon SWF

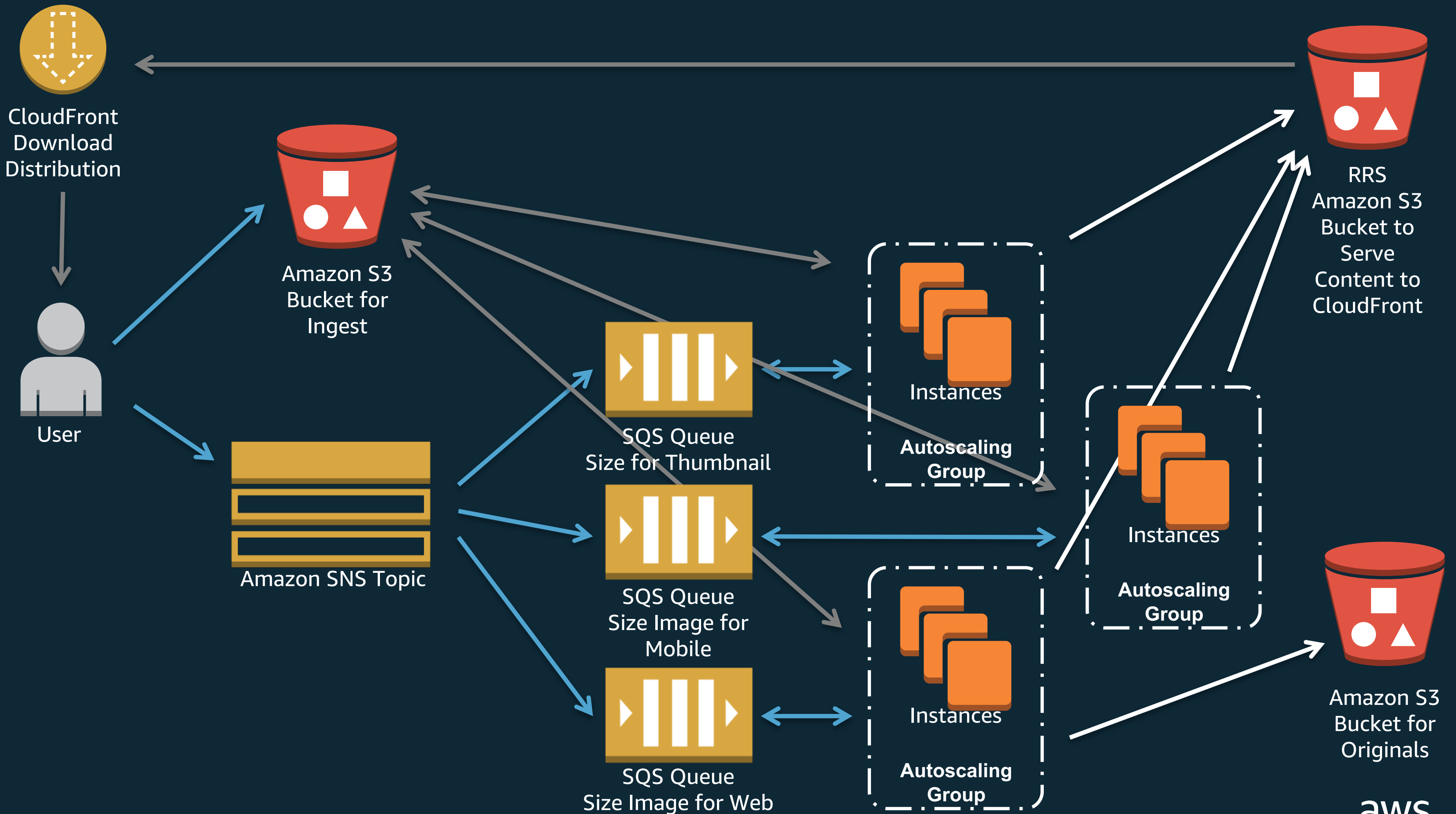
If someone has a service for it already, use that instead of building it yourself

