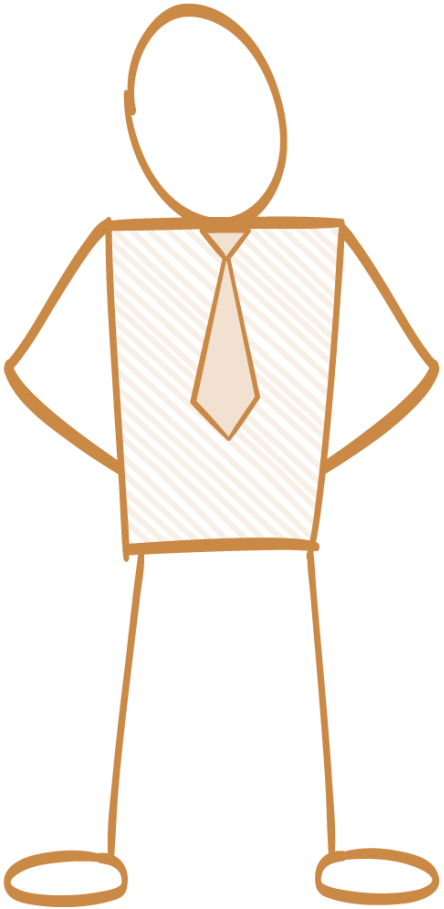
Architecting on AWS
Student Guide
Version 3.1

100-ARC-31-EN-SG



Module A: Reference Architecture

DocStore

by aMedia Corp.

DocStore

aMedia DocStore allows customers to upload and manage their documents

DocStore

aMedia DocStore allows customers to upload and manage their documents

- **Upload and access content from anywhere** – browser and mobile apps
- **PDF**, **Office docs**, **images**, **videos** – text content extracted, indexed, and searchable

DocStore

Two Account Types

- Free
  - 5GB storage
  - Search on name, created/modified, and tags
    - No full-text indexing
  - All content may be securely downloaded by owner

DocStore

Two Account Types

- Premium
  - 20GB storage
  - Full-text indexing and search for all documents (OCR on images)
  - All content may be securely downloaded by owner

DocStore

Two Major Components

DocStore

#1: Web Application

- Allow users to upload, manage, and view all documents
- All content uploaded directly to S3

DocStore

#1: Web Application

- Both powered by public DocStore API
- Both must scale
- Both must be highly available

DocStore

#1: Web Application

- Multi-region deployment for global support
- User can log in to any region
  - Always redirected to “home” region (i.e., region where account was created)

DocStore Requirements

Account Types

- **Free** – 5GB cap, basic document indexing, download original files
- **Premium** – 20GB cap, full-text indexing/search + OCR

Web App/API

- **User authentication**
- **Session state stored off-instance**
- **Powered by API tier**
- **Highly Available**

Content

- **Stored in S3**
- **Efficient upload/download to/from S3**
- **Static assets in one S3 bucket; distributed globally**

Deployment

- **Oregon (us-west-2) and Sydney (ap-southeast-2)**
- **Replicate minimal information (i.e., don't replicate uploaded content)**

DocStore

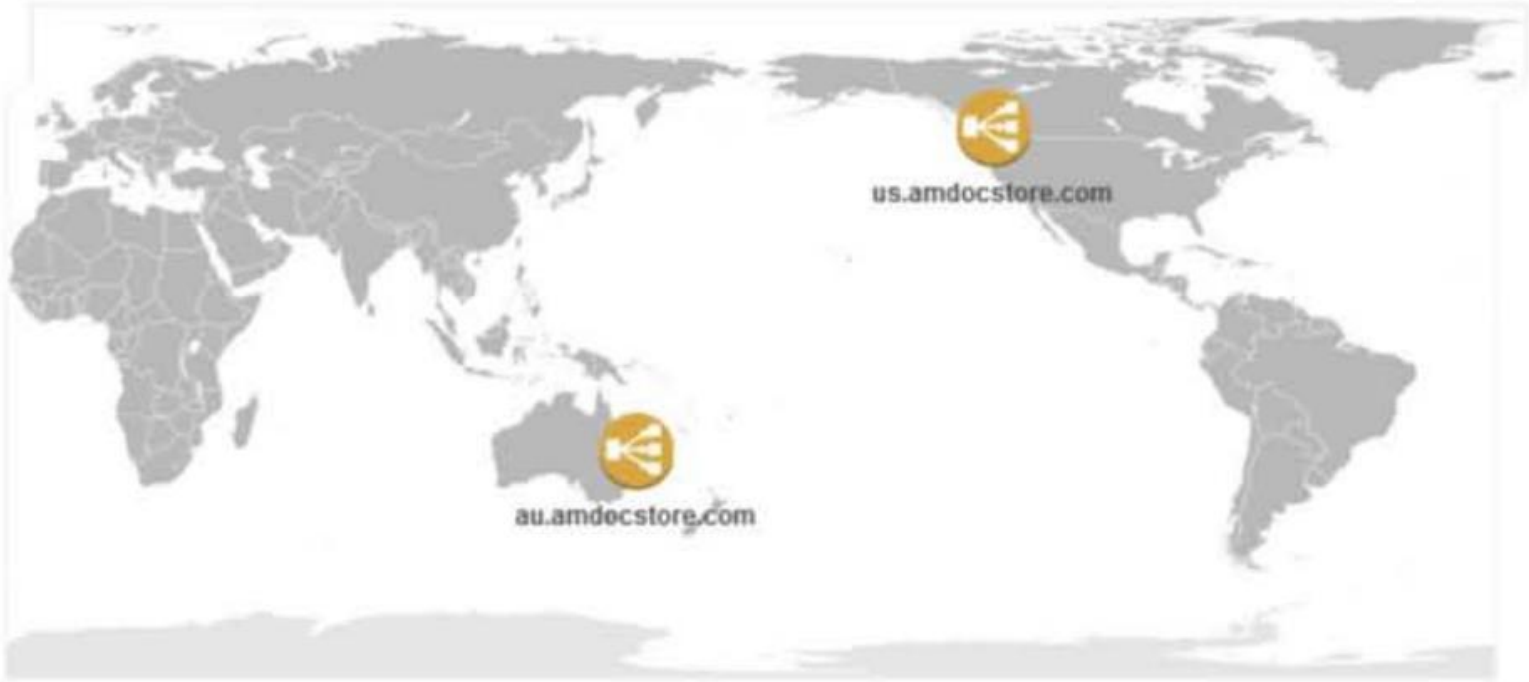
Sample Architecture

Application deployed in two regions: us-west-2 (Oregon) and ap-southeast-2 (Sydney)

DocStore

Each ELB has a region-specific CNAME in Route 53

DocStore



Latency Based Routing feature of **Route 53** directs customers to closest endpoint

DocStore

amazon web services

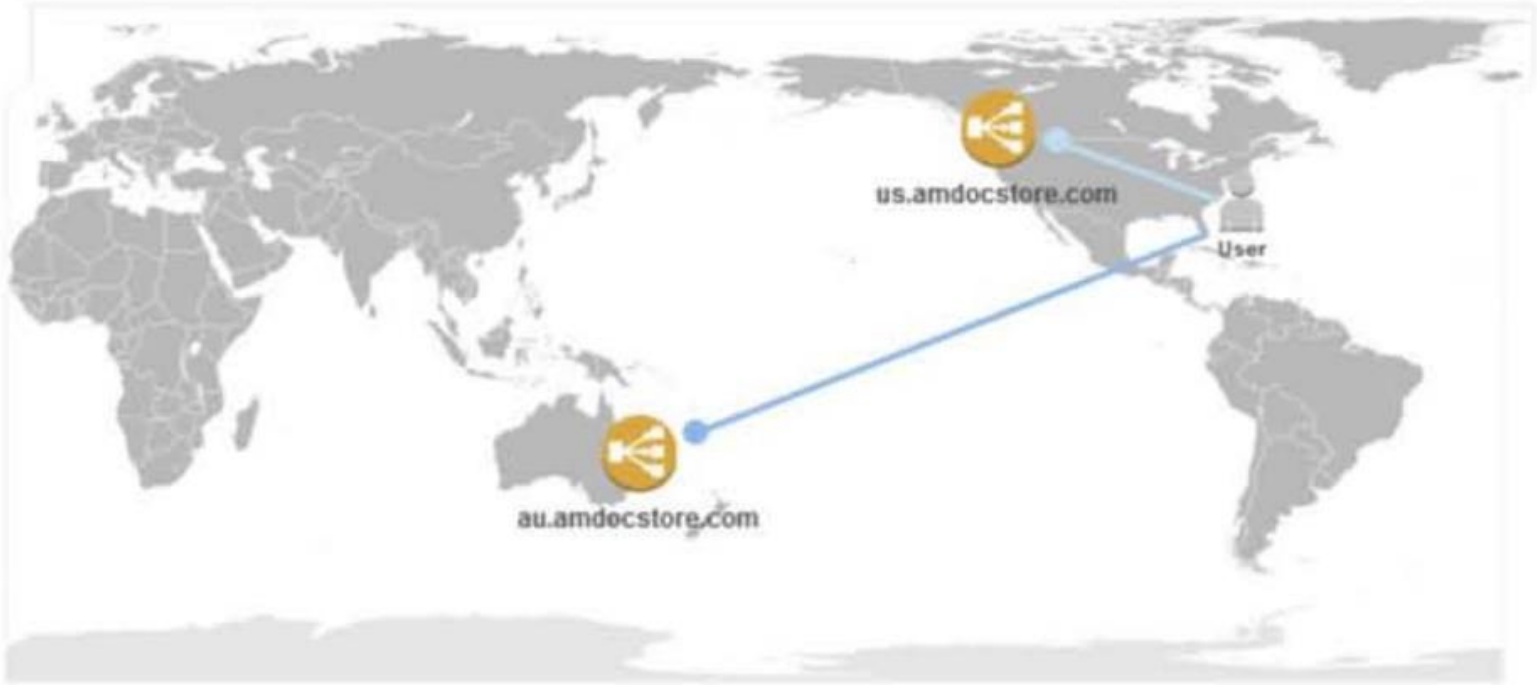
Closer to Oregon? That's where you go. In Perth? Head to the ELB in Sydney.

DocStore

amazon web services

Closer to Oregon? That's where you go.

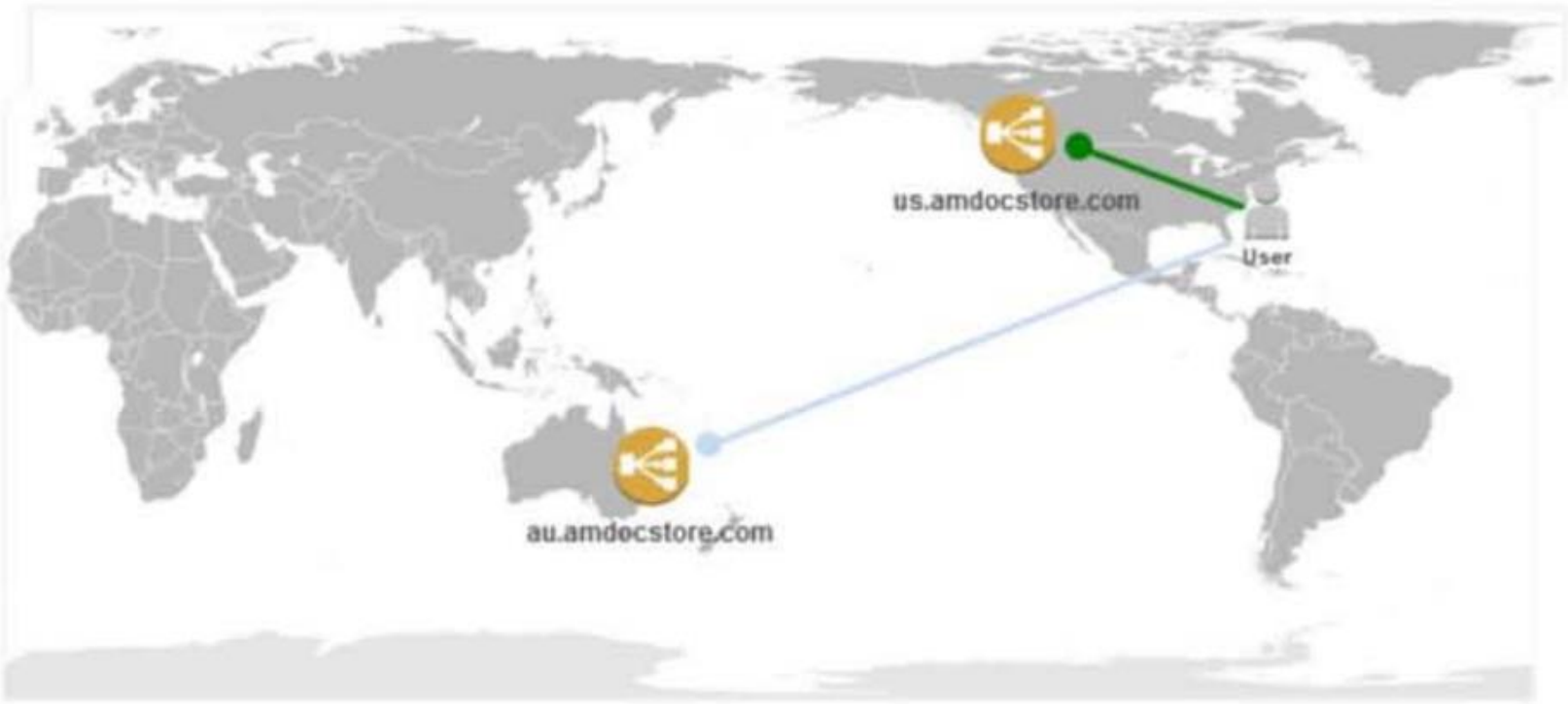
DocStore



amazon web services

Closer to Oregon? That's where you go.

DocStore



amazon web services

Let’s focus on one region

DocStore

Route 53 LBR directs a user to the **nearest Elastic Load Balancer** (in this case, the ELB in us-west-2)

DocStore

DocStore

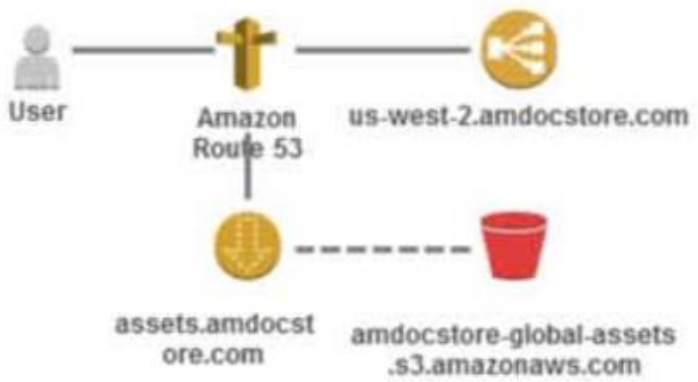
DocStore stores all **static web assets** (CSS, images, JavaScript, etc.) **in S3**

DocStore

This bucket exists in the us-west-2 origin, but is the **canonical source for static asset requests globally**

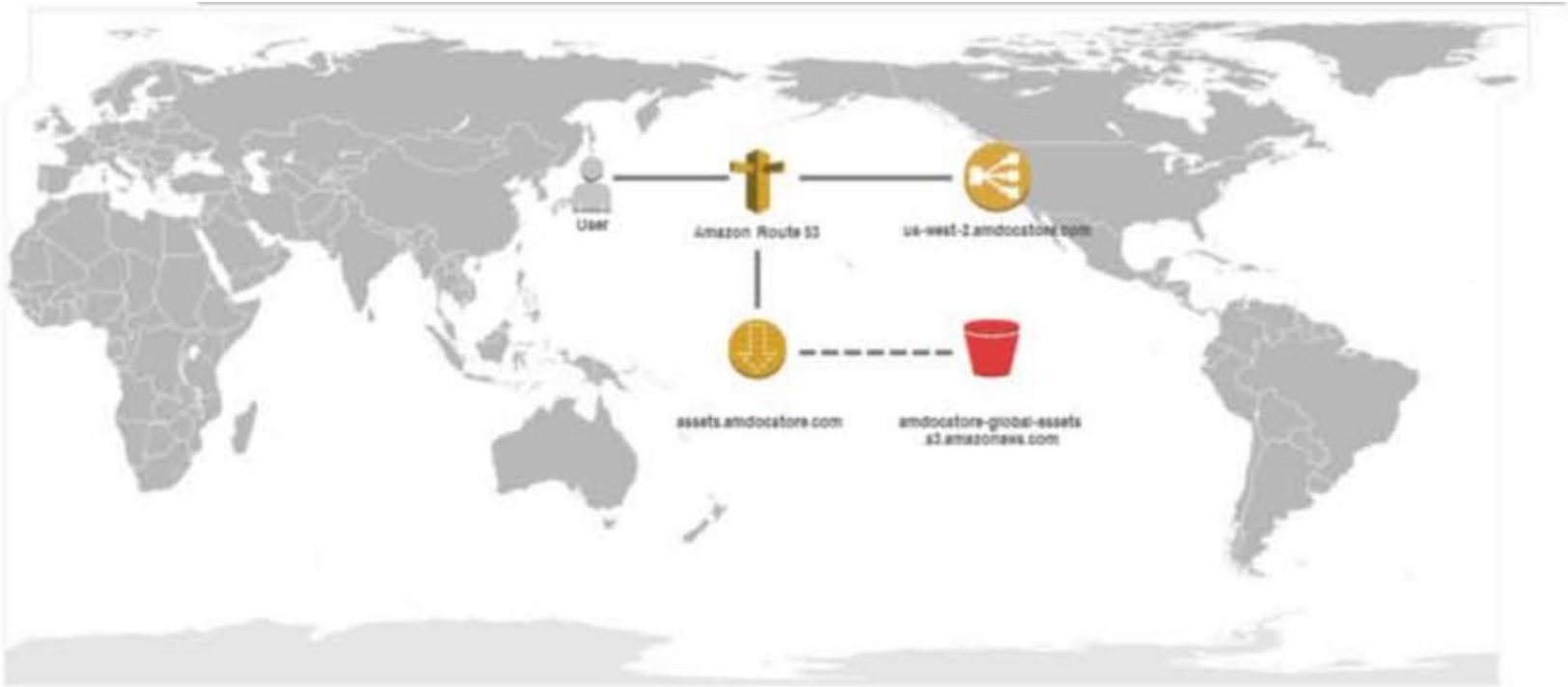
CloudFront distributes static web assets in S3 to end users with low latency using a **global network of edge locations**

DocStore



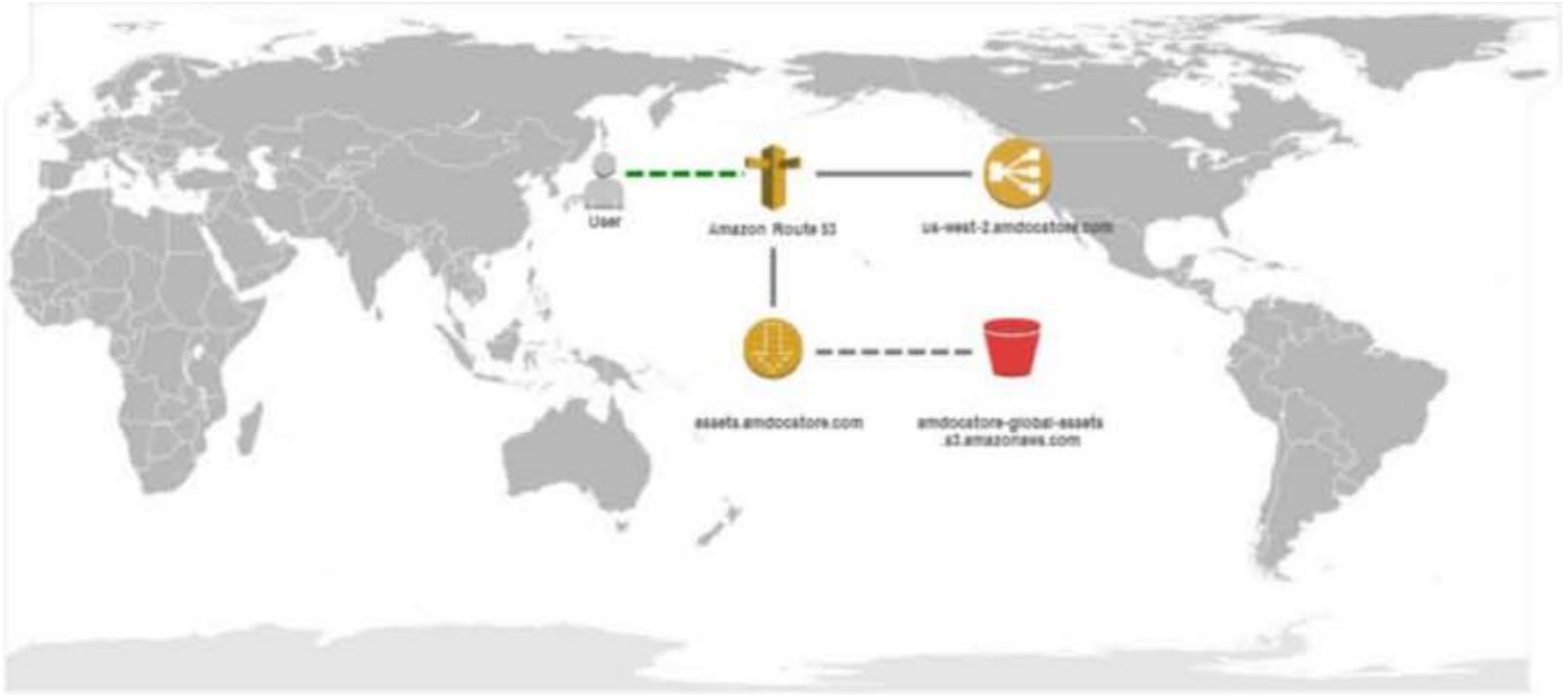
Let's see how a user in **Tokyo** would access
http://assets.amdocstore.com/css/style.css

DocStore



Any user request for assets.amdocstore.com will **automatically route to the nearest edge location**

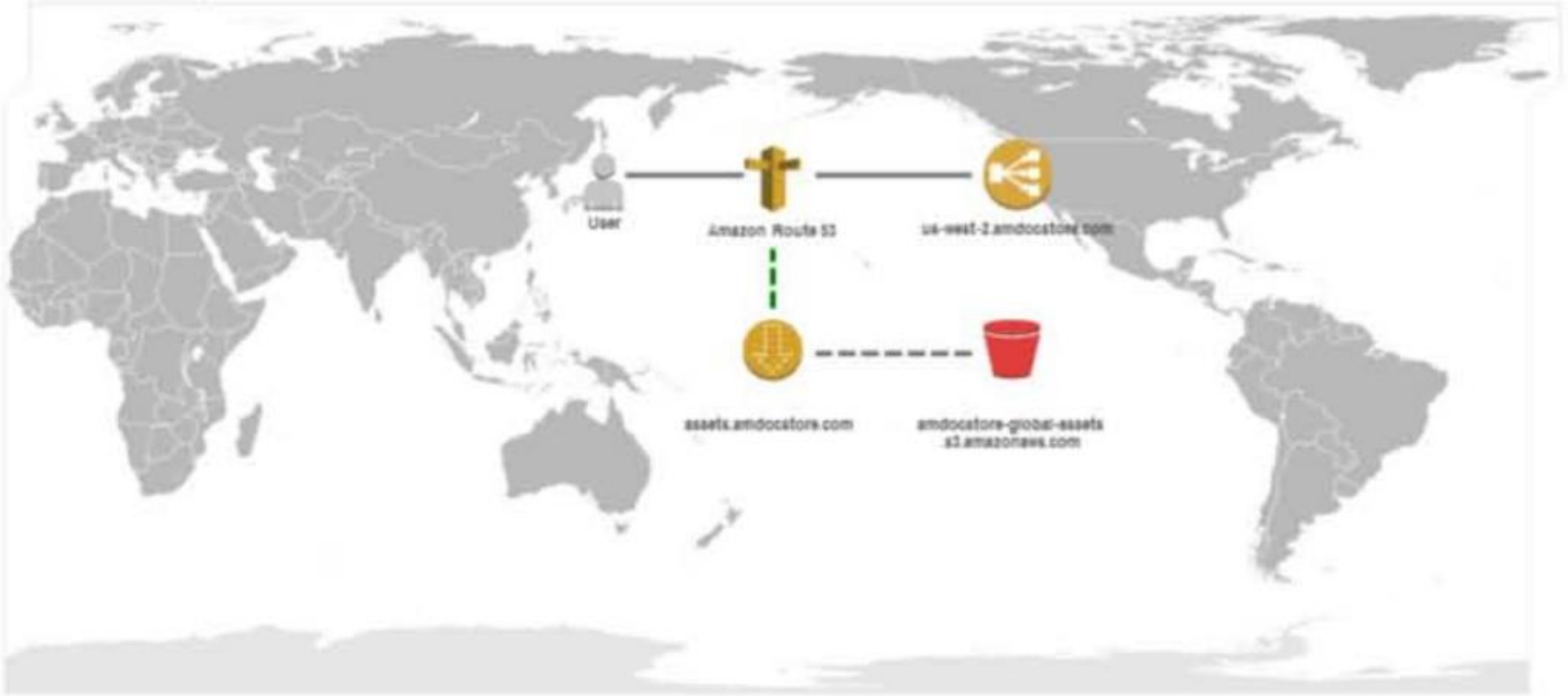
DocStore



amazon web services

Any user request for assets.amdocstore.com will **automatically route to the nearest edge location**

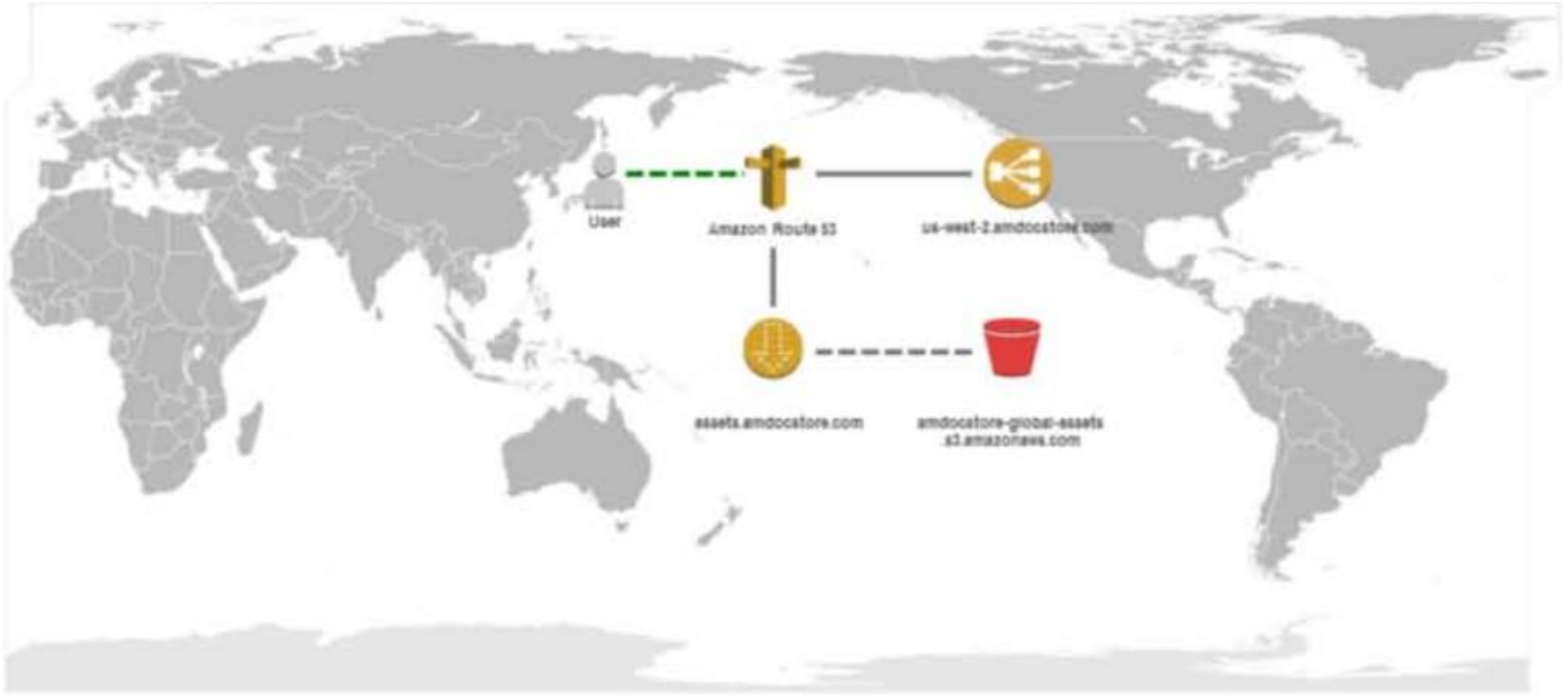
DocStore



amazon web services

Any user request for assets.amdocstore.com will **automatically route to the nearest edge location**

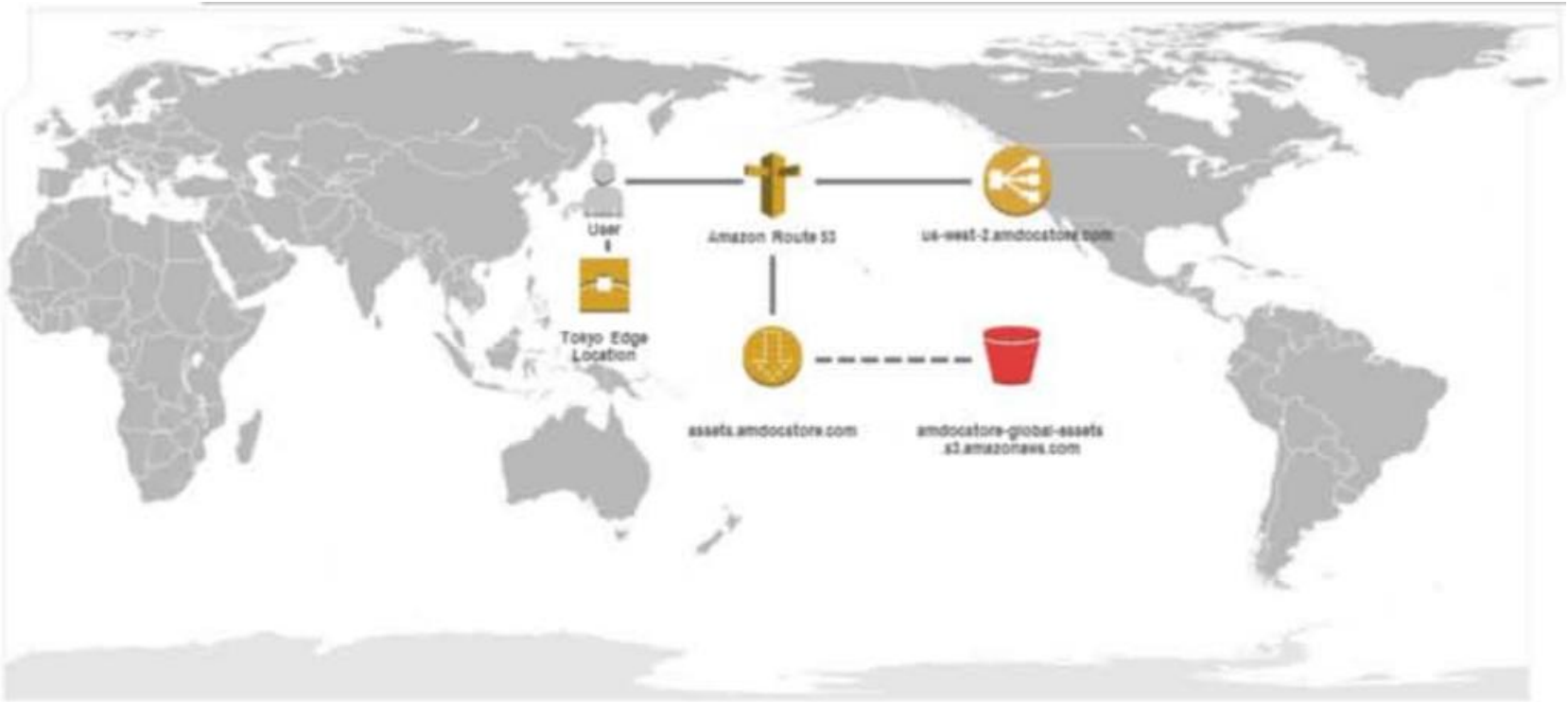
DocStore



amazon web services

Any user request for assets.amdocstore.com will **automatically route to the nearest edge location**

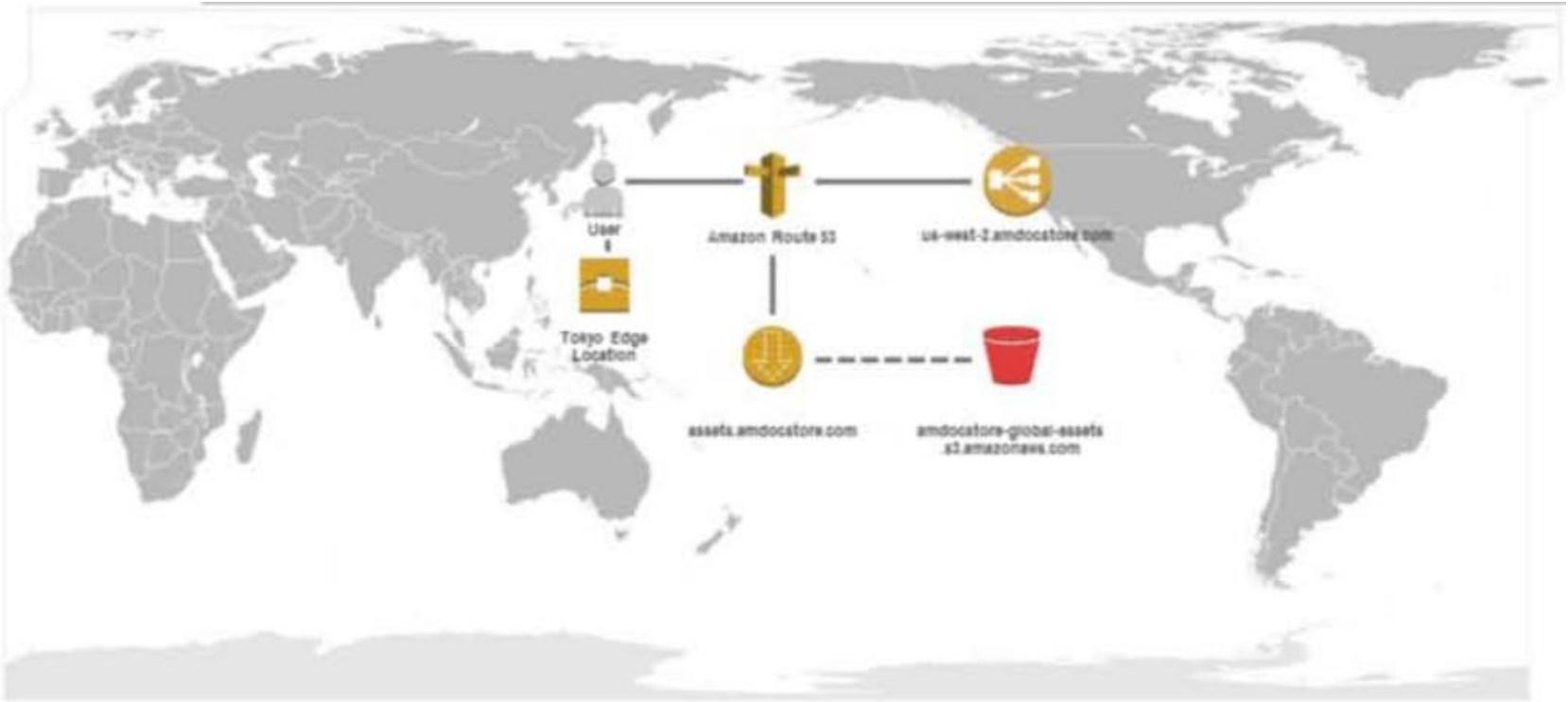
DocStore



amazon web services

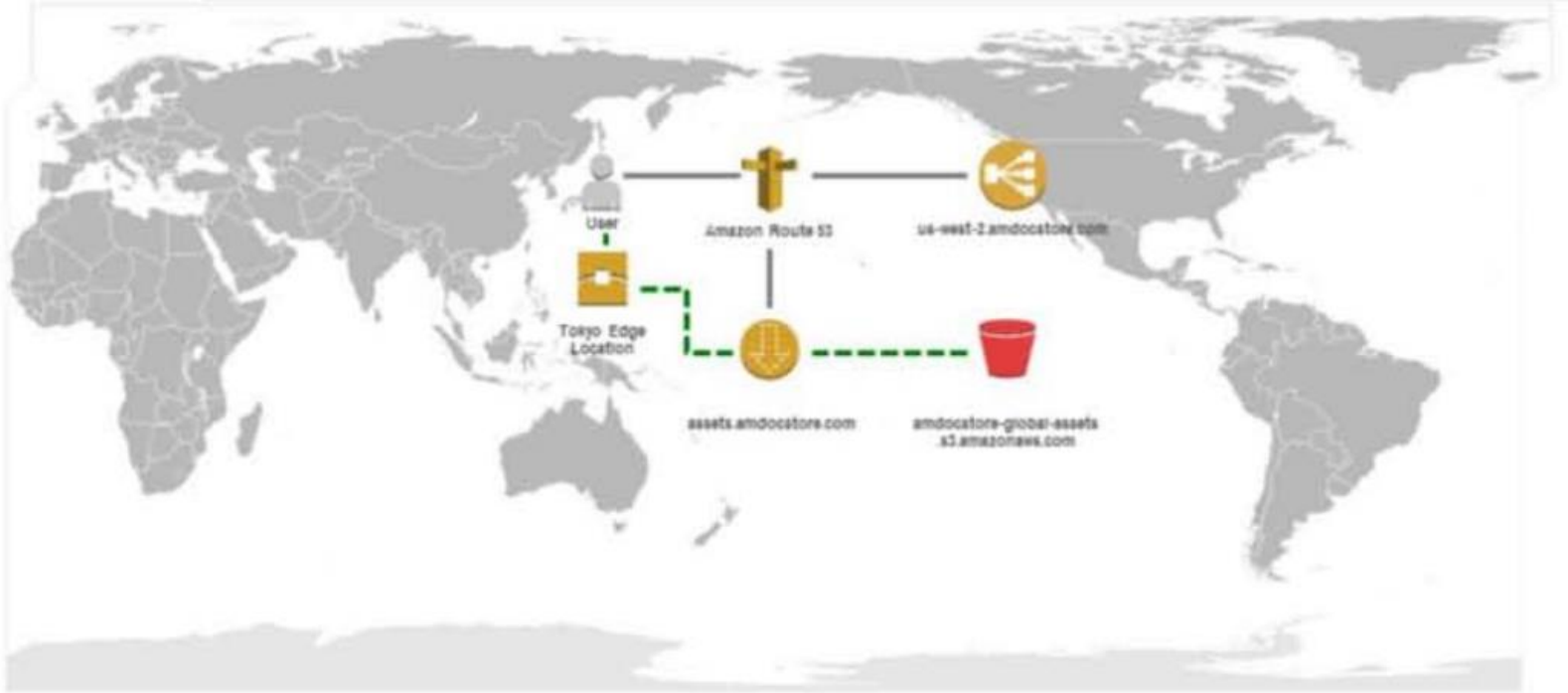
The requested **file will be served from the Tokyo edge** if it is in the cache

