SWINBURNE UNIVERSITY OF TECHNOLOGY

Cloud Computing Architecture

Lecture 10
Routing across Regions – Route53
Architectural Design Patterns
Sample Architectures

Reminders

- Assignment 3
  - Due 9 am **Thursday 29 October** (Week 12)- submit to Canvas
  - LATE SUBMISSIONS WILL NOT BE ACCEPTED.
  - If you plan to submit Assignment 3 you must register for an interview using the 'Txl}}hv#¡#Dvvljqp hqw#6#Iqwhuylhz v#by **Friday 23 October**
  - If you are doing a group assignment don't forget to create your group under Canvas | People | Assignment 3 by **Friday 23 October**

SWINBURNE UNIVERSITY OF TECHNOLOGY

Last week – Decoupled Architectures

- ☐Loose coupling
- ☐Message Driven Architectures
  - ☐Message Queues
  - ☐MoM middleware, JMS, ActiveMQ
  - ☐AWS SQS (Simple Message Queue)
- ☐Event Driven Architectures
  - ☐Events and Complex Event Processing
  - ☐Publish Subscribe (pub-sub) Systems
  - ☐AWS SNS (Simple Notification Service)

Quizzes:
ACA Mod 6 Decoupling you Infrastructure

SWINBURNE UNIVERSITY OF TECHNOLOGY

3

This week

- Routing across Regions – Route53
- High availability and reliability
  - More high availability patterns
  - Disaster recovery patterns
- Sample Architectures

SWINBURNE UNIVERSITY OF TECHNOLOGY

This week

- **Routing across Regions – Route53**
- High availability and reliability
  - More high availability patterns
  - Disaster recovery patterns
- Sample Architectures

SWINBURNE UNIVERSITY OF TECHNOLOGY

Amazon Route 53

Amazon Route 53 is **an authoritative DNS service from AWS** with the following characteristics:

- DNS translates domain names (like www.amazon.com) into IP addresses.

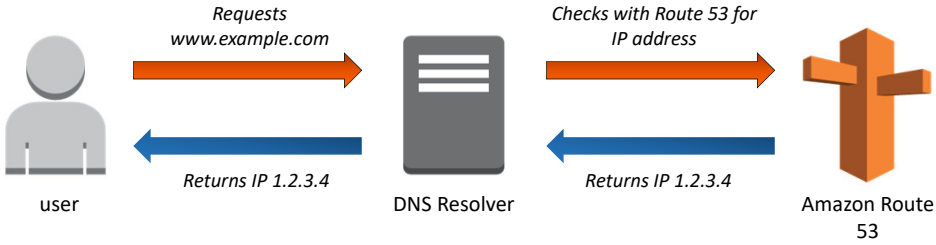
The name refers to the fact that DNS servers respond to queries on port 53.

Amazon Route 53

Elastic load balancers will distribute traffic within a Region. If you want to go outside a Region, you have to use Amazon Route 53. Route 53 is a high availability, authoritative DNS service that translates domain names into IP addresses.

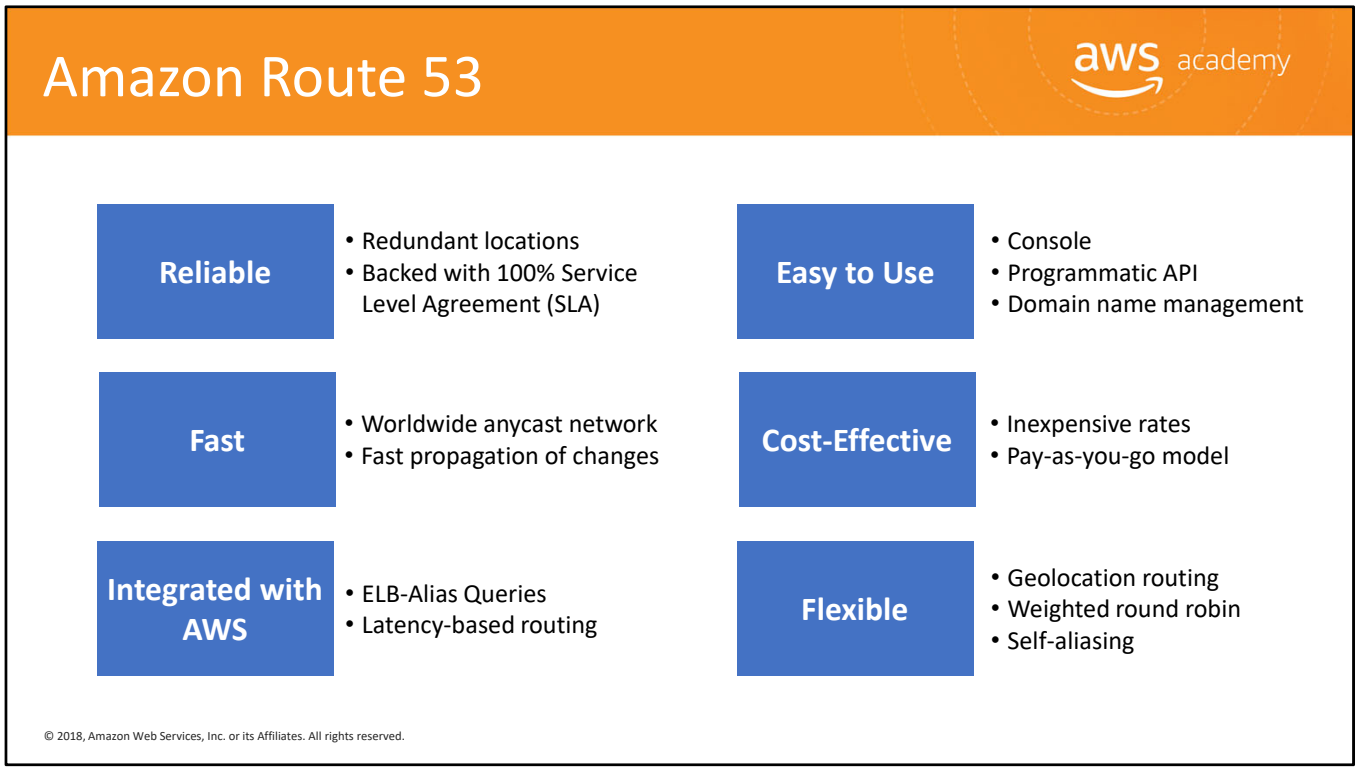
The service is called Route 53 because DNS servers respond to queries on the User Datagram Protocol—or UDP—port 53. Route 53 is one of our only services that has a 100% availability service-level agreement.

Amazon Route 53 DNS Resolution



DNS is how we look up things on the internet.

Here, we see the basic pattern that Route 53 follows when a user initiates a DNS request. When http://example.com is typed in, it goes to a DNS resolver. The DNS resolver checks with your domain in Route 53, gets the IP address of 1.2.3.4, and returns it to the user. That allows the user to get to the website without having to memorize the IP address.



Amazon Route 53 is a Tier 0 service, meaning that it is designed and maintained with the highest level of availability in mind.

Route 53 is more than just a DNS service. You can use it to distribute loads across Regions, it has redundant locations backed by 100% SLA, it is fast and utilizes worldwide anycast (which is network addressing and routing methodology in which a single destination address has multiple routing paths), changes are quickly propagated, and it is accessible via the AWS console or programmatic APIs. It is inexpensive with a pay-as-you-go model. You can set up latency-based routing, which means that your customers will be directed to the fastest server based on their latency.

You can also use geolocation rounding, which is helpful if you are in a business where data has to stay in the country of origin. For example, if a customer originates in Germany, their network traffic never leaves Germany.

Weighted round robin—or WRR—is a network scheduling discipline where the number of packets served is in proportion to the assigned weight, and in inverse proportion to the size of the packets. For example, if you're testing a new application, and you only want to send a small amount of that traffic to your new application to see if everything is working properly, you can use WWR.

Route 53 routing schemes

- **Simple routing:** Single server environments.
- **Weighted round robin:** Assign weights to resource record sets to specify the frequency.
- **Latency-based routing:** Helps to improve your global applications.
- **Health check and DNS failover:** Fail over to a backup site if your primary site becomes unreachable.

What kinds of routing does Amazon Route 53 support? Route 53 supports simple routing in a single server environment, and it supports weighted round robin, which assigns the weights according to the desired access frequency. Route 53 also supports latency-based routing, which provides the fastest connection to the application. Finally, it also offers health checks and DNS failover if the primary site becomes unreachable.

Simple routing—or round robin—uses a simple routing policy for when you have a single resource that performs a given function for your domain, such as one web server that serves content for the example.com website. In this case, Amazon Route 53 responds to DNS queries based only on the values in the resource record set, like the IP address in an A record.

Weighted round robin allows you to assign weights to resource record sets in order to specify the frequency with which different responses are served. You might want to use this capability to do A/B testing, where you send a small portion of traffic to a server where you made a software change. For instance, suppose you have two record sets that are associated with one DNS name: one with weight 3, and one with weight 1. In this case, 75% of the time, Amazon Route 53 will return the record set with weight 3 and 25% of the time Amazon Route 53 will return the record set with weight 1. Weights can be any number between 0 and 255.

Latency-based routing (or LBR) helps you improve your application's performance for a global audience. LBR works by routing your customers to the AWS endpoint (for example Amazon EC2 instances, Elastic IP addresses or load balancers) that provides the fastest experience based on actual performance measurements of the different AWS Regions where your application is running.

Additional Route 53 Supports

- **Geolocation routing:** Specify **geographic locations** by continent, by country, or by state in the United States.
- **Geoproximity routing with traffic biasing:** Route traffic based on the physical distance between your users and your resources.
- **Multivalue answers:** Ability to return multiple health-checkable IP addresses in response to DNS queries as a way to use DNS to improve availability and load balancing.

Additionally, Route 53 supports geolocation and geoproximity routing, which both enable location-based routing. Multivalue answer routing lets you configure Route 53 to return multiple health-checkable IP addresses so you can use DNS to improve availability and load balancing.

Geolocation routing lets you choose the resources that serve your traffic based on the geographic location of your users, or the origin of DNS queries. When you use geolocation routing, you can localize your content and present some or all of your website in the language of your users. You can also use geolocation routing to restrict distribution of content to only the locations where you have distribution rights. Another possible use is for balancing loads across endpoints in a predictable, easy-to-manage way, so that each end-user location is consistently routed to the same endpoint.

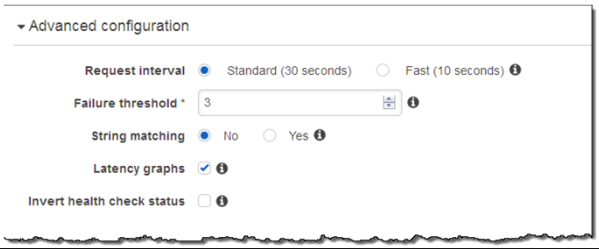
With DNS failover, Amazon Route 53 can help detect an outage of your website and redirect your end users to alternate locations where your application is operating properly. When you enable this feature, Amazon Route 53 health-checking agents will monitor each location or endpoint of your application to determine its availability. You can take advantage of this feature to increase the availability of your customer-facing application.

Geoproximity routing lets you route traffic based on the physical distance between your users and your resources if you're using Route 53 traffic flow. You can also route more or less traffic to each resource by specifying a positive or negative bias. When you create a traffic flow policy, you can specify either an AWS Region, if you're using AWS resources, or the latitude and longitude for each endpoint.

With multiple answers, if you want to route traffic approximately randomly to multiple resources, such as web servers, you can create one multivalue answer record for each resource and, optionally, associate an Amazon Route 53 health check with each record. For example, suppose you manage an HTTP web service with a dozen web servers that each have their own IP address. No one web server could handle all of the traffic, but if you create a dozen multivalue answer records, Amazon Route 53 responds to DNS queries with up to eight healthy records in response to each DNS query. Amazon Route 53 gives different answers to different DNS resolvers. If a web server becomes unavailable after a resolver caches a response, client software can try another IP address in the response.

Amazon Route 53 DNS Failover

- Route 53 tracks health status of resources and take action when error occurs. Configure failover with Amazon Route 53.
- Configure backup and failover scenarios for your own applications.
- Improve health checks by combining multiple health checks, domain name–based health checks, string matching, specifying the request interval, and more.

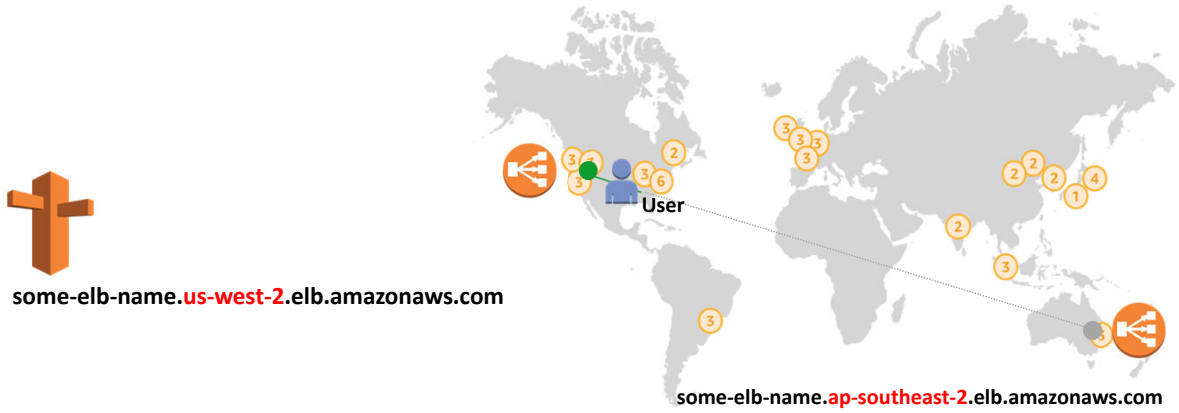


Route 53 lets you track the health status of your resources and take action when an error occurs. It is straightforward to configure failover with Amazon Route 53.

This is a snapshot of what the console would look like if you use Route 53. You can configure backup and failover scenarios for your applications. You can improve health checks by combining multiple health checks, using domain name-based health checks, string matching, and specifying the request interval. The process of configuring these Route 53 settings is minimal.

Note that for string matching, the health check looks for a particular string in the page body, within the first 5 KB of content. You could use string matching to confirm whether the DB instance is working by matching on a string that the page would contain only if it successfully connects to the DB. Failover to the static backup site would then be triggered in the following situations: the web server goes down, the DB instance goes down, if the web server hangs, or if the web server returns invalid content.