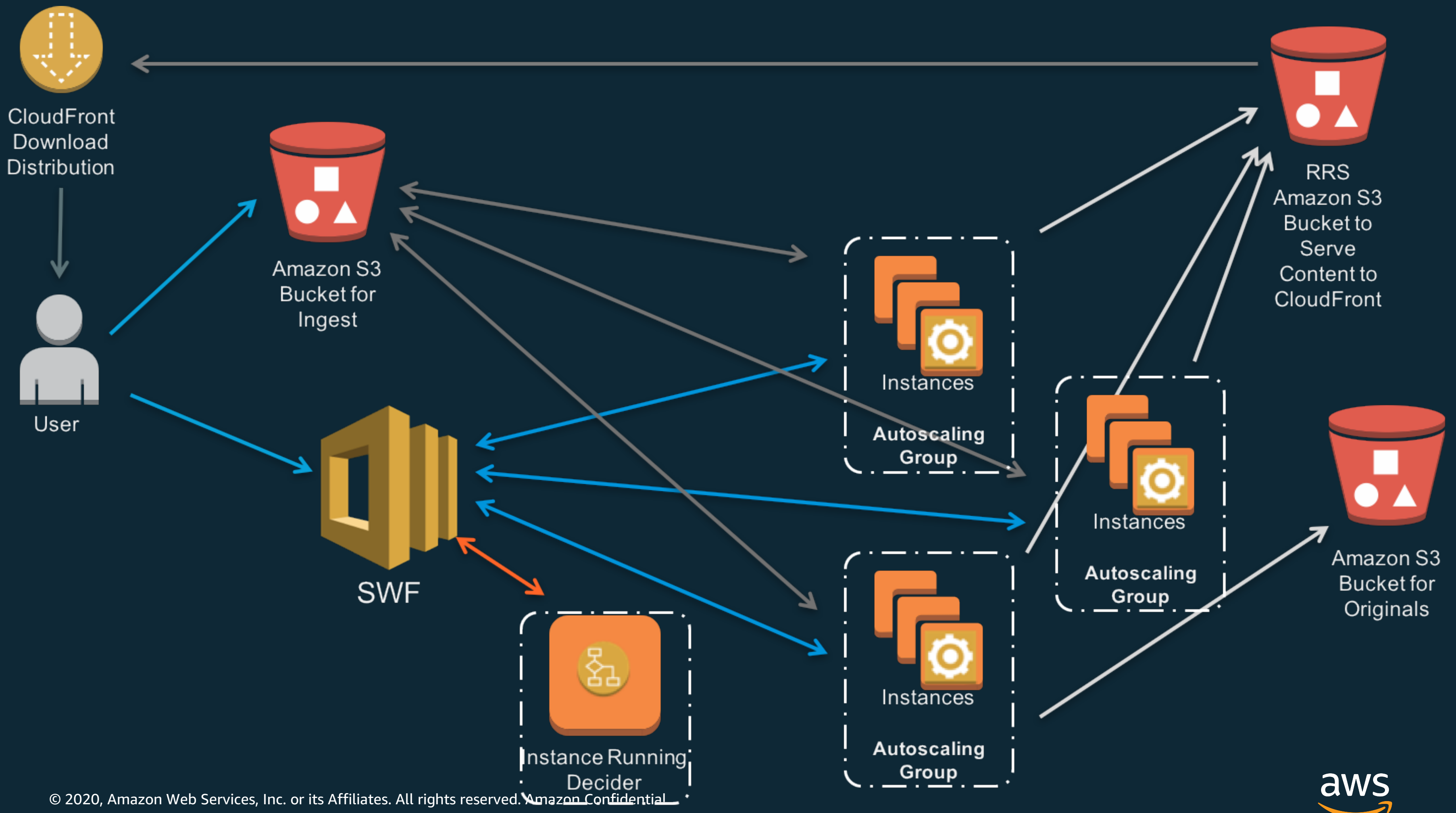

CloudFront Download Distribution
Amazon S3 Bucket for Ingest
RRS Amazon S3 Bucket to Serve Content to CloudFront
User
Amazon SNS Topic
SQS Queue Size for Thumbnail
SQS Queue Size Image for Mobile
SQS Queue Size Image for Web
Instances
Autoscaling Group
Instances
Autoscaling Group
Instances
Autoscaling Group
Amazon S3 Bucket for Originals
aws

Amazon Simple Workflow Service

Provides an orchestration tool across your infrastructure
Can act as a middle layer to pass messages and setup tasks
Lets you break down individual tasks into different workers
Lets you define logic between workers
Lets you make a worker task from anything that can be scripted
Includes built-in retries, timeouts, logging
Features built-in reliability, scalability, and low cost

Your code = Deciders & Workers


CloudFront Download Distribution
RRS Amazon S3 Bucket to Serve Content to CloudFront
Amazon S3 Bucket for Ingest
User
Instances
Autoscaling Group
SQS Queue Size for Thumbnail
Instances
Autoscaling Group
Amazon SNS Topic
SQS Queue Size Image for Mobile
Instances
Autoscaling Group
Amazon S3 Bucket for Originals
SQS Queue Size Image for Web