DocStore



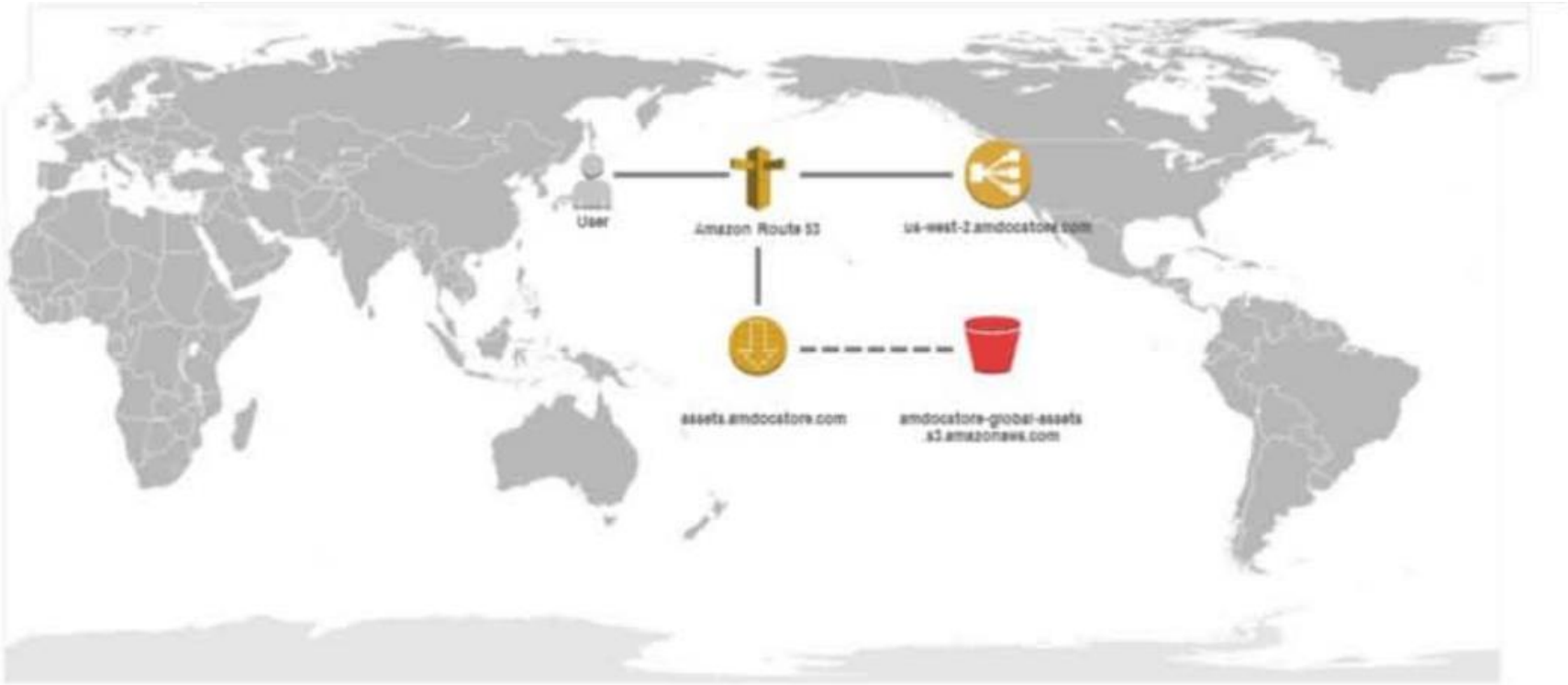
The requested file will be served from the Tokyo edge if it is in the cache, or **retrieved from the origin and cached at the edge** for future requests

DocStore



Let’s go back to our application deployment in **us-west-2**

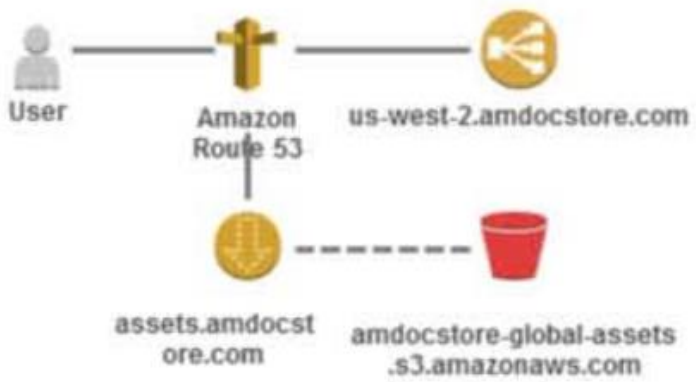
DocStore



amazon web services

Let’s go back to our application deployment in **us-west-2**

DocStore

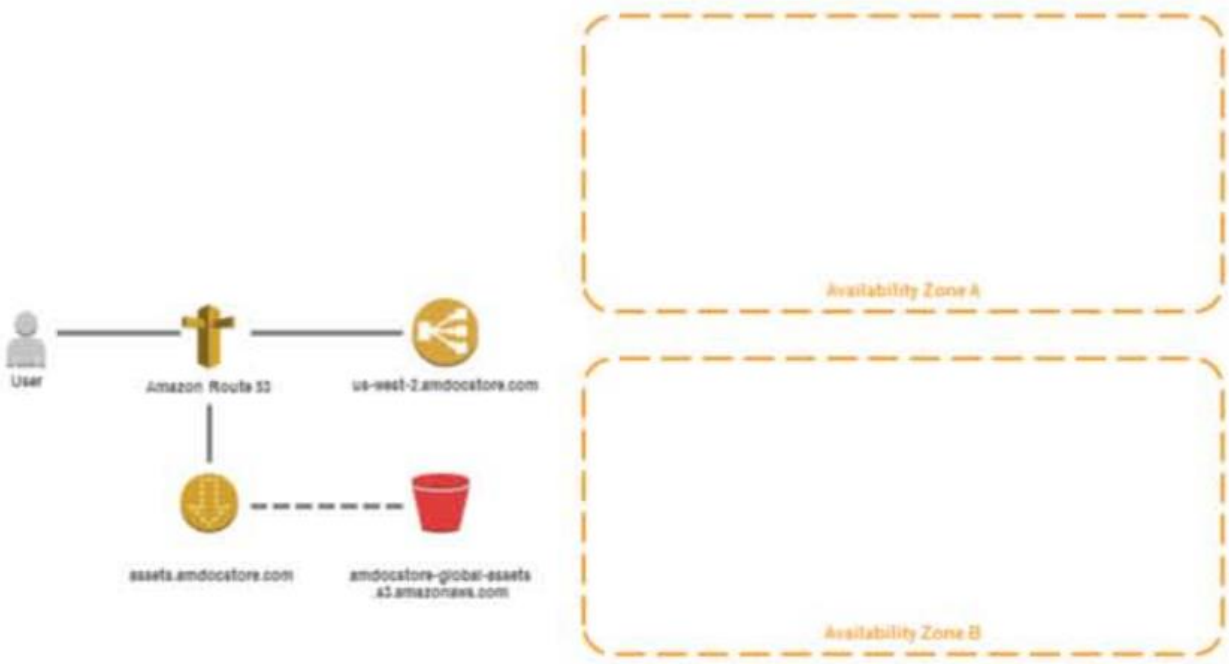


DocStore

Let’s go back to our application deployment in **us-west-2** and **focus on the browser component of DocStore**

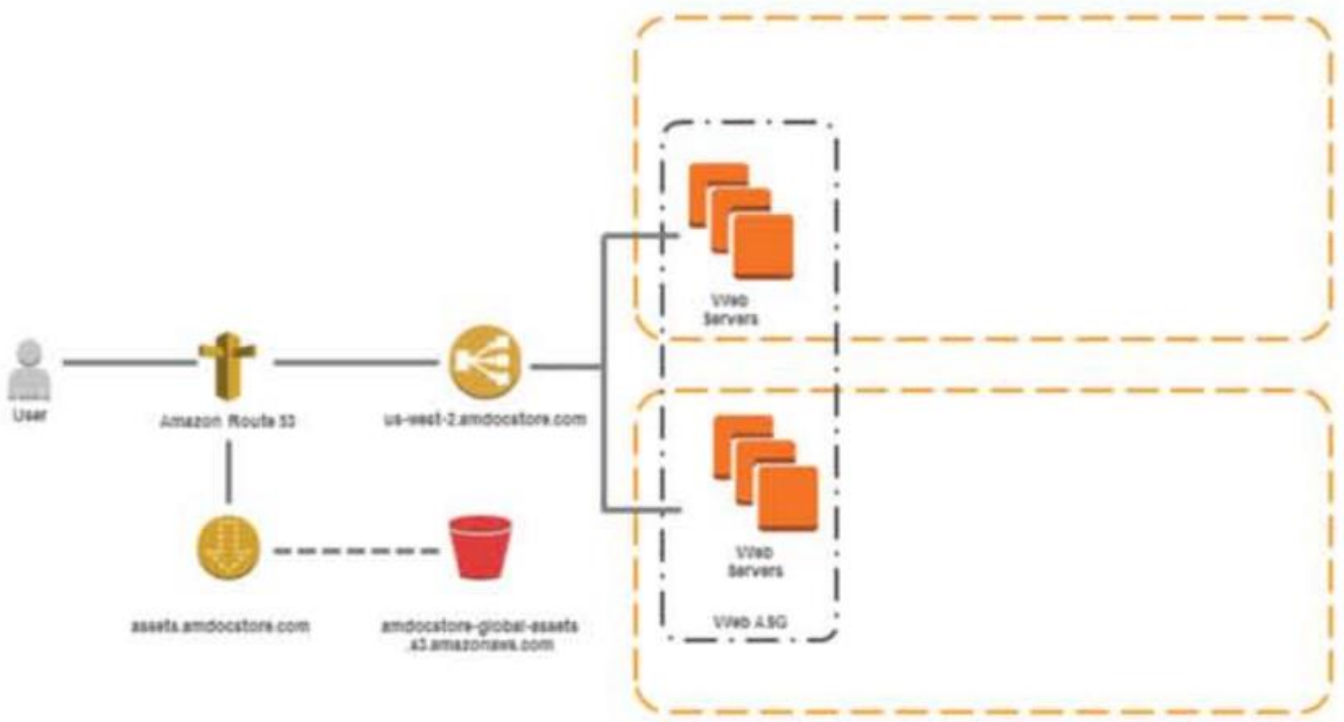
Web servers will be deployed across multiple Availability Zones using Auto Scaling

DocStore



Web servers will be deployed across multiple Availability Zones using Auto Scaling

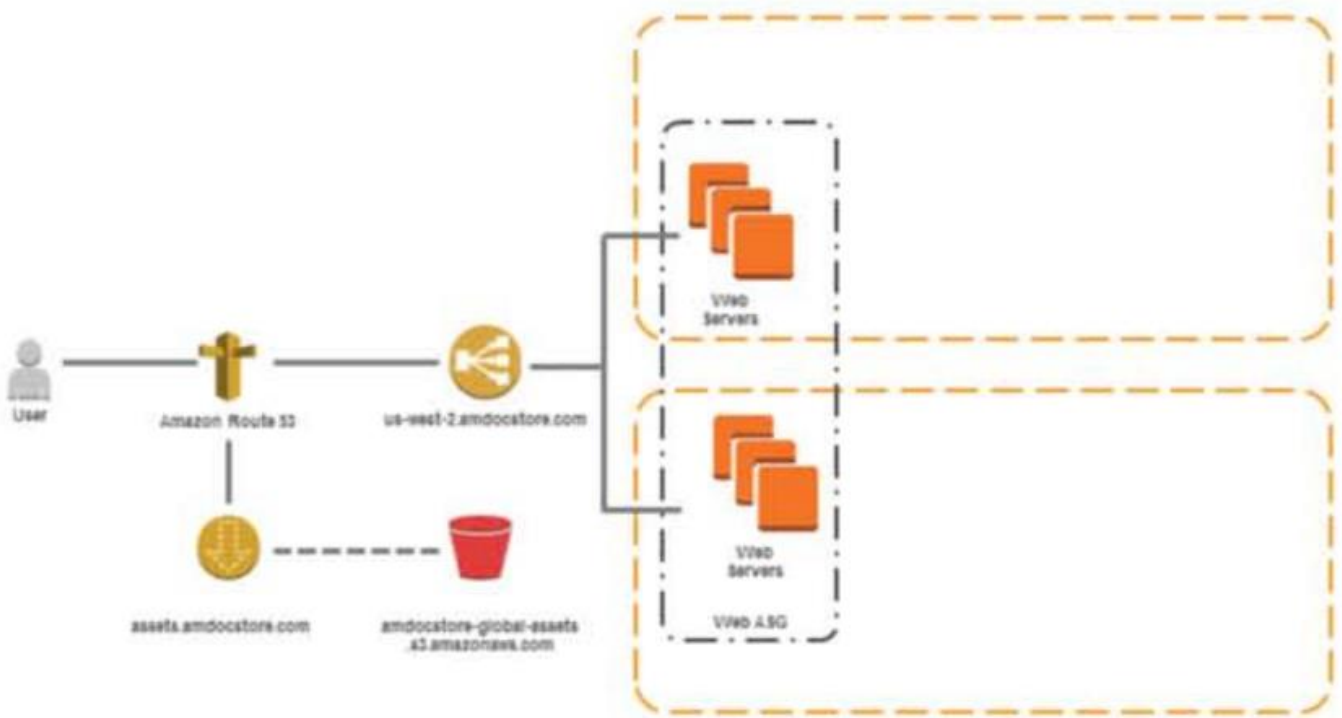
DocStore



amazon web services

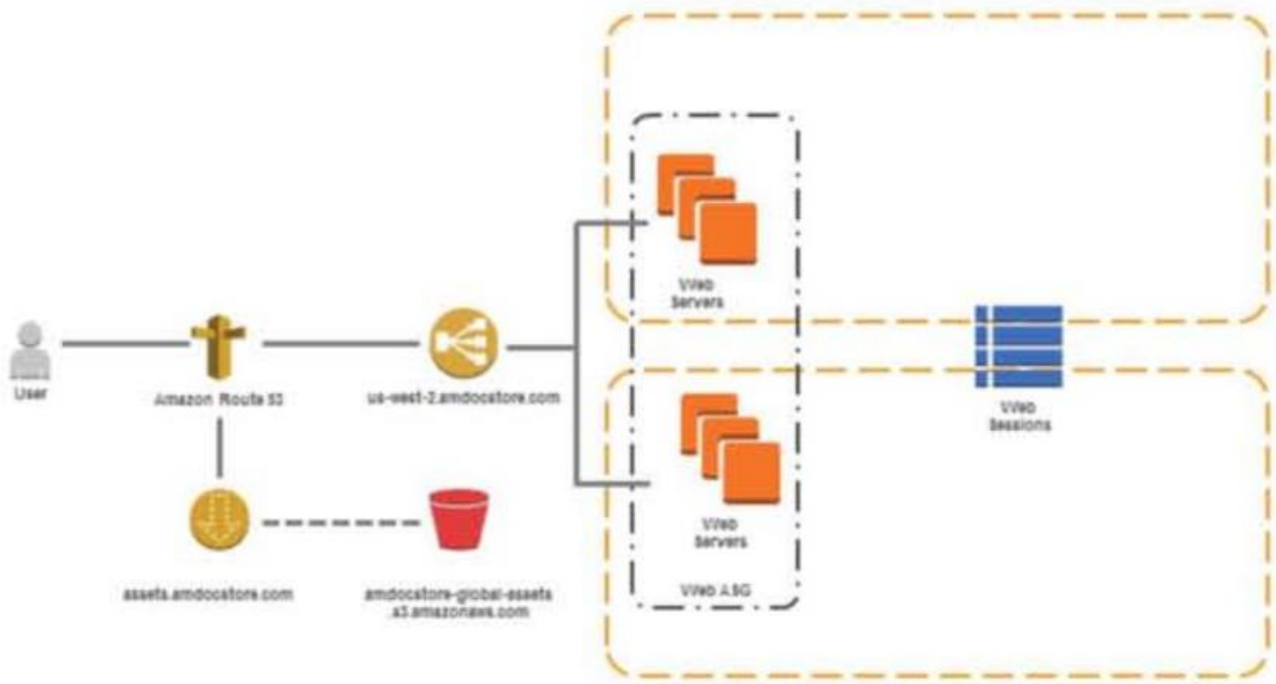
Web servers host the DocStore Interface that users will access with their web browsers

DocStore



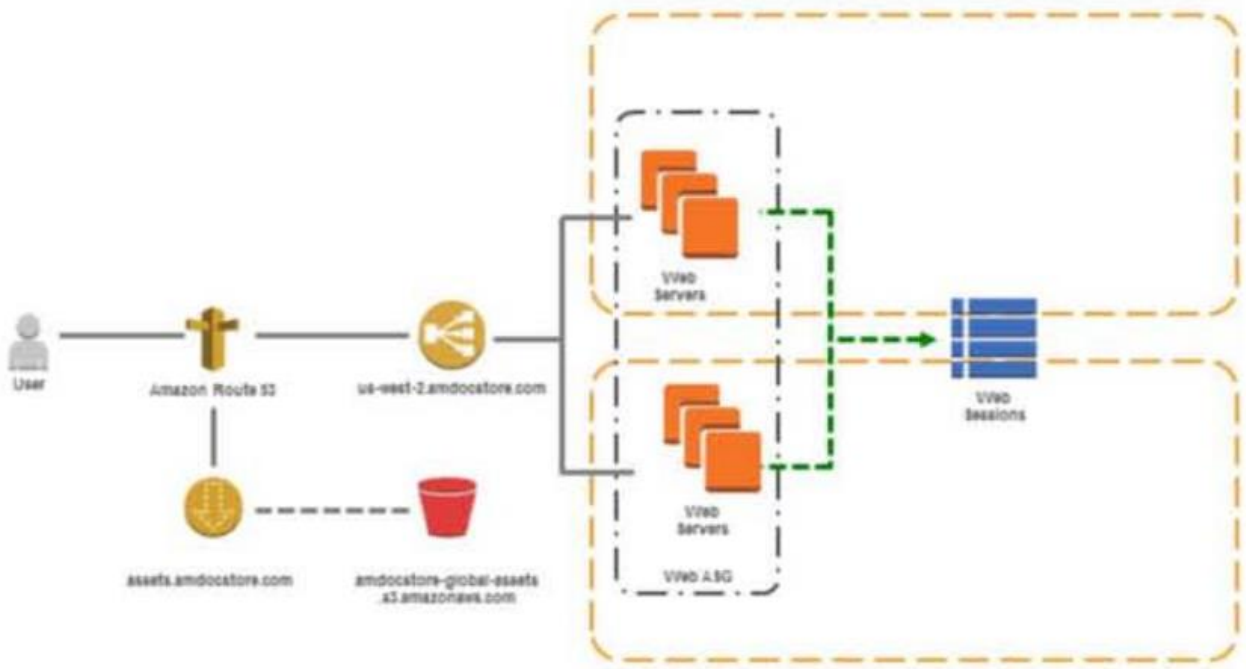
Session state is stored out of the web tier in a DynamoDB table. DynamoDB is a regional service, meaning the data is automatically distributed across AZs

DocStore



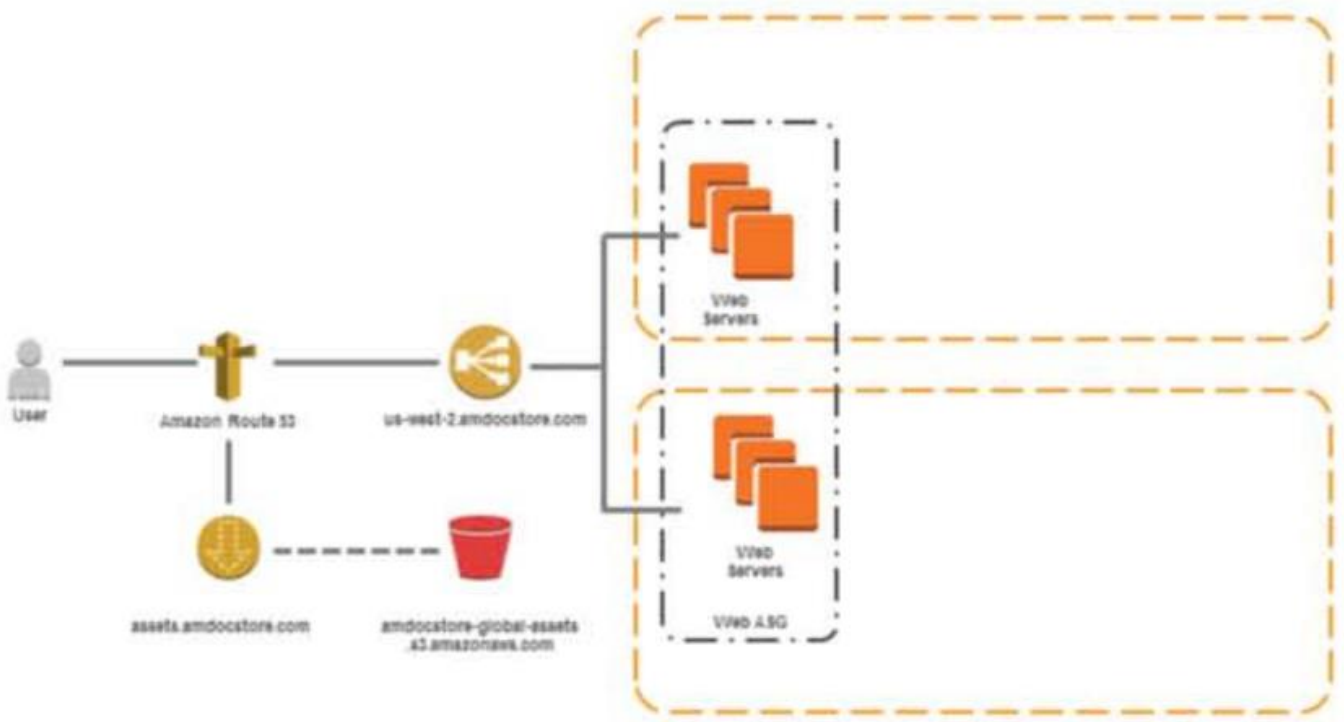
Session state is stored out of the web tier in a DynamoDB table. DynamoDB is a regional service, meaning the data is automatically distributed across AZs

DocStore



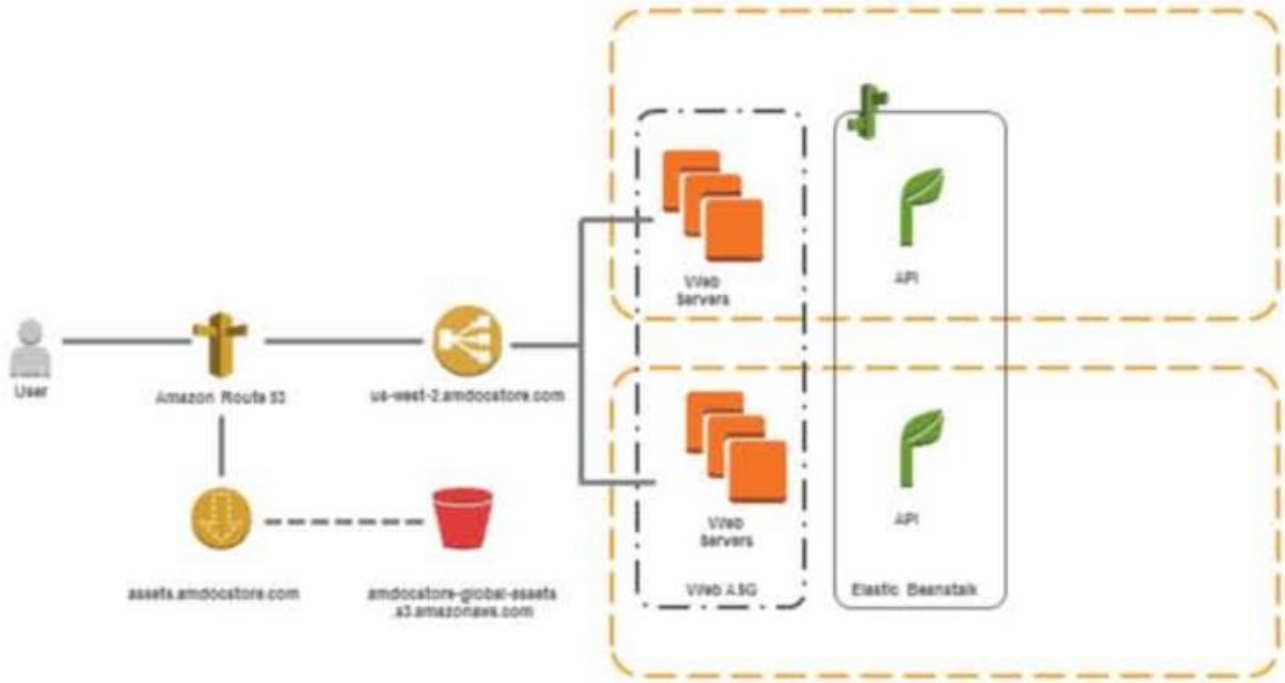
Core functionality is provided by the **DocStore API**. Web servers consume this API

DocStore



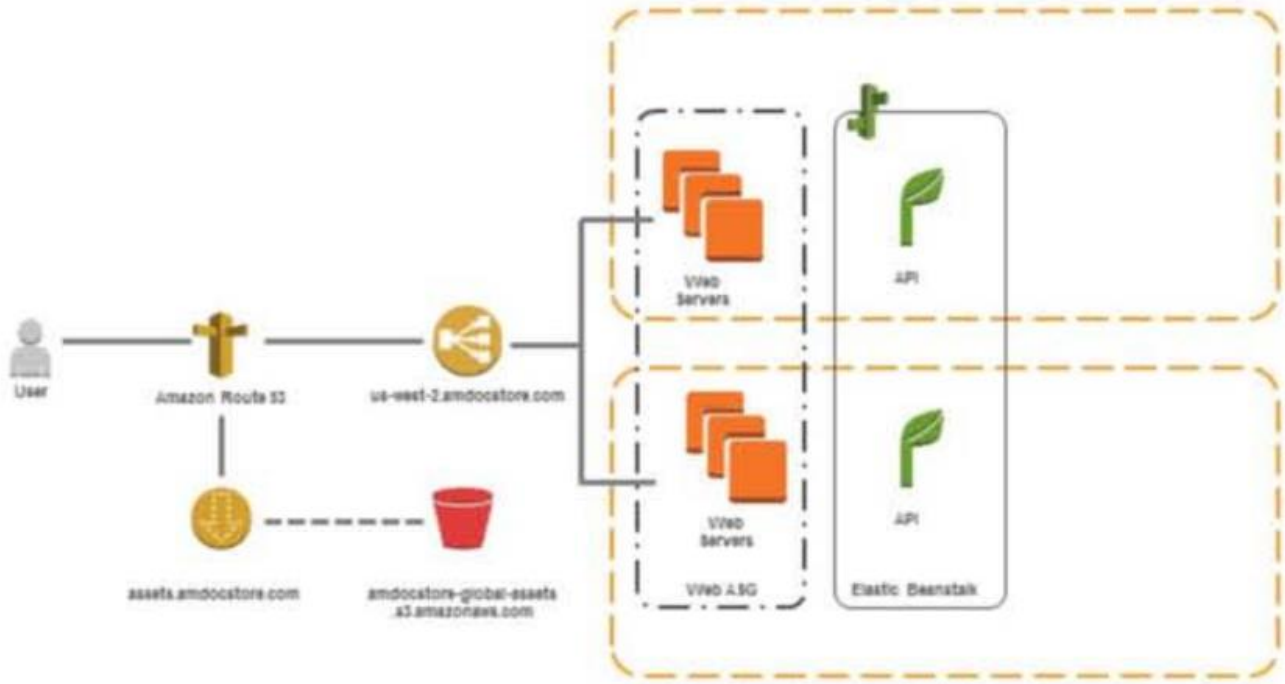
The DocStore API will be deployed as an application in Elastic Beanstalk container

DocStore



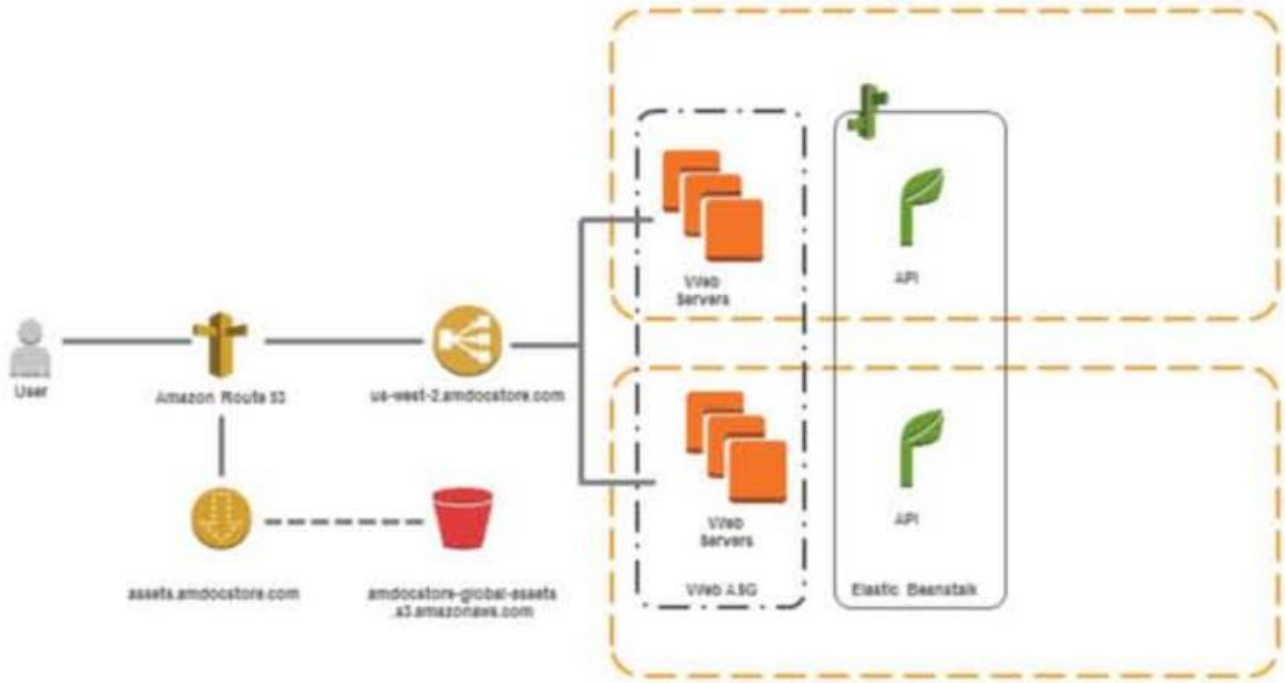
Elastic Beanstalk provides Multi-AZ deployment, Auto Scaling, and load balancing

DocStore



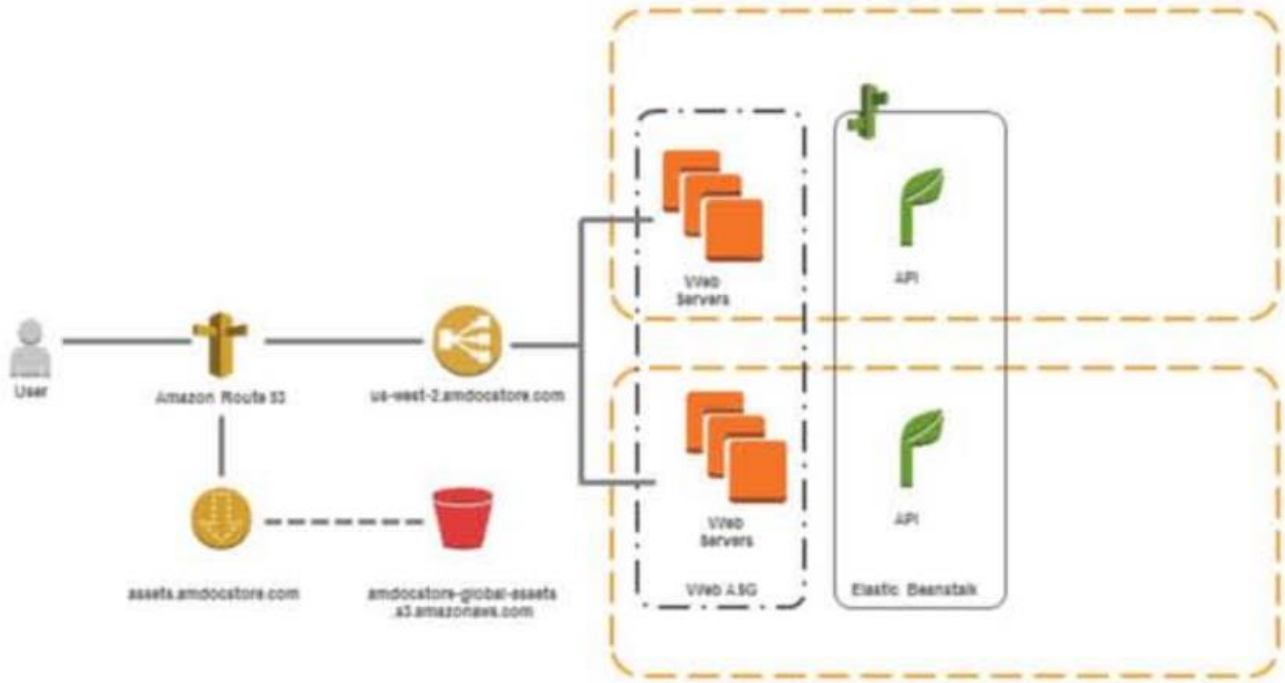
Elastic Beanstalk supports Java, .NET, Python, PHP, Ruby, and node.js applications