Use Case: Multi-Region Deployment



Name	Type	Value
amgogreen.com	ALIAS	some-elb-name.us-west-2.elb.amazonaws.com
amgogreen.com	ALIAS	some-elb-name.ap-southeast-2.elb.amazonaws.com

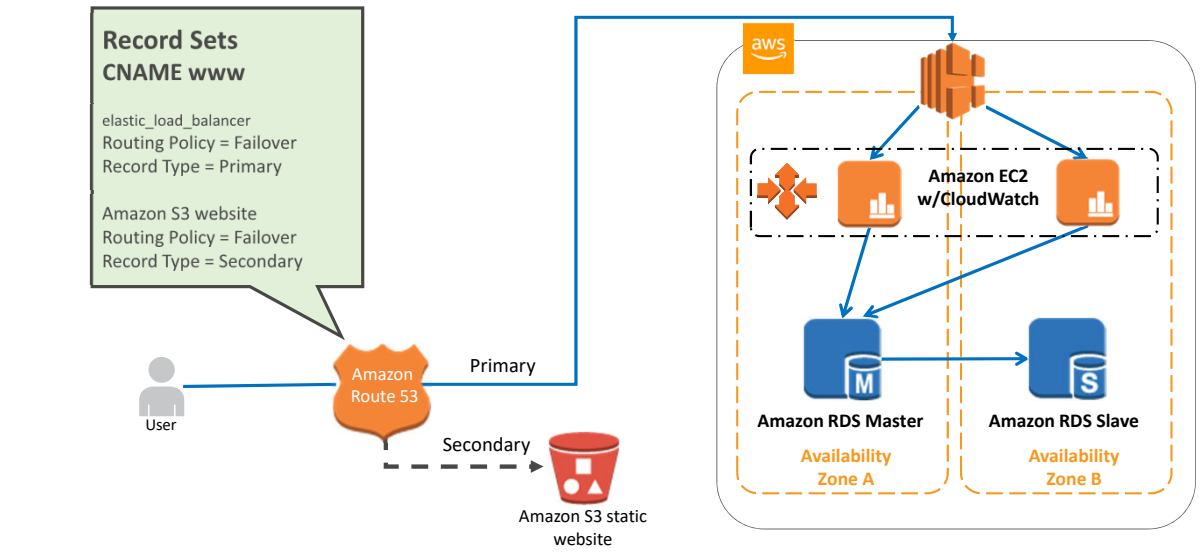
Here is an example of using a multi-Region deployment. There are two deployments, one in the us-west-2 Region on the west coast of the United States, and one in the Asia Pacific Southeast Region. Both load balancers will respond to http://amgogreen.com . The load balancers will direct the user to the closest http://amgogreen.com.

In this example, the user is automatically directed to the Elastic Load Balancing load balancer that's also in the United States, because it's geographically closer to the user. This situation offers a better user experience. If the west coast site went down, all traffic would automatically be redirected to the Asia Pacific Southeast domain.

Benefits include:

- Latency-based routing to the Region
- And load balancing routing to the Availability Zone

Typical Architecture



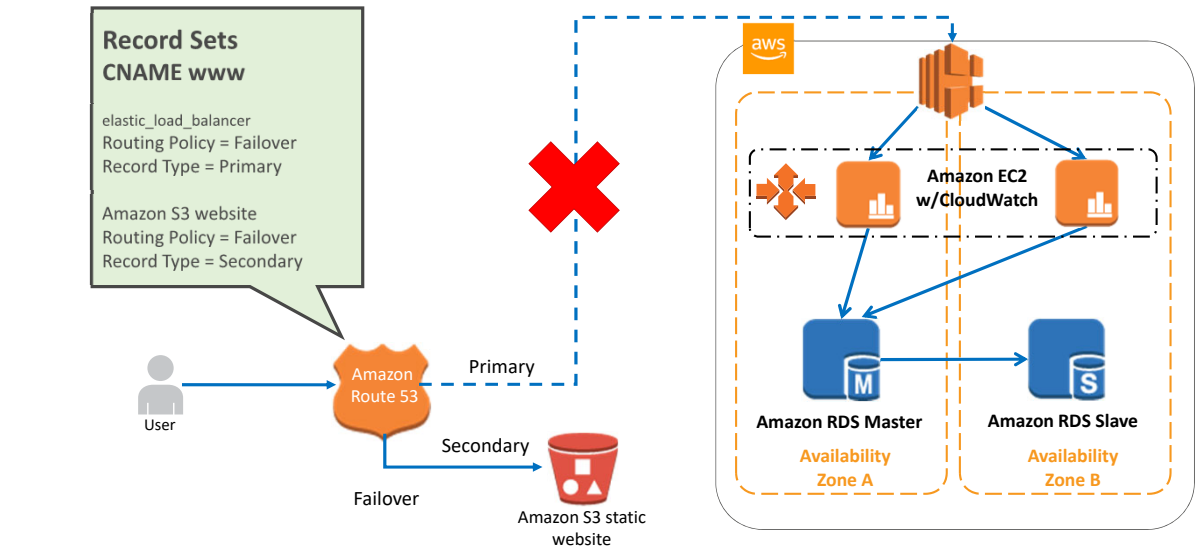
This slide shows an example of how DNS failover works in a typical architecture for a multi-tiered web application. Route 53 passes traffic to an ELB load balancer, which then passes the traffic to a fleet of Amazon EC2 instances that use Auto Scaling.

We can do the following tasks with Route 53 to ensure high availability:

First, create two DNS records for the CNAME www with a routing policy of Failover Routing. The first record is the primary, and it points to the ELB load balancer for your web application. The second record is the “Secondary” route policy, and it points to your static Amazon S3 website. If something happened to the primary link, traffic would automatically be redirected to the static website that’s hosted in the Amazon S3 bucket.

Use Route 53 health checks to make sure the primary is up. If the primary is up, all traffic will default to your web application stack.

Typical Architecture



This slide shows an example of how DNS failover works in a typical architecture for a multi-tiered web application. Route 53 passes traffic to an ELB load balancer, which then passes the traffic to a fleet of Amazon EC2 instances that use Auto Scaling.

We can do the following tasks with Route 53 to ensure high availability:

First, create two DNS records for the CNAME www with a routing policy of Failover Routing. The first record is the primary, and it points to the ELB load balancer for your web application. The second record is the “Secondary” route policy, and it points to your static Amazon S3 website. If something happened to the primary link, traffic would automatically be redirected to the static website that’s hosted in the Amazon S3 bucket.

Use Route 53 health checks to make sure the primary is up. If is the primary is up, all traffic will default to your web application stack.

Getting Started With Amazon Route 53

Guides are offered for each of the following tasks:

- Creating a domain that uses Amazon Route 53 as the DNS service.
- Migrating an existing domain to Amazon Route 53.
- Creating a subdomain that uses Amazon Route 53 without migrating the parent domain.
- Migrating a subdomain to Amazon Route 53 without migrating the parent domain.
- Creating alias record sets for Elastic Load Balancing.

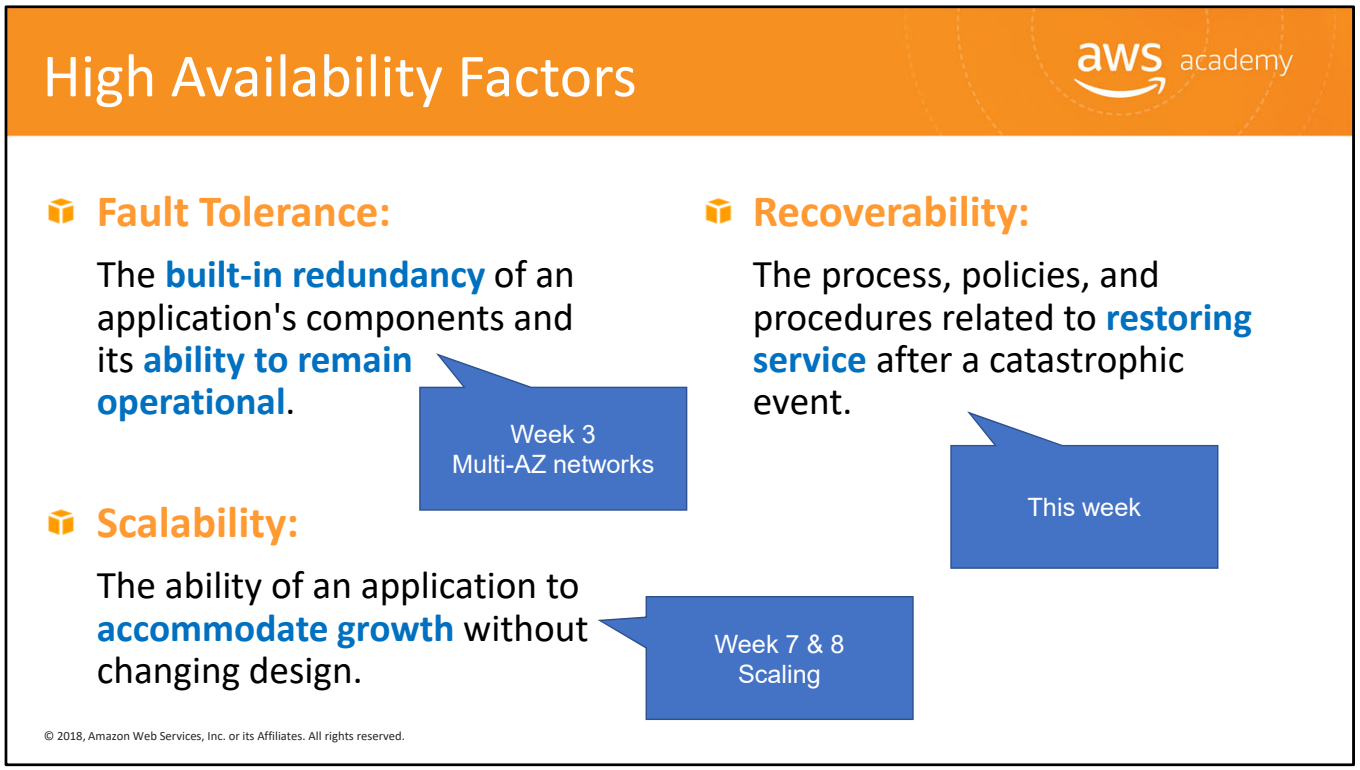
This slide shows some guides on getting started with Amazon Route 53.
Go to each link to learn more. To create a domain that uses Amazon Route 53 as the DNS service select the link. http://docs.aws.amazon.com/Route53/latest/DeveloperGuide/

- To migrate an existing domain to Amazon Route 53 select the link. http://docs.aws.amazon.com/Route53/latest/DeveloperGuide/MigratingDNS.html
- To create a subdomain that uses Amazon Route 53 without migrating the parent domain select the link. http://docs.aws.amazon.com/Route53/latest/DeveloperGuide/CreatingNewSubdomain.html
- To migrate a subdomain to Amazon Route 53 without migrating the parent domain select the link. http://docs.aws.amazon.com/Route53/latest/DeveloperGuide/MigratingSubdomain.html
- To create alias record sets for Elastic Load Balancing select the link. http://docs.aws.amazon.com/Route53/latest/DeveloperGuide/resource-record-sets-choosing-alias-non-alias.html

This week

- Routing across Regions – Route53
- **High availability and reliability**
  - More high availability patterns
  - Disaster recovery patterns
- Sample Architectures

SWINBURNE UNIVERSITY OF TECHNOLOGY



Fault tolerance, recoverability, and scalability are the prime factors that determine the overall availability of your application.

Fault Tolerance is often confused with high availability, but fault tolerance refers to the built-in redundancy of an application's components and its **ability to remain operational** even if some of the components of that system fail. Fault tolerance relies on specialized hardware to detect a hardware fault and instantaneously switch to a redundant hardware component, whether the failed component is a processor, memory board, power supply, I/O subsystem, or storage subsystem. The fault tolerant model does not address software failures, which is by far the most common reason for downtime.

Scalability is a question of how quickly your application's infrastructure can respond to increased capacity needs to ensure that your application is available and performs within your required standards. It does not guarantee availability, but is one part of your application's availability.

Recoverability is often overlooked as a component of availability. In the event a natural disaster makes one or more of your components unavailable, or destroys your primary data source, can you restore service quickly and without lost data? We will not discuss specific disaster recovery strategies in this module.

It is these non-functional requirements that typically define the design of your

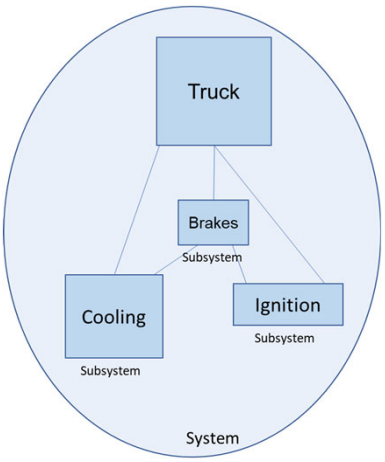
infrastructure. While a highly available and fault-tolerant environment may span multiple Availability Zones and AWS Regions, there are costs associated with this design that must be balanced with the availability requirements.

Reliability vs. Availability

Reliability – A measure of **how long a resource/subsystem/system** performs its intended function.

Two common measures of reliability:

- **Mean Time Between Failure (MTBF)** – Total time in service/number of failures
- **Failure Rate** – Number of failures/total time in service



Availability – A measure of the **percentage of time** the resources are operating normally.

- A percentage of uptime (such as 99.9%) over a period of time (commonly a year).
- **Availability** – Normal Operation Time/Total Time
- **Common Shorthand** – Refers only to the number of 9s; for example, 5 nines is 99.999% available.

Reliability is a measure of how long a resource performs its intended function. What's the difference between that and availability?

As a matter of a fact, **reliability** is closely related to **availability**. **Reliability** is a measure of how long a resource performs its intended function while **availability** is a measure of the percentage of time the resources are in an operable state.

As we looked at the services, we often saw numbers like 99.99%. Those numbers refer to the availability, or the percentage of time that a system or application is correctly performing the operations expected. A common shorthand refers only to the number of9s. For example, 5 nines is 99.999% available.

For more information about reliability, select the link.
https://d0.awsstatic.com/whitepapers/architecture/AWS-Reliability-Pillar.pdf.

This week

- Routing across Regions – Route53
- **High availability and reliability**
  - **More high availability patterns**
  - Disaster recovery patterns
- Sample Architectures

SWINBURNE UNIVERSITY OF TECHNOLOGY

Multi-AZ Pattern

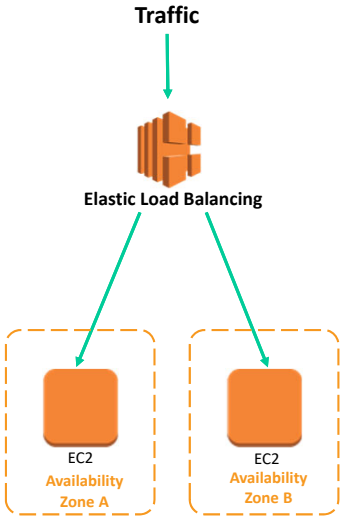
aws academy

Pros:

- If an Availability Zone fails, the system is still available as a whole.

Implementation:

- Create an AMI for your instance.
- Spin up multiple instances using that AMI in multiple AZs.
- Create a load balancer in multiple AZs and attach the instances.
- Confirm instances are attached to load balancer and are in a healthy state.



As we begin, let's review a multi-Availability Zone, or multi-AZ, pattern.

In this example, if an Availability Zone fails, the system is still available as a whole.

To implement a multi-AZ deployment:

- Create an Amazon Machine image—or AMI—for your instance.
- Spin up multiple instances using that AMI in multiple Availability Zones.
- Create a load balancer in multiple Availability Zones, and attach the instances.
- Finally, confirm that the instances are attached to the load balancer, and that they are in a healthy state.