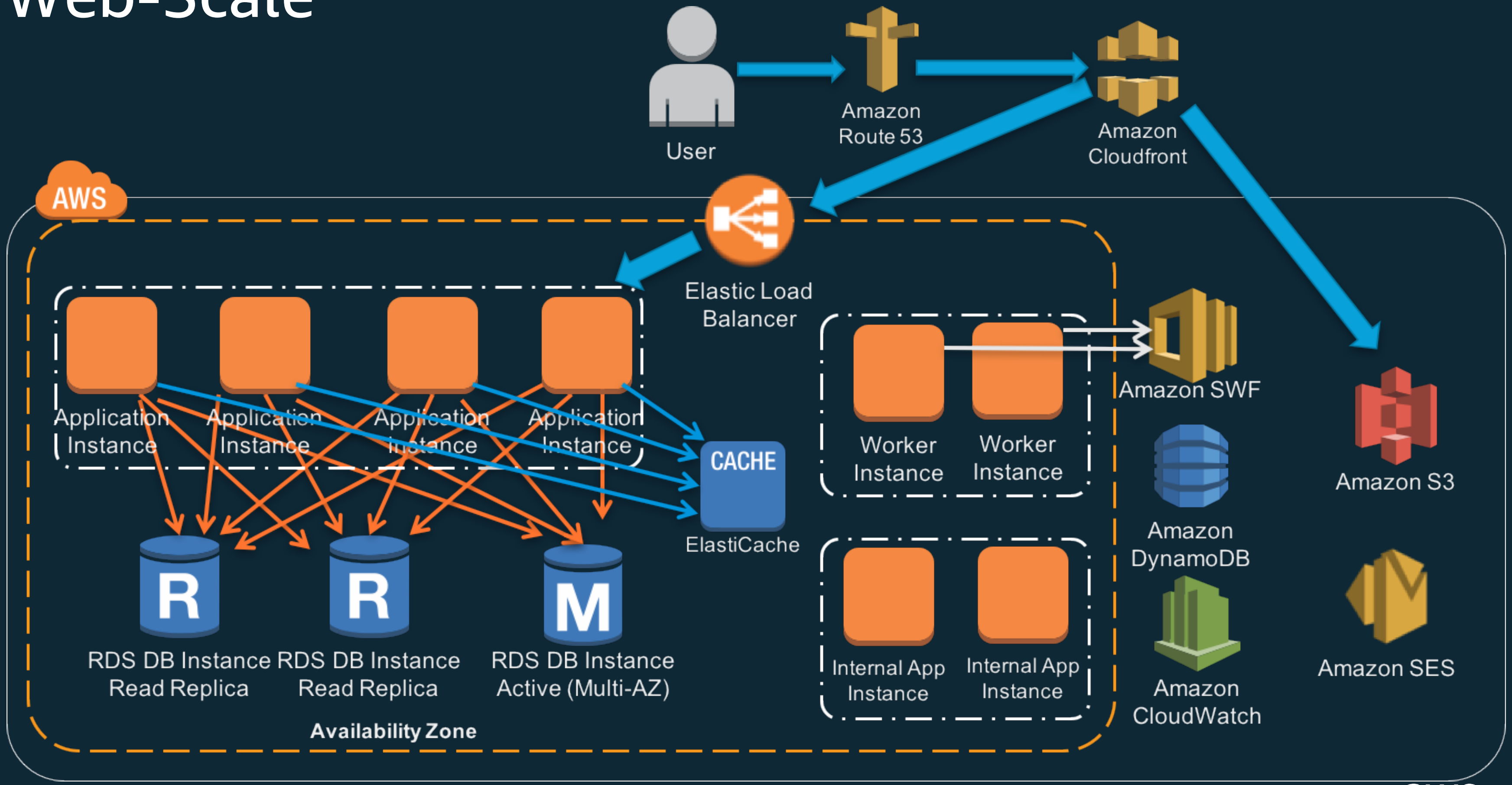
CloudFront Download Distribution
Amazon S3 Bucket for Ingest
RRS Amazon S3 Bucket to Serve Content to CloudFront
User
Instances
Autoscaling Group
SWF
Instances
Autoscaling Group
Amazon S3 Bucket for Originals
Instances
Autoscaling Group
Instance Running Decider

Web-Scale

Reaching web-scale will require most or all of the following:

- Multi-AZ
- Elastic Load Balancing between tiers
- Auto Scaling
- Service-oriented architecture
- Serving content smartly (S3/CloudFront)
- Caching database queries
- Moving state off tiers that auto-scale

Web-Scale


User
Amazon Route 53
Amazon Cloudfront
AWS
Elastic Load Balancer
Application Instance
Application Instance
Application Instance
Application Instance
CACHE
ElastiCache
RDS DB Instance Read Replica
RDS DB Instance Read Replica
RDS DB Instance Active (Multi-AZ)
R
R
M
Availability Zone
Worker Instance
Worker Instance
Internal App Instance
Internal App Instance
Amazon SWF
Amazon DynamoDB
Amazon CloudWatch
Amazon S3
Amazon SES