DocStore



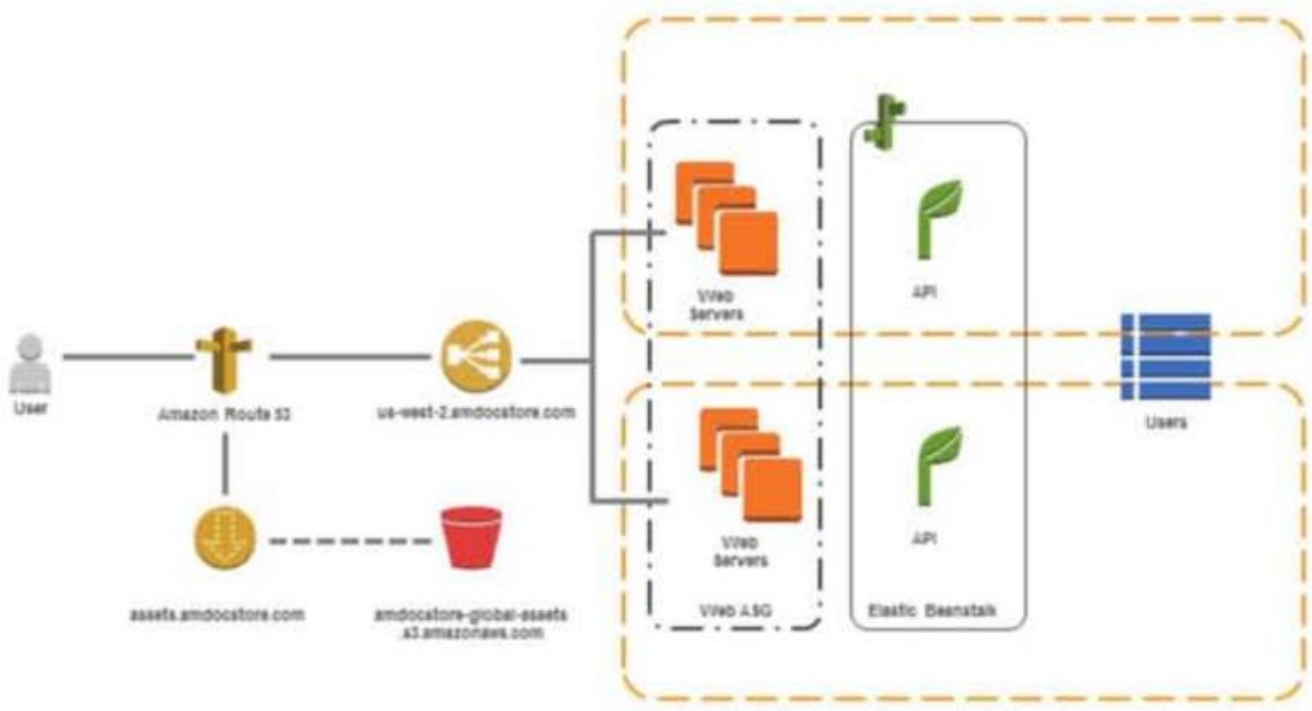
The API will use several different data stores for different types of data

DocStore



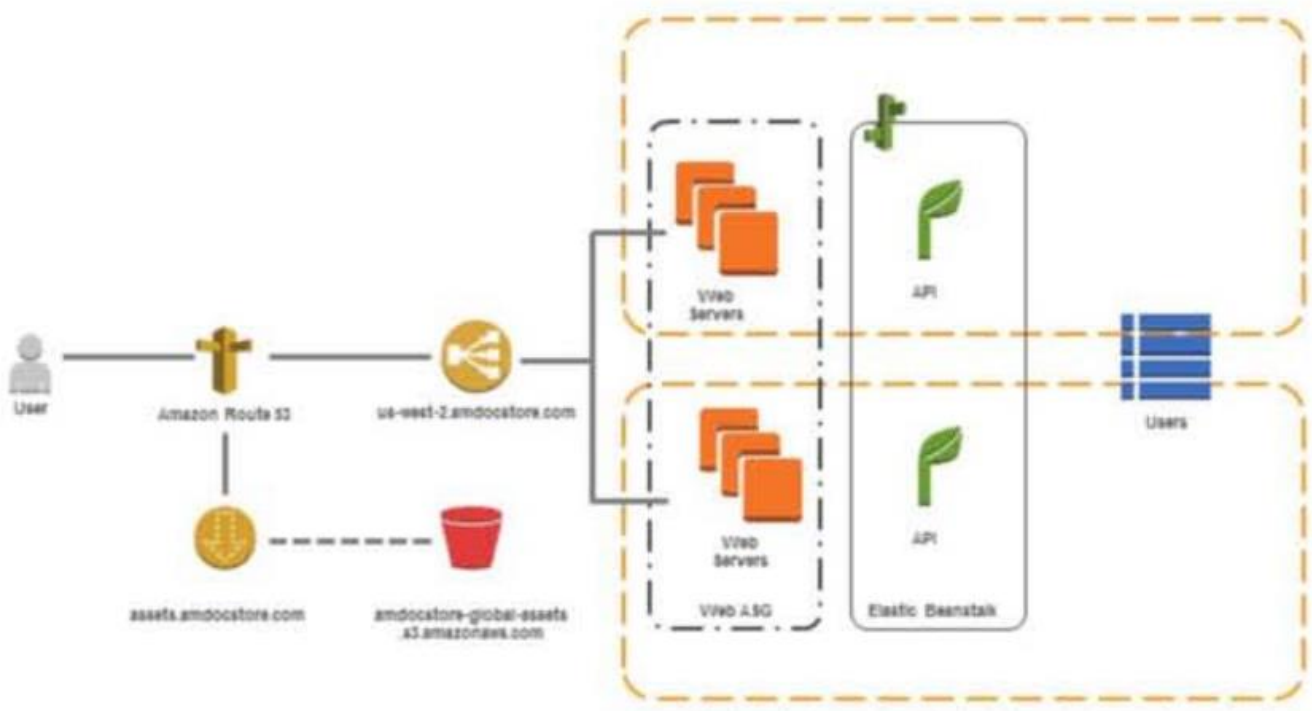
User account information will be stored in a **DynamoDB** table

DocStore



Let’s see what happens when a **new user signs up for DocStore**

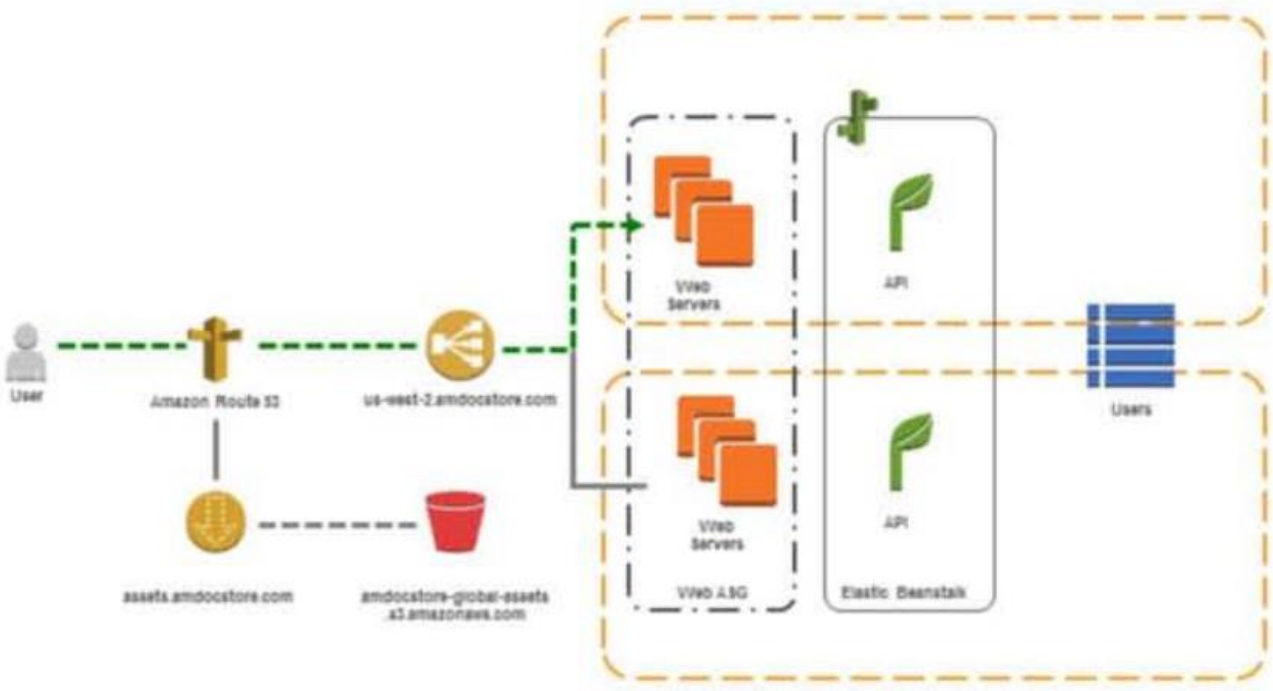
DocStore



amazon web services

Create Account: User submits sign-up form to web servers with username, password, etc.

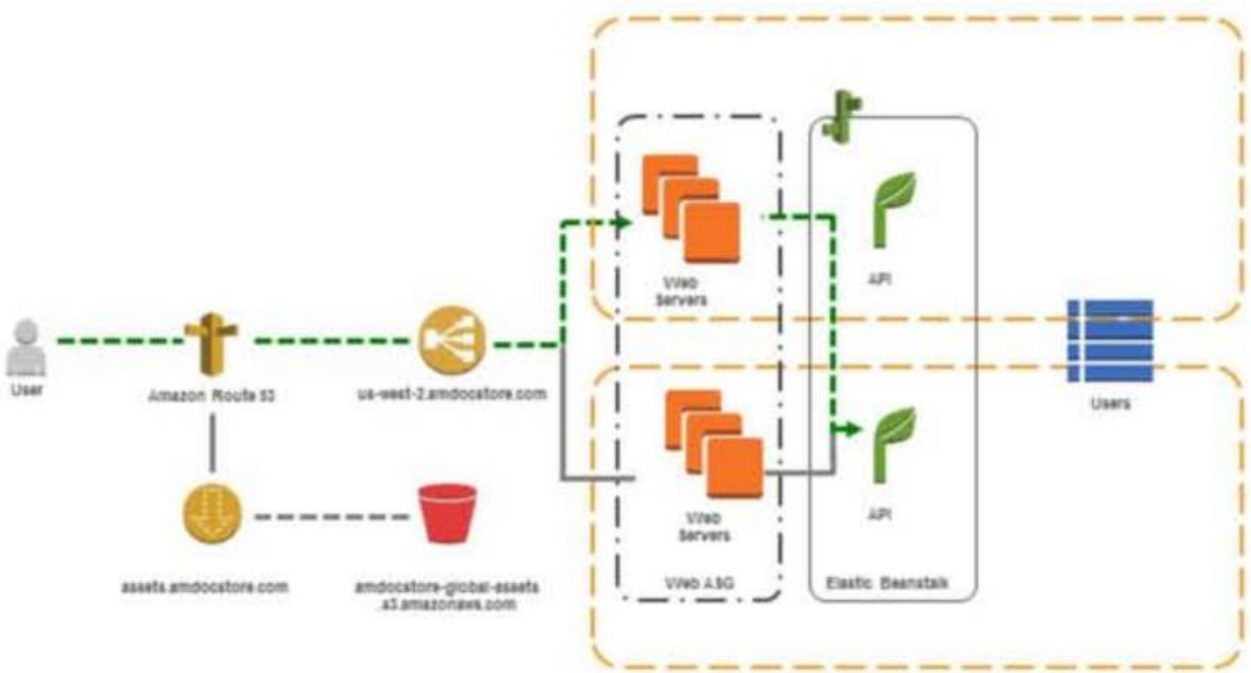
DocStore



amazon web services

Create Account: Web servers invoke CreateAccount API

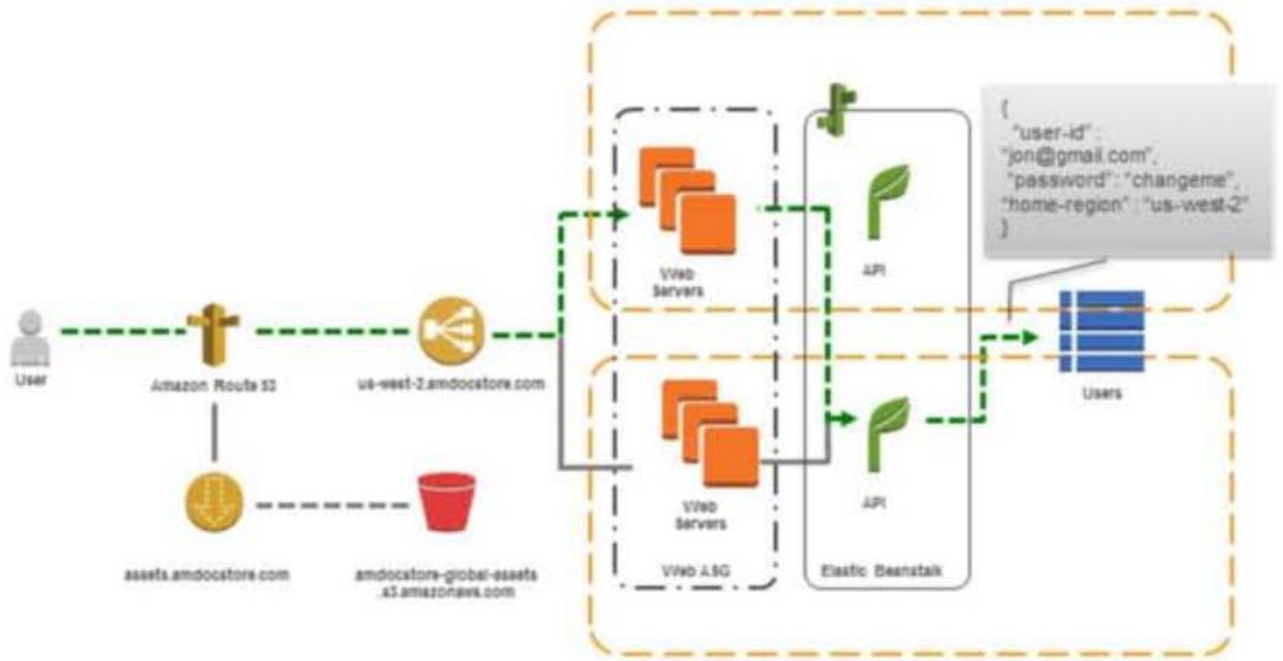
DocStore



amazon web services

Create Account: API creates a **new item** in the datastore

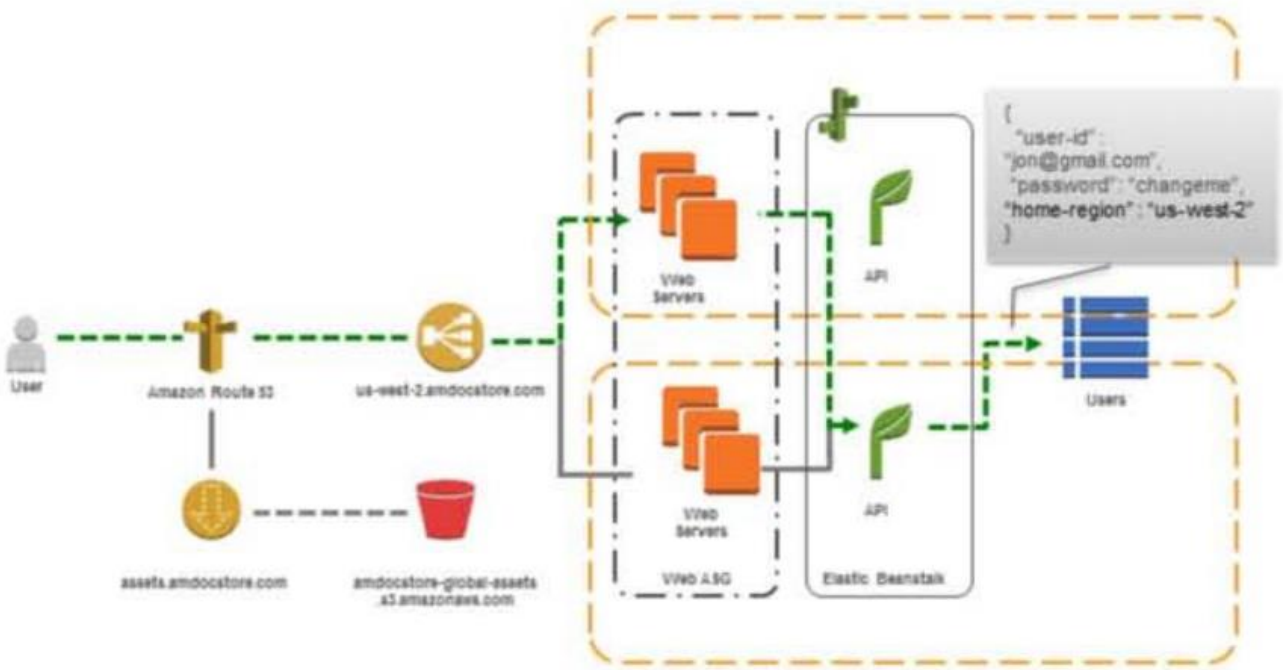
DocStore



amazon web services

Create Account: The API app deployed in Elastic Beanstalk has a configuration setting making it aware of the region it is running in

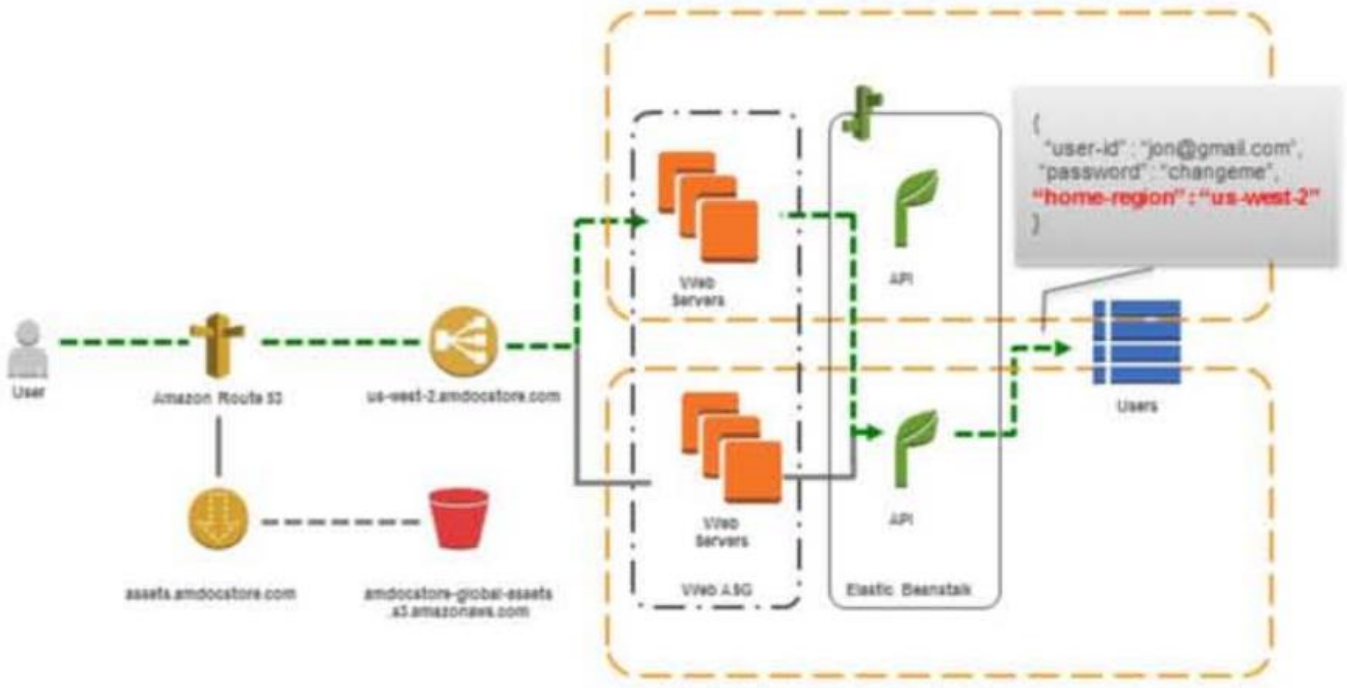
DocStore



amazon web services

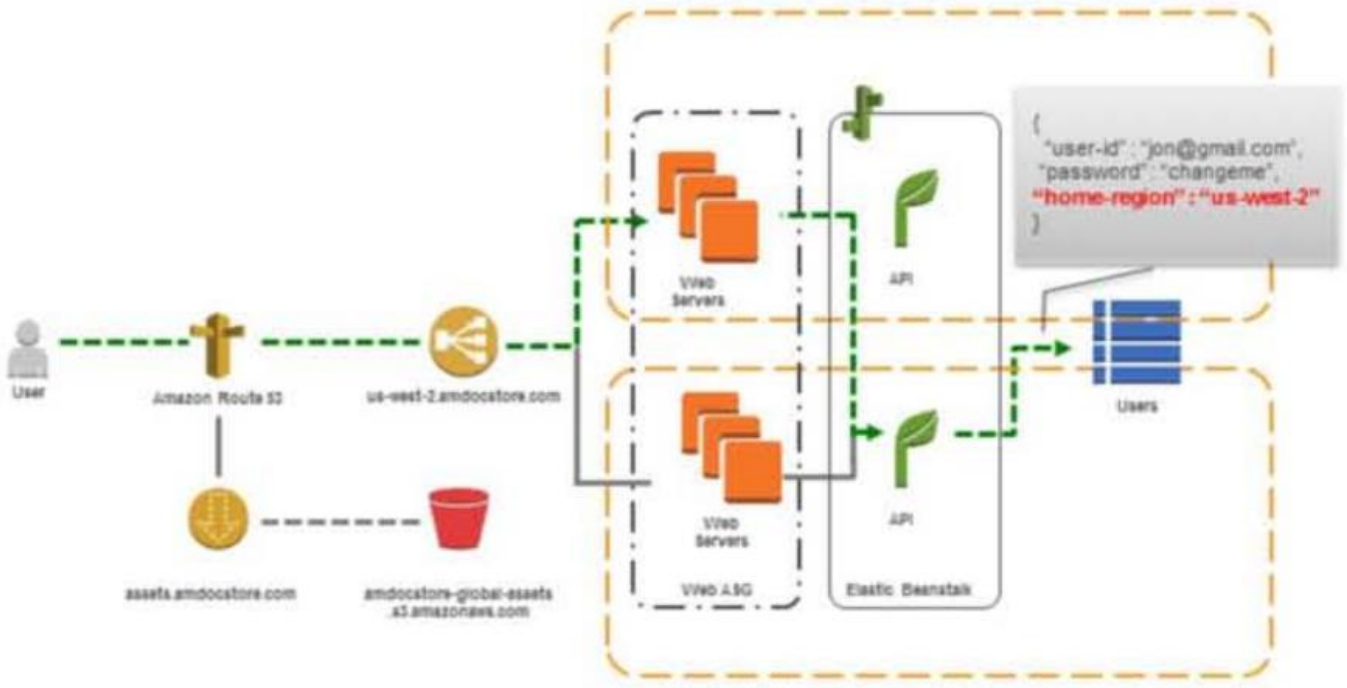
Create Account: In addition to standard information, the API stores the user's "home region", i.e., the region the user signed up in

DocStore



Create Account: Replicating account data to our Sydney deployment will allow the API in that region to redirect the user to his home region if he ever accesses from there

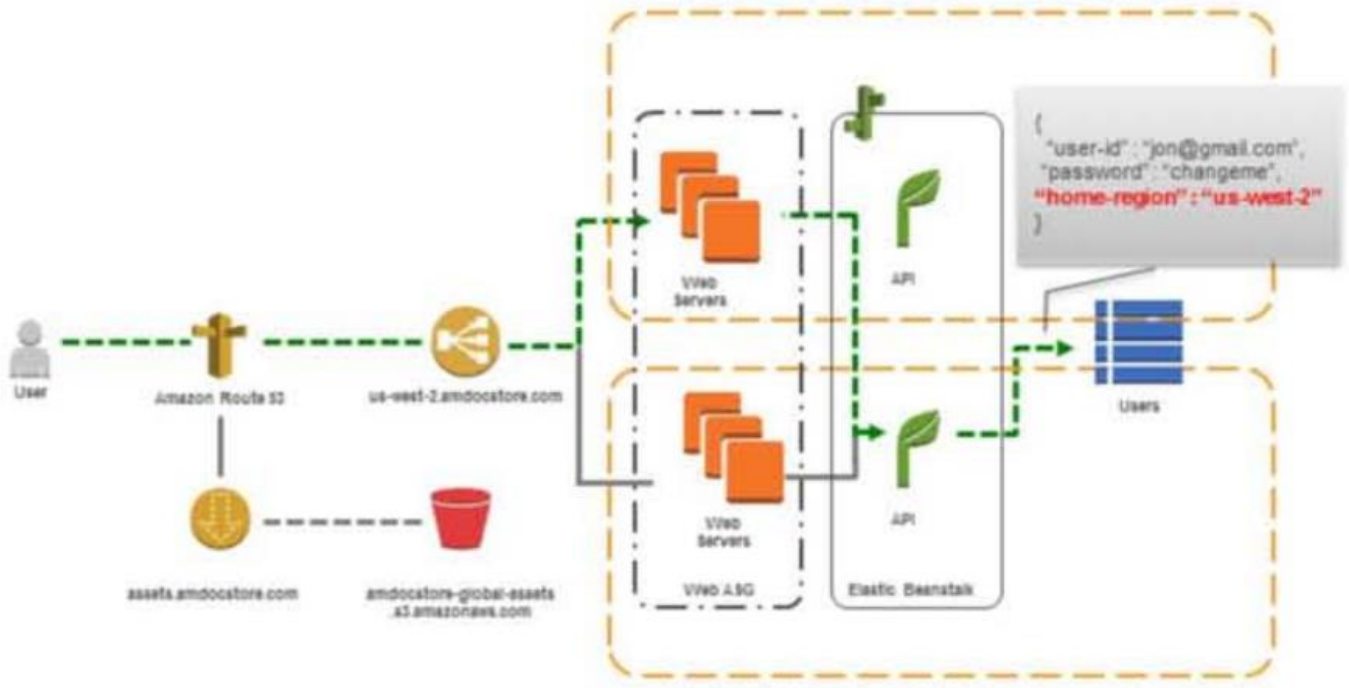
DocStore



amazon web services

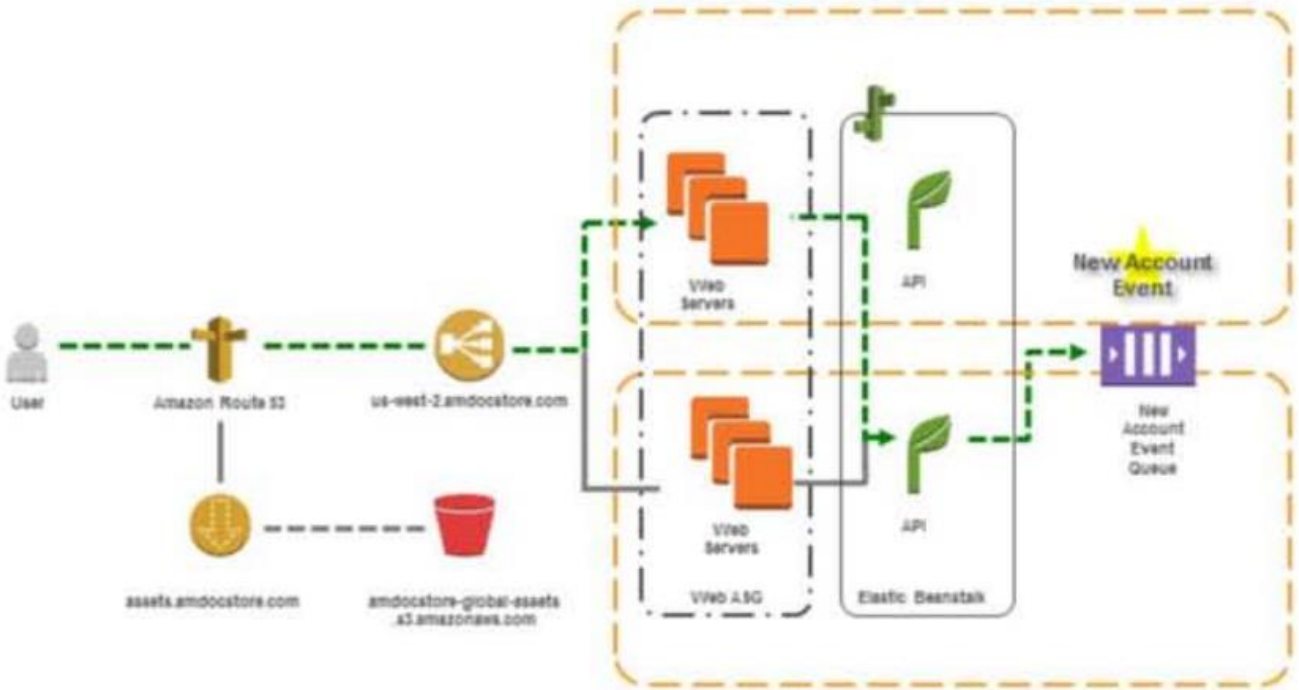
Create Account: We'll consider account signup an event that other components of our application might be interested in.

DocStore



Create Account: We'll let listeners interested in the event (i.e., a cross-region account replication service) receive the event by putting a message in a queue

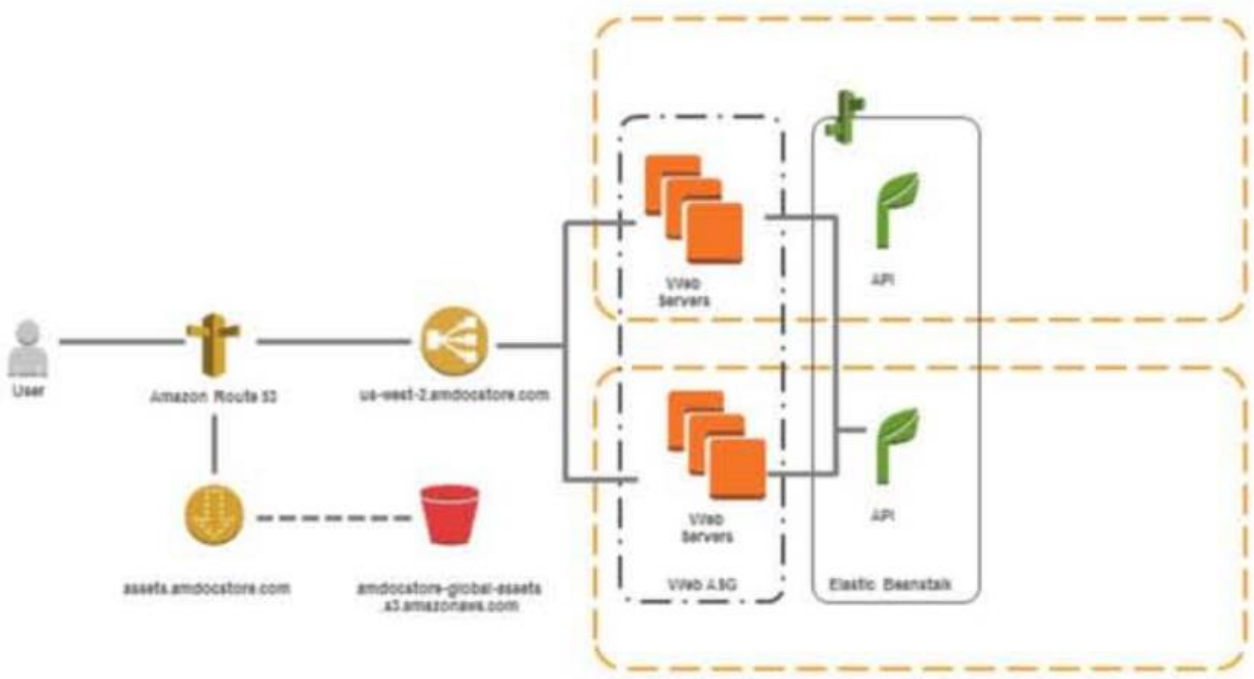
DocStore



amazon web services

Documents uploaded by a user are stored in an S3 bucket

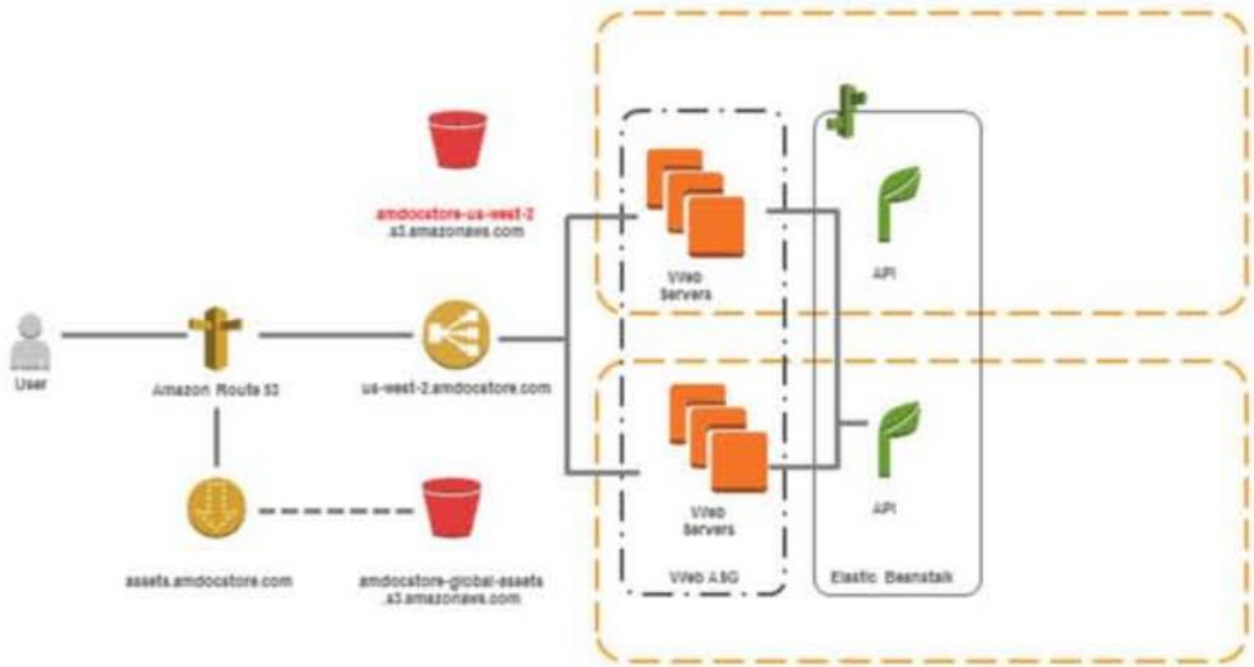
DocStore



amazon web services

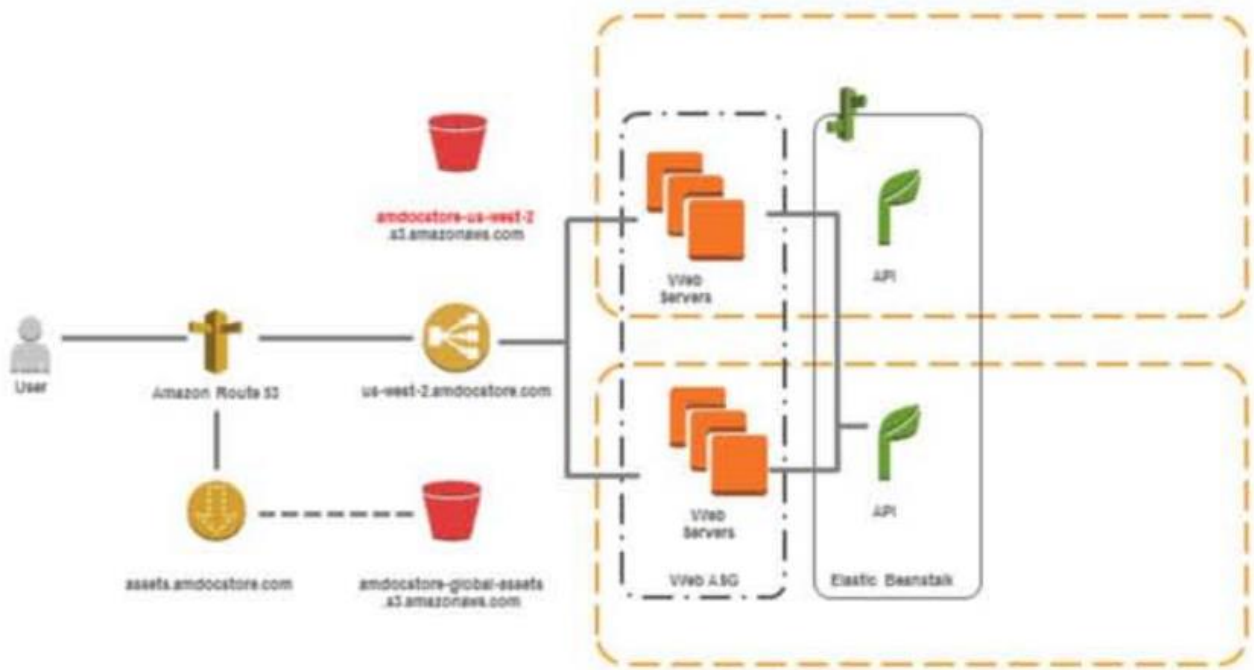
The S3 bucket name includes the name of the region it was created it in.