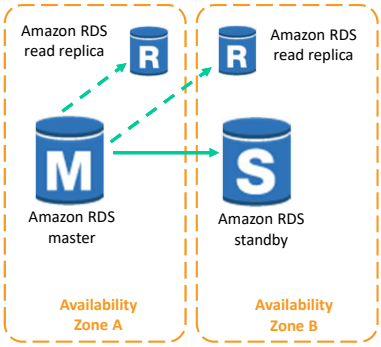
High-Availability Database Pattern

Pros:

- One connection string for master and slave with automatic failover.
- Maintenance does not bring down DB but causes failover.
- Read replicas take load off of master.

Implementation:

- Create an Amazon RDS instance (Aurora, MariaDB, MySQL, Oracle, PostgreSQL or SQL Server).
- Deploy in multiple Availability Zones.
- Create read replicas for each zone (Aurora, MariaDB, MySQL, or PostgreSQL).



In this example, the pros of a high availability database pattern include:

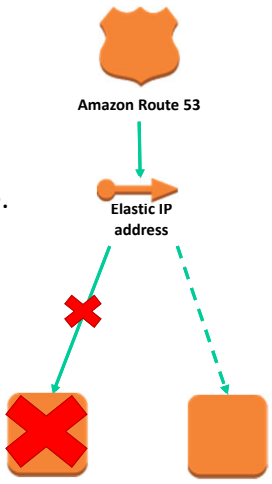
- There is one connection string for the master and slave databases with automatic failover.
- Maintenance does not bring down the database, but it instead causes a failover.
- Finally, read replicas take the load off of the master.

To implement this pattern:

- Create an Amazon Relational Database Service—or Amazon RDS—instance by using Aurora, MariaDB, MySQL, Oracle, PostgreSQL or SQL Server.
- Deploy in multiple Availability Zones.
- Create read replicas for each Availability Zone in Aurora, MariaDB, MySQL, or PostgreSQL.

Floating IP Address Pattern

Problem: Your instance fails or you need to upgrade it, so you need to push traffic to another instance with the same public IP address.
Solution: Use an Elastic IP address.



Pros:

- Since we are moving the Elastic IP address, DNS will not need to be updated.
- Fallback is as easy as moving the Elastic IP address back to the original instance.
- Elastic IP addresses can be moved across instances in different zones in the same region.

Implementation:

- Allocate the Elastic IP address for the EC2 instance.
- Upon failure or upgrade, launch a new EC2 instance.
- Disassociate the Elastic IP address from the EC2 instance and associate it to the new EC2 instance.

Let's discuss a scenario for the floating IP address pattern.

In this scenario, your instance fails, or you need to upgrade it. Thus, you need to push traffic to another instance with the same public IP address.

A solution for this situation would be to use an Elastic IP address.

- Because we are moving the Elastic IP address, the Domain Name System—or DNS—will not need to be updated.
- Fallback is as easy as moving the Elastic IP address back to the original instance.
- Elastic IP addresses can be moved across instances in different Availability Zones that are in the same Region.

To implement this pattern:

- Allocate the Elastic IP address for the Amazon EC2 instance.
- Upon failure or upgrade, launch a new Amazon EC2 instance.
- Disassociate the Elastic IP address from the Amazon EC2 instance and associate it to the new Amazon EC2 instance.

Floating Interface Pattern

Problem: When an instance fails or needs to be upgraded, traffic must be pushed to another instance with the same public and private IP addresses and the same network interface.

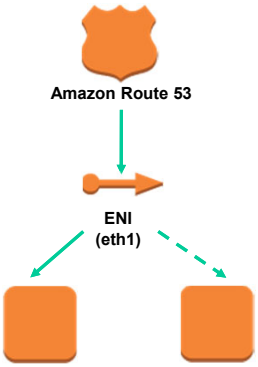
Solution: Deploy your application in VPC and use an elastic network interface (ENI) on eth1 that can be moved between instances.

Pros:

- DNS will not need to be updated.
- Easy rollback: move the ENI back to the original instance.
- Anything pointing to the public or private IP on the instance will not need to be updated.
- ENIs can be moved across instances in a subnet.

Implementation:

- Allocate the ENI for the instance.
- Upon failure or upgrade, launch a new instance.
- Detach the ENI from the instance and attach it to the new instance.



Now, let's review a scenario for a floating interface pattern.

When an instance fails or needs to be upgraded, traffic must be pushed to another instance with the same public and private IP addresses, and the same network interface.

A solution is to deploy your application in a virtual private cloud—or VPC—and use an elastic network interface on eth1. The network interface can be moved between instances.

The advantages are that:

- The DNS will not need to be updated.
- It's easy to roll back your changes by moving the network interface back to the original instance.
- Anything that points to the public or private IP address on the instance will not need to be updated.
- And the network interfaces can be moved across instances in a subnet.

To implement this pattern:

- Allocate the network interface for the instance.
- If you experience failure or need to upgrade, you can launch a new instance.
- Detach the network interface from the instance, and attach it to the new instance.

State-Sharing

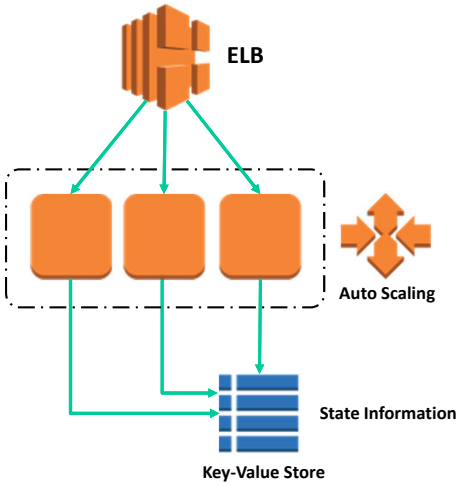
Problem: You want your application to be **stateless** in order to better scale horizontally.
Solution: Move state off your server into a **key-value store.**

Pros:

- This lets you use the scale-out pattern without having to worry about inheritance or loss of state information.

Implementation:

- Use Amazon ElastiCache and DynamoDB for data storage.
- Prepare a data store for storing the state information.
- Use, as a key in the data store, an ID that identifies the user (a session ID or user ID), and store the user information as a value.
- Store, reference, and update the state information in the data store, instead of storing it in the web/APP server.



Sharing application state is an important feature to implement for an automatically scaled fleet.

In this scenario, you want your application to be stateless so that you can better scale it horizontally.

The solution for this scenario is to move the state off of your server, and into a key-value store, or KVS.

This solution lets you use the scale-out pattern without having to handle inheritance or deal with the loss of state information.

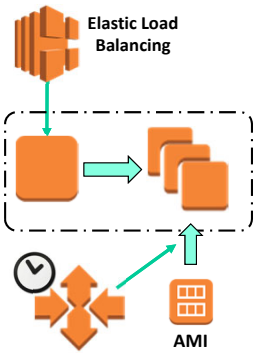
To implement this feature:

- Use Amazon ElastiCache and Amazon DynamoDB for data storage.
- Prepare a data store for storing the state information.
- When you define keys in the data store, use an identifier that identifies the user—such as a session ID or user ID—and store the user information as a value.
- Finally, store, reference, and update the state information in the data store, instead of storing it in the web or application server.

Scheduled Scale-Out

Problem: An application's traffic does not scale organically, but has huge jumps at specific periods of the day or for an event.

Solution: Use **Scaling by Schedule** or **Scaling by Policy**.



Advantages:

- Scale in advance of a traffic spike you know will occur.

Implementation:

- Create a customized AMI.
- Create a Launch Config for your Auto Scaling group.
- Create an Auto Scaling group for your instances (behind a load balancer).
- Options:
  - Create Schedule Update to launch or terminate instances at a specified time.
  - Create Scale by Recurrence policy that will automatically scale your instances based upon cron.

In this example of a scheduled scale-out, an application's traffic does not scale organically. Instead, the application traffic has huge jumps at specific periods of the day or for an event.

A solution for this scenario would be to use Scaling by Schedule or Scaling by Policy.

Using these features allows you to scale resources in advance of a traffic spike that you know will occur.

To implement this solution:

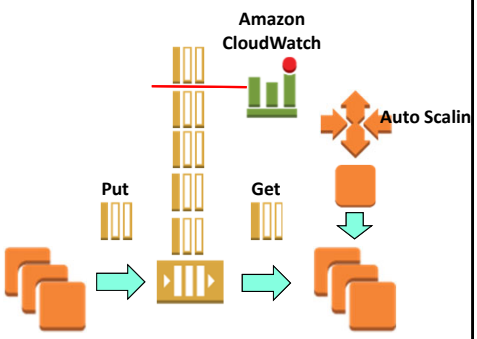
- Create a customized AMI.
- Create a Launch Config for your Auto Scaling group.
- Create an Auto Scaling group for your instances, and make sure that the Auto Scaling group is behind a load balancer.

To do this, you can:

- Create a Schedule Update to launch or terminate instances at a specified time.
- Or create a Scale by Recurrence policy that will automatically scale your instances based upon cron.

Job Observer Pattern

Problem: You need to manage resources against the depth of your work queue.
Solution: Create an Auto Scaling group to scale compute resources based upon queue depth.



Pros:

- Compute scales with job size, providing efficiency and savings.
- Job can be completed in a shorter timeframe.
- Even if a work item fails to complete, process is resilient.

Implementation:

- Work items for batch job placed in Amazon SQS queue as messages.
- Auto Scaling group should be created to scale compute resources up or down based upon Amazon CloudWatch queue depth metric.
- Batch processing servers retrieve work items from Amazon SQS to complete job.

In this scenario for the job observer pattern, you need to manage resources against the depth of your work queue.

The solution for this scenario is to create an Auto Scaling group that will scale your compute resources based upon your queue depth.

The advantages of this solution are that:

- Compute resources scale with the job size, which provides efficiency and savings.
- The job can be completed in a shorter timeframe.
- And even if a work item fails to complete, the process is resilient.

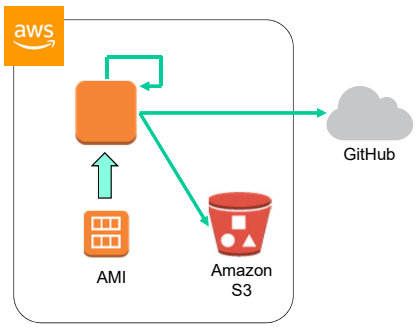
To implement this pattern:

- Place the batch job in the queue from Amazon Simple Queue service—or Amazon SQS—as messages.
- An Auto Scaling group should be created to scale compute resources up or down, based on the Amazon CloudWatch queue depth metric.
- Finally, batch processing servers retrieve work items from Amazon SQS to complete the job.

Bootstrap Instance

Problem: Code releases happen often. Creating a new AMI every time you have a release and managing these AMIs across multiple regions is difficult.

Solution: Develop a base AMI, and then bootstrap the instance during the boot process to install software, get updates, and install source code so that your AMI rarely or never changes.



Pros:

- Do not need to update AMI regularly and move customized AMI between regions for each software release.

Implementation:

- Identify a base AMI to start from.
- Create a repository where your code is located.
- Identify all packages and configs that need to occur at launch of the instance.
- During launch, pass user data to your EC2 instances that will be executed to bootstrap your instance.

Let's review a scenario for a bootstrap instance.

Code releases happen often, so it is difficult to create a new AMI every time you have a release, and to manage these AMIs across multiple Regions.

A solution for this scenario is to develop a base AMI. You then bootstrap the instance during the boot process to install software, get updates, and install source code so that your AMI rarely or never changes.

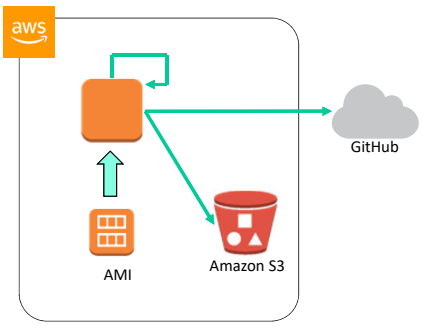
The advantage of this solution is that you do not need to update the AMI regularly, and you also do not need to move the customized AMI between Regions for each software release.

To implement this solution:

- Identify a base AMI to start from.
- Create a repository to store your code.
- Identify all packages and configuration files that need to occur when the instance launches.
- During the launch of the instance, you can pass the user data to your Amazon EC2 instances. This data will be executed to bootstrap your instance.

Bootstrap Instance: Example

```
66      "LaunchConfig" : {
67        "Type" : "AWS::AutoScaling::LaunchConfiguration",
68        "Metadata" : {
69          "Comment1" : "Configure the bootstrap helpers to install the Apache Web Server and PHP",
70          "Comment2" : "The website content is downloaded from the index.zip file",
71
72          "AWS::CloudFormation::Init" : {
73            "config" : {
74              "packages" : {
75                "yum" : {
76                  "httpd"        : [],
77                  "php"          : []
78                }
79              },
80
81              "sources" : {
82                "/var/www/html" : "https://s3.amazonaws.com/bhol/index.zip"
83              },
84
85
86              "services" : {
87                "sysvinit" : {
88                  "httpd" : {
89                    "enabled"       : "true",
90                    "ensureRunning" : "true"
91                  }
92                }
93              }
94            }
95          }
96        },
```



Displayed is an example of a bootstrap instance.

This week

- Routing across Regions – Route53
- **High availability and reliability**
  - High availability patterns
  - **Disaster recovery patterns**
- Sample Architectures

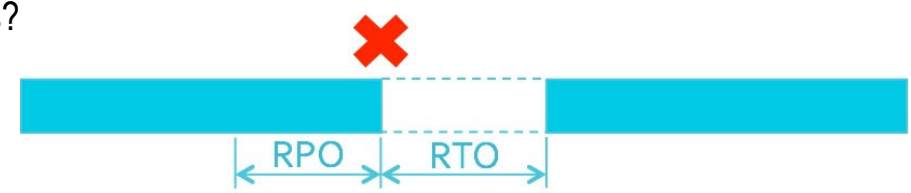
SWINBURNE UNIVERSITY OF TECHNOLOGY

High Availability: Driven By Your Requirements

- Recovery Time Objective (**RTO**)
  How quickly must the system recover?

- Recovery Point Objective (**RPO**)
  How much data can you afford to lose?

- How much money do you need to invest to meet those objectives?



Essentially, non-functional requirements define the design of your infrastructure. A highly available and fault-tolerant environment expands multi-AZ and cross-regions; however, there are costs associated with that design. We discussed in the *Designing for Cost* module that pricing can differ from region to region.