DocStore



The S3 bucket name includes the name of the region it was created it in. Users **upload documents to the bucket in their home region**

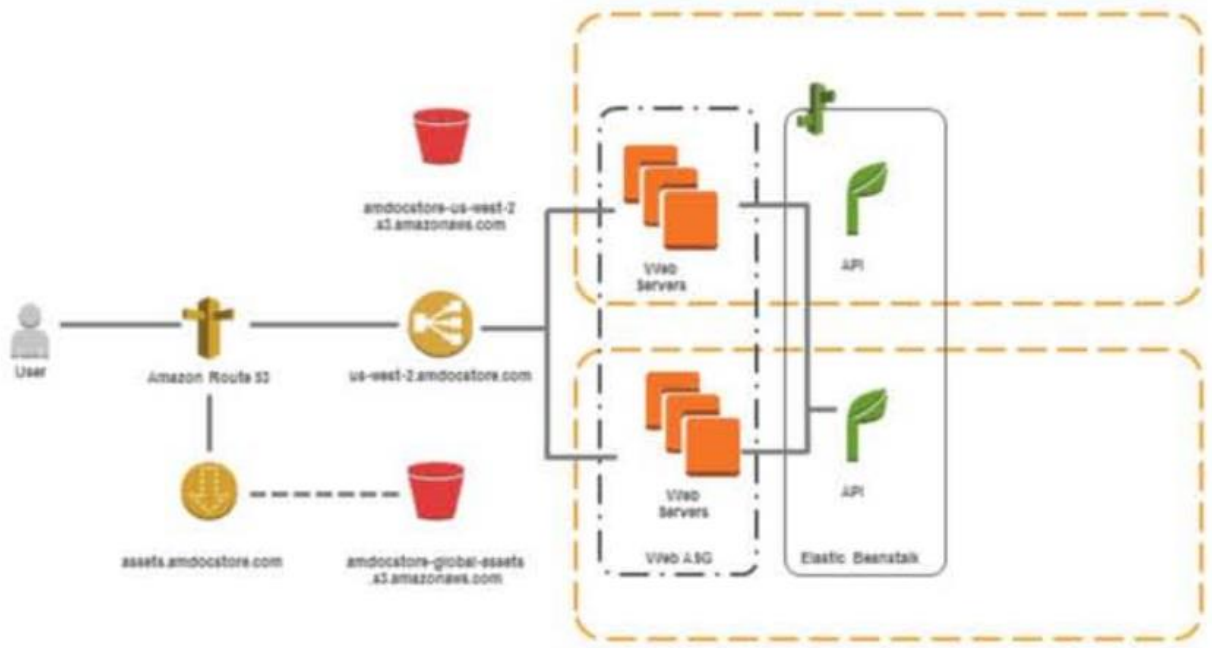
DocStore



amazon web services

Let's see what happens when a user wants to **upload a PDF document**

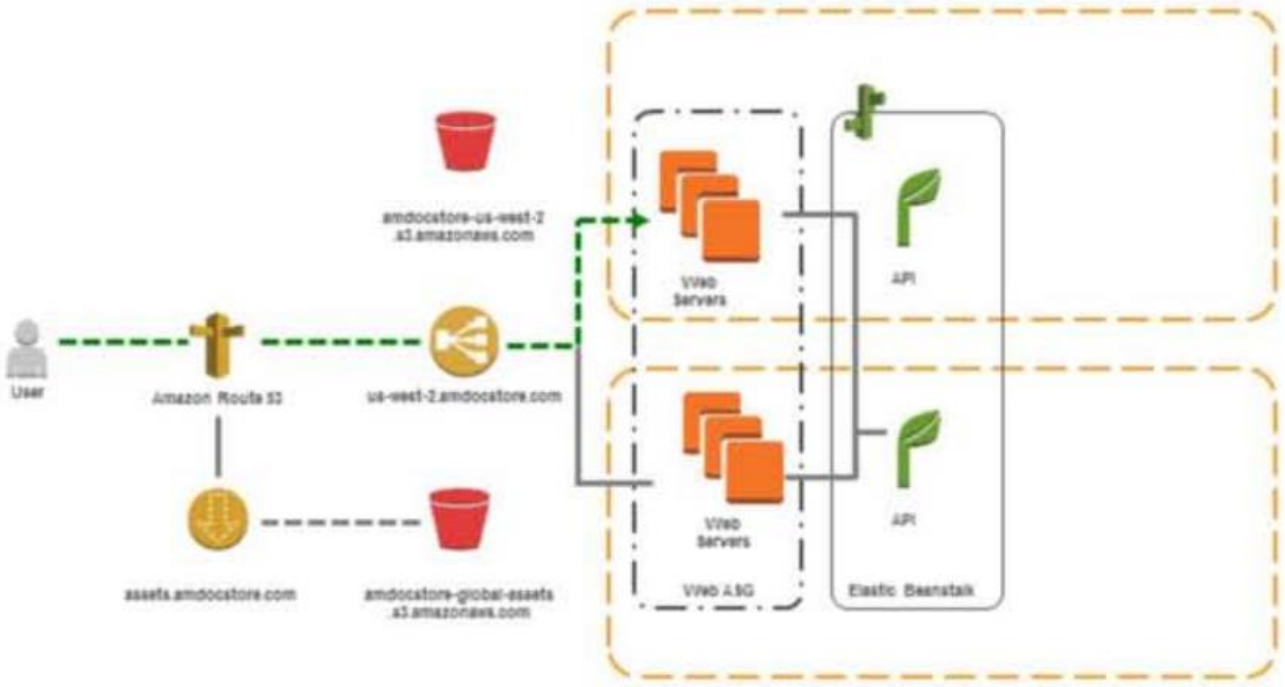
DocStore



amazon web services

Upload Document: An authenticated user requests the upload document page from the web servers

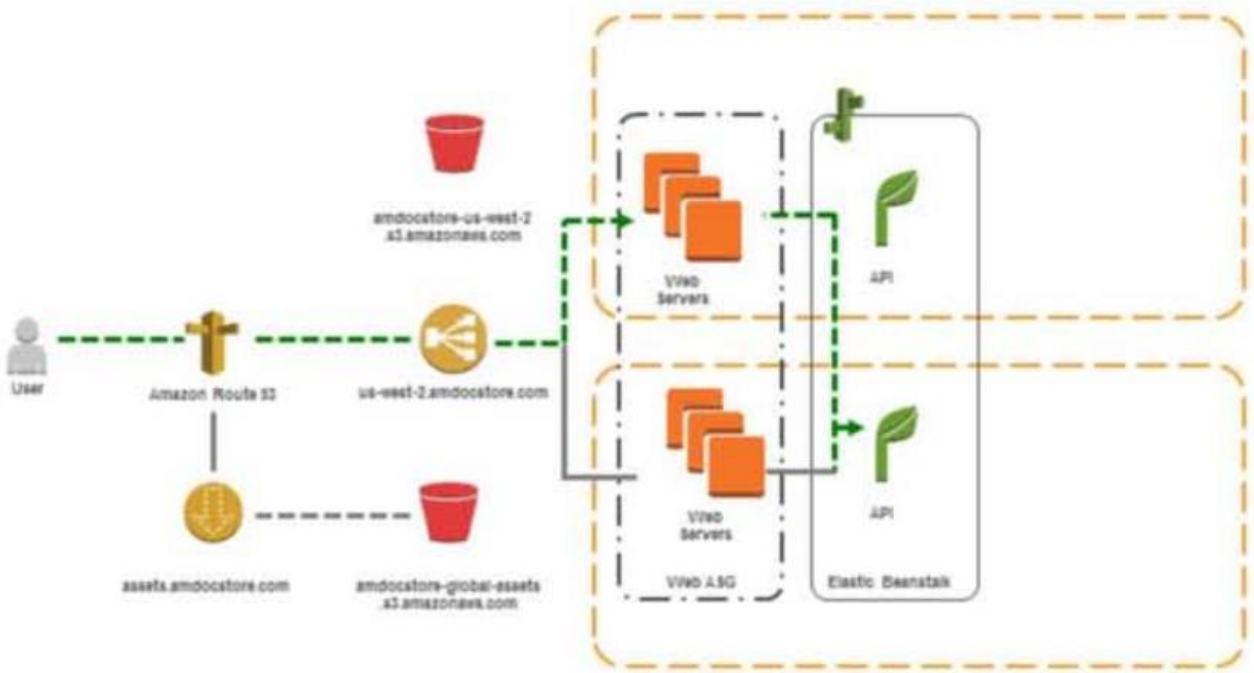
DocStore



amazon web services

Upload Document: The web servers request a signed HTLM form from the API and return it to the user's browser

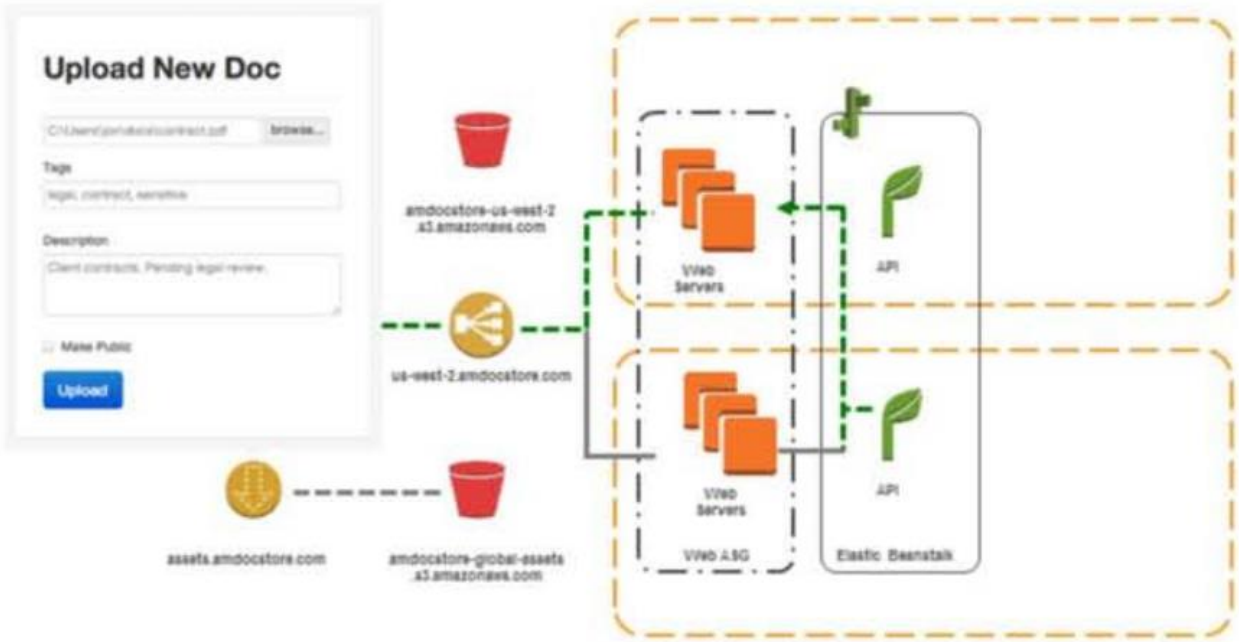
DocStore



amazon web services

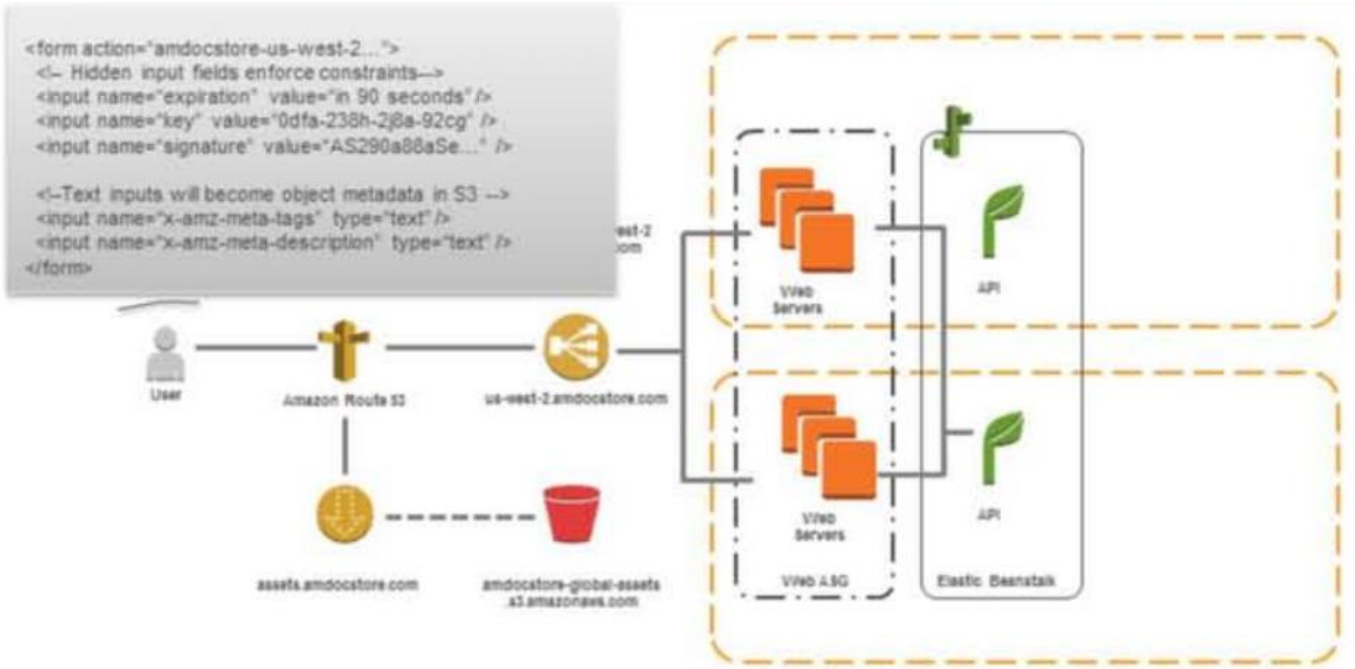
Upload Document: The signed form is delivered to the user's browser

DocStore



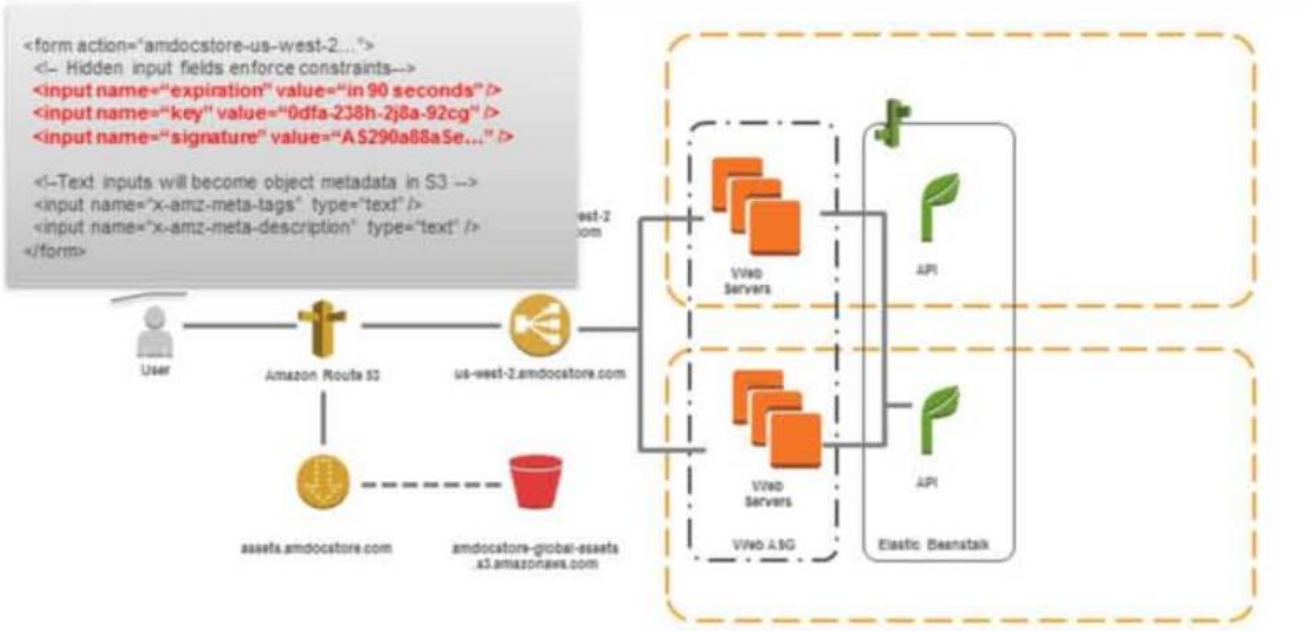
Upload Document: The signed form allows the user to upload his document **directly to S3**

DocStore



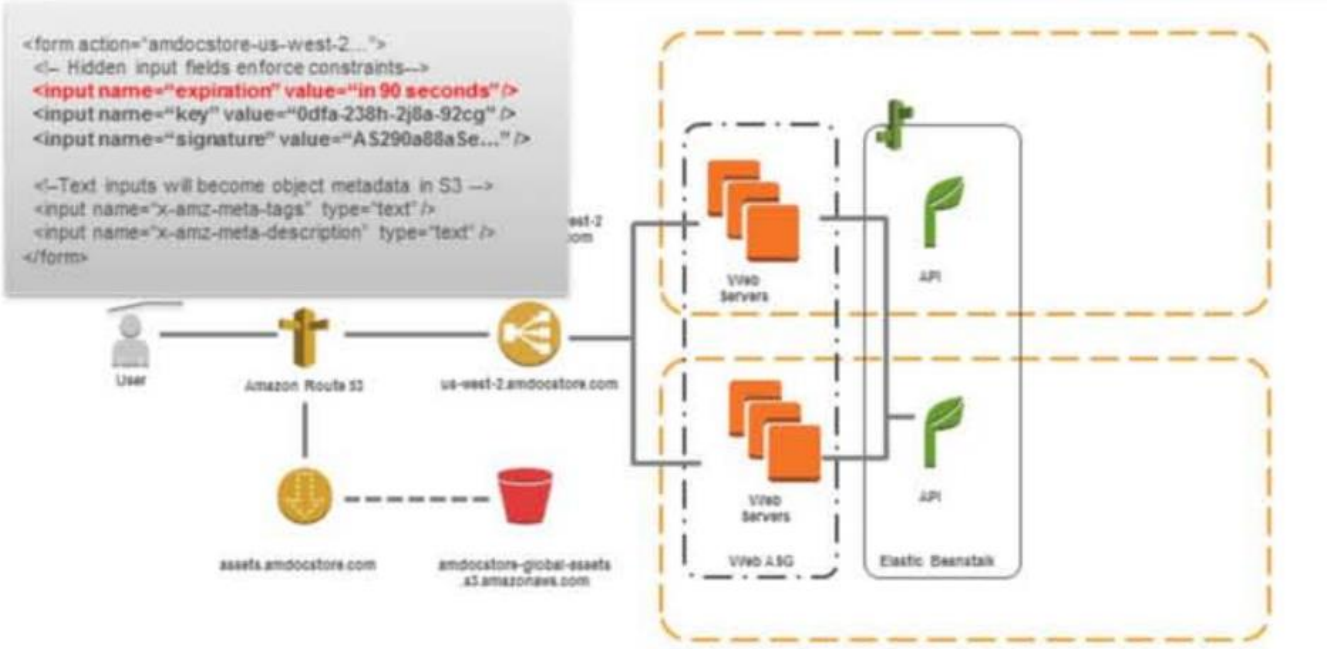
Upload Document: The form includes some **hidden inputs that constrain** what the user can upload.

DocStore



Upload Document: The form includes some **hidden inputs that constrain** what the user can upload. **The expiration** indicates for how long the form is valid

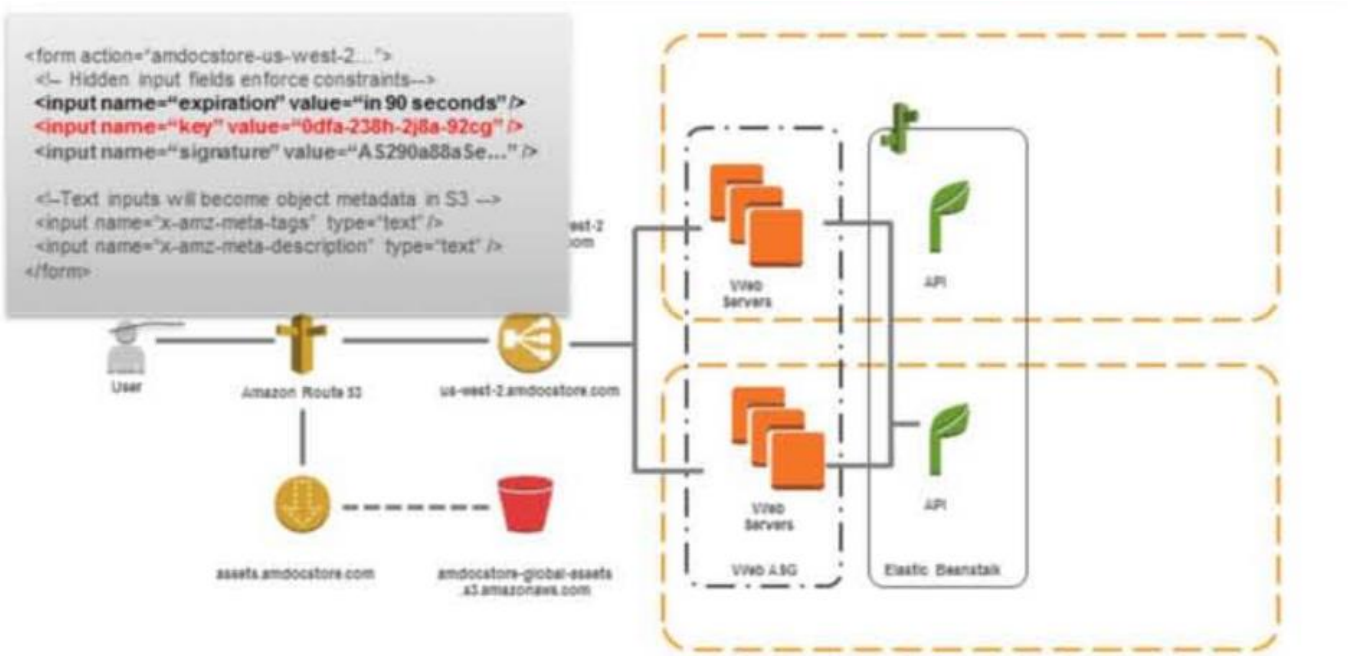
DocStore



amazon web services

DocStore

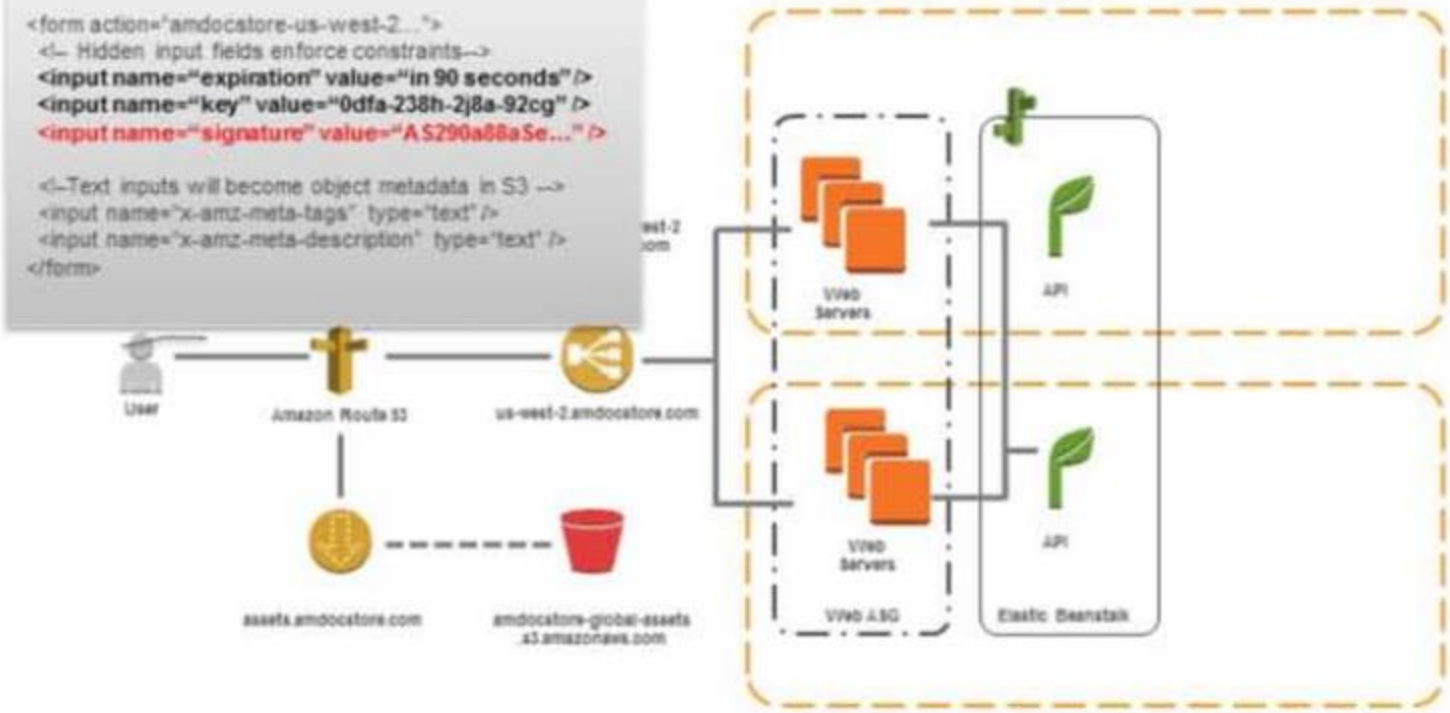
Upload Document: The form includes some **hidden inputs that constrain** what the user can upload. **The key** allows us to specify what the object will be called in S3 (thus avoiding naming conflicts)



Upload Document: The form includes some hidden inputs that constrain what the user can upload. The signature ensures the upload will be rejected if the form is tampered with

DOCSTORE

```
<form action="amdocstore-us-west-2...">
  <!-- Hidden input fields enforce constraints-->
  <input name="expiration" value="in 90 seconds" />
  <input name="key" value="0dfa-238h-2j8a-92cg" />
  <input name="signature" value="AS290a88aSe..." />

  <!--Text inputs will become object metadata in S3 -->
  <input name="x-amz-meta-tags" type="text" />
  <input name="x-amz-meta-description" type="text" />
</form>
```



amazon web services

Upload Document: User-editable text input fields prefixed with x-amz-meta-will be associated with the uploaded object as S3 object metadata

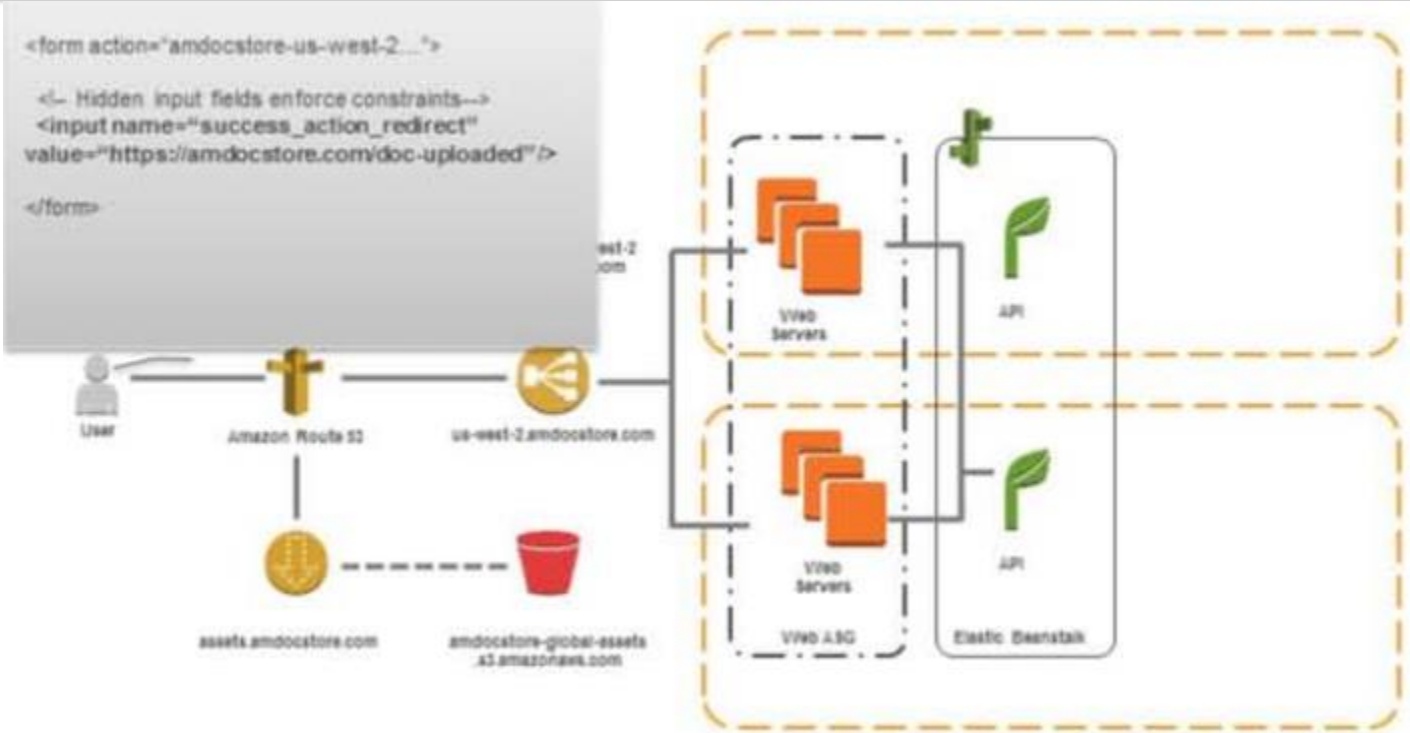
DOCSTORE

amazon web services

Upload Document: Finally, the success_action_redirect specified when the API generated the form tells S3 where to redirect the user's browser when the upload completes

DOCSTORE

```
<form action="amdocstore-us-west-2....">

  <!-- Hidden input fields enforce constraints-->
  <input name="success_action_redirect"
value="https://amdocstore.com/doc-uploaded" />

</form>
```

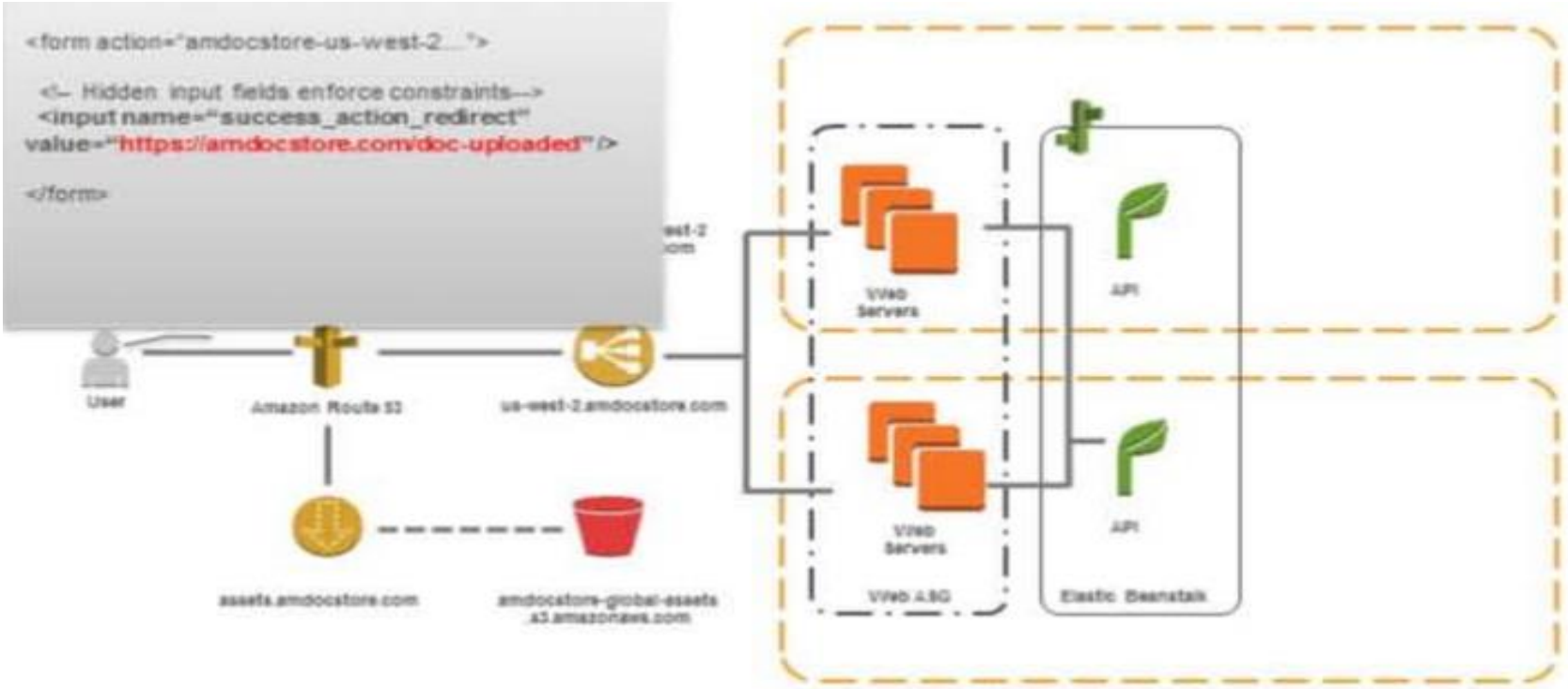


amazon web services

Upload Document: We send the user to a special page on the web servers that will capture this event

DOCSTORE

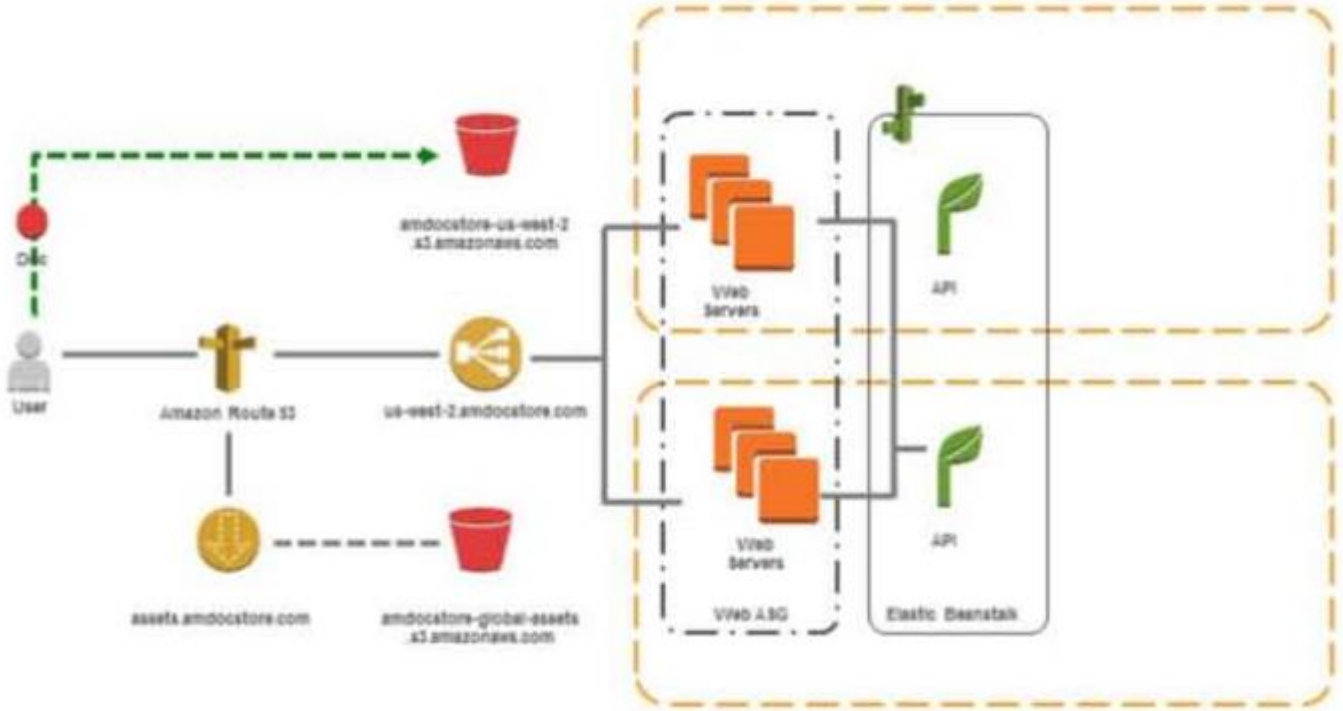
```
<form action="amdocstore-us-west-2...">

  <!-- Hidden input fields enforce constraints-->
  <input name="success_action_redirect"
value="https://amdocstore.com/doc-uploaded" />

</form>
```



amazon web services

Upload Document: While uploading the document to S3

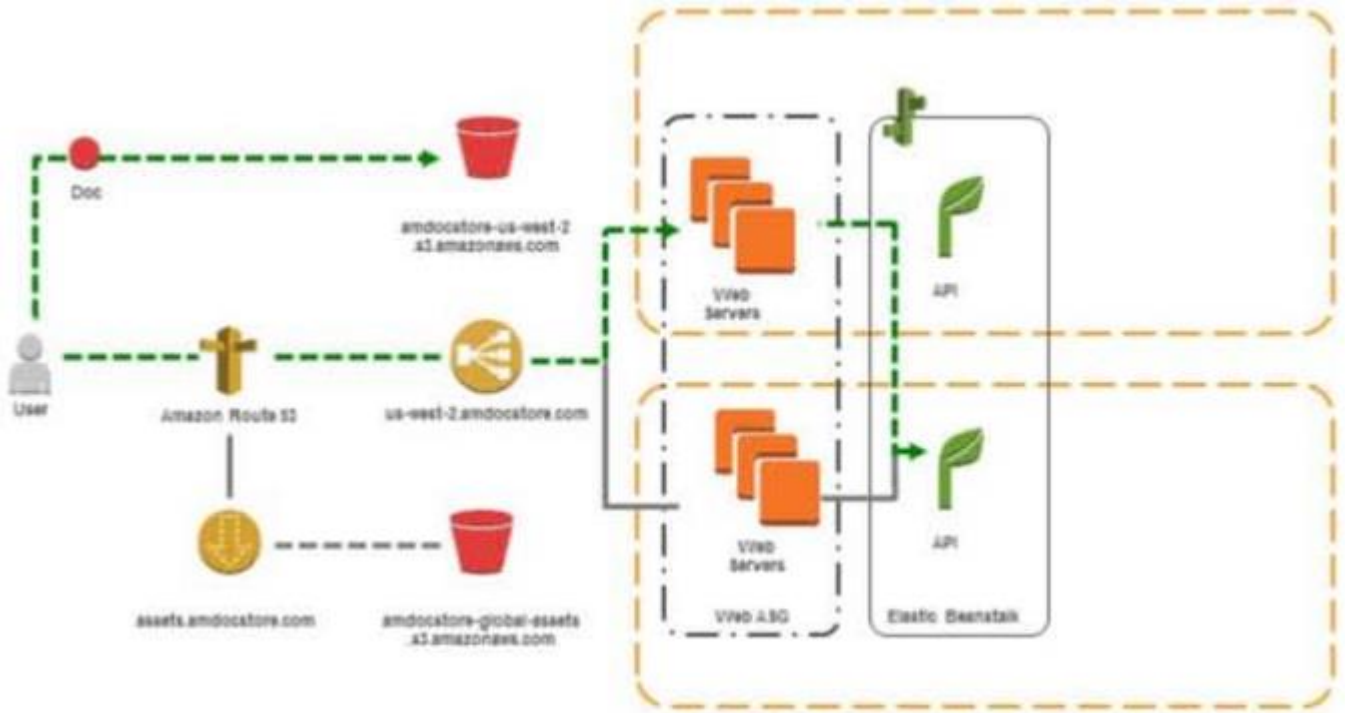
DOCSTORE



amazon web services

Upload Document: While uploading the document to S3, we asynchronously notify the web servers that the upload has begun

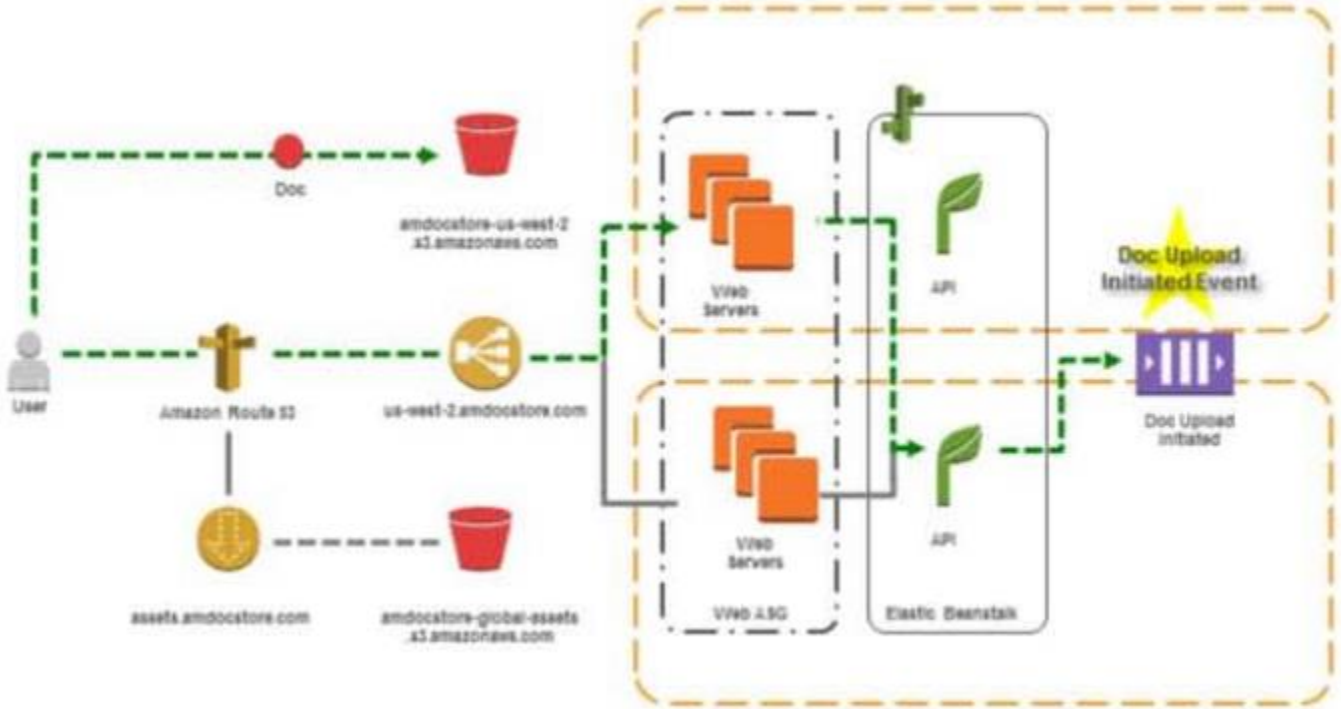
DOCSTORE



amazon web services

Upload Document: The API places a message in a queue indicating as much. Other components of DocStore may be interested in this event later

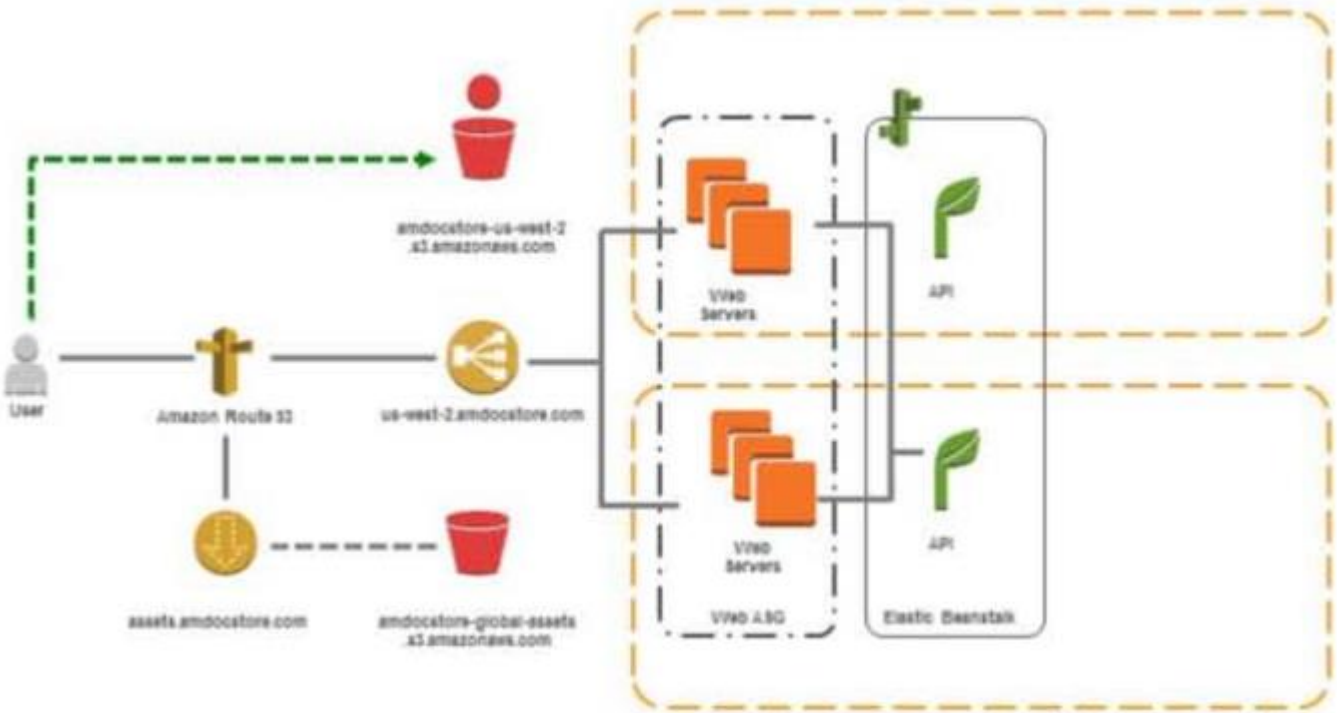
DOCSTORE



amazon web services

Upload Document: When the upload completes

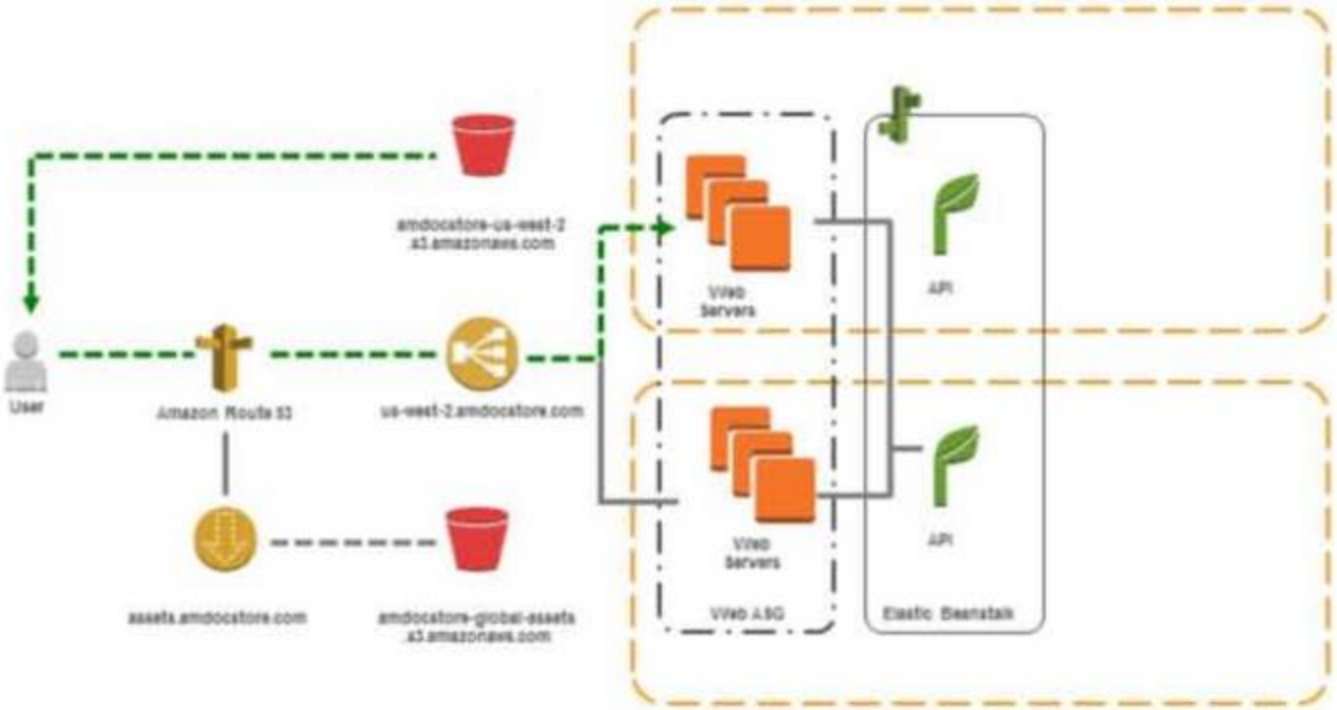
DOCSTORE



amazon web services

Upload Document: When the upload completes S3 redirects the users to the URL we specified earlier

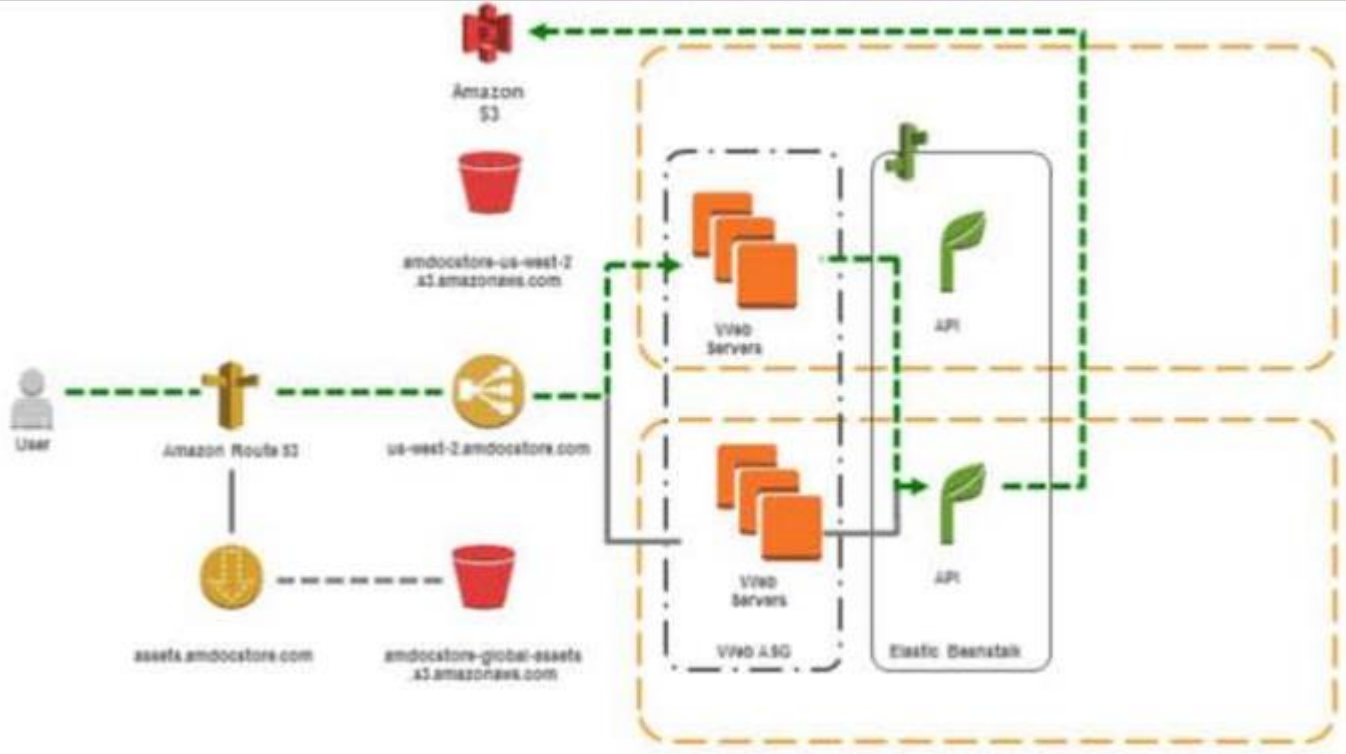
DOCSTORE



amazon web services

Upload Document: The API retrieves metadata about the object from the S3 API

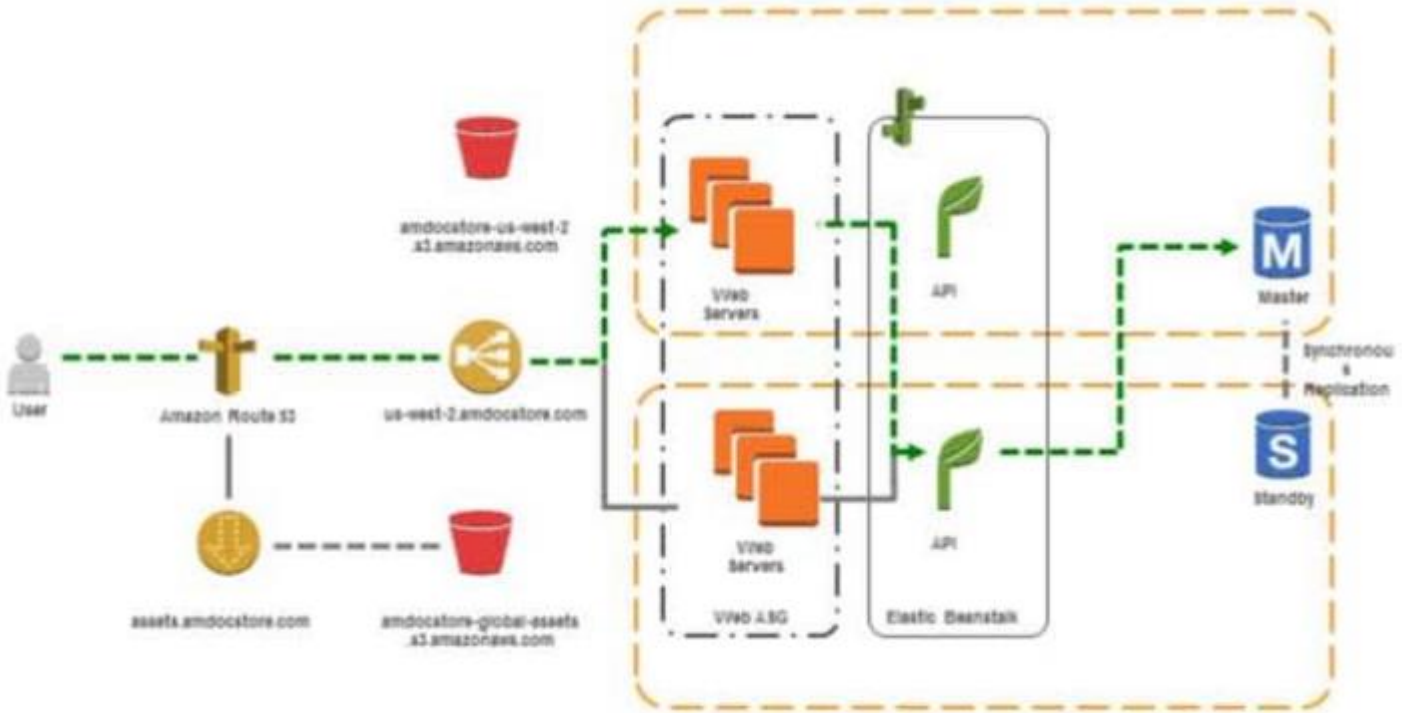
DOCSTORE



amazon web services

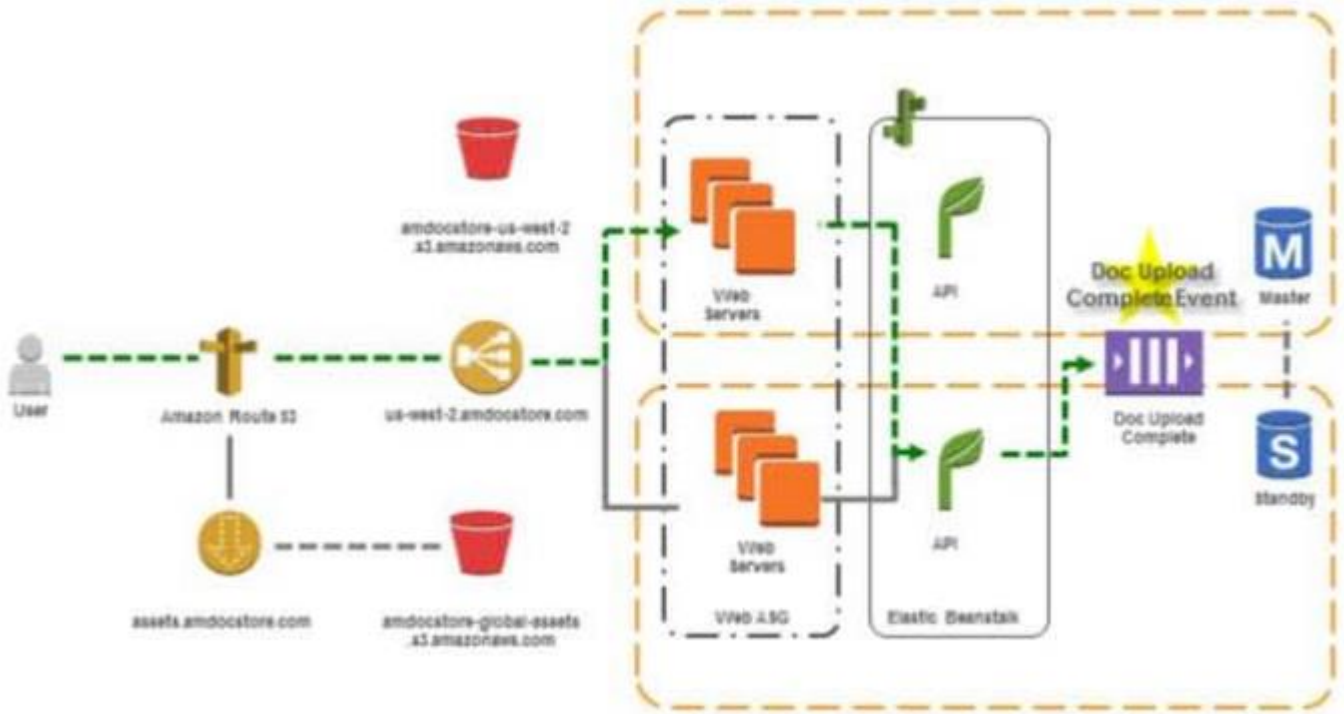
Upload Document: The Object metadata is stored in an RDS instance running in Multi-AZ mode for high availability

DOCSTORE



Upload Document: Finally, the API places a message in a queue indicating that the document upload was complete, and includes the ID of the newly uploaded object in S3

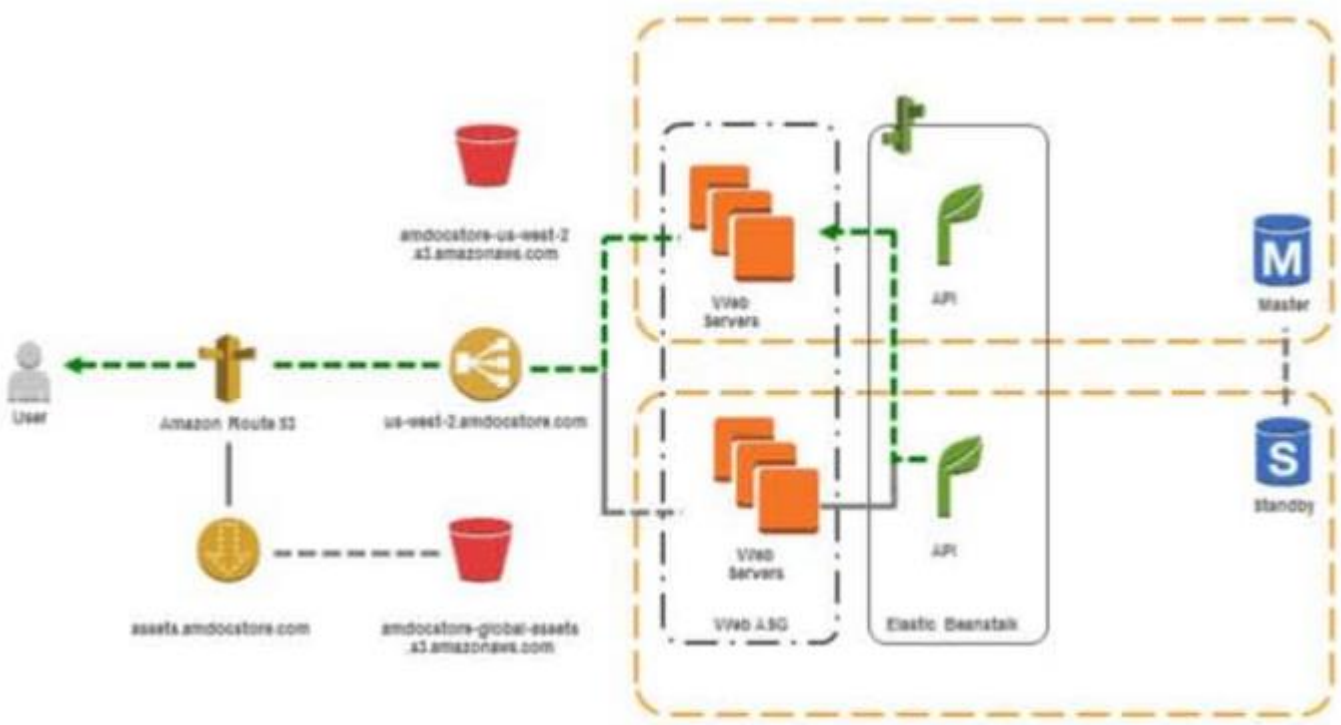
DOCSTORE



amazon web services

Upload Document: The user is redirected to a list of his documents

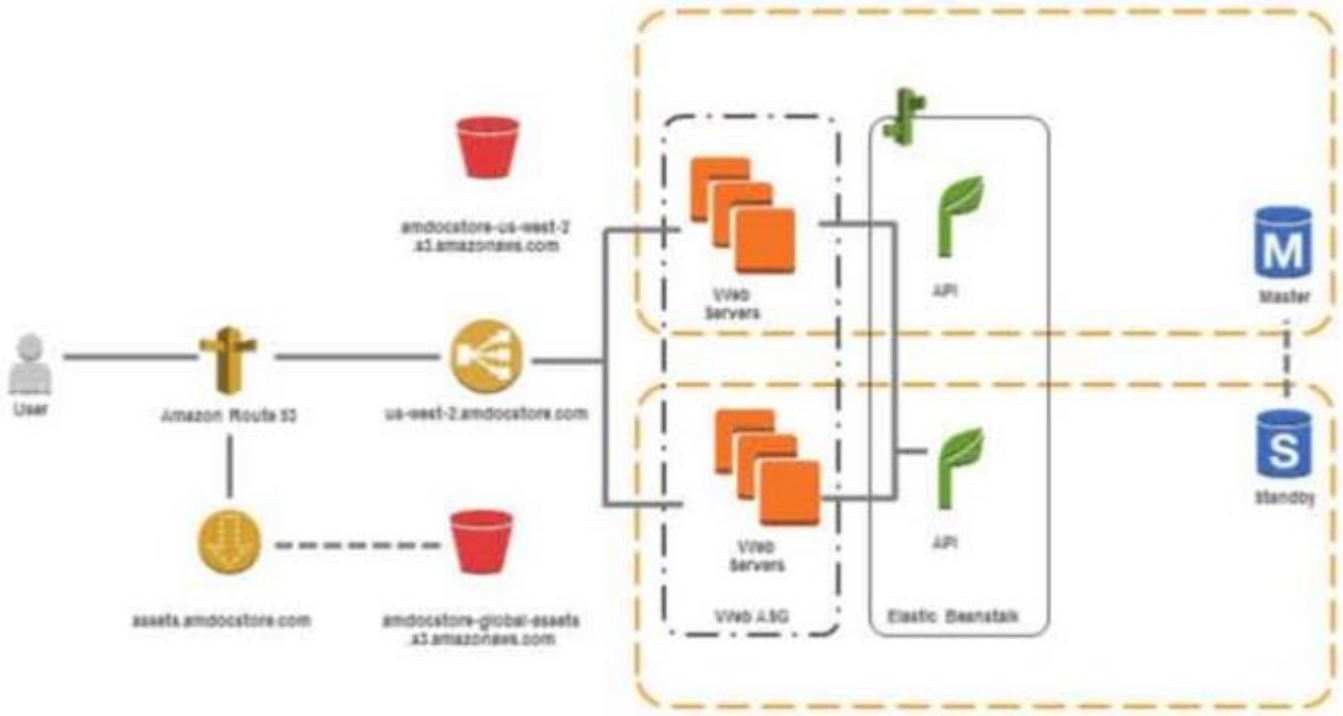
DOCSTORE



amazon web services

Let's see what happens when a user wants to view and download his documents

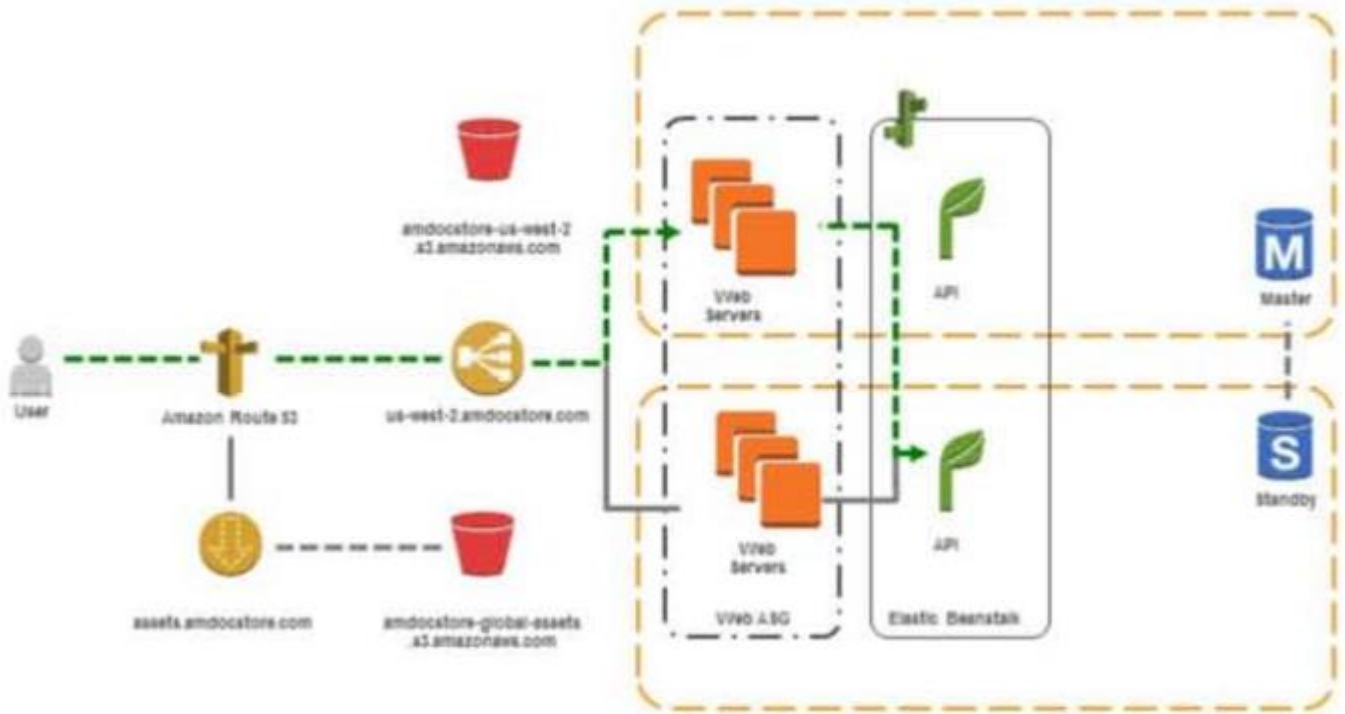
DOCSTORE



amazon web services

View and Download Docs: An authenticated user requests the web site URL to view his docs