Disaster Recovery on AWS

Example architectural patterns (sorted from highest to lowest RTO/RPO):

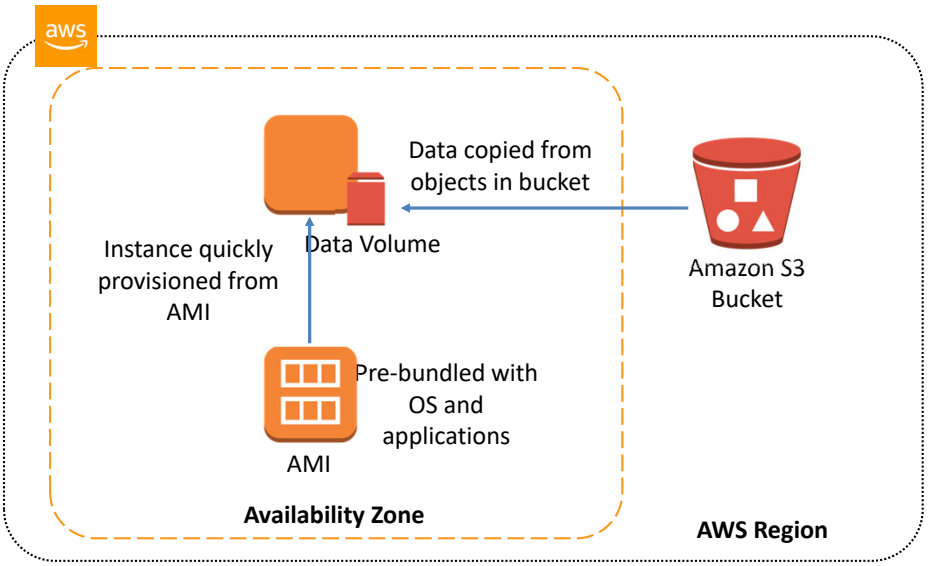
- Backup and Restore
- Pilot Light
- Fully Working Low-Capacity Standby
- Multi-Site Active-Active

Let’s cover some common practices for disaster recovery—or DR—inside of AWS.

The example architectural patterns involve recovery time objectives, or RTOs, and recovery point objectives, or RPOs. They are sorted from the highest RTO and RPO to the lowest RTO and RPO.
As a reminder, the recovery time objective is the time that your business is down during an outage. The recovery point objective is the amount of data that will be lost.

- The first example is when you want to perform a simple backup and restore operation.
- The second example is the “pilot light” architecture.
- The third example is a fully working low-capacity standby architecture.
- And the last example is a multi-site active-active architecture.

Backup and Restore



Here is an example of how you would restore the system using the backup and restore method. In this scenario, you should already have an Amazon Machine Image, or AMI, that you can use to provision an instance. After you provision an instance, you can then copy the data over from your Amazon S3 bucket. Then, you switch Amazon Route 53 to point to your site in AWS.

Backup and Restore

Advantages

- Simple to get started.
- Extremely cost effective (mostly backup storage).

Preparation Phase

- Take backups of current systems.
- Store backups in Amazon S3.
- Describe procedures to restore from backup on AWS.
  - Know which AMI to use; build your own as needed.
  - Know how to restore system from backups.
  - Know how to switch to new system.
  - Know how to configure the deployment.

The advantages of the backup and restore option are that it's simple to get started, and it's extremely cost-effective. In this scenario, you will mostly pay for backup storage.

For the preparation phase:
Take backups of current systems.
Store backups in Amazon S3.
Describe procedures to restore from backups on AWS.

- Know which AMI to use, and build your own AMI as needed.
- Know how to restore the system from backups.
- Know how to switch to a new system.
- And know how to configure the deployment.

Backup and Restore

In case of disaster

- Boot up required infrastructure.
  - Amazon EC2 instances with prepared AMIs, Elastic Load Balancing, etc.
  - Use AWS CloudFormation to automate deployment of core networking.
- Retrieve backups from Amazon S3.
- Restore system from backup.
- Switch over to the new system.
  - Adjust DNS records to point to AWS.

Objectives

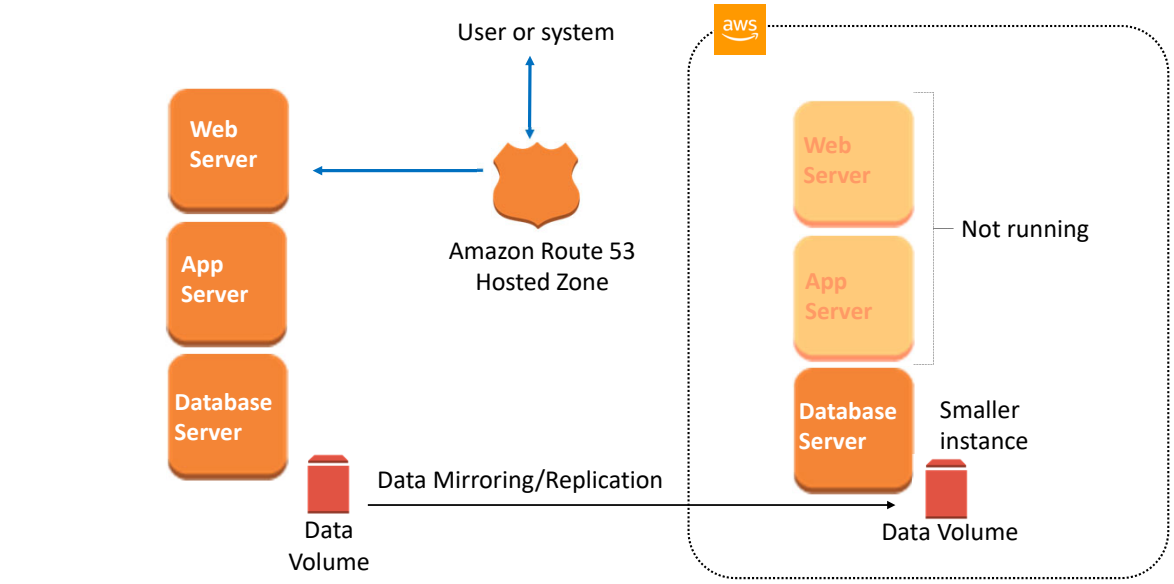
- **RTO:** as long as it takes to bring up infrastructure and restore system from backups.
- **RPO:** time since last backup.

In case of a disaster, you can:

- Retrieve backups from Amazon S3.
- Boot up the required infrastructure, including Amazon Elastic Compute Cloud—or Amazon EC2—instances with prepared AMIs, Elastic Load Balancing, and so on. Use AWS CloudFormation to automate the deployment of core networking.
- Restore your system from a backup.
- And then switch over to the new system, adjusting Domain Name System—or DNS—records to point to AWS.

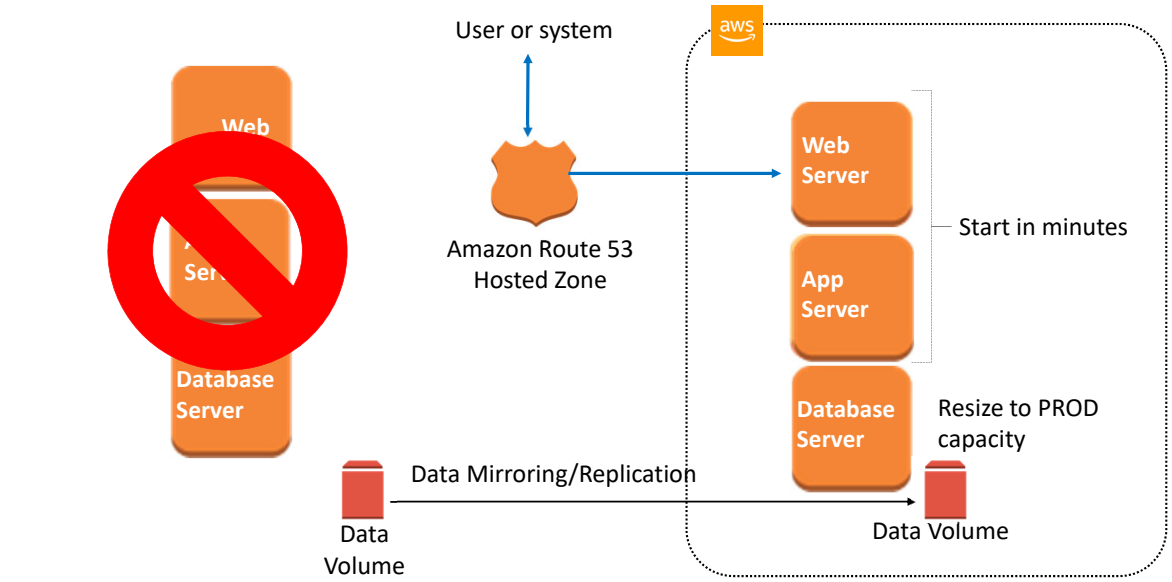
For the backup and restore option, the recovery time objective—or RTO—is as long as it takes to launch the infrastructure and restore your system from backups.
The recovery point objective—or RPO—is the time since your last backup.

Pilot Light



For this example of a “pilot light” architecture, you have an on-premises system, and Amazon Route 53 points to your on-premises system. You have a smaller database instance in the AWS Cloud where you mirror data or replicate data from your on-premises system. You have two servers that represent your web tier and your application tier, but they are not on and running yet.

Pilot Light



If the primary system fails, you grow your database server to the size that it needs to be to take on production data traffic. Next, you turn on your web server and application server. You then point Amazon Route 53 to your new servers. This entire process can happen in minutes.

Pilot Light

Advantage

- Very cost-effective (Uses fewer 24/7 resources).

Preparation Phase

- Take backups of current systems
- Store backups in S3
- Describe procedure to restore from backup on AWS
  - Know which AMI to use; build your own as needed
  - Know how to restore system from backups
  - Know how to switch to new system
  - Know how to configure the deployment

An advantage of the pilot light architecture is that it's very cost-effective. It uses fewer resources than an active-active situation.

For the preparation phase, consider the time that it takes to handle backups of your current systems. Store backups in Amazon S3, and describe the procedure you use to restore your system from backups on AWS.

- Know which AMI to use, and build your own as needed.
- Know how to restore your system from backups.
- Know how to switch to a new system.
- And know how to configure the deployment.

Pilot Light

In case of disaster

- Automatically bring up resources around the replicated core data set.
- Scale the system as needed to handle current production traffic.
- Switch over to the new system.
  - Adjust DNS records to point to AWS.

Objectives

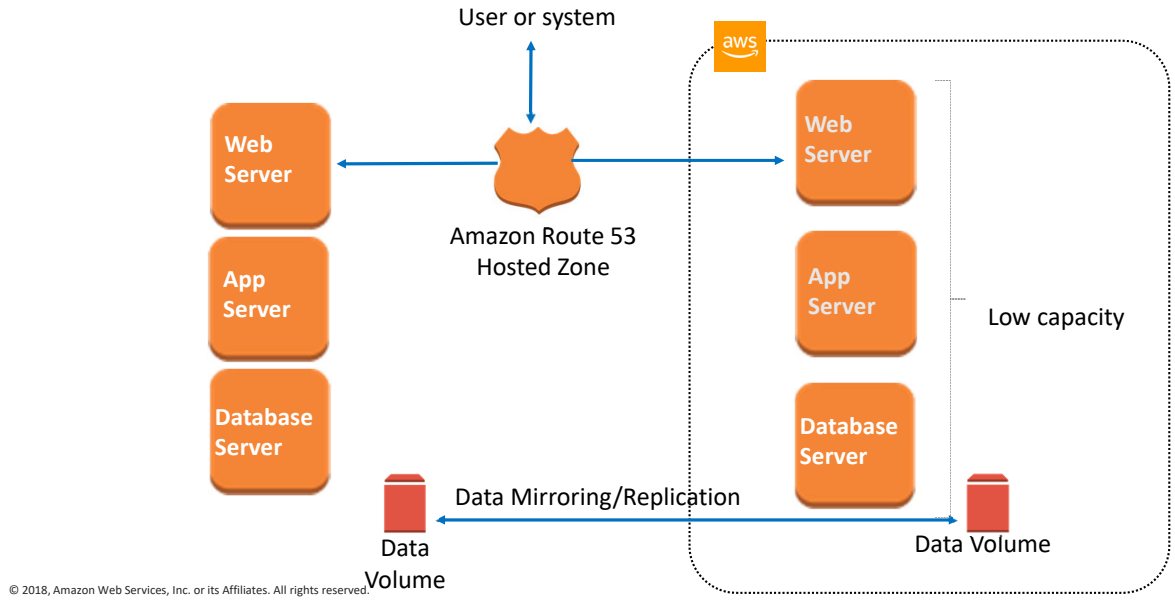
- **RTO:** as long as it takes to detect need for DR and automatically scale up replacement system.
- **RPO:** depends on replication type.

In case of a disaster, you can:

- Automatically bring up your resources around the replicated core data set.
- Scale the system as needed to handle current production traffic.
- Finally, switch over to the new system, and adjust the DNS records to point to AWS.

The RTO is as long as it takes for you to detect that you need disaster recovery, and to automatically scale up a replacement system.
The RPO will depend on how you replicate your data between your on-premises system and the AWS Cloud.

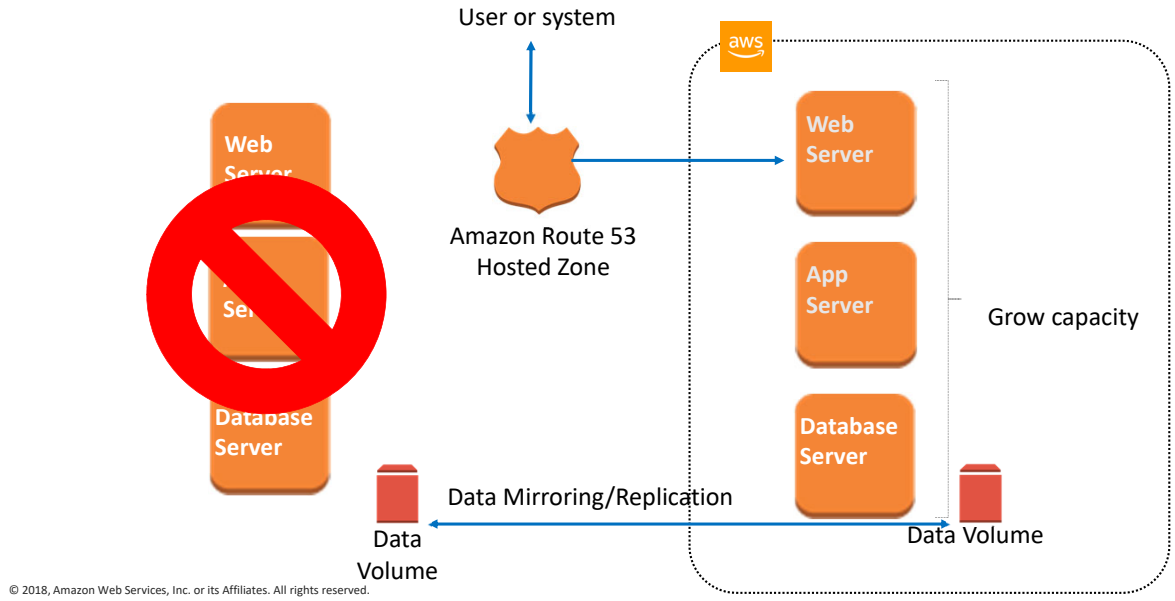
Fully Working Low-Capacity Standby



Now, let's discuss a fully working low-capacity standby architecture. A low-capacity standby architecture is like the next level of the pilot light architecture.

In this scenario, you have two running systems: the main system and a low-capacity system that runs on AWS. You can use Amazon Route 53 to distribute requests between the main system and the cloud system.