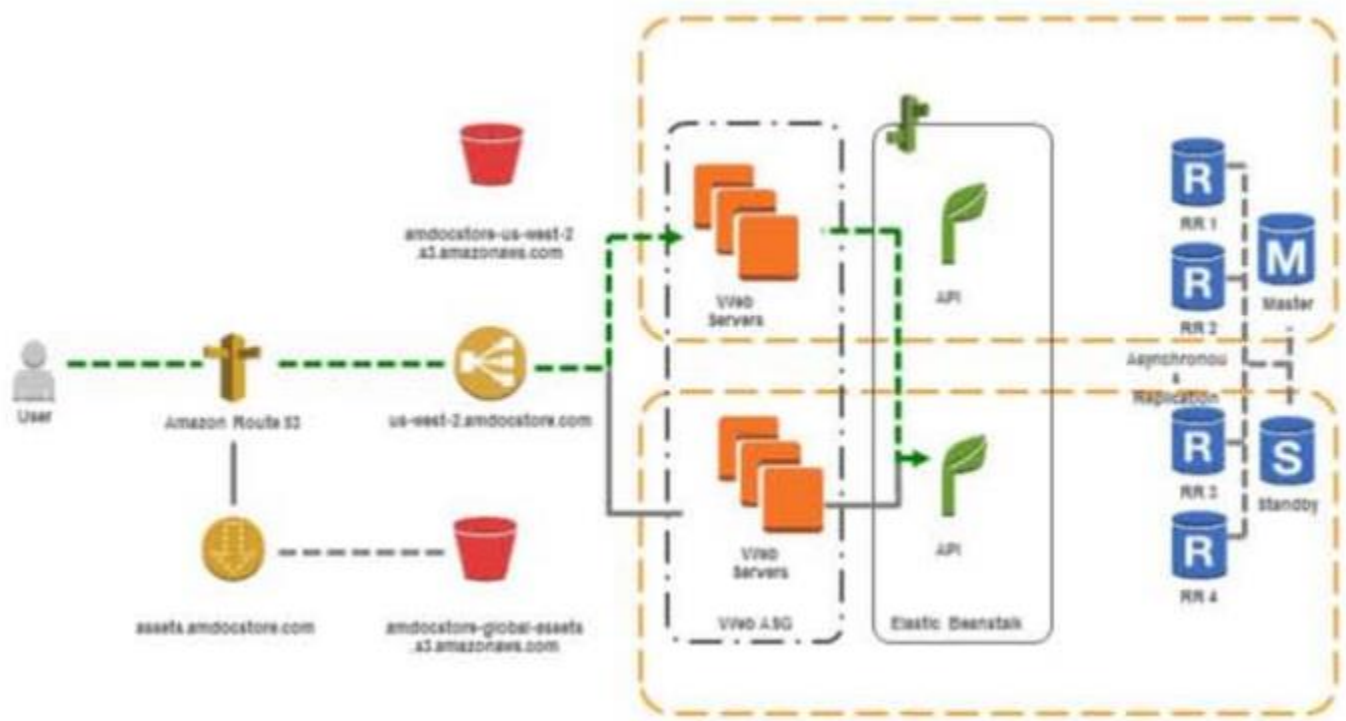
DOCSTORE



amazon web services

View and Download Docs: To increase read performance of the relational database, multiple RDS Read Replicas are provisioned. Read Replicas are suitable for read workloads, like the api selecting a list of documents owned by a user

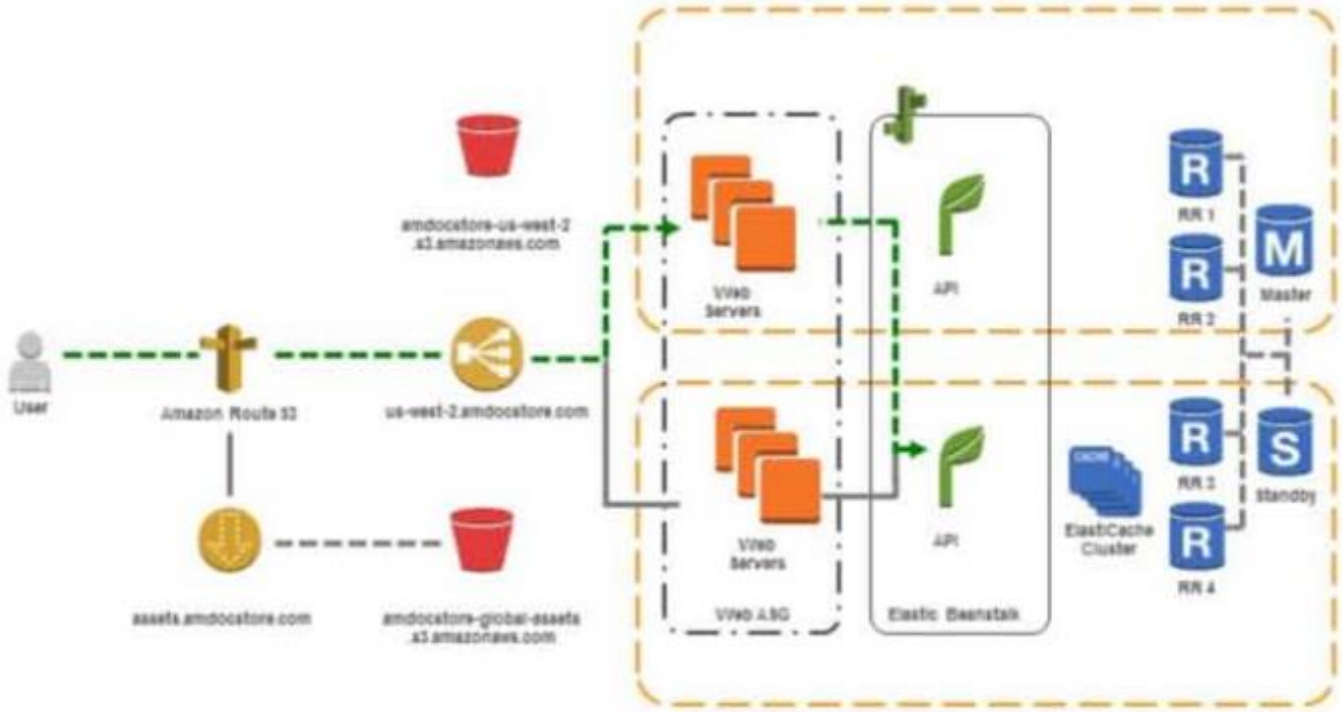
DOCSTORE



amazon web services

View and Download Docs: ElasticCache Cluster in front of the Read Replicas would provide additional performance when listing the user's documents

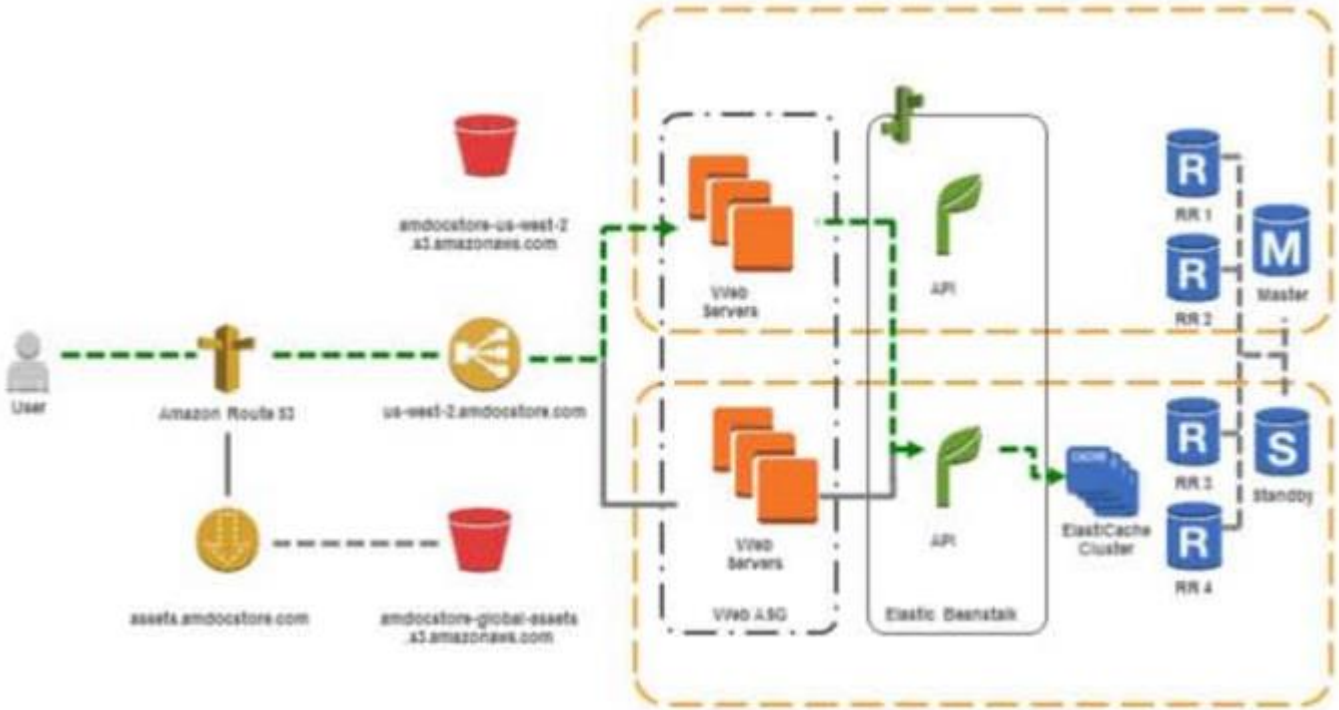
DOCSTORE



amazon web services

View and Download Docs: The API first checks the cache for a recent list of the user's documents

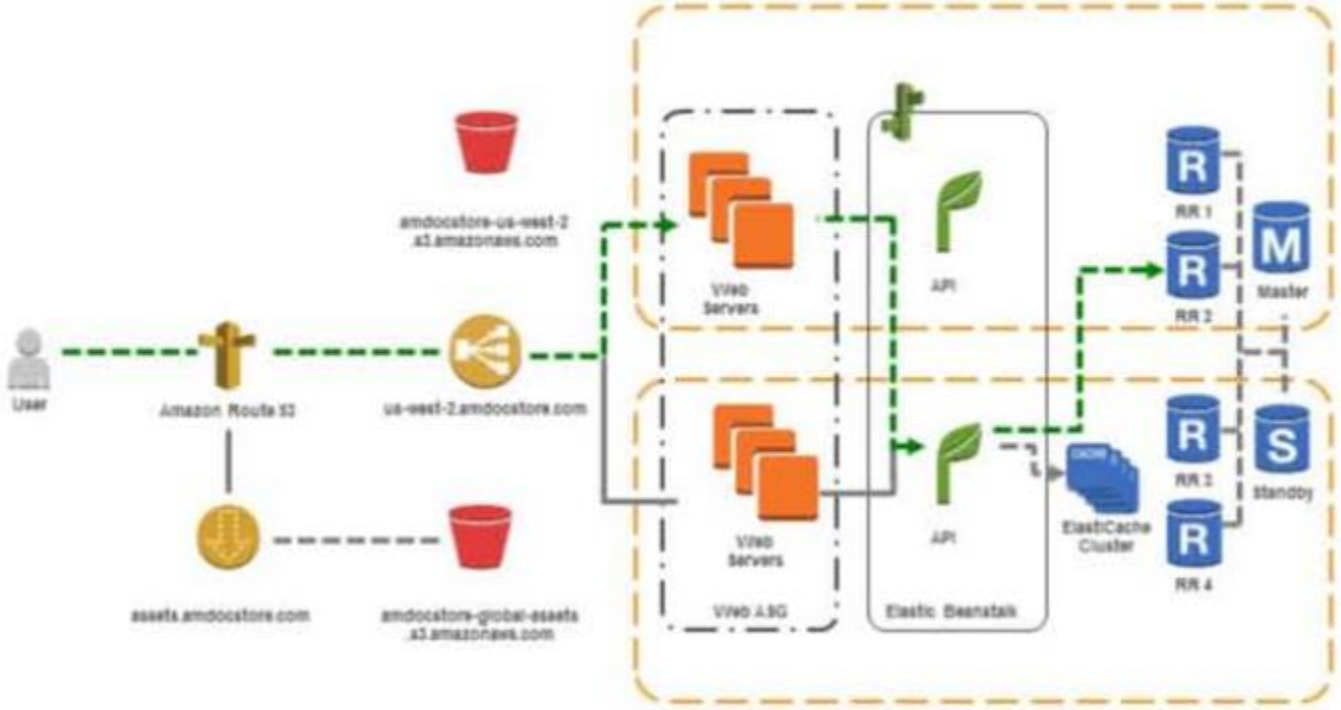
DOCSTORE



amazon web services

View and Download Docs: If no cached data is available, the API chooses one of the Read Replicas and queries for the list of documents

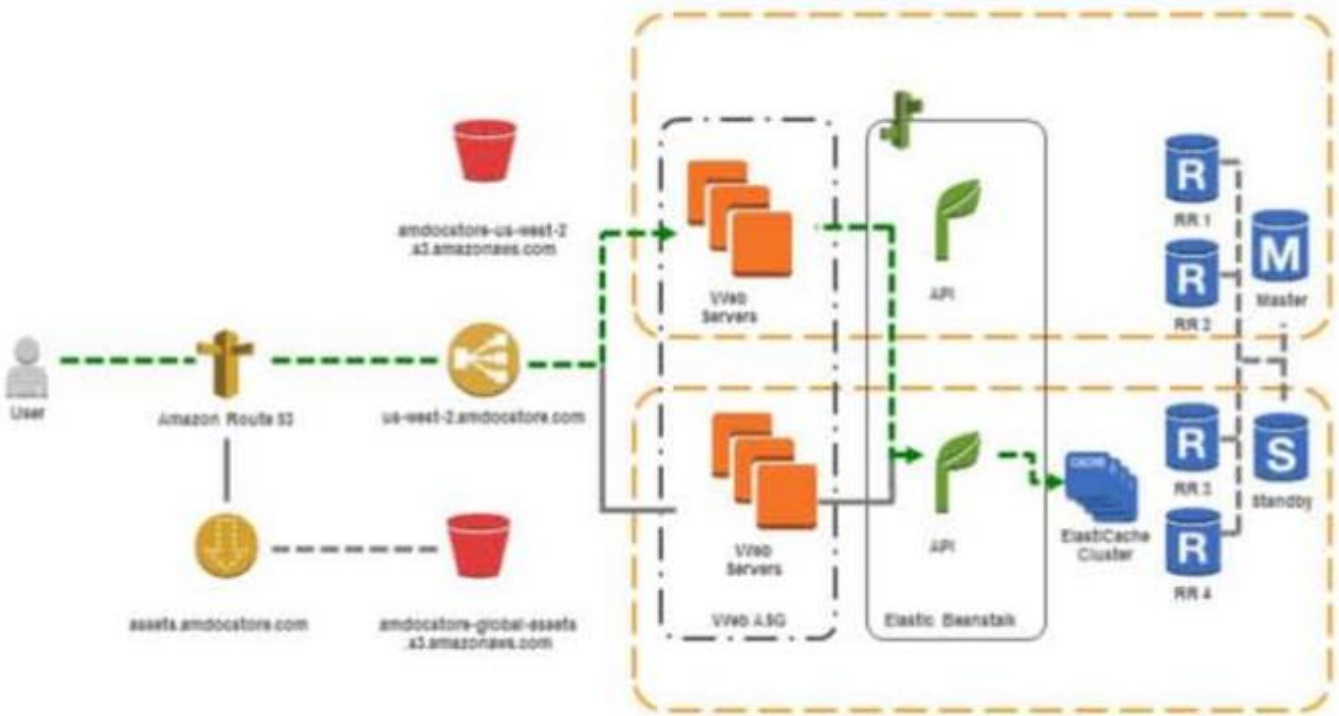
DOCSTORE



amazon web services

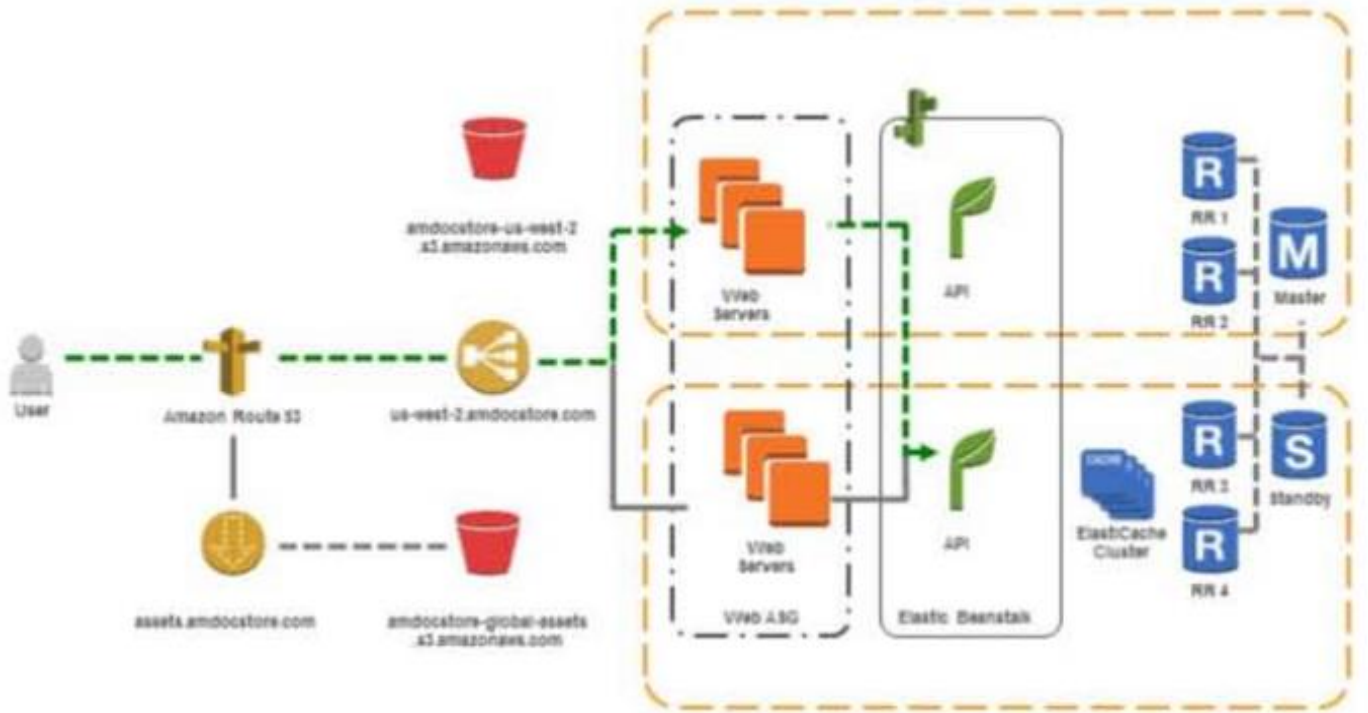
View and Download Docs: The result of the query is stored in the cache for future access

DOCSTORE



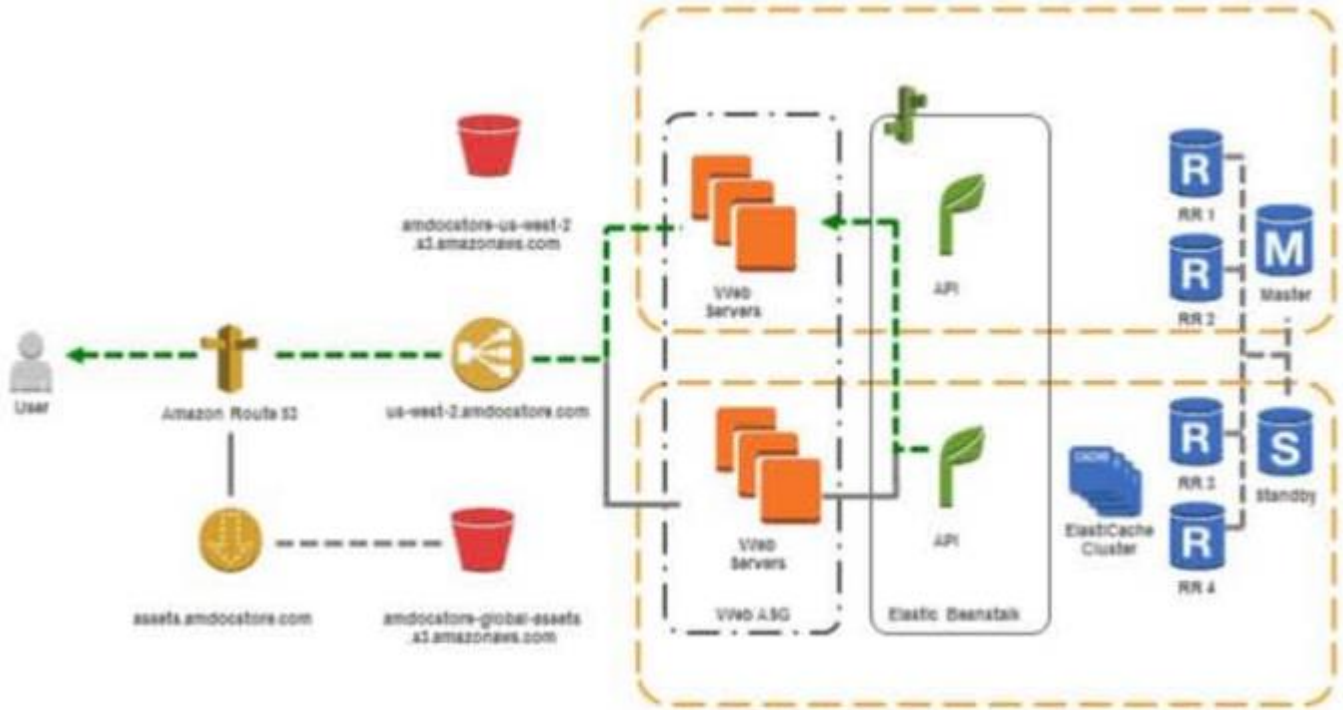
View and Download Docs: For each document, the API constructs a secure, auto-expiring, signed URL that will allow the user to download the document directly from S3

DOCSTORE



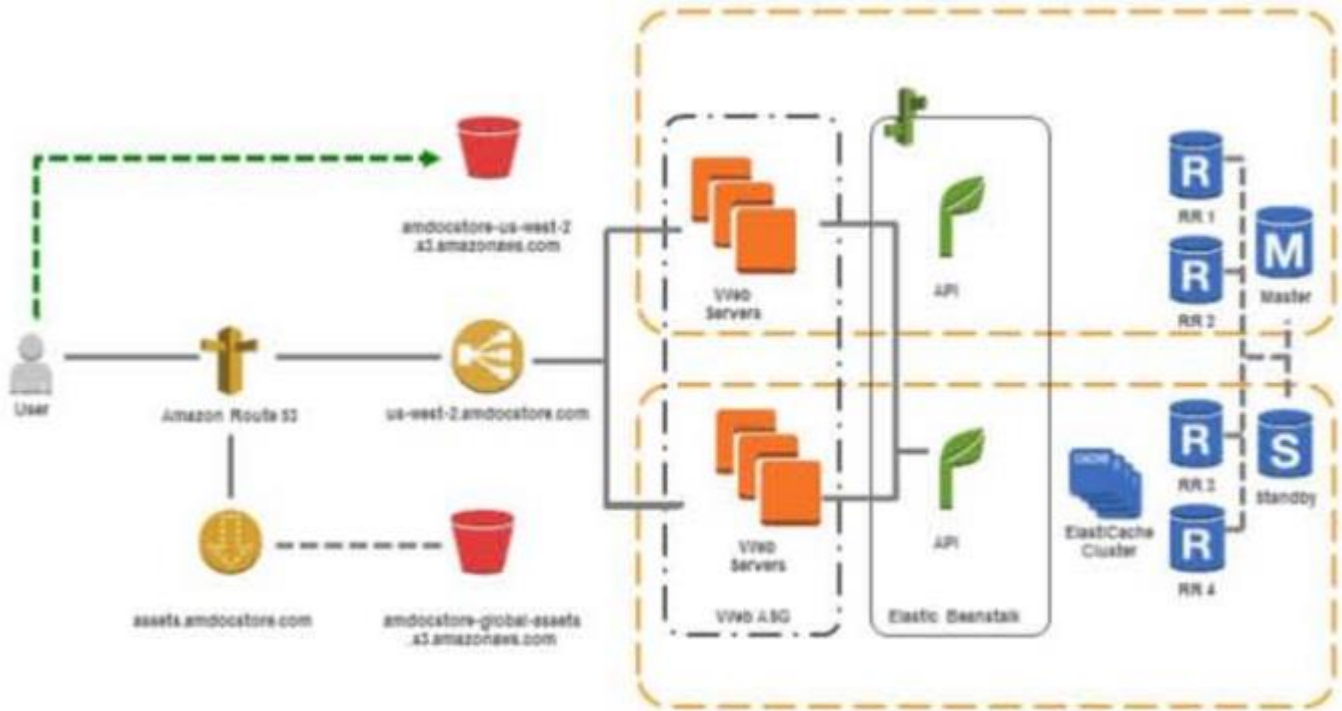
View and Download Docs: The list of documents- and links for direct, secure download-are delivered to the user's browser

DOCSTORE



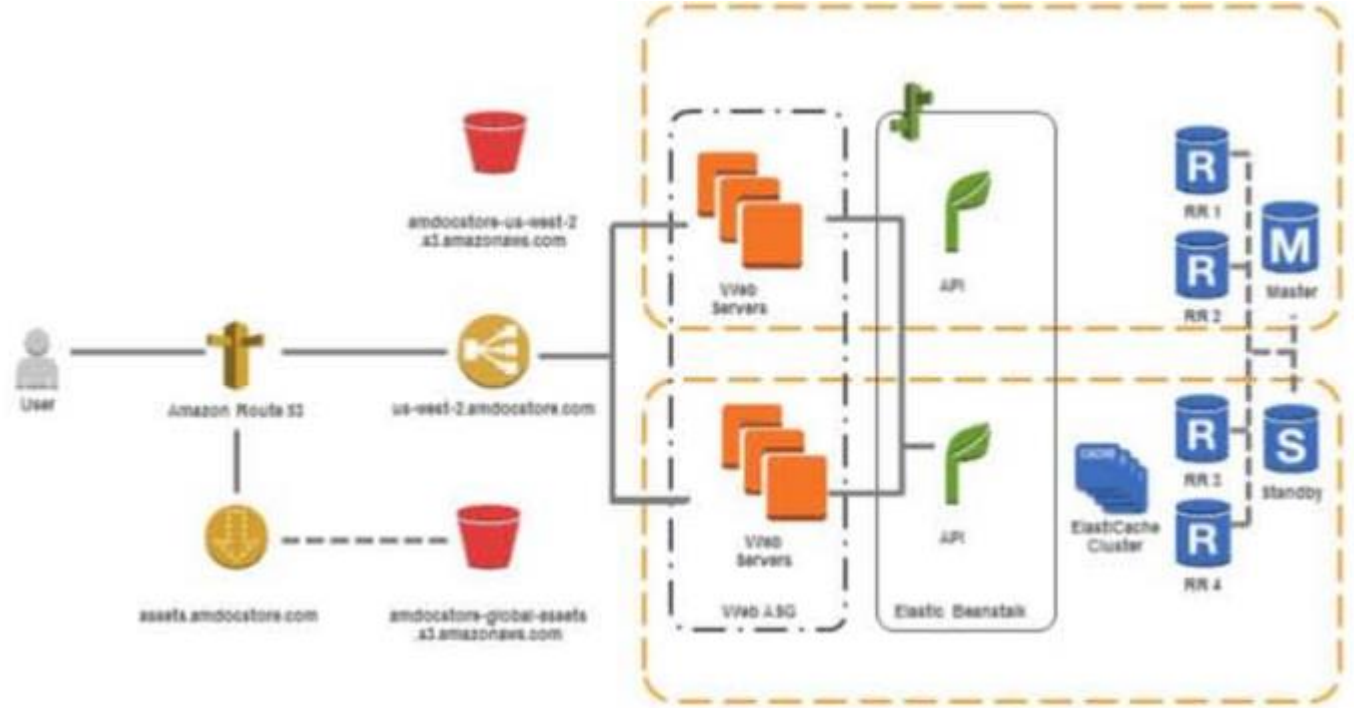
View and Download Docs: The user may download documents directly from S3

DOCSTORE



View and Download Docs: Let's look at options for security

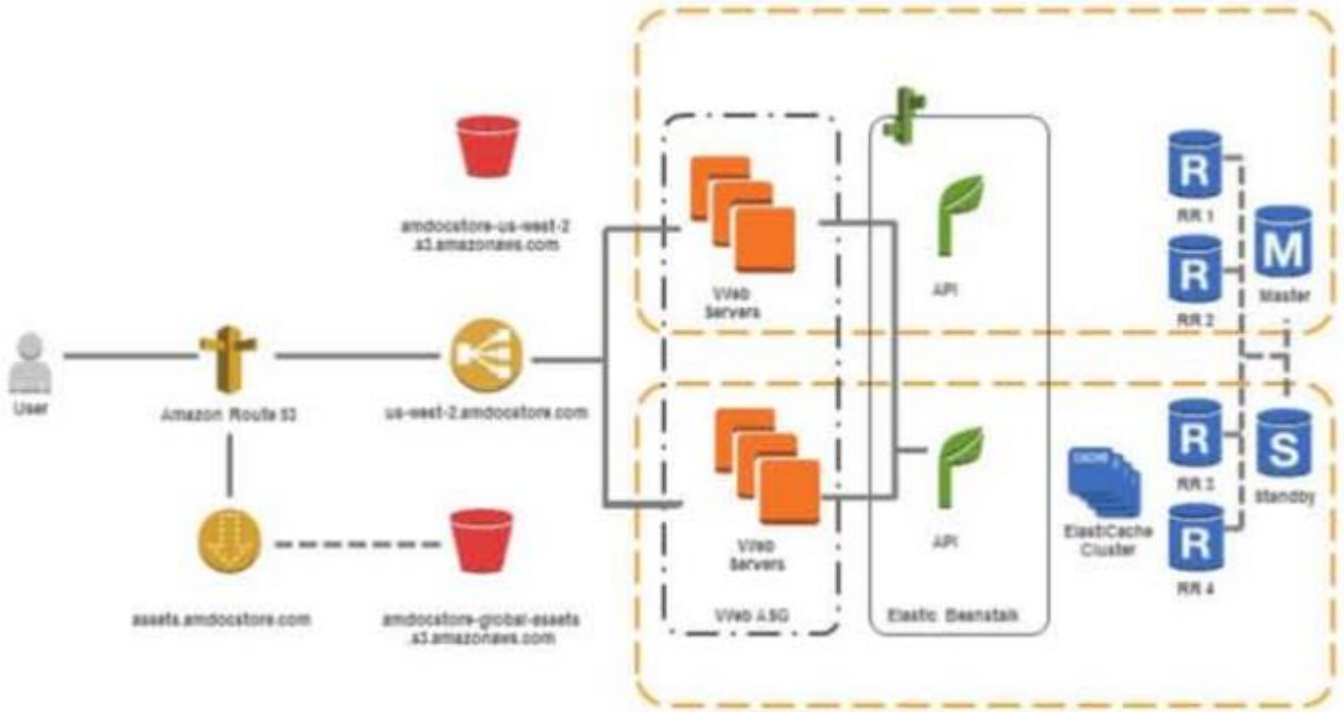
DOCSTORE



amazon web services

View and Download Docs: Where can security measures and controls be implemented in this architecture?

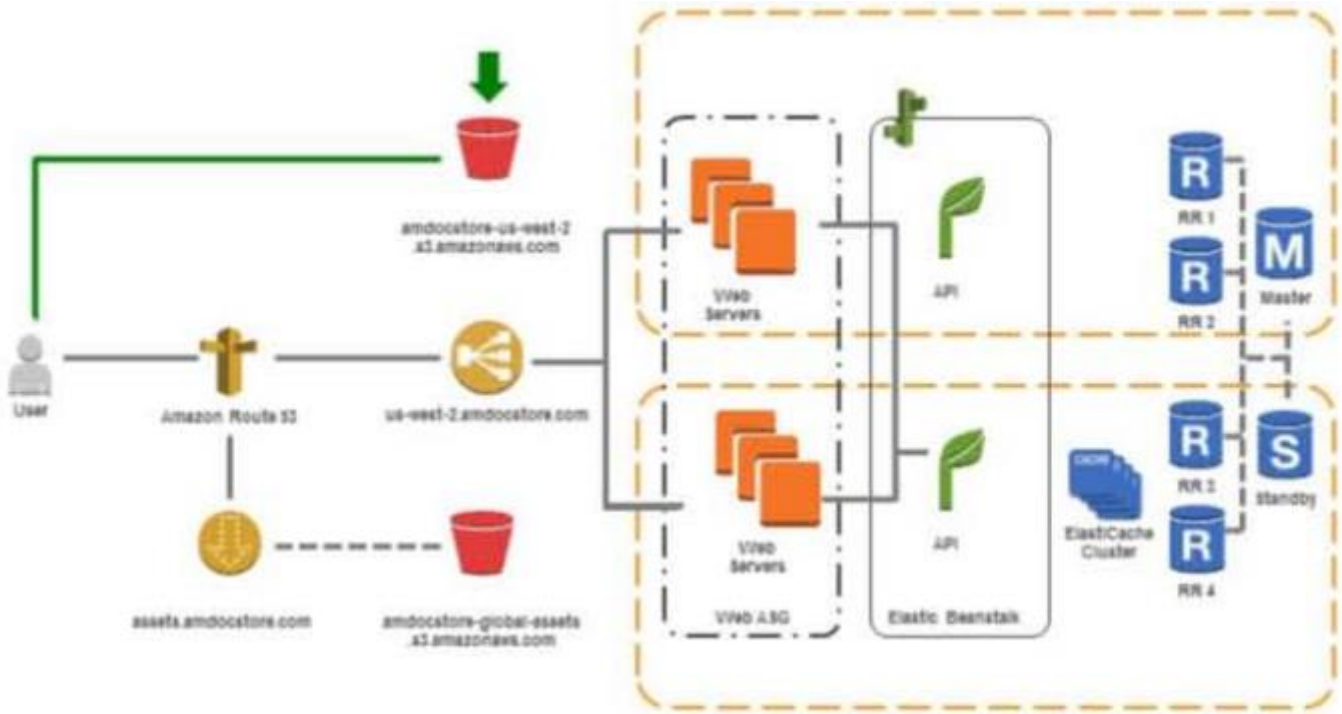
DOCSTORE



amazon web services

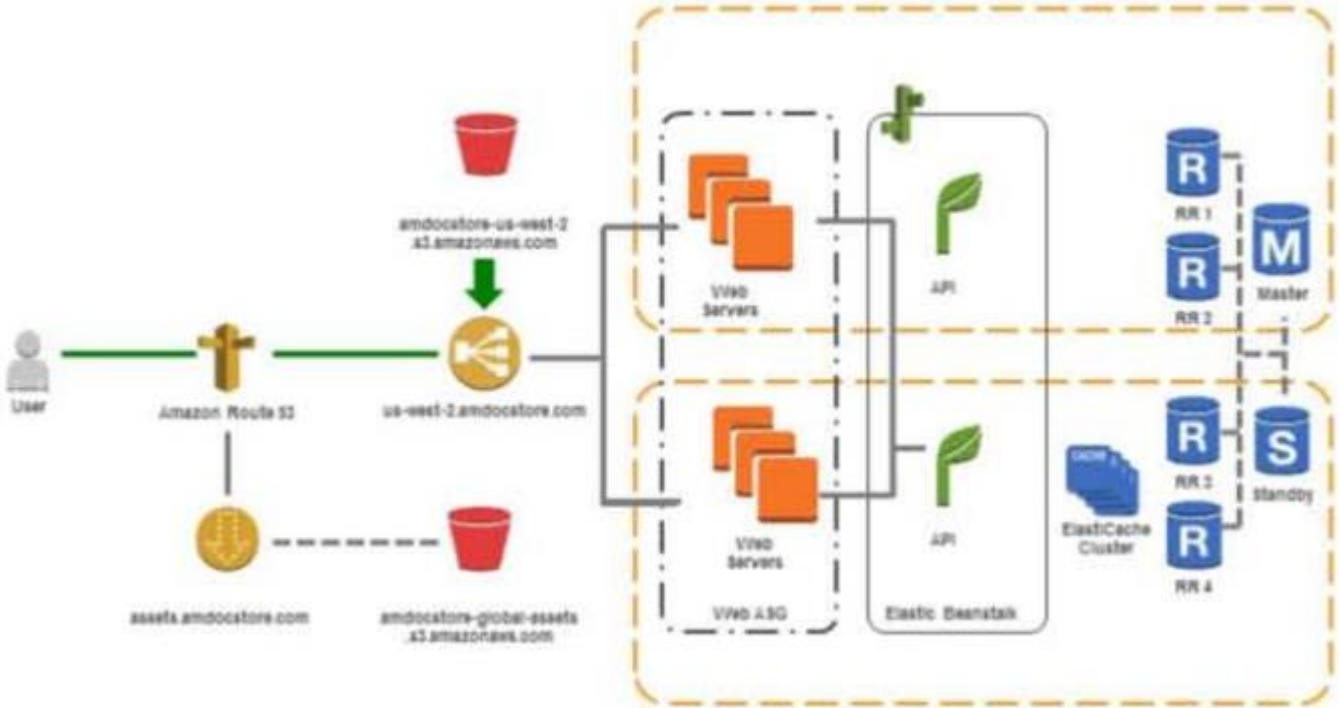
View and Download Docs: User downloads documents directly from S3 using a secure, signed URL and over SSL