Fully Working Low-Capacity Standby



If an actual catastrophic event takes over your on-premises infrastructure, you only need to change your DNS records to point to your AWS Cloud site.
Then, you can grow capacity by using horizontal scaling or vertical scaling of your instances. In this case, Amazon Route 53 switches over to the secondary system, which is designed to automatically scale up its capacity if there is a failover from the primary system.

Fully Working Low-Capacity Standby

aws academy

Advantages

- Can take some production traffic at any time
- Cost savings (IT footprint smaller than full disaster recovery)

Preparation

- Similar to Pilot Light.
- All necessary components running 24/7, but not scaled for production traffic.
- Best practice: continuous testing.
  - "Trickle" a statistical subset of production traffic to DR site.

Some of the advantages of a fully working low-capacity standby architecture include that it can take some production traffic at any time, and there is a cost savings because the IT footprint is smaller than that of a full disaster recovery.

The preparation is similar to the pilot light scenario.
All necessary components and instances are running 24/7, but it's not scaled for production traffic.
A best practice for the fully working low-capacity standby is continuous testing. You can use the "trickle" test method, and have a statistical subset of your production traffic go to your disaster recovery site. This method can alert you to any issues in your infrastructure before you need to rely on your disaster recovery site to handle all of your production traffic.

Fully Working Low-Capacity Standby

aws academy

In case of disaster

- Immediately fail over most critical production load.
  - Adjust DNS records to point to AWS.
- (Auto) Scale the system further to handle all production load.

Objectives

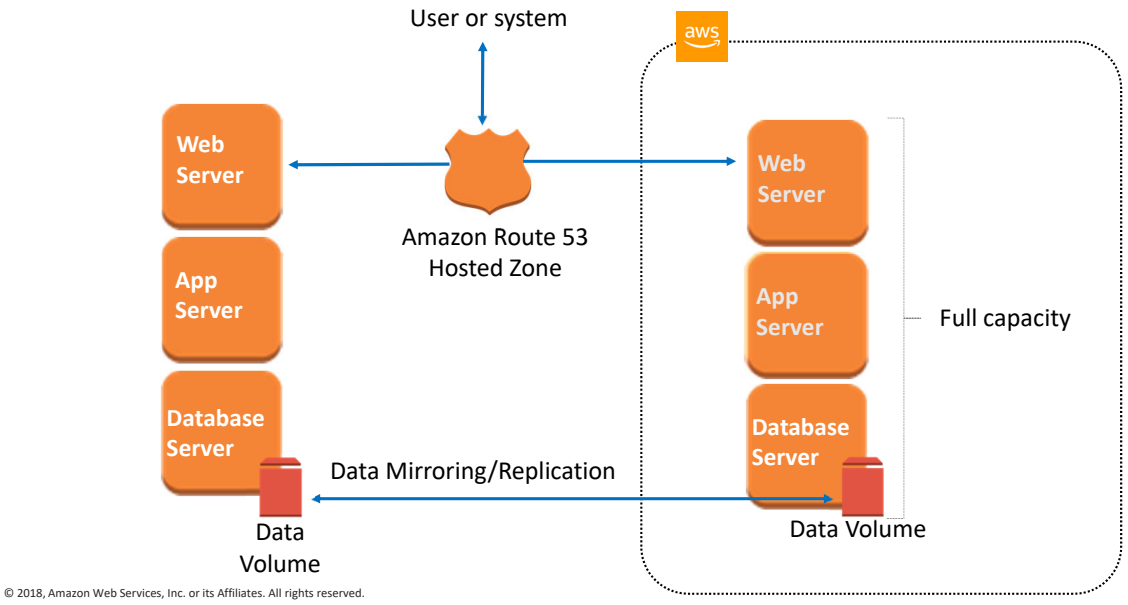
- **RTO:** for critical load: as long as it takes to fail over; for all other load, as long as it takes to scale further.
- **RPO:** depends on replication type.

In case of a disaster, you can:

- Immediately fail over the most critical production load.
- Adjust your DNS records to point to AWS.
- And Auto Scale the system further to handle all production load traffic.

Your RTO can take the critical load immediately for as long as it takes to fail over. For all other loads, it will be as long as it takes to scale further.
The RPO depends on your replication type.
To clarify, does this mean use the Amazon EC2 Auto Scaling feature, or to automatically scale?
Does this sentence express the idea that the RTO is the time it takes for the critical load to fail over?
The use of “it” here is ambiguous. Does it refer to RTO?

Multi-Site Active-Active



For a multi-site active-active architecture, you have a full-size, production-capable infrastructure within the AWS Cloud. In this case, all of the servers are already operating at full capacity and can take the load at any time. You also perform data mirroring and replication between two sites.

Multi-Site Active-Active

aws academy

Advantage

- Can take all production load at any moment.

Preparation

- Similar to Low-Capacity Standby.
- Fully scaling in/out with production load.

In Case of Disaster

- Immediately fail over all production load.

Objectives

- **RTO:** as long as it takes to fail over.
- **RPO:** depends on replication type.

The advantage of a multi-site active-active architecture is that it can take on all of the production load at any time.

The preparation for this architecture is similar to that of a low-capacity standby, except you that don't need to scale in or out with the production load because the environment is already at full capacity.

In the case of a disaster, you can immediately fail over your entire production load. To fail over, point your DNS record to ensure that all traffic is sent to your disaster recovery site within the AWS Cloud.

The RTO is as long as it takes to fail over, and the RPO depends on your replication type.

Best Practices for Being Prepared

Start simple and work your way up

- Backups in AWS as a first step.
- Incrementally improve RTO or RPO as a continuous effort.

Check for any software licensing issues.

Exercise your DR Solution

- Practice "Game Day" exercises.
- Ensure backups, snapshots, AMIs, etc. are working.
- Monitor your infrastructure.

One of the best practices for being prepared is to start simple and work your way up.
As a first step, make sure your backups in AWS work.
Also, incrementally improve the RTO and the RPO as a continuous effort.

Another best practice is to always check for any software licensing issues.

Finally, you should exercise your disaster recovery solution.

- Practice "game day" exercises.
- Ensure that your backups, snapshots, AMIs, and other recovery resources work, and make sure that everyone knows your disaster recovery plan thoroughly.
- Make sure that you monitor your infrastructure so that if anything happens to your production environment, you can quickly implement your disaster recovery plan.

This week

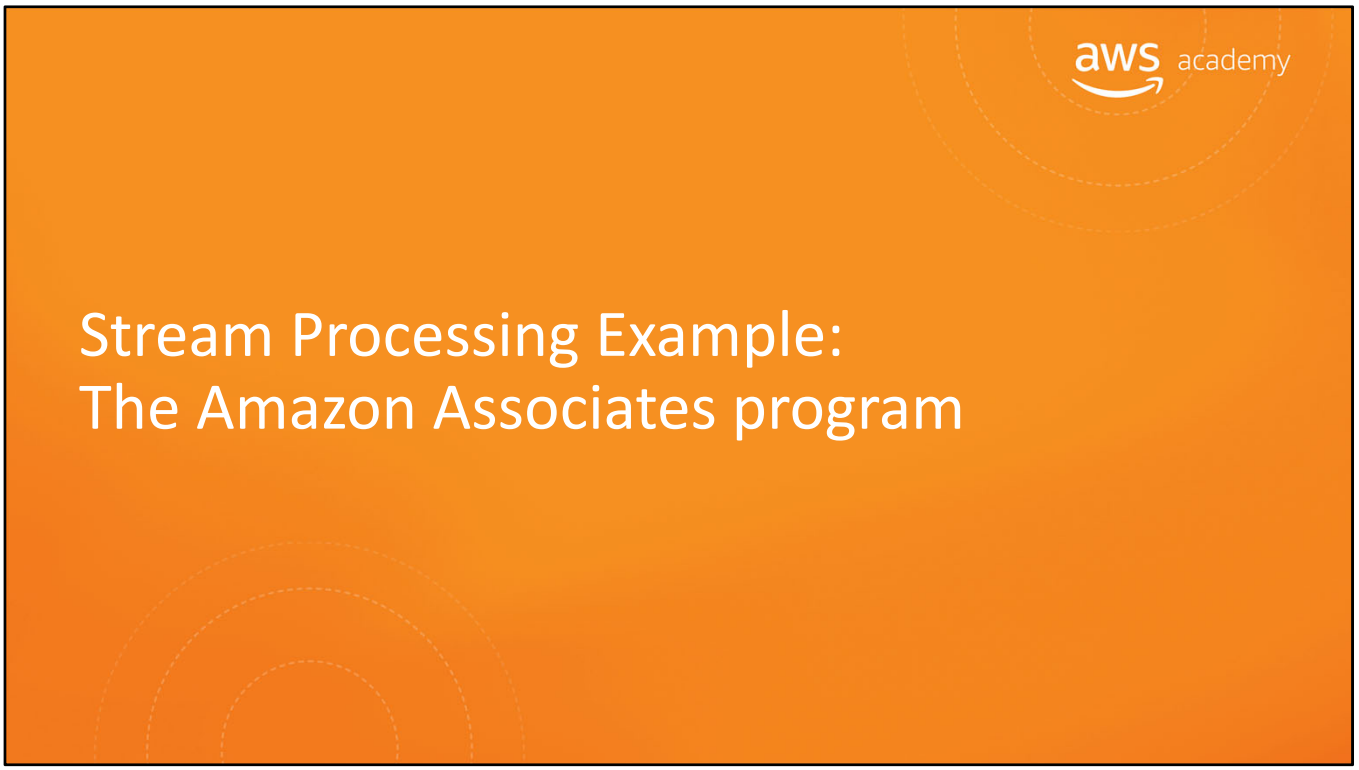
- Routing across Regions – Route53
- High availability and reliability
  - High availability patterns
  - Disaster recovery patterns
- **Sample Architectures**
  - **Stream processing**
  - Sensor network data ingestion
  - Mobile gaming
  - Transcoding and serving video files

These might give you some ideas for Assignment 3

For detailed descriptions
aws.training
ACA Module 15
Section 2

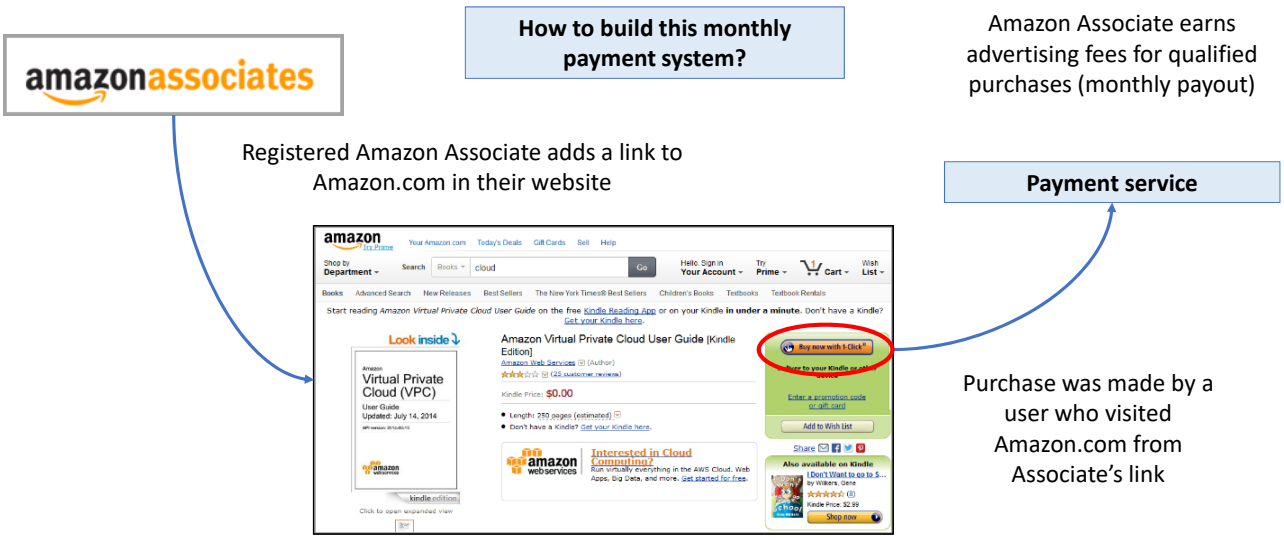
SWINBURNE UNIVERSITY OF TECHNOLOGY

46



Next, we'll discuss a stream processing example for the Amazon Associates program.

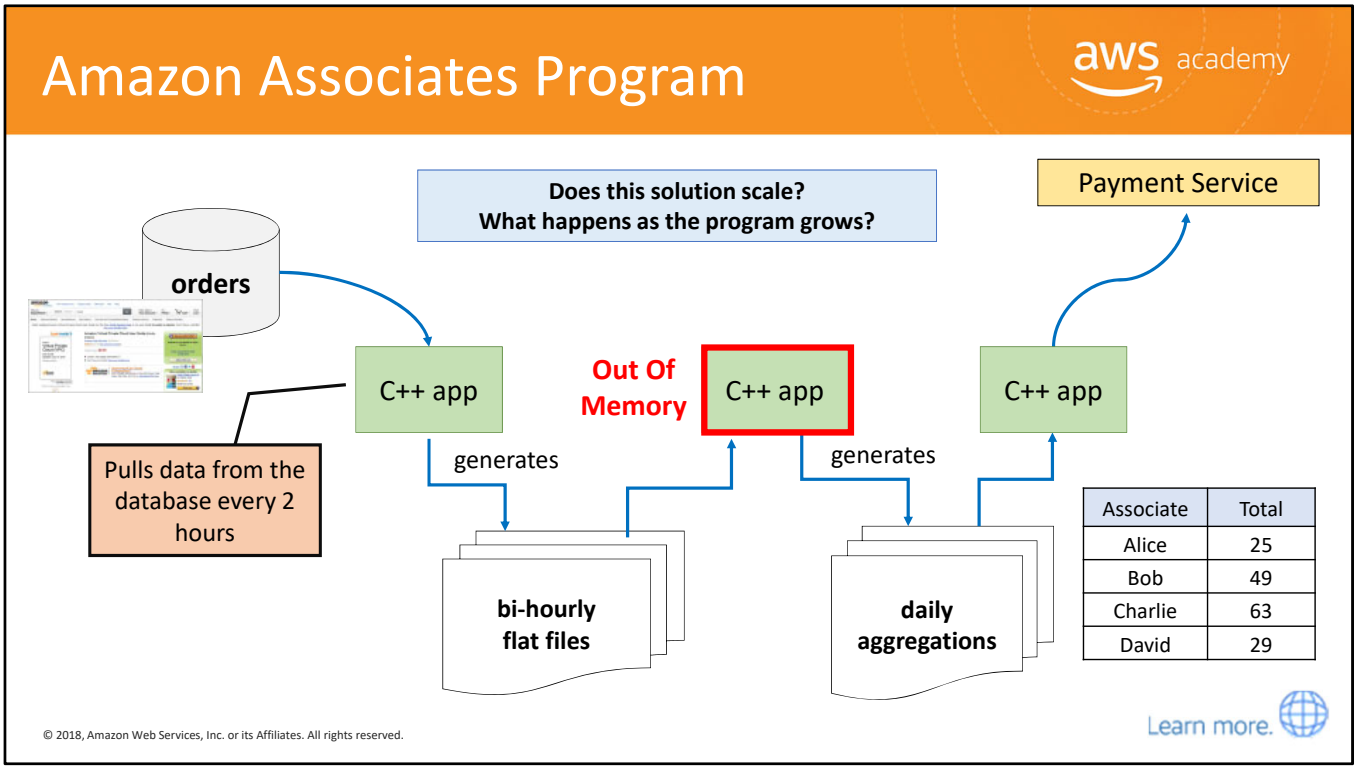
Gen 1: Amazon Associates Program



In this example, a registered Amazon Associate adds a link to Amazon.com in their website.

A user visits Amazon.com by using the Associate's link and makes a purchase.

The Amazon Associate earns advertising fees for qualified purchases as a monthly payout.



With the Amazon Associates program, you add a link to Amazon.com on your website, and if people use the link and buy something, Amazon will give you a share of that sale, up to 15%.

How much to pay each associate every month is an extremely important problem. It is critical that Amazon is correct, but still it seems like a fairly straightforward problem. Amazon tracks every sale that is attributed to an associate link, adds up the sales at the end of the month, and pays the associate at the appropriate rate. The reality is a little more complicated, though, because Amazon must track an order throughout its lifecycle to know whether the order was cancelled or returned.

This slide illustrates the system that Amazon built over 12 years ago. A single-threaded C++ application connected directly to the production orders database every two hours, and exported the orders that it was interested in into a series of flat files. Next, another C++ application ran once a day to aggregate the bi-hourly files into a daily view.

Finally, another system ran once a month to scan over the daily aggregates, calculate how much to pay people, and give that information to the payments service, which was run by a totally separate team.

This system ran very smoothly until about 2008, when the scale of the associates program grew to the point where the daily aggregations system started to run out of memory. The program needed a way to process large volumes of data quickly.

Select the link to learn more about the Amazon Associate's Program.

https://affiliate-program.amazon.com/gp/associates/join/landing/main.html

Amazon EMR

aws academy

Amazon EMR is managed platform that simplifies running big data frameworks, such as Apache Hadoop and Apache Spark, **on AWS Cloud**.

What is Hadoop?

- An open source project written in Java, managed by the Apache Software Foundation.
- Built for:
  - Handling uncertainty
  - Handling high data variety, volume, and velocity.
  - Providing solution flexibility: Write your own data applications.
  - Processing large batches of data (such as logs).

To solve the problem Amazon Elastic MapReduce, or EMR, was created. Amazon EMR is a managed cluster platform that simplifies running big data frameworks, such as Apache Hadoop and Apache Spark, on the AWS Cloud so that users can process and analyze large amounts of data.

Hadoop is an open source project written in Java, and is managed by the Apache Software Foundation.
It uses a distributed processing architecture called Amazon EMR in which a task is mapped to a cluster of commodity servers for processing. Each piece of work that is distributed to the cluster servers can be run or re-run on any of the servers.

The cluster servers use Hadoop Distributed File System—or HDFS—to store data locally for processing. The results of the computation that is performed by those servers are then reduced to a single output set. One node, which is designated as the master node, controls the distribution of tasks. The master node can automatically handle server failures.

Hadoop provides a number of benefits:

First, Hadoop handles uncertainty by facilitating data navigation, discovery, and ad-hoc data analysis. Hadoop allows you to compensate for unexpected

occurrences by allowing you to analyze large amounts of data quickly in order to form a response.

Hadoop also handles high data variety, volume, and velocity. Unlike traditional database systems, Hadoop can process structured, unstructured, or semi-structured data. This includes virtually any data format that is currently available, such as Extensible Markup Language, or XML; comma-separated values, or CSV, files; text; log files; objects; SQL; JavaScript Object Notation, or JSON; and binary formats. In addition to natively handling many types of data, Hadoop can transform data into formats that allows for better integration into your existing data sets. Also, you can store data with or without a schema, and you can perform large-scale extract, transform, load—or ETL—processes to transform your data.

Hadoop provides flexibility so you can write your own data applications. Because Hadoop is open source, there are a number of ecosystem projects that can help you analyze the multiple types of data that Hadoop can process and analyze. These projects allow you tremendous flexibility when you develop big data solutions. Hadoop's programming frameworks—including Hive, Pig, and so on—can support almost any big data use case for your applications.

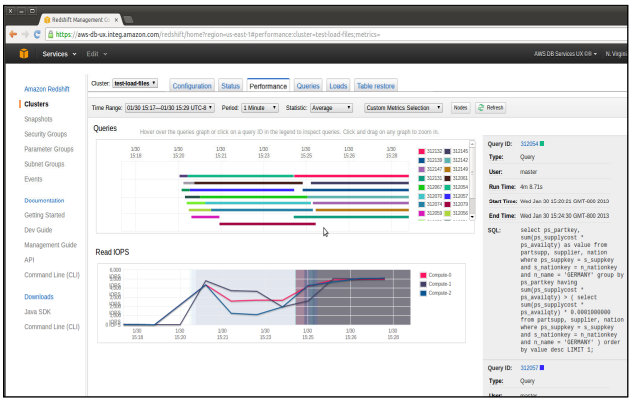
Hadoop can process large batches of data, such as logs. Because of Hadoop's distributed architecture, Hadoop clusters can affordably handle tremendous amounts of data. Adding additional data processing capability is as straightforward as adding additional servers to your cluster for linear scaling.

Amazon Redshift

aws academy

Amazon Redshift is a fully managed, petabyte-scale **data warehouse** service.

- Provision in minutes.
- Monitor query performance.
- Point and click resize.
- Built-in security.
- Automatic backups.

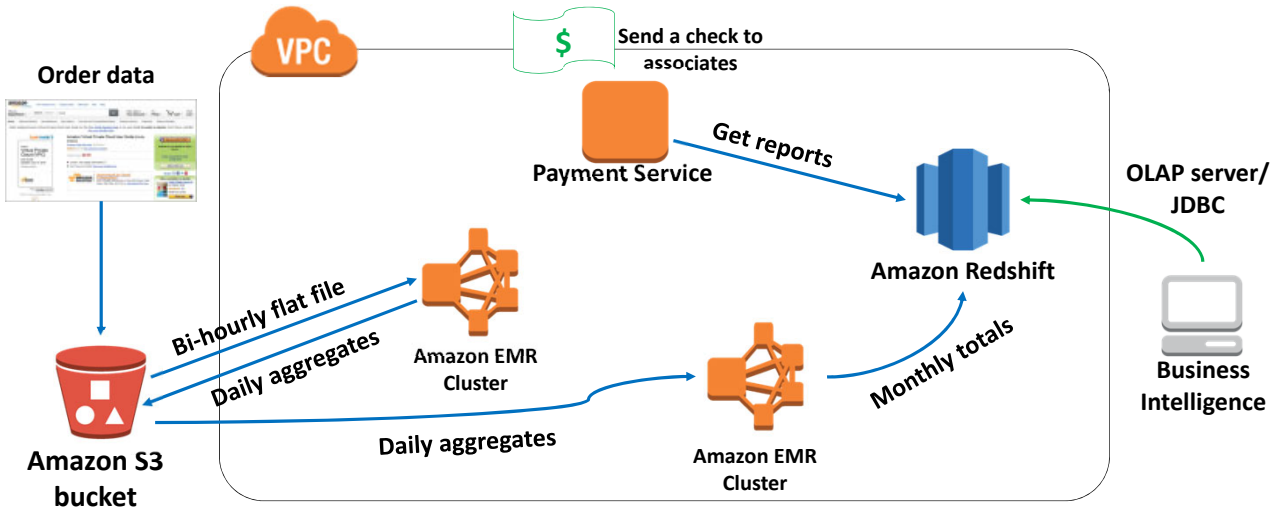


You can build business intelligence with Amazon Redshift. Amazon Redshift is a fully managed, petabyte-scale data warehouse service.

With Amazon Redshift, you can:

- Provision a resource in minutes
- Monitor the performance of your query
- Resize via point and click
- Use built-in security
- And configure automatic backups

Gen 2: EMR for Amazon Associates



To implement Amazon EMR as a solution for the Amazon Associates program's architecture, the team developed a system that publishes the raw orders stream to a set of registered listeners. The Amazon Associates team wrote an application that listens to the orders stream, filters it, retrieves any additional data it might need about the order from other services, and places it into Amazon Simple Storage Service, or Amazon S3. Then, a Hadoop cluster starts on a daily basis, and it pulls 60 days of order history out of Amazon S3. It adds the month-to-date sales of every associate, computes pay rates, and stores all of that data back on to Amazon S3. After the architecture adopted a distributed solution, larger and larger data sizes no longer posed a problem. As the volume of data grew, the Amazon Associates team did not have to struggle to figure out how to fit the data in memory on a single machine. All the team needed to do was to add machines to their cluster. The code that runs in the cluster doesn't change at all.

To solve all these issues, Amazon moved to Amazon EMR, which can be considered to be like running Hadoop in the cloud.

There are significant advantages to Amazon EMR:
- Cluster setup and resizing takes minutes.
- You can create application-specific cluster configurations.
- A container-managed service makes software installation and upgrades trivial.
- And the cost is usually much lower.

The total cost of an on-premises deployment in comparison to Amazon EMR includes:
- The time and expense spent in sizing and ordering hardware.
- The development effort.

- And the cost of maintaining the system and administrative personnel.

What's Next?

aws academy

Generation 2 solution achieved the following:

- Replaced the file system with Amazon S3.
- Used Amazon EMR to remove the performance bottleneck that was caused by the single-threaded C++ application.
- Added analytics and data insight with Amazon Redshift.

How can we further improve this system as the load increases?

- **Real-time** data processing.

So, what's next?
The second generation of the Amazon Associates program achieved the following changes:

- The file system was replaced with Amazon S3.
- Amazon EMR was used to remove the performance bottleneck that the single-threaded C++ application caused.
- And features for analytics and data insight with Amazon Redshift were added.

How can this system be further improved while the load increases? One improvement is adding real-time data processing.

Batch Versus Streaming

aws academy

Batch processing (“cold” data)

Hourly server logs
- System issues from an hour ago

Weekly, monthly bill
- Spending in past billing cycle

Daily customer preference report from clickstream logs

Daily fraud reports
- Was there fraud yesterday?

Stream processing (“hot” data)

Compiling Amazon CloudWatch metrics
- Monitor issues immediately

Real-time spending alerts, caps
- Respond to sudden cost increases

Real-time analysis
- What to offer the customer now

Real-time detection
- Block fraudulent use now

Batch processing typically involves querying large amounts of “cold” data. In batch processing, you can retrieve hourly server logs that display system issues from several hours ago. You receive a weekly or monthly bill that shows your spending across the past billing cycle. A daily customer preference report is available from clickstream logs, and daily fraud reports to tell you whether you experienced any fraud yesterday.

Stream processing is in real time, and it typically involves querying small amounts of “hot” data. It takes only a short amount of time to get answers.
Stream processing compiles Amazon CloudWatch metrics immediately, and it also monitors issues. You have access to real-time spending alerts and caps. You can also respond to sudden cost increases. You receive real-time analysis, and you can make customer offers immediately. Real-time fraud detection also allows you to immediately block fraudulent use.

Amazon Kinesis

aws academy

Collect, process, and analyze video and data streams in real time.

Amazon Kinesis Video Streams

Capture, process, and store video streams for analytics and machine learning.

Amazon Kinesis Data Streams

Build custom applications that analyze data streams using popular stream processing frameworks.

Amazon Kinesis Data Firehose

Load data streams into AWS data stores.

Amazon Kinesis Data Analytics

Analyze data streams with SQL.

With **Amazon Kinesis**, you can collect, process, and analyze video and data streams in real-time. You can get timely insights and react quickly to new information. Amazon Kinesis offers key capabilities to cost-effectively process streaming data at any scale, and it offers you the flexibility to choose the tools that best suit the requirements of your application.

Amazon Kinesis Video Streams is a fully managed AWS service that you can use to capture, process, and store video streams for analytics and machine learning.

You can use **Amazon Kinesis Data Streams** to collect and process large streams of data records in real time. With Amazon Kinesis Data Streams, you can build custom applications that analyze data streams by using popular stream-processing frameworks.

Amazon **Kinesis Data Firehose** is a fully managed service for loading data streams into AWS data stores. Amazon Kinesis Data Firehouse delivers real-time streaming data to destinations such as Amazon S3; Amazon Redshift; Amazon Elasticsearch Service, or Amazon ES; and Splunk.

With **Amazon Kinesis Data Analytics**, you can process and analyze streaming data by using standard SQL. The service enables you to quickly author and run powerful SQL code against streaming sources to perform time series analytics,

feed real-time dashboards, and create real-time metrics.

Amazon Kinesis Data Streams

Near-real-time performance

Perform continual processing on streaming big data. Processing latencies fall to a few seconds, compared with the minutes or hours associated with batch processing.

High throughput, elastic

Seamlessly scale to match your data throughput rate and volume. Easily scale up to gigabytes per second. It can scale in or out based on operational or business needs.

Build near real-time apps

Client libraries enable developers to design and operate near real-time streaming data processing applications.

Low cost

Cost-efficient for workloads of any scale. Get started by provisioning a small stream, and pay low hourly rates only for what you use.

Let’s review the key benefits of Amazon Kinesis Data Streams.

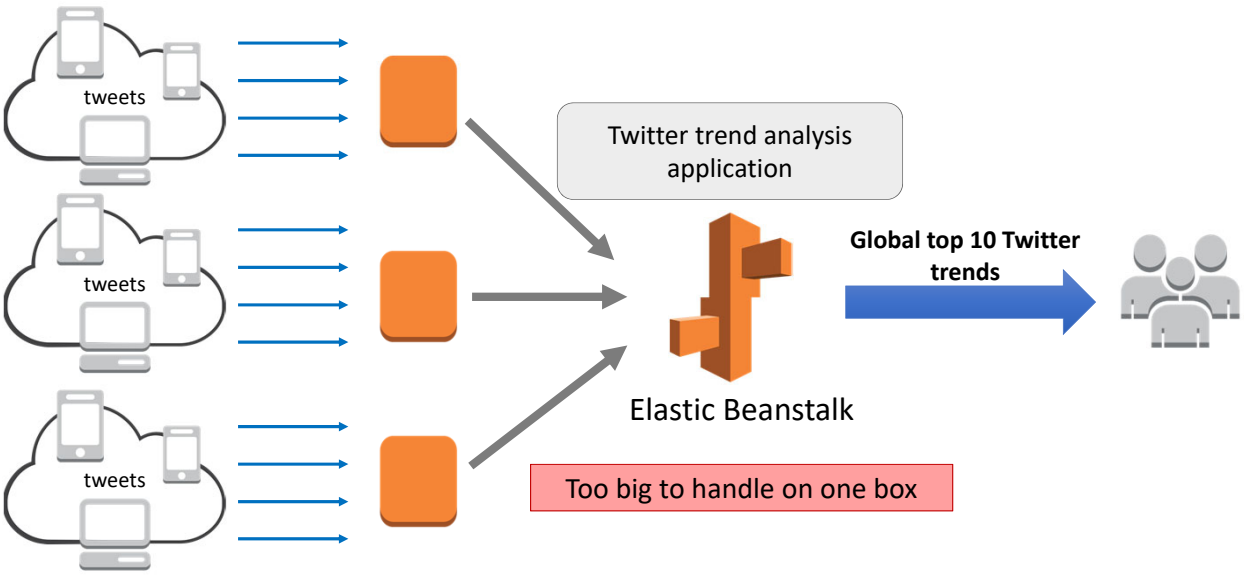
Amazon Kinesis Streams offers near-real-time performance, which allows you to perform continual processing on streaming big data. The processing latencies are reduced to a few seconds, compared with the minutes or hours that are associated with batch processing.