DOCSTORE



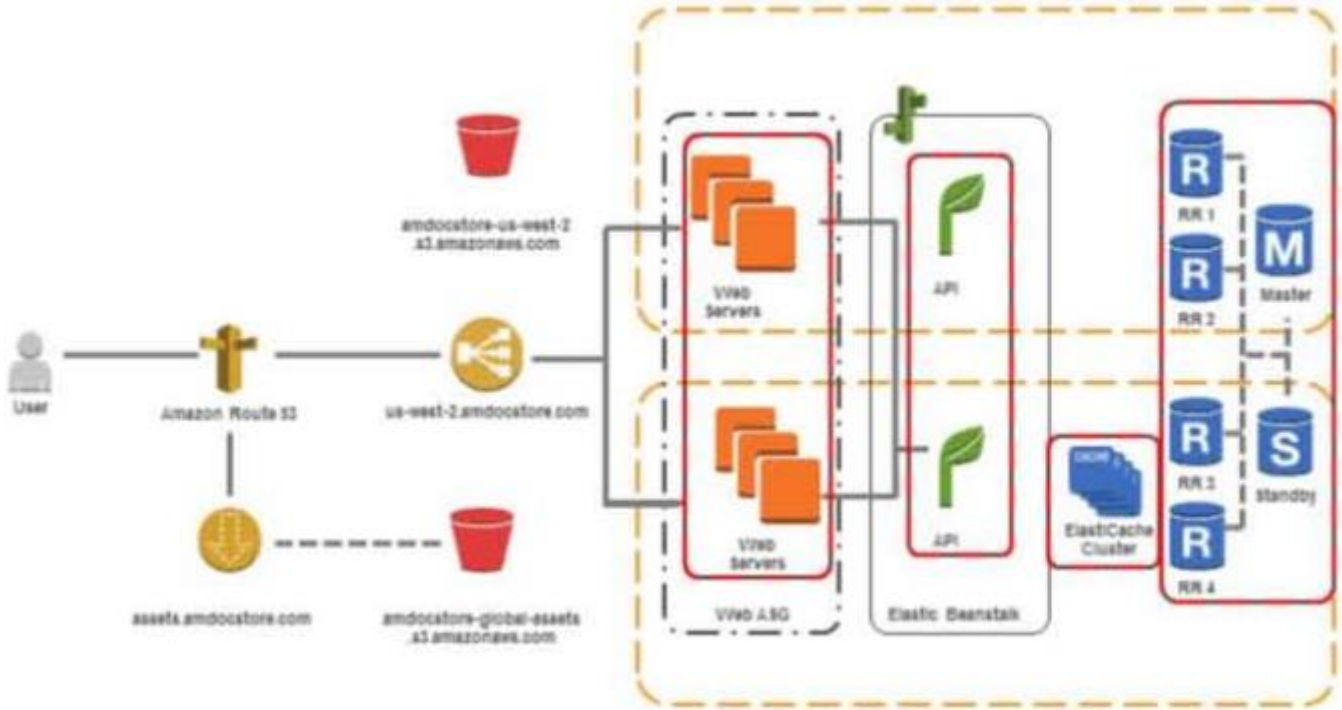
View and Download Docs: SSL connection between user and ELB

DOCSTORE



View and Download Docs: Security Groups restrict inbound traffic

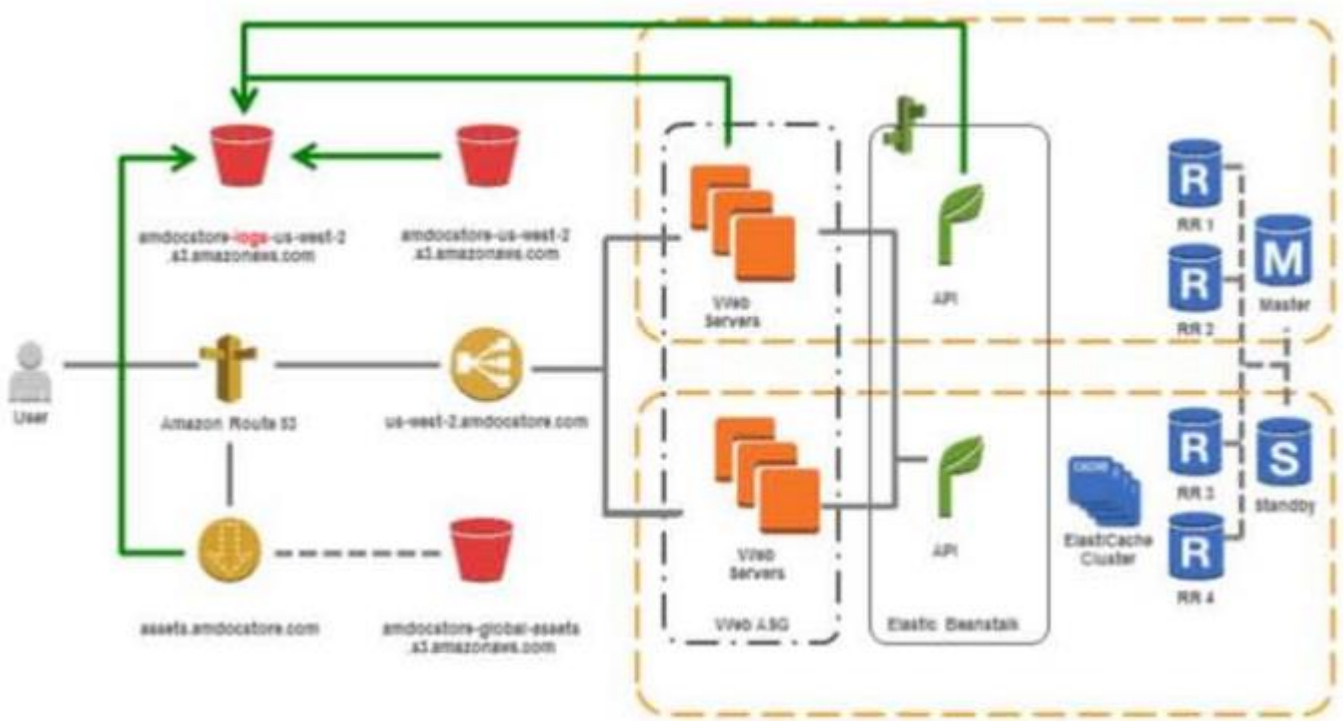
DOCSTORE



amazon web services

View and Download Docs: Access logs from CloudFront, EC2, and S3 are pushed to S3

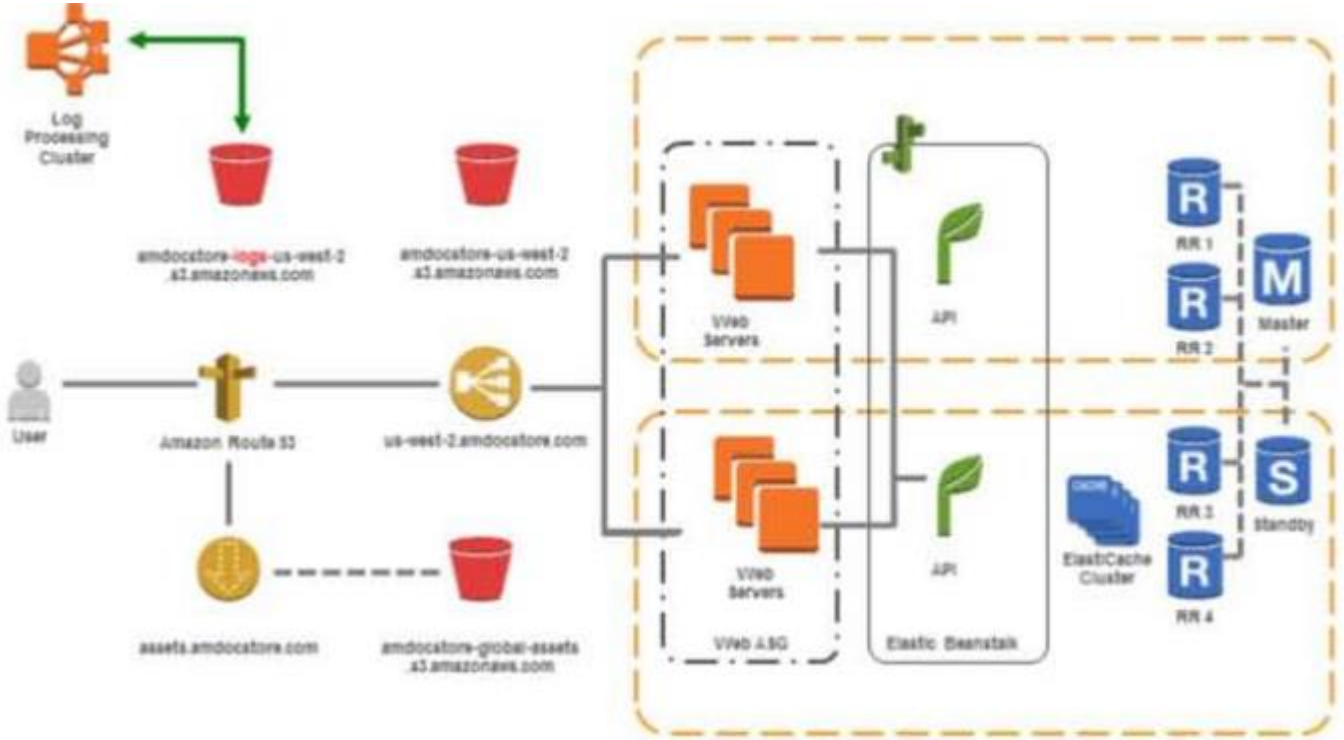
DOCSTORE



amazon web services

View and Download Docs: Logs are processed on a schedule using Elastic MapReduce cluster

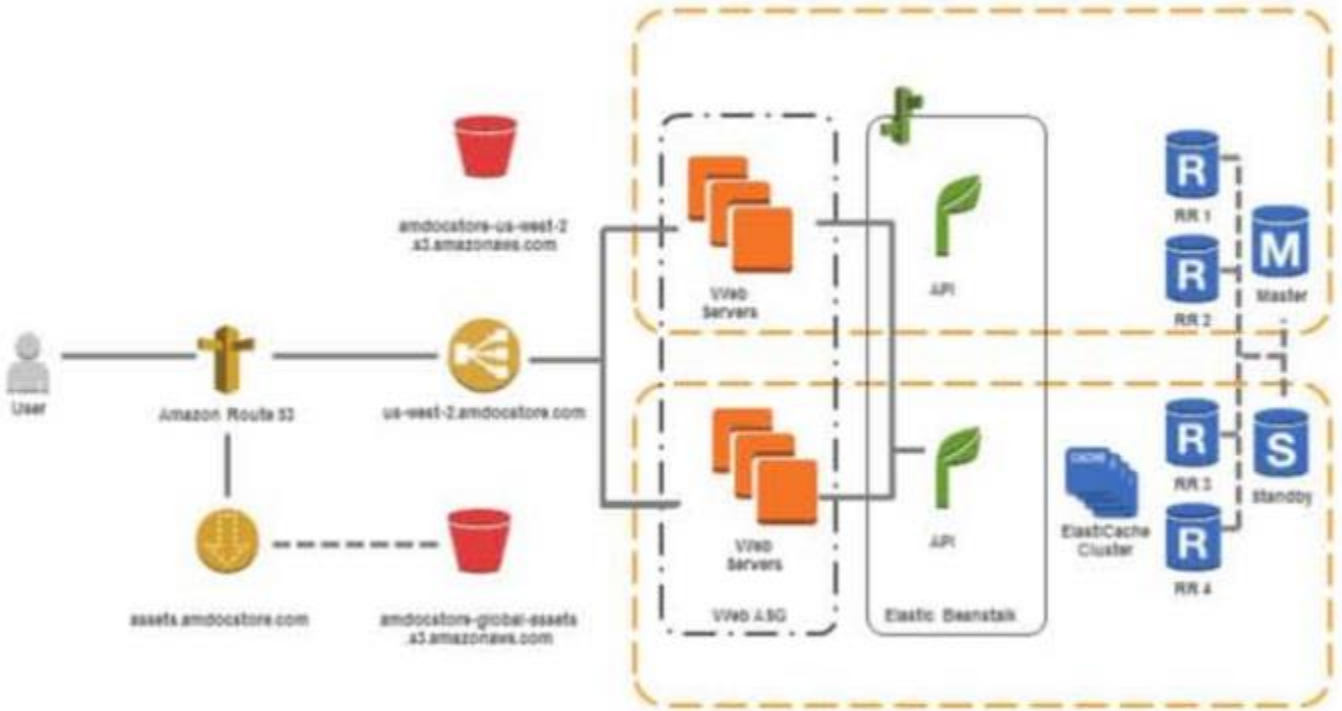
DOCSTORE



amazon web services

Let's focus on the networking environment

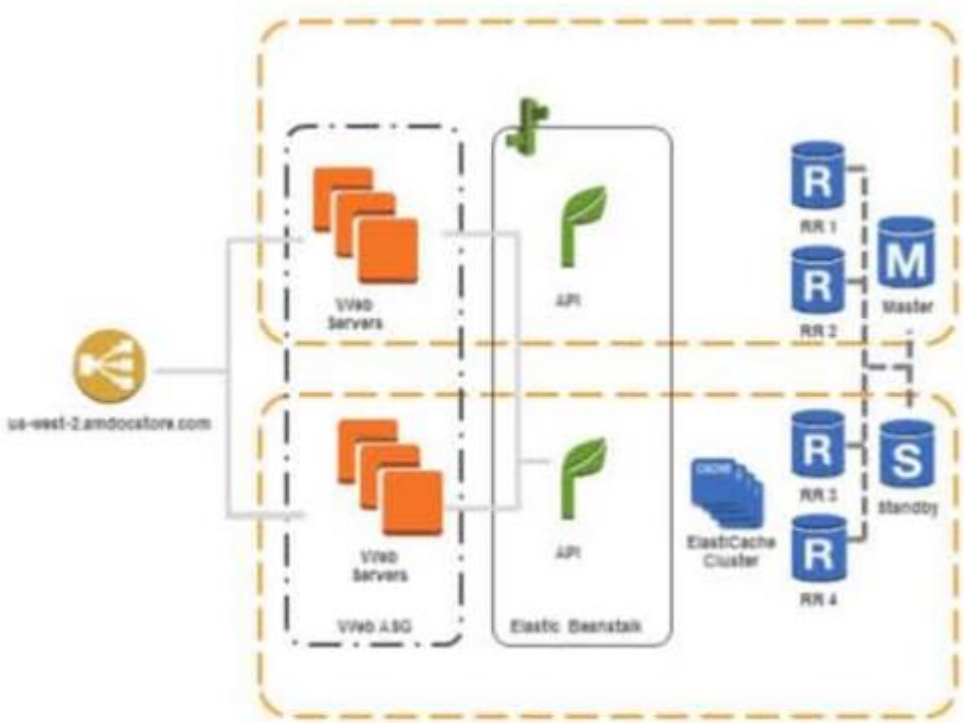
DOCSTORE



amazon web services

Let's focus on the networking environment

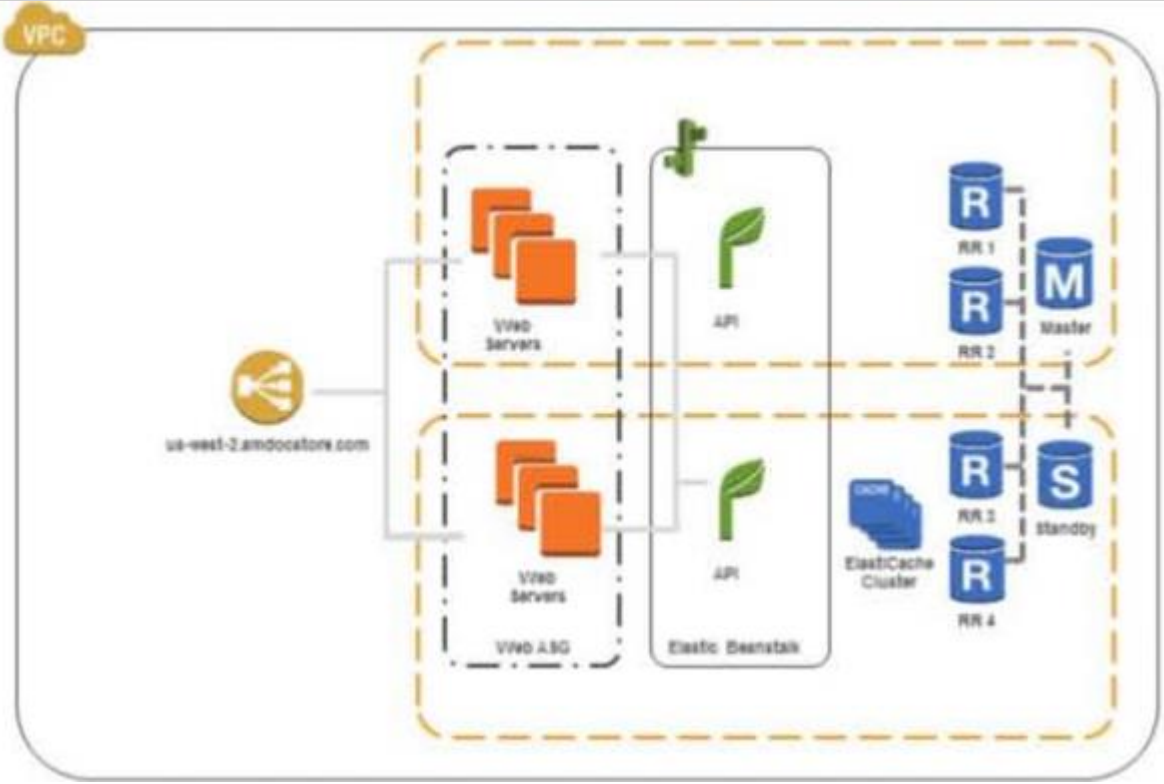
DOCSTORE



amazon web services

VPC: this VPC is in us-west-2 and has a 10.0.0.0/16 CIDR block

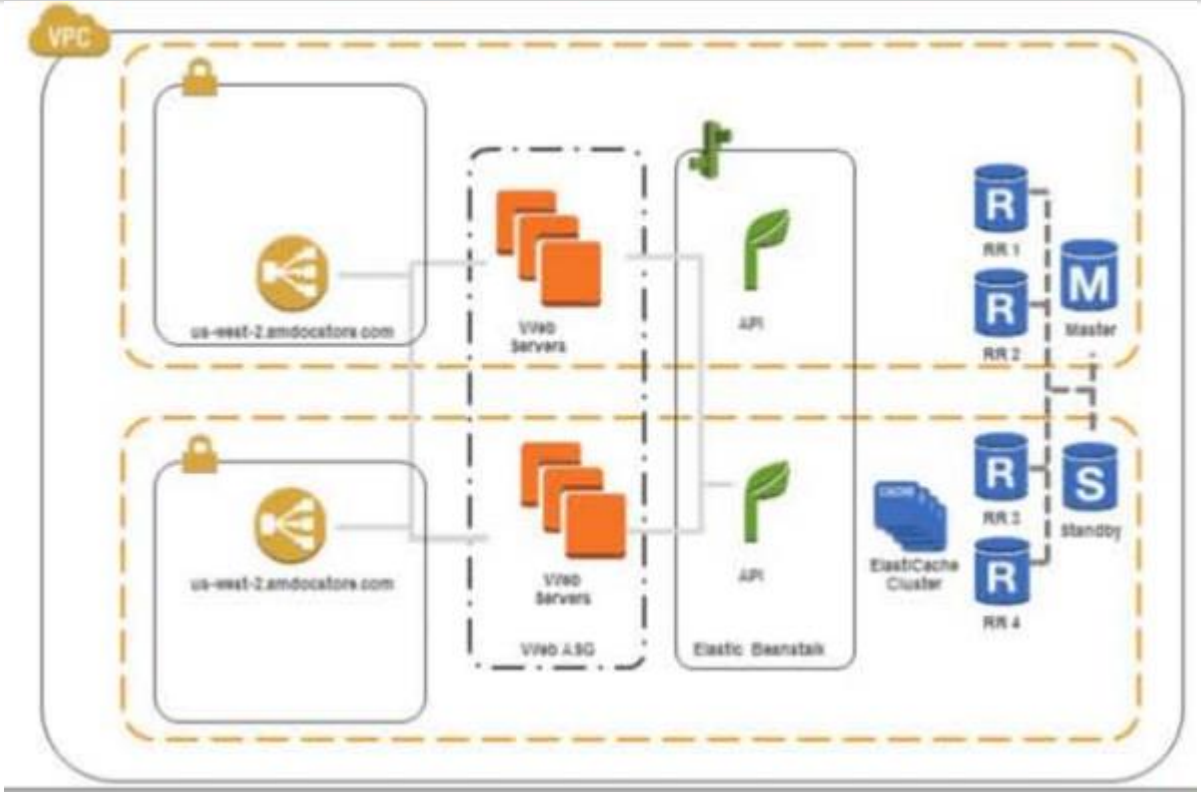
DOCSTORE



amazon web services

VPC: We define two subnets that our ELB will run in (note: although we show the ELB in multiple subnets, this is still a single ELB)

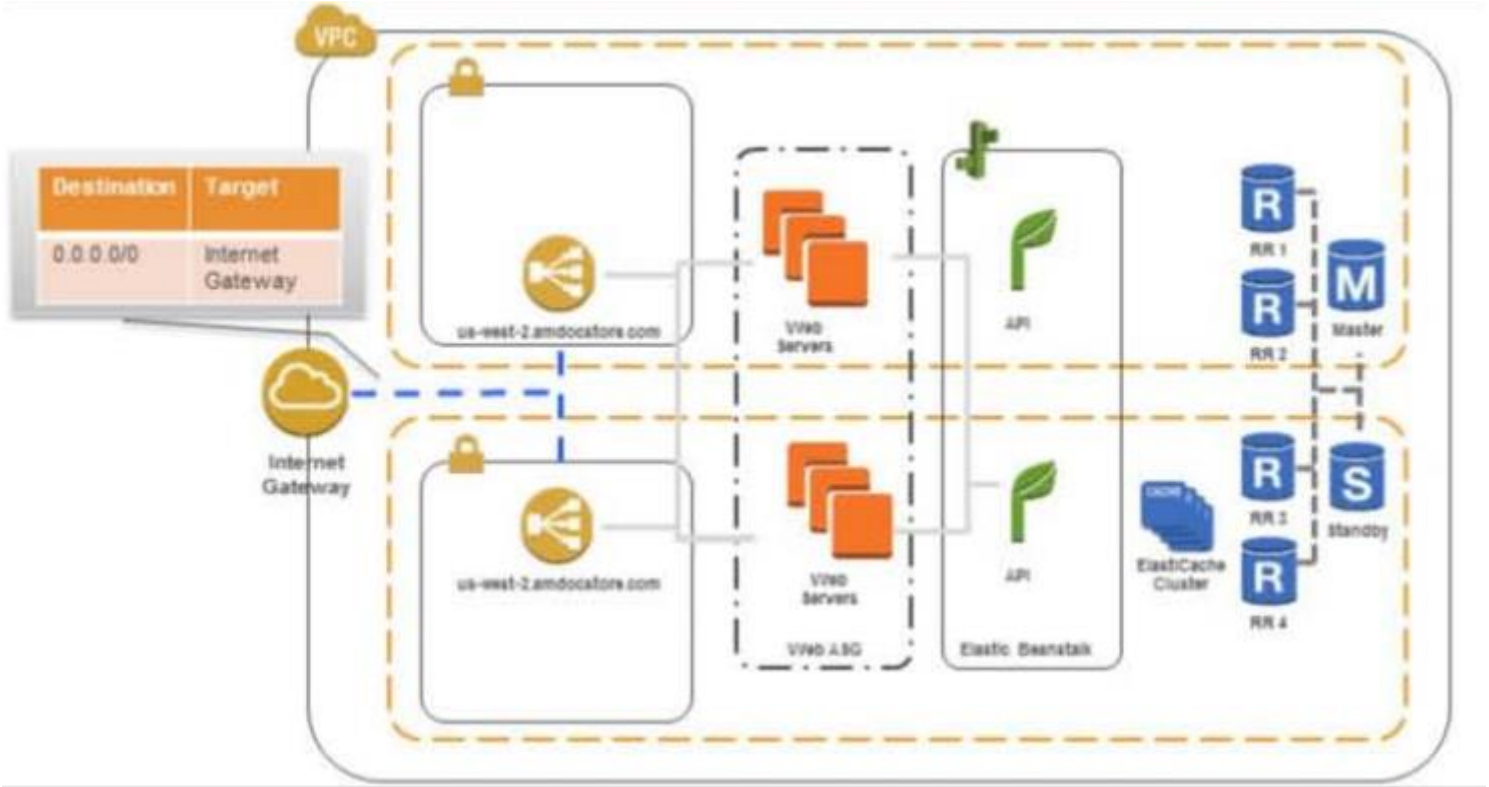
DOCSTORE



amazon web services

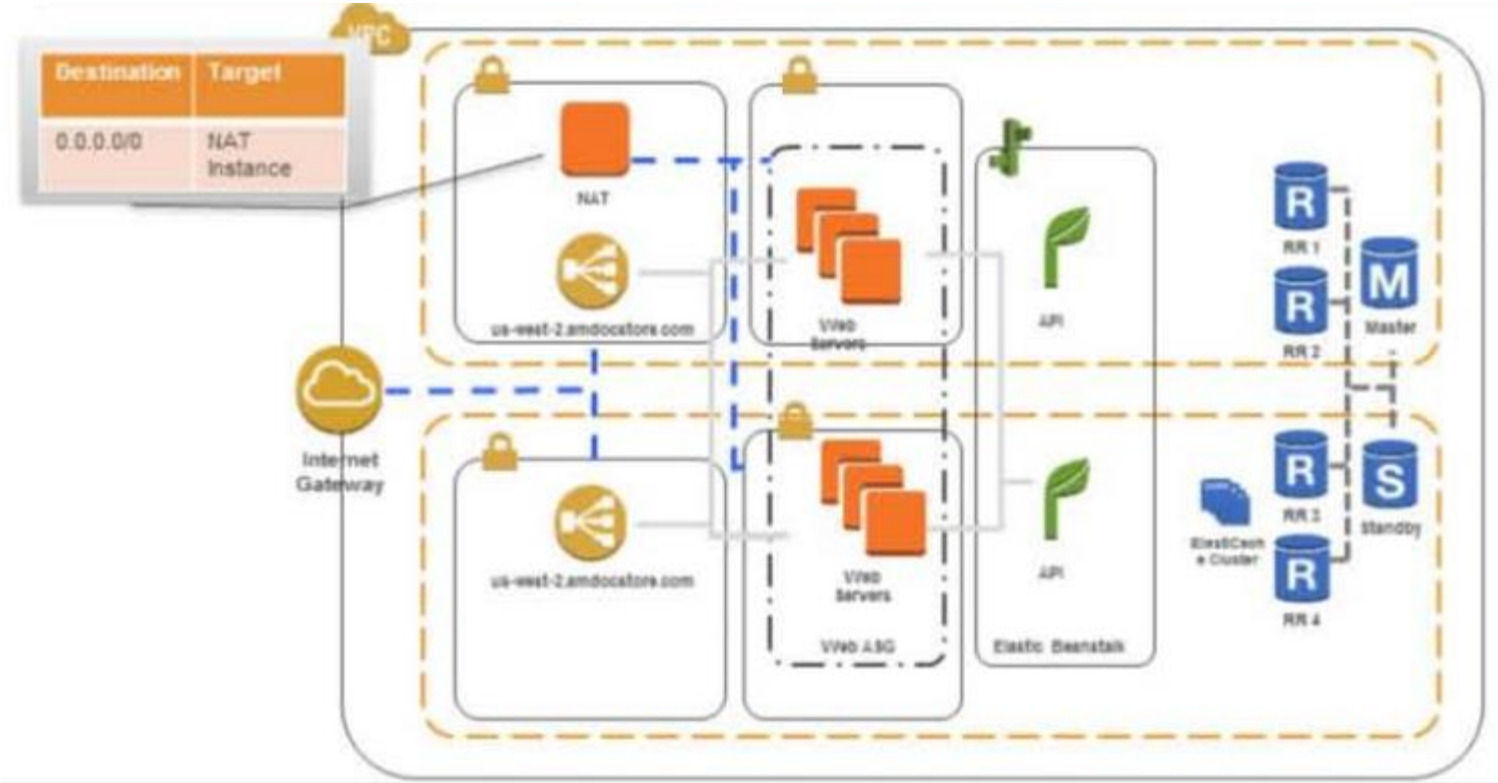
VPC: These subnets should be public. We will attach an Internet Gateway and define a Route Table entry to make them public

DOCSTORE



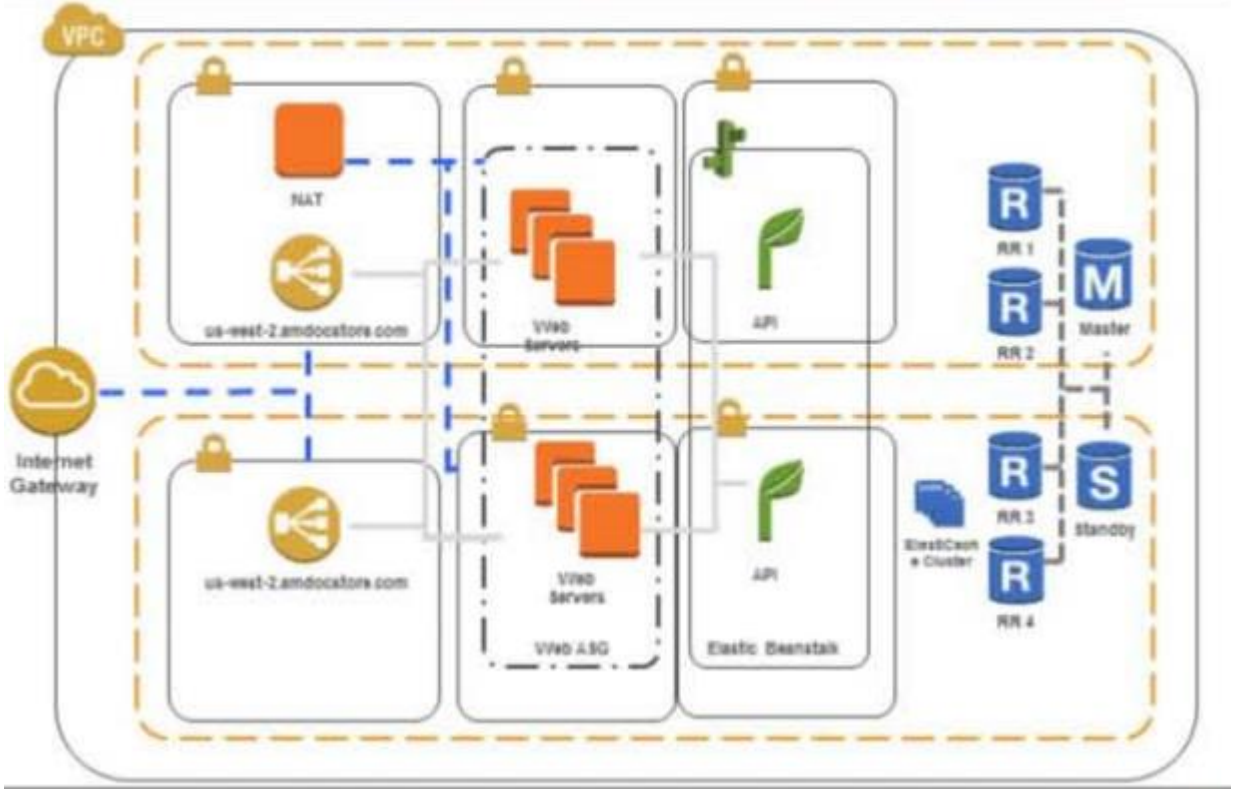
VPC: Web servers run in a private subnet, but have outbound internet access via an EC2 instance in the public subnet running NAT

DOCSTORE



VPC: The DocStore API running on Elastic Beanstalk is deployed in both a public and private subnet

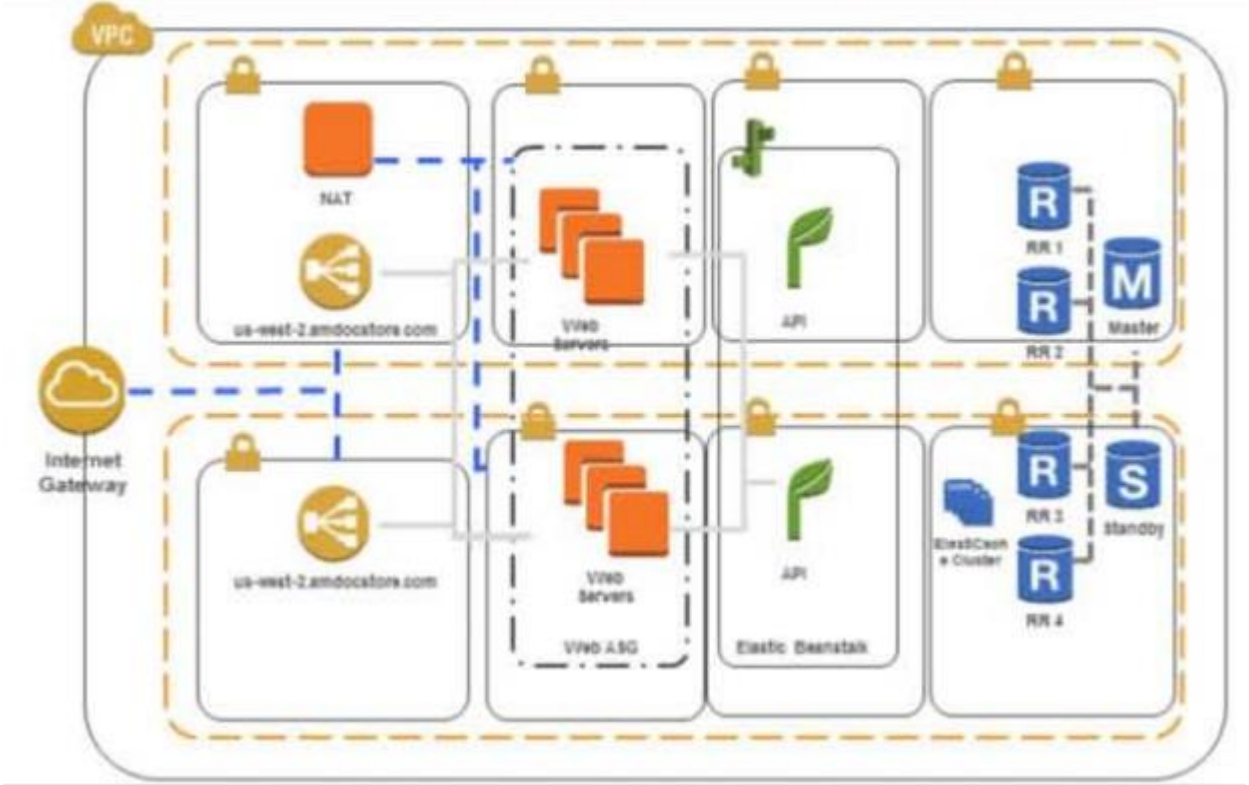
DOCSTORE



amazon web services

VPC: Finally, the RDS and ElastiCache instances are deployed in a private subnet

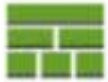
DOCSTORE



amazon web services

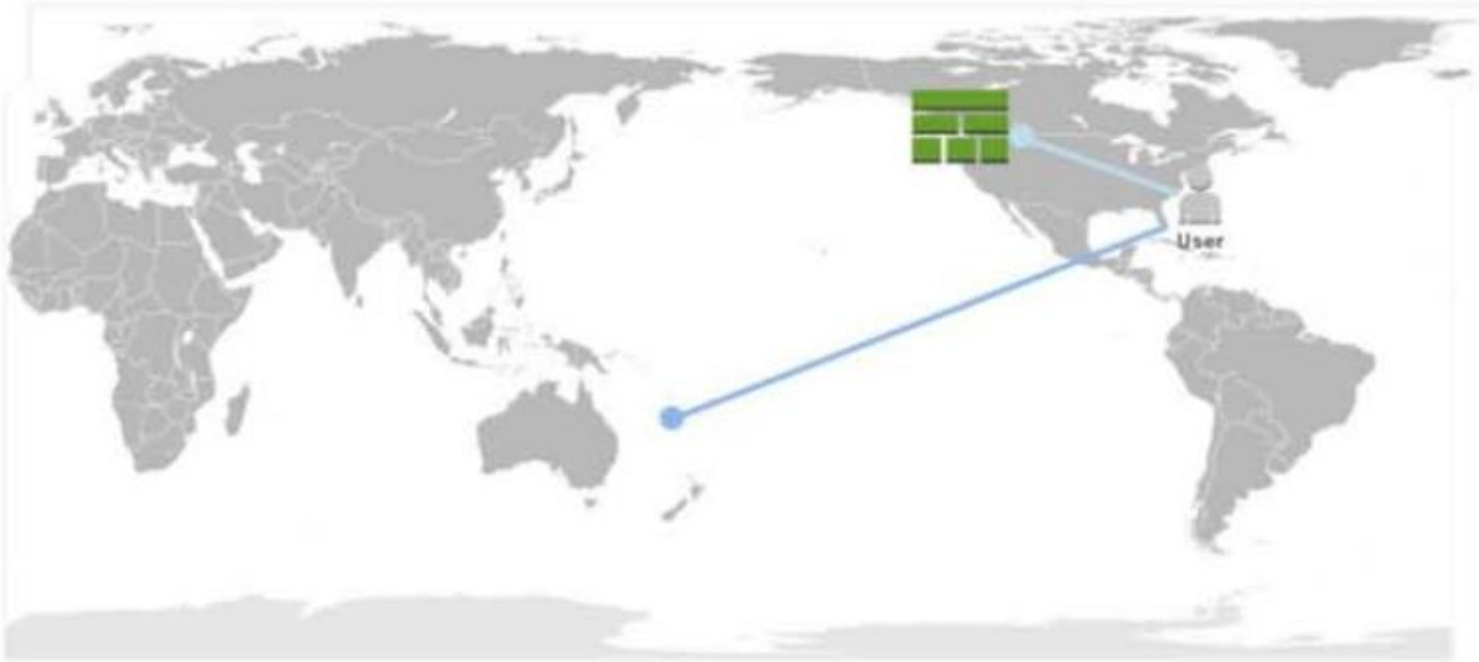
CloudFormation: Entire application is deployed as a stack from a template file