Client libraries enable developers to design, build, and operate applications that use near-real-time streaming data processing.

Amazon Kinesis Streams has high throughput that is also elastic. You can seamlessly scale to match the throughput rate and volume of your data. You can scale up to gigabytes per second. Amazon Kinesis Streams can scale in or out based on your operational or business needs.

Amazon Kinesis Streams is also cost-efficient for workloads of any scale. You can get started by provisioning a small stream, and pay low hourly rates only for what you use.

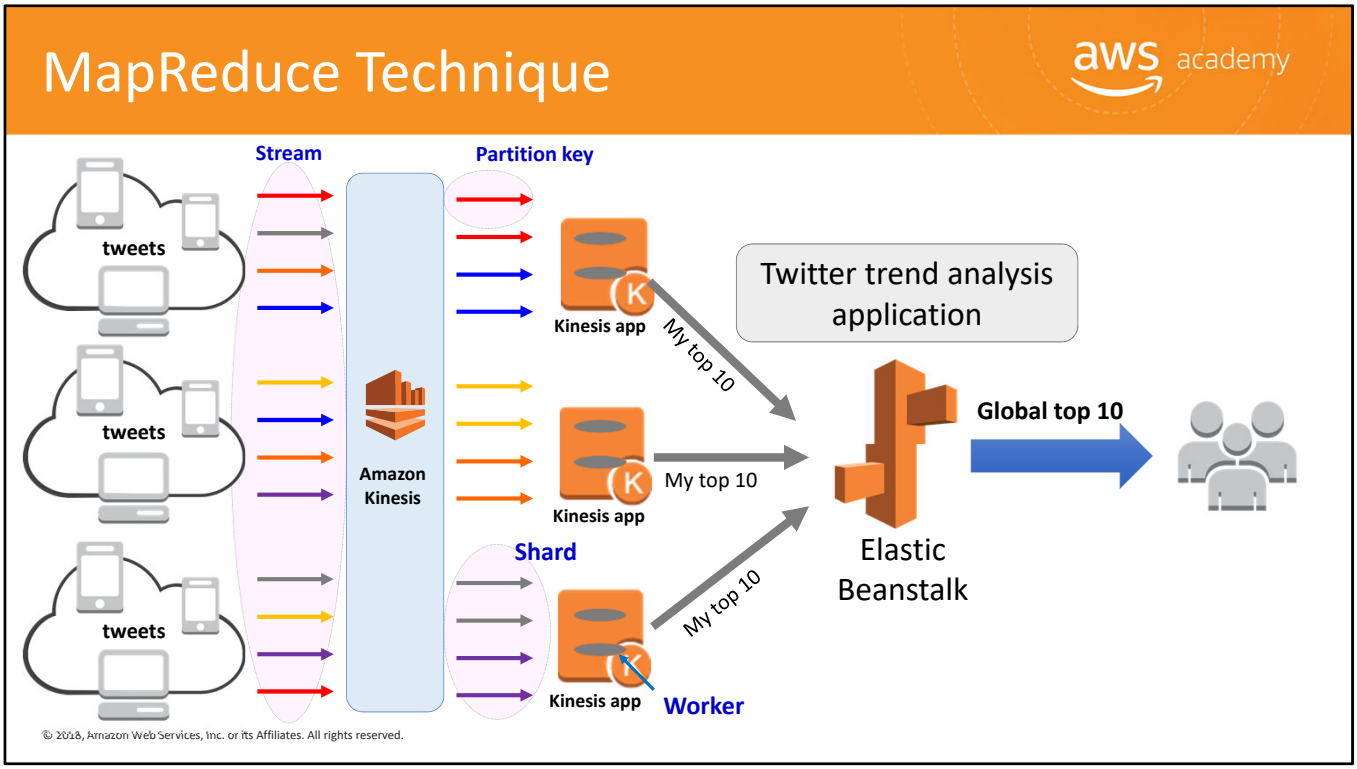
Twitter Trend Analysis Application



Let's view an example of an application for Twitter trend analysis. The planned functionality for the twitter-trends.com website is to display the top 10 current trends on Twitter. On AWS, you might begin by creating a basic AWS Elastic Beanstalk website.

Unfortunately, the Twitter firehose feed is quickly becoming too big to process on a single machine. You will need to divide up the work so that it can be done in parallel on multiple machines. The workload must then be combined into an output that can be vended by the Elastic Beanstalk web server.

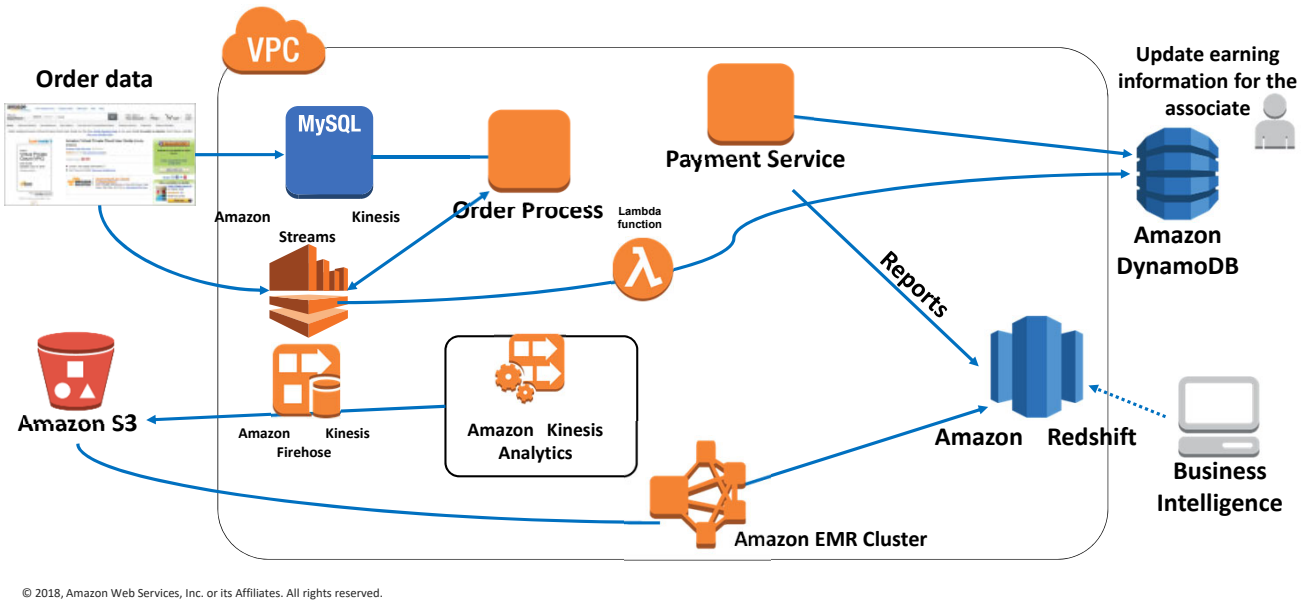
In this case, you could compute a local top-10 value for all the tweets coming into each processing instance, and then combine all top-10 lists into a global top-10 list in the web server. The problem is that, without some sorting or grouping function, each processing instance will receive a random selection of tweets. This situation makes it impossible for any one instance to know how many tweets are occurring for any specific topic.



You can solve this problem by applying a technique from the MapReduce world. If you are computing the top 10 tweeted topics for a standalone batch of tweets, you can treat the topic of each tweet as a partition key (for example, #WorldCup). Each map task would take in a fraction of all tweets, group them by topic, and send the count for each topic to the appropriate reduce task that is responsible for that topic.

You can do something equivalent by introducing an intermediate stage that takes in the stream of tweets and orders them by topic. Each processing box would then pull only the tweets for the set of topics that it is responsible for. This would enable each box to be an authoritative counter of tweets for some subset of all currently active tweet topics. This process is essentially the streaming equivalent of the MapReduce processing paradigm.

Real-Time Order Processing



In this example:

- Every order that Amazon.com receives is pushed to Amazon Kinesis.
- If the order is linked to an Associate program—that is, it's a qualified order—that order information is stored in an Amazon S3 bucket. Because the qualified order is mapped to a specific Amazon associate, the information can be stored in Amazon DynamoDB. The associate can log into a web application to see how many people came to their website and purchased a product at Amazon.com.
- The Amazon EMR cluster retrieves the qualified order information from the Amazon S3 bucket, and then the monthly total is stored in Amazon Redshift. A business intelligence application can then run some analysis on the information.
- A payment service then pulls the report, and sends payment checks to the associates every month.

This week

- Routing across Regions – Route53
- High availability and reliability
  - High availability patterns
  - Disaster recovery patterns
- **Sample Architectures**
  - Stream processing
  - **Sensor network data ingestion**
  - Mobile gaming
  - Transcoding and serving video files

SWINBURNE UNIVERSITY OF TECHNOLOGY

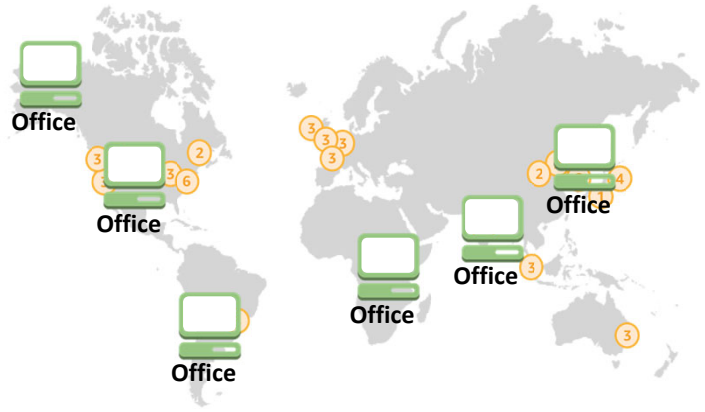
Sensor Network Data Ingestion and Processing Example: Flu Outbreak Heat Map

Now, we will discuss an example of sensor network data ingestion and processing that is focused on an outbreak heat map for influenza, or the flu.

Example: Flu Data

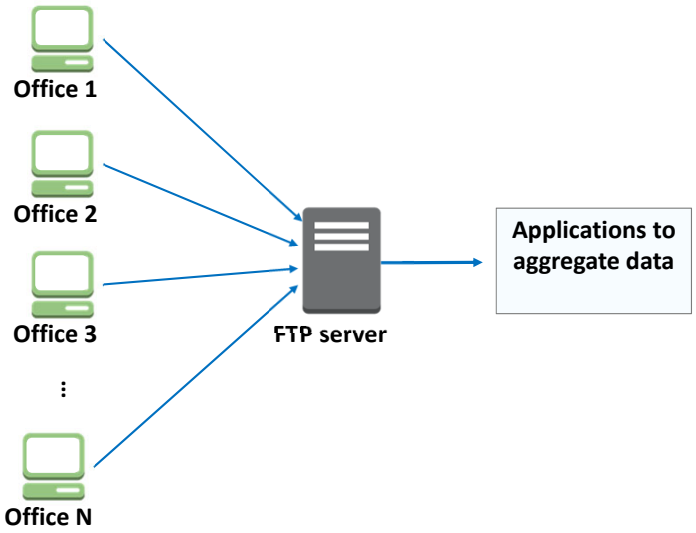
Scenario: Influenza outbreak around the world

Mission:
Based on the data you collect, generate heat maps to understand public health



In this scenario, the Government Health Organization wants to understand a flu outbreak that is happening around the world. Their mission is to understand the state of public health by collecting data, and using the data to generate heat maps.

Generation 1: Flu Heat Map



Does this scale?

Is there a concern for a single point of failure?

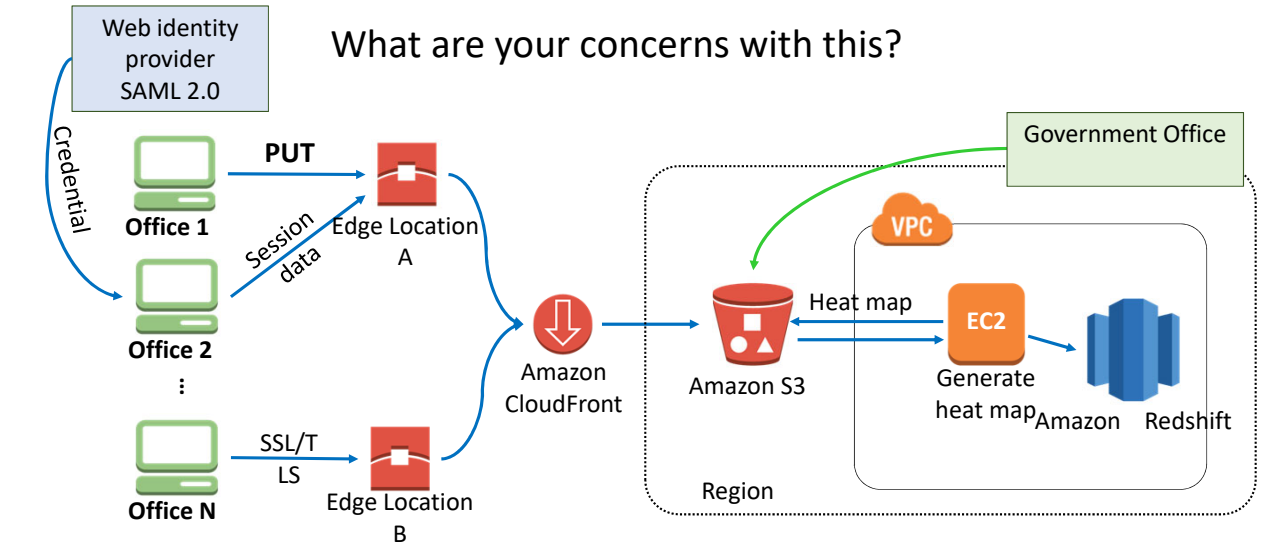
What AWS services would you use?

This scenario is based on a real-world example. It is not rare to see a government organization that uses very basic technology. In this case, all of the federal offices send the flu data to a central file transfer protocol—or FTP—server.

This solution does not scale, and the FTP server can become a single point of failure. Because of the nature of file systems, this solution does not generate a near-real-time heat map for flu cases.

Could you use Amazon Kinesis? The issue with Amazon Kinesis is that its payload size is limited. The data that comes from global offices probably exceeds the allowed payload size for Amazon Kinesis.