DOCSTORE



CloudFormation: The sample template file is used to deployed to us-west-2

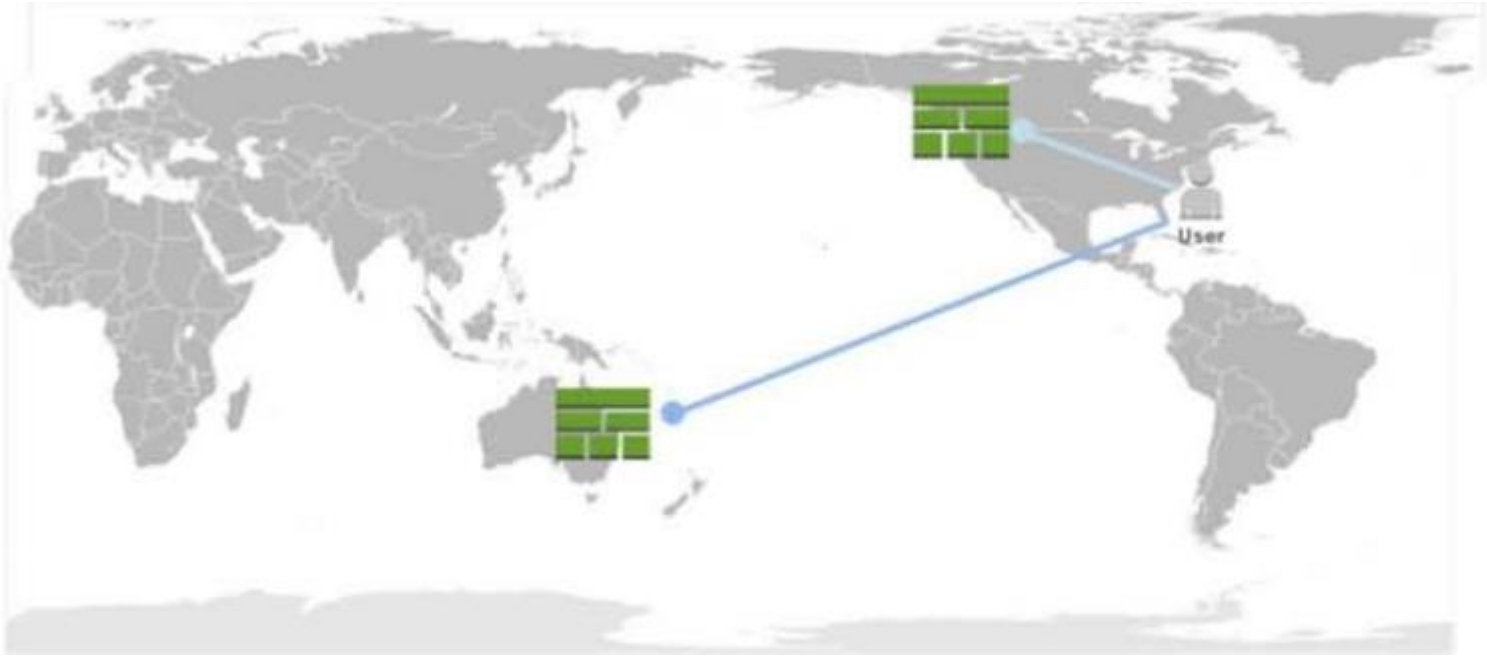
DOCSTORE



amazon web services

CloudFormation: and ap-southeast-2

DOCSTORE



amazon web services

#2. Batch processing Back-End

#2. Batch processing Back-End

- Sync user account
- Enforce max storage limits
- Extract and index based on document type
- Detailed dashboard with system-wide totals:
  - # docs, avg, doc size, total storage

#2. Batch processing Back-End

- Responds to events triggered by Web interface

#2. Batch processing Back-End

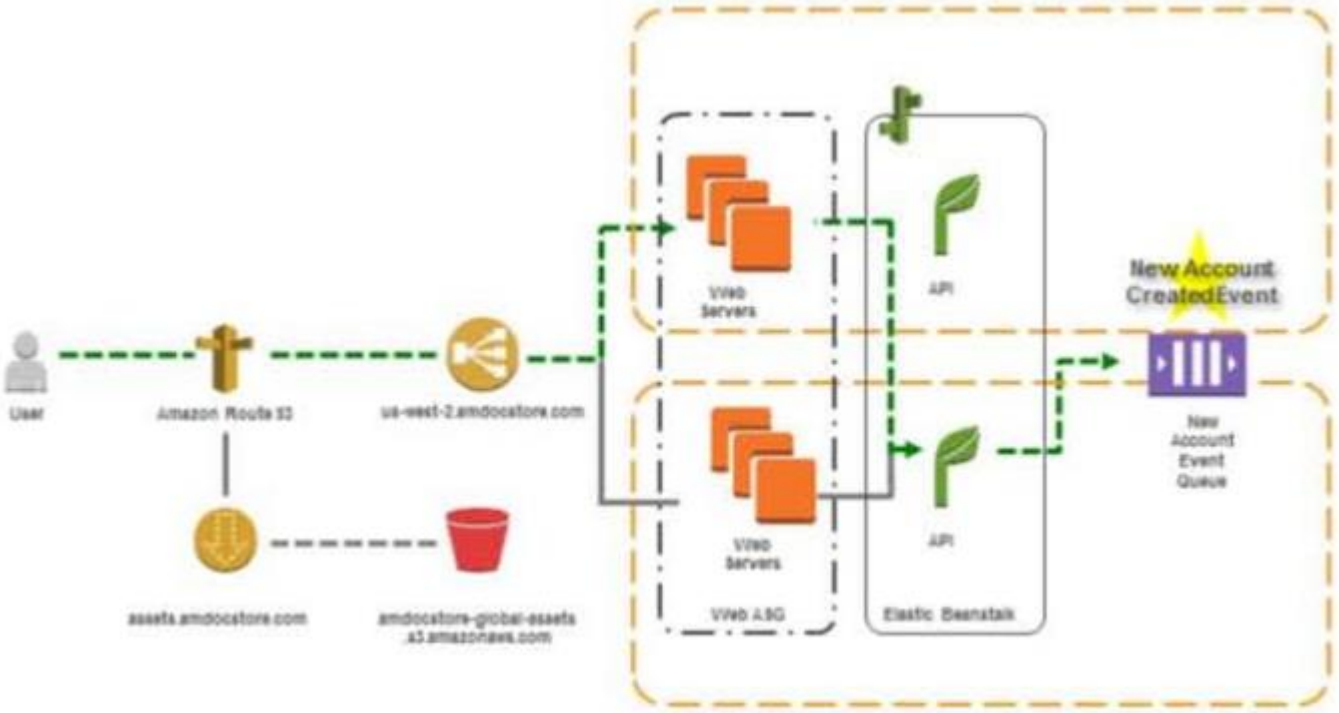
- Responds to events triggered by Web interface

#2. Batch processing Back-End

- Responds to events triggered by Web and Mobile interfaces

New Account Create Event: Triggered when a user creates a new account. Event message in queue includes username, password and home region

DOCSTORE



amazon web services

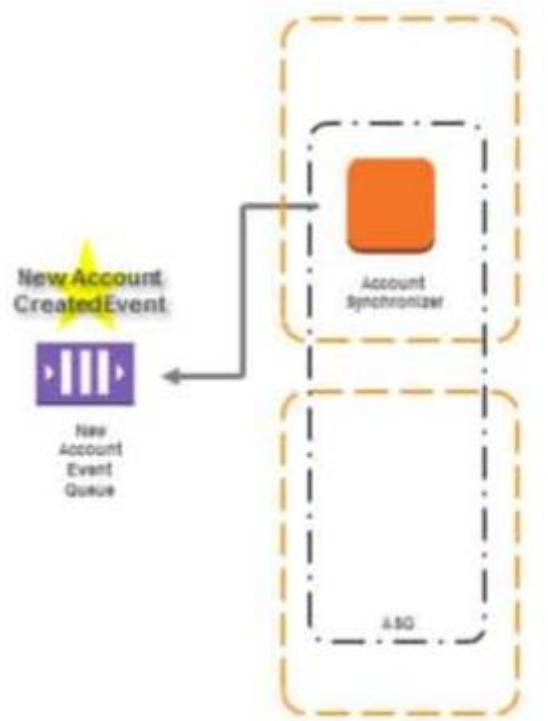
New Account Created Event: Triggered when a user creates a new account, event message in queue includes username, password and home region

DOCSTORE

amazon web services

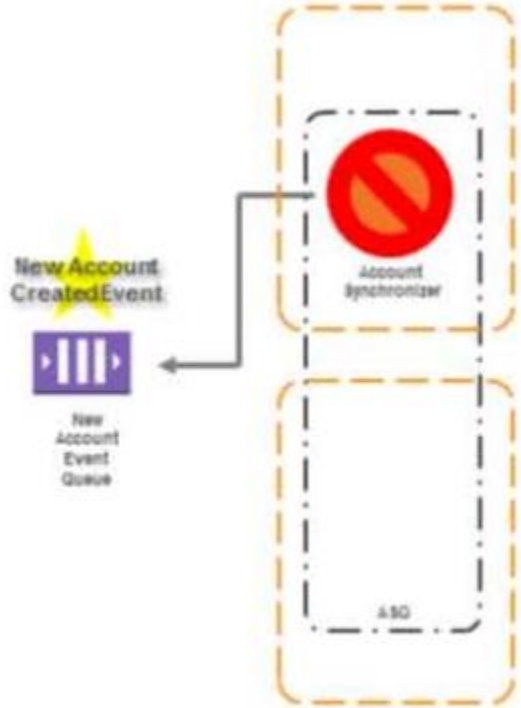
New Account Created Event: An instance running the Account Synchronization service polls the New Account Event Queue

DOCSTORE



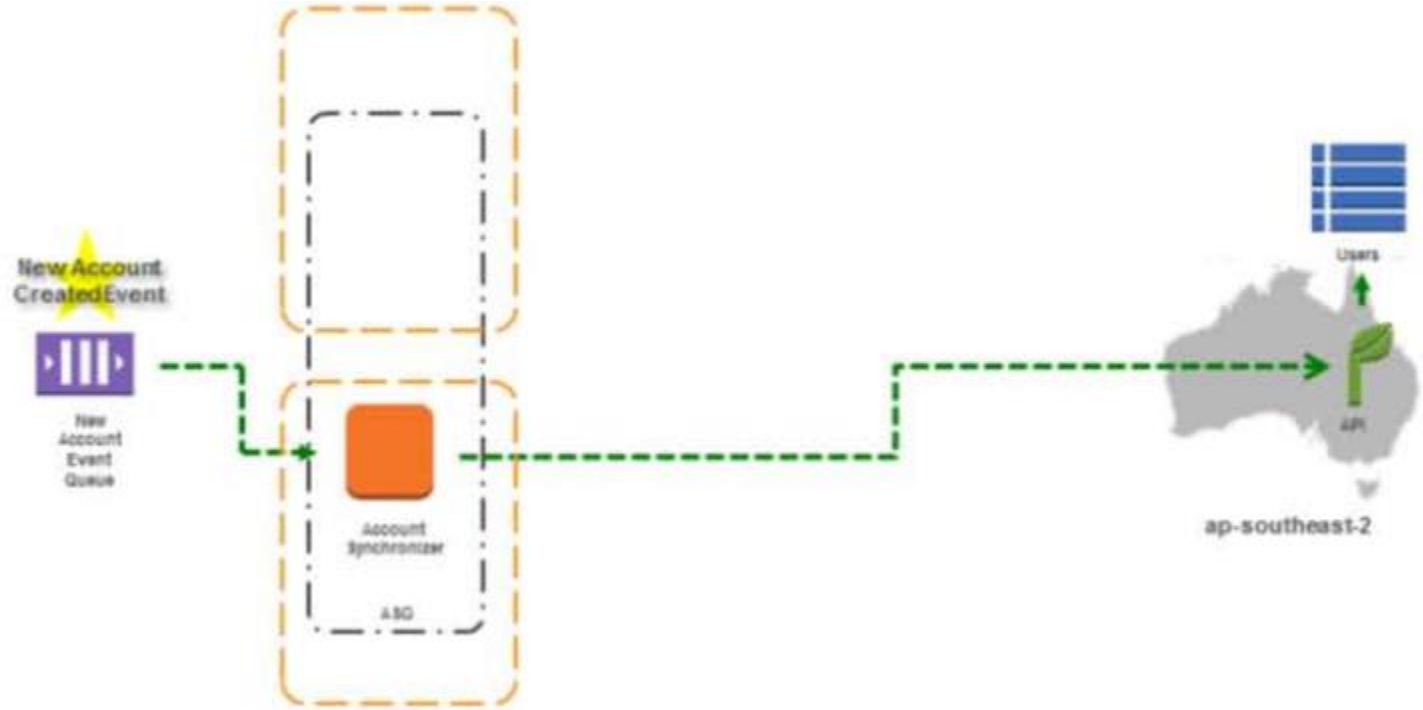
amazon web services

New Account Created Event: If the instance fails, Auto Scaling will launch a replacement



New Account Created Event: Account Synchronizer invokes API in ap-southeast-2 to synchronize new user's account data

DOCSTORE



amazon web services

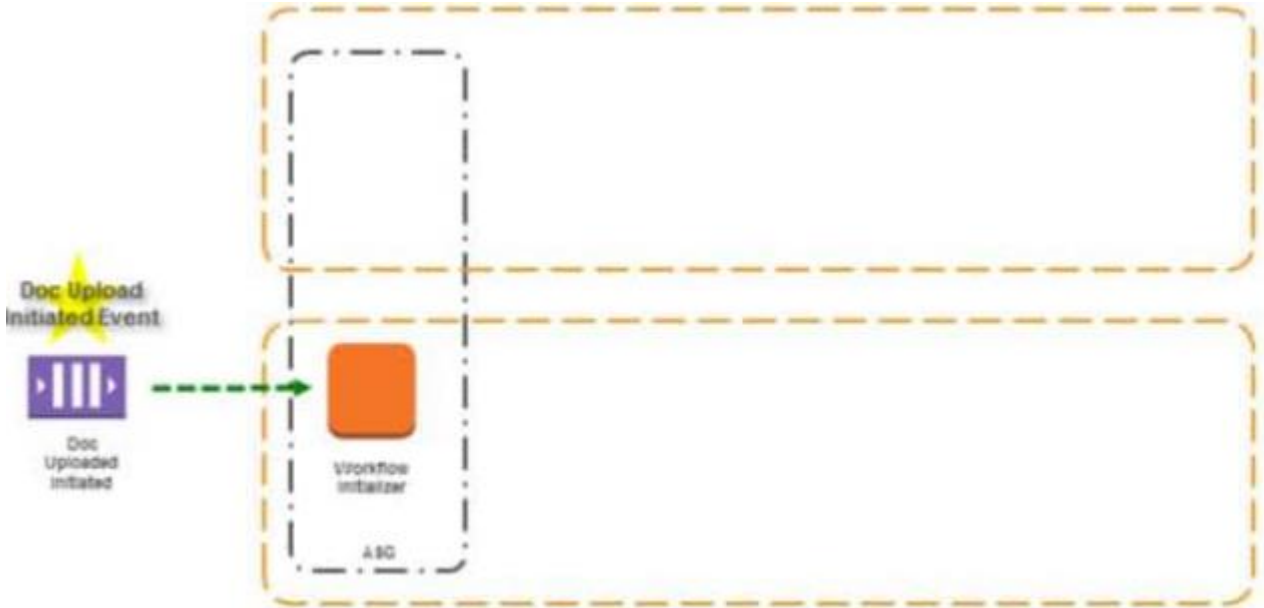
New Account Created Event : When a user begins uploading a document to S3, an event is triggered so the upload can be tracked via workflow

DOCSTORE

amazon web services

New Account Created Event : When an event is received from the queue, the workflow initializer kicks off a SWF workflow execution

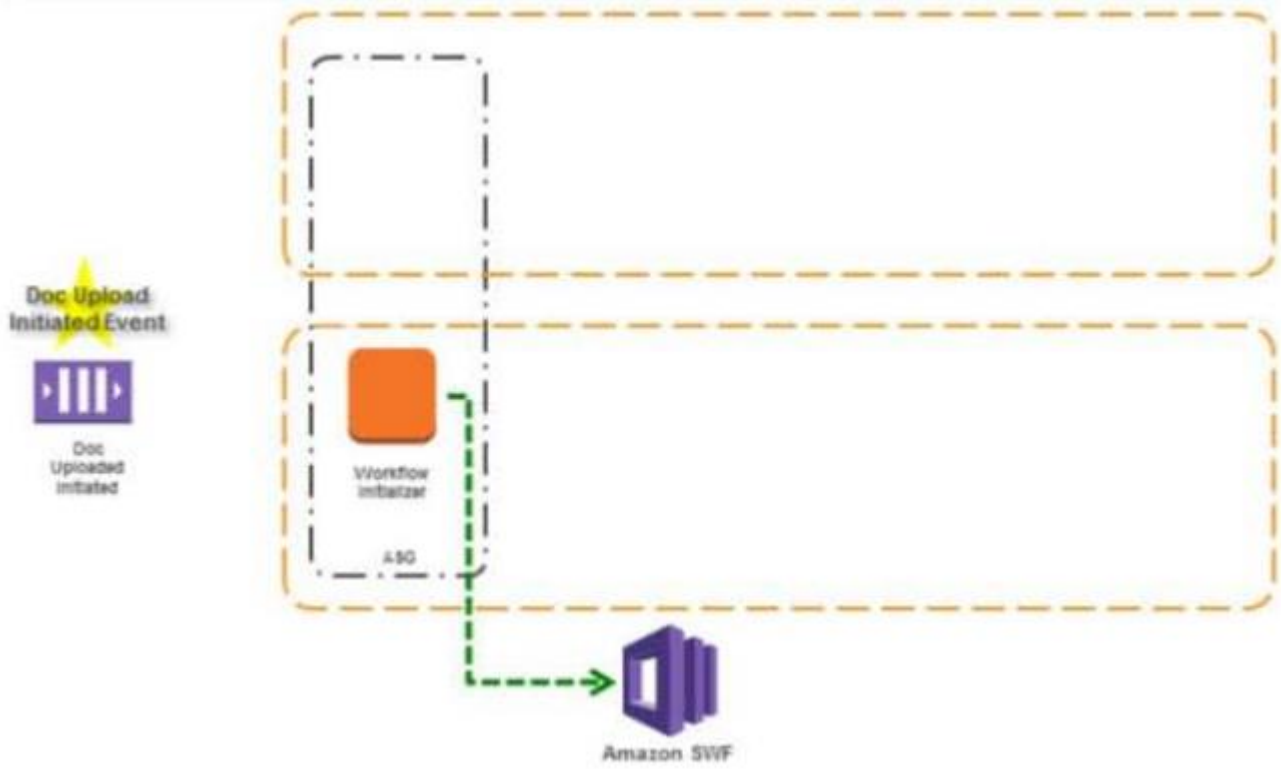
DOCSTORE



amazon web services

Doc Uploaded Initiated Event: When a event is received from the queue, the workflow initializer kicks off a SWF workflow execution

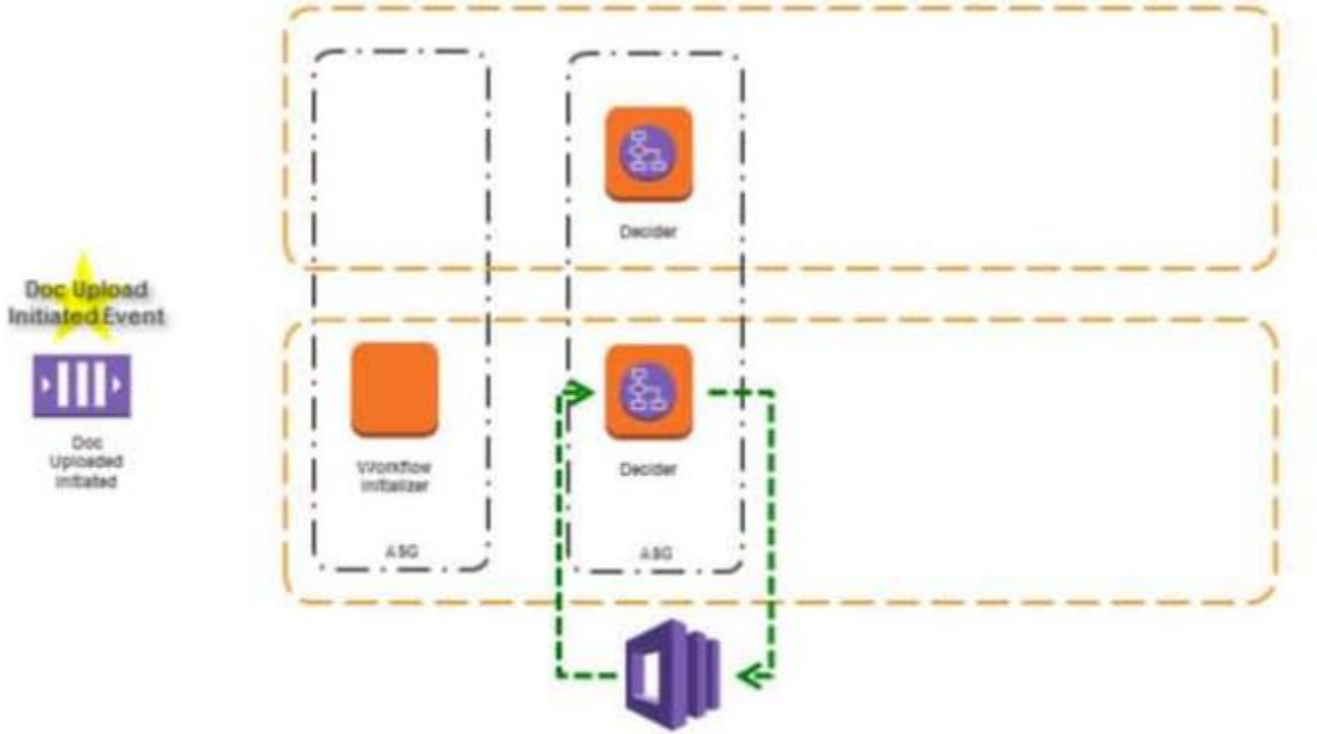
DOCSTORE



amazon web services

Doc Uploaded Initiated Event: A an instance running Decider code receives the workflow execution started task, and schedules a Check Upload Status activity

DOCSTORE



amazon web services

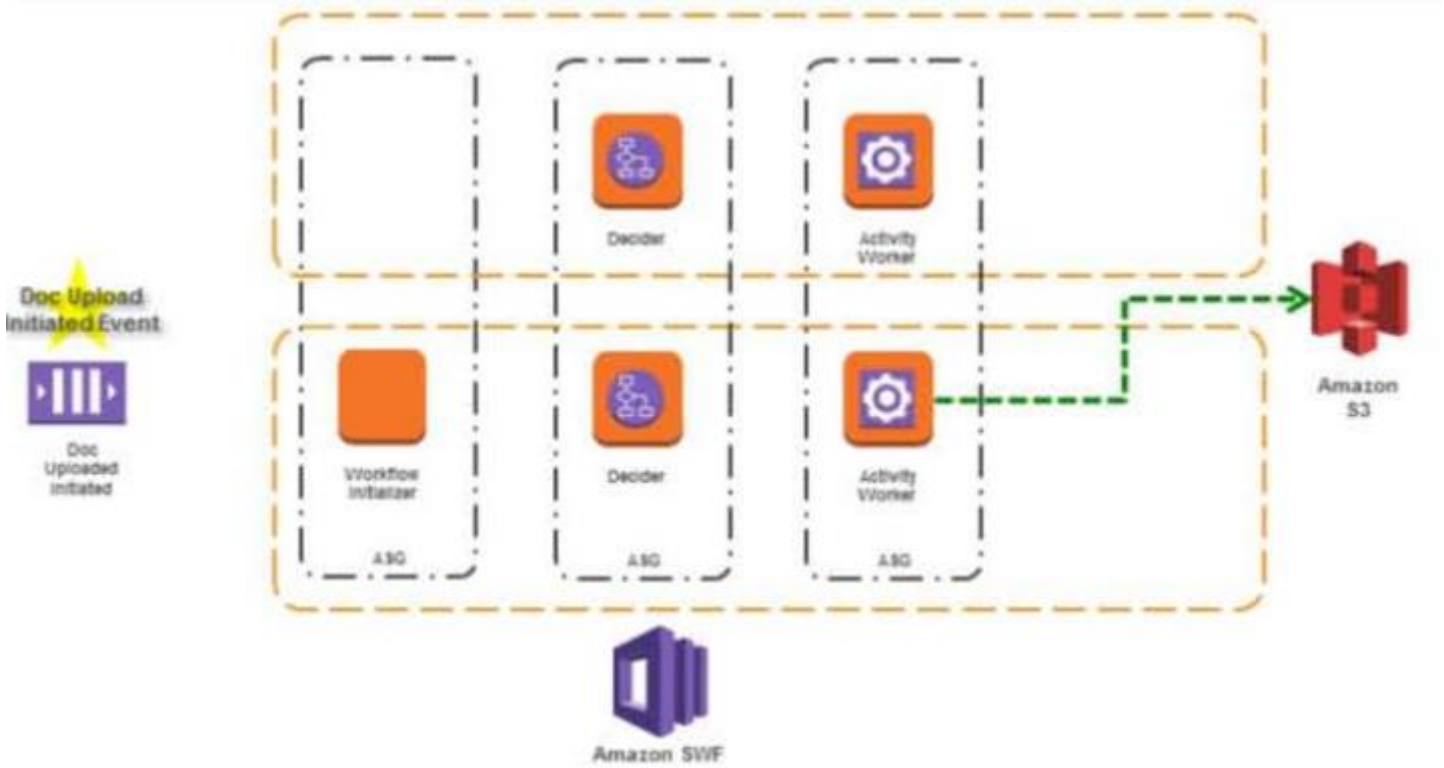
Doc Uploaded Initiated Event: A an instance running the Activity Worker code receives the Check Upload Status task

DOCSTORE

amazon web services

Doc Uploaded Initiated Event: The Activity Worker periodically checks S3 to determine if the doc upload has completed

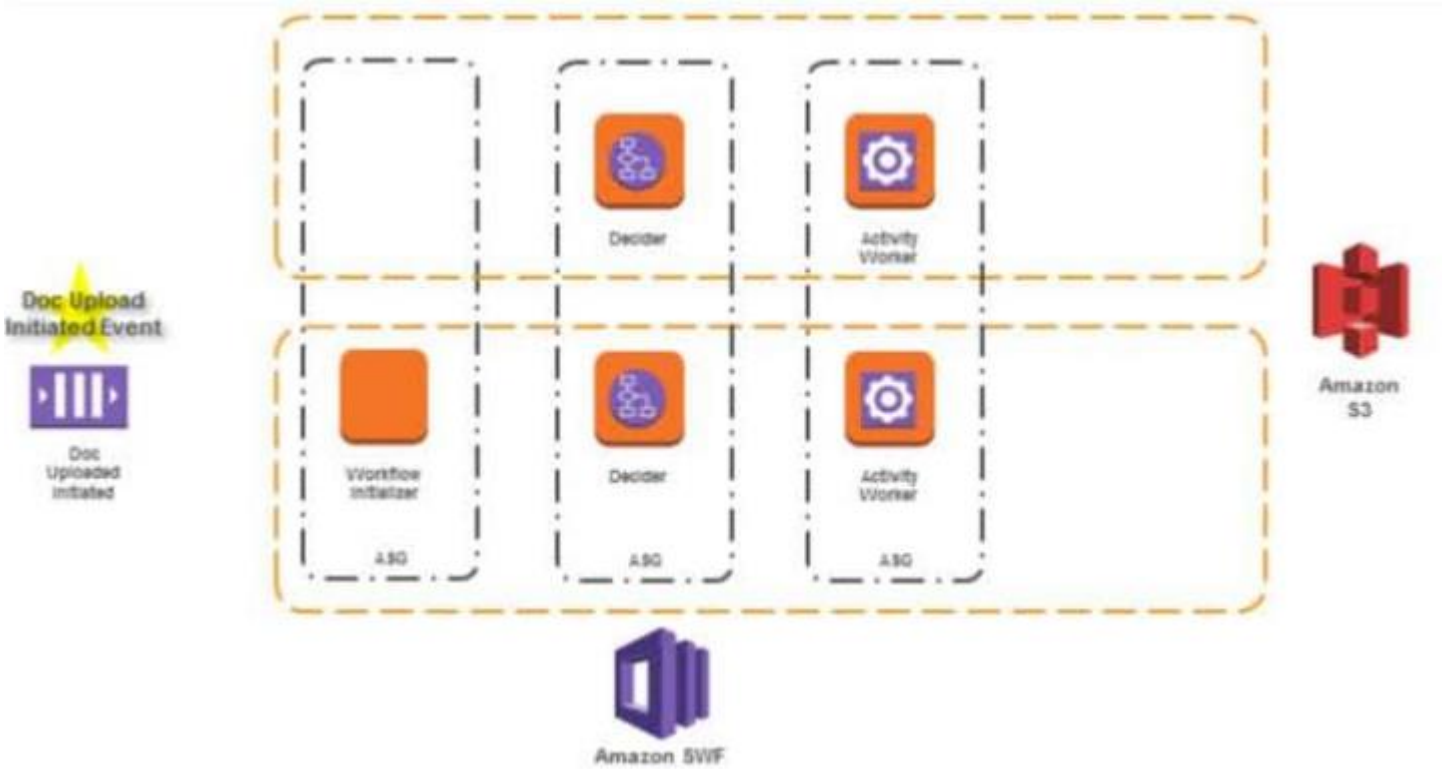
DOCSTORE



amazon web services

Doc Uploaded Initiated Event: Once the document is deleted in S3, a final is scheduled in the near future to ensure that the document has been property processed by the Doc Upload Complete workflow

DOCSTORE



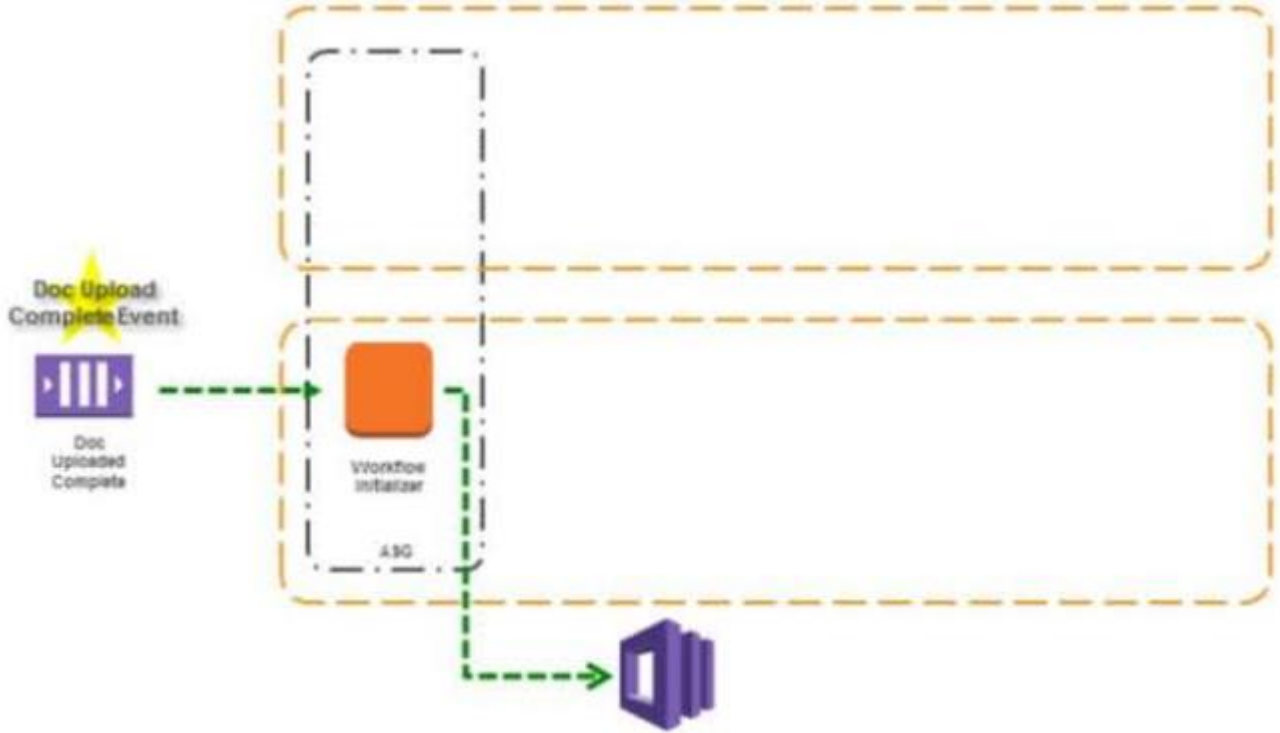
Doc Uploaded Initiated Event: When a document upload has been complete, it needs to be processed

DOCSTORE

amazon web services

Doc Uploaded Initiated Event: A workflow initializer determines the appropriate workflow to initiate