Generation 2: Flu Heat Map

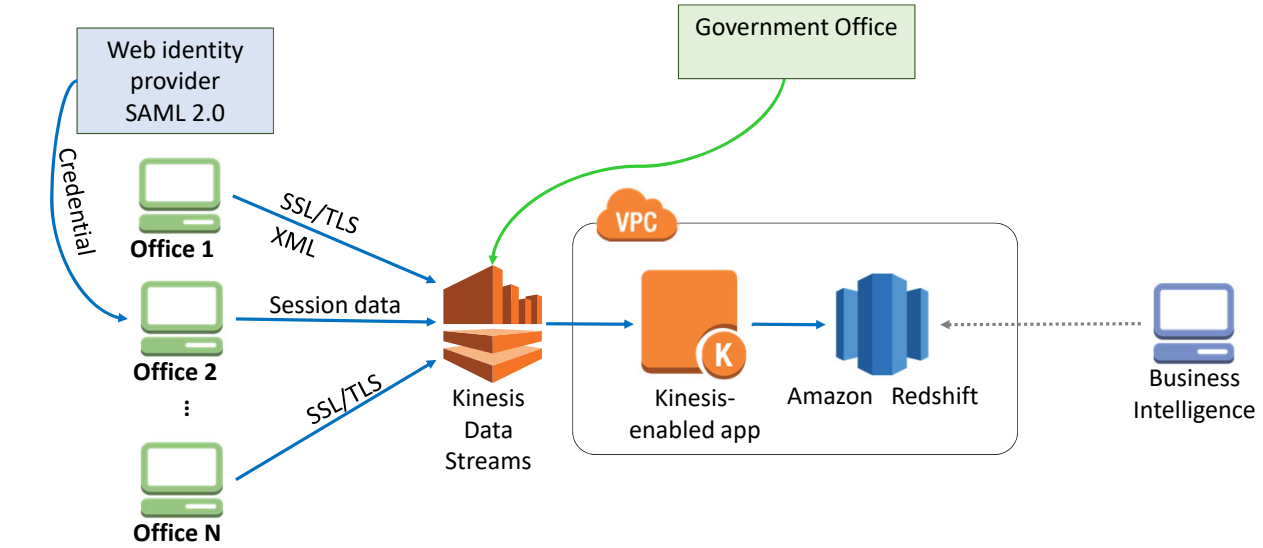


What are your concerns with this example?

One concern is that Amazon CloudFront is compatible with this use case, but it does not cache the following operations:

- POST
- PUT
- PATCH
- DELETE
- Or OPTIONS

Generation 3: Flu Heat Map



In this case, the payload on each report is not very large. Kinesis provides the speed that you want for processing the data.

This week

- Routing across Regions – Route53
- High availability and reliability
  - High availability patterns
  - Disaster recovery patterns
- **Sample Architectures**
  - Stream processing
  - Sensor network data ingestion
  - **Mobile gaming**
  - Transcoding and serving video files

SWINBURNE UNIVERSITY OF TECHNOLOGY

Application Backend Example: Mobile Gaming

Now, let’s review an example of an application backend for mobile gaming.

Mobile Games

Mobile app revenue is projected to grow continuously.

Mobile back-end technologies

- HTTP-based
- External Social APIs
- Save state
- Database
- Static data store
- Mobile push
- Analytics

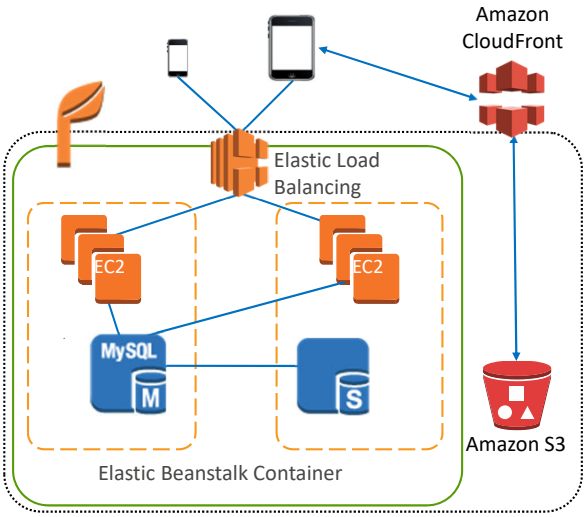
Mobile app revenue is projected to grow continuously.

Mobile backend technologies include:
- HTTP-based technologies
- External social application programming interfaces, or APIs
- Save state
- Databases
- Static data stores
- Mobile push
- And analytics

A few mobile gaming customers that use AWS include:
- Supercell, which offers Hay Day and Clash of Clans
- Halfbrick, which offers Fruit Ninja
- Wooga, which offers Diamond Dash and Pocket Village
- Ubisoft, which used AWS to launch 10 social games in 18 months
- SEGA, which migrated forums and some game servers to AWS and reported a 50% savings in server costs
- FunPlus Game, a successful Facebook game company that began without physical servers but quickly moved to AWS to gain agility and reduce costs

Mobile Games Backend Concepts

- Think in terms of APIs
- GET friends, leaderboards
- HTTP + JSON
- Multiplayer servers
- Binary assets
- Game analytics



Mobile game backends are becoming more and more like the backends of web applications. For example, gaming backends use:

- APIs to provide data and functionality
- GET requests to retrieve data about friends and leaderboards
- Hypertext Transfer Protocol—or HTTP—and JSON
- Multiplayer servers
- Binary assets
- And game analytics

Improve Your Game

aws academy

Sentiment analysis

- Enjoying game
- Engaged
- Like or dislike new content
- Stuck on a level
- Bored
- Abandonment

Players' behavior

- Hours played day or week
- Number of sessions per day
- Level progression
- Friend invites or referrals
- Response to mobile push
- Money spent per week

What information can help improve your game?

Let's review some examples of how to use data analysis to gain insights from sentiment analysis and player behavior.

To get insight into how your application is perceived, it can be helpful to collect data about the behaviors of your users. This means ingesting and processing streams of data as your application is being used, and then performing analysis on that data to gain insights from it.

For example, you might want to know if users are enjoying the game, if they are engaged, and whether they like or dislike new content. Understand why they're stuck on a level, bored and have abandoned the game.

You can analyze players' behavior to include hours played per day or week, the number of sessions per day, their level progression, and whether they invite or refer friends. You could also analyze the response to mobile push, and how much money your users spend per week.

Data Analytics for Gaming

aws academy

Batch processing examples:

- What game modes do people like best?
- How many people have downloaded?
- Where do most player characters die?
- How many daily players are there on average?

Real-time processing examples:

- What game modes are people playing now?
- Are more or fewer people downloading today?
- Are player characters dying in the same places, or different places?
- How many people are playing today? Is there any variance?

Downloadable content is additional content for a video game that is distributed through the internet by the game's official publisher or other third-party content producers. It can be of several types, ranging from aesthetic outfit changes to a new, extensive storyline that is similar to an expansion pack.

Some examples of batch processing include:

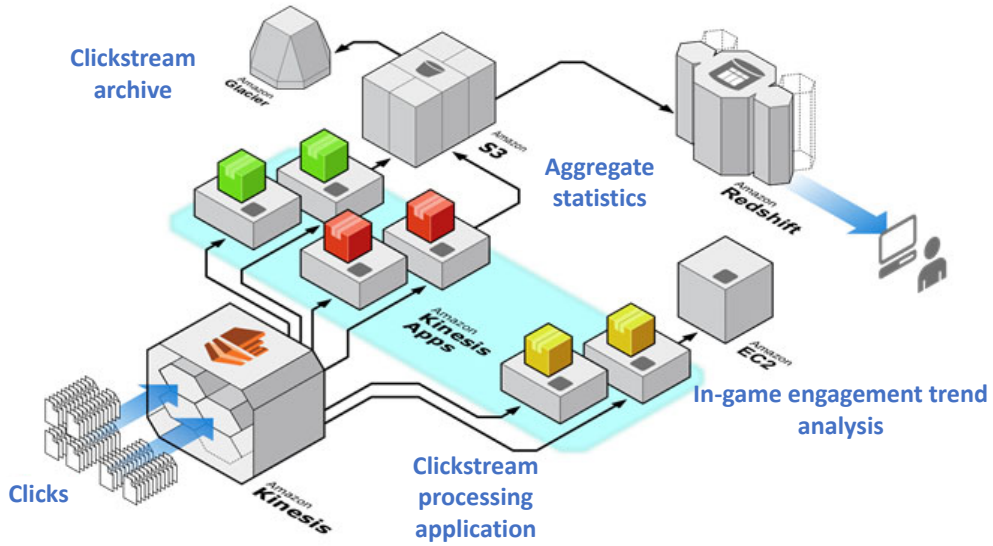
- What game modes do people like best?
- How many items people have downloaded?
- Where do most player characters die?
- How many daily players are there on average?

Examples of real-time processing include:

- What game modes are people playing now?
- Are more or fewer people downloading content today?
- Are player characters dying in the same places, or different places?
- How many people are playing today? Is there any variance in the number of players?

Both types of analysis are valuable, but when you can quickly determine what your players love, you can make your game have more of the things that your players want.

Data Analytics for Gaming



This example shows a reference architecture. Let's review some use cases for data analytics in gaming.

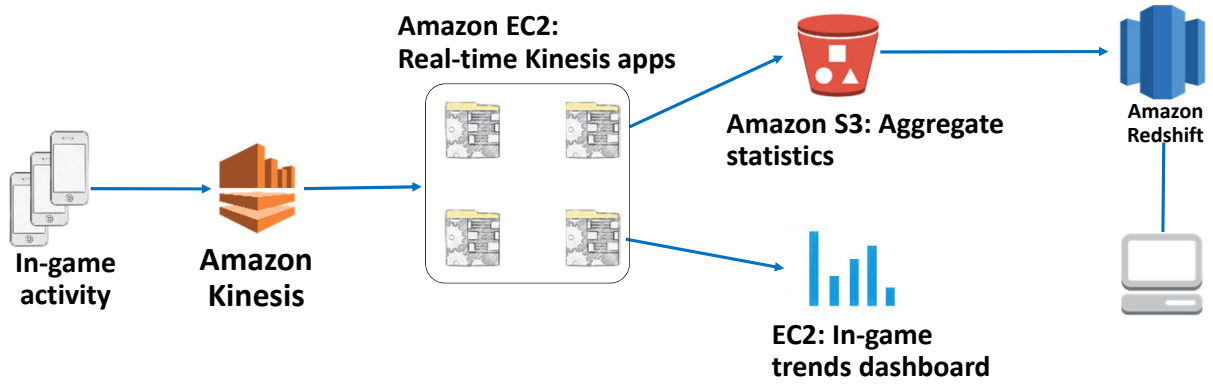
Use cases for data analytics in gaming include:
- Tracking player performance to keep games balanced and fair.
- Tracking game performance metrics by geography or region for online games that have a global player base.
- Tracking sales performance of items sold in-game.
- Tracking performance of online advertisements served in-game.
- Accelerating the delivery of the metrics that are described above to near-real-time.
- Retrieving continual in-game metrics on user engagement.
- Accessing real-time analytics on user engagement in the game.
- Receiving optimized predictions of when or where to sell in-game purchases.

The business benefits for using data analytics in gaming include:
- Reducing the operational burden on your developers.
- Scaling your data analytics to match highly variable gaming loads without overpaying for spare capacity or experiencing increased time-to-results when usage is up.
- Trying more experiments with data experiments, and finding new and game-changing metrics and analysis.
- Ingesting and aggregating gaming logs that were previously uncollected or unused.
- Accelerating the time-to-results when you batch process game logs so that you can reduce the delivery of gaming metrics from every 48 hours to every 10 minutes.
- Deliver continuous or real-time game insight data from hundreds of game servers.

Case Study: Supercell

aws academy

Analytics pipeline for **Clash of Clans**



Learn more.

Supercell is a gaming company that is based in Finland, and it was founded in 2010 by six game-industry veterans. It is one of the fastest-growing social game developers in the world. With a staff of just over 100 employees, Supercell has three games:-Hay Day, which is a social farming game; Clash of Clans, which is a social resource management game with strategic combat; and Boom Beach, which is a combat strategy game. These games attract tens of millions of players on iOS and Android devices every day.

As an example, let's discuss Clash of Clans. Data is loaded into Amazon Redshift for business intelligence tasks, and the data is also moved into Amazon S3 for long-term archiving. The team can restore the vaults from Amazon Glacier if they need to perform tasks such as recovering data, loading retrospective information into a new data warehouse instance, or augmenting their reports with data that was not originally available.

SuperCell uses Amazon Kinesis with Amazon EC2, Amazon S3, Amazon Glacier, and Amazon Redshift to gain a complete view of how their customers are playing their games. These AWS services also provide resources for analytics. The team can dive into player data to answer specific questions, such as who are the high scorers, which areas of the game world are most popular, or how their most recent updates are performing.

Sami Yliharju, who is the Services Lead at Supercell, states: "We are using Amazon Kinesis for real-time delivery of game insight data sent by hundreds of our game engine servers. Amazon Kinesis also offloads a lot of developer burden in building a real-time, streaming data

ingestion platform, and enables Supercell to focus on delivering games that delight players worldwide."

You can learn more about Supercell by selecting the link.
http://aws.amazon.com/solutions/case-studies/supercell/

This week

- Routing across Regions – Route53
- High availability and reliability
  - High availability patterns
  - Disaster recovery patterns
- **Sample Architectures**
  - Stream processing
  - Sensor network data ingestion
  - Mobile gaming
  - **Transcoding and serving video files**

SWINBURNE UNIVERSITY OF TECHNOLOGY

Example: Transcoding and Serving Video Files

Finally, let’s discuss an example for transcoding and serving video files.

Transcoding Video Files

aws academy

Challenging to set up:

- Difficult to choose the right transcoding settings.
- Complicated to build a system that performs well.

Difficult to adapt:

- Amount of content can scale up quickly.
- Output formats and device types change frequently.

Complex cost issues:

- Building for peak is uneconomical.
- Storage and delivery costs on a large scale can be prohibitive.

Transcoding video files can be a challenging process to set up. It's difficult to choose the right transcoding settings, and it is complicated to build a system that performs well.

The transcoding process can be difficult to adapt, so the amount of content can scale up quickly as output formats and device types change frequently.

Complex cost issues include building for the peak, which is not economical. In addition, storage and delivery costs on a large scale can be prohibitive.

It can be difficult to adapt, so the amount of content can scale up quickly as output formats and device types change frequently.

Complex cost issues including building for the peak, which is uneconomical and storage and delivery costs on a large scale can be prohibitive.

Amazon Elastic Transcoder

Amazon Elastic Transcoder is a highly scalable and cost-effective way for developers and businesses to convert—(or "transcode"—video files from their source format into versions that will play on devices like smartphones, tablets, and personal computers. You can use Elastic Transcoder to convert files from different media formats into H.264, VP9, VP8, AAC, MP4, and MPEG-DASH files at different resolutions, bitrates, and frame rates. You can also set up transcoding pipelines to transcode files in parallel.

Any customer with media assets that are stored in Amazon S3 can use Elastic Transcoder. Examples include developers who create applications or websites that publish user-generated content, enterprises and educational establishments that convert training and communication videos, and content owners and broadcasters that need to convert media assets into web-friendly formats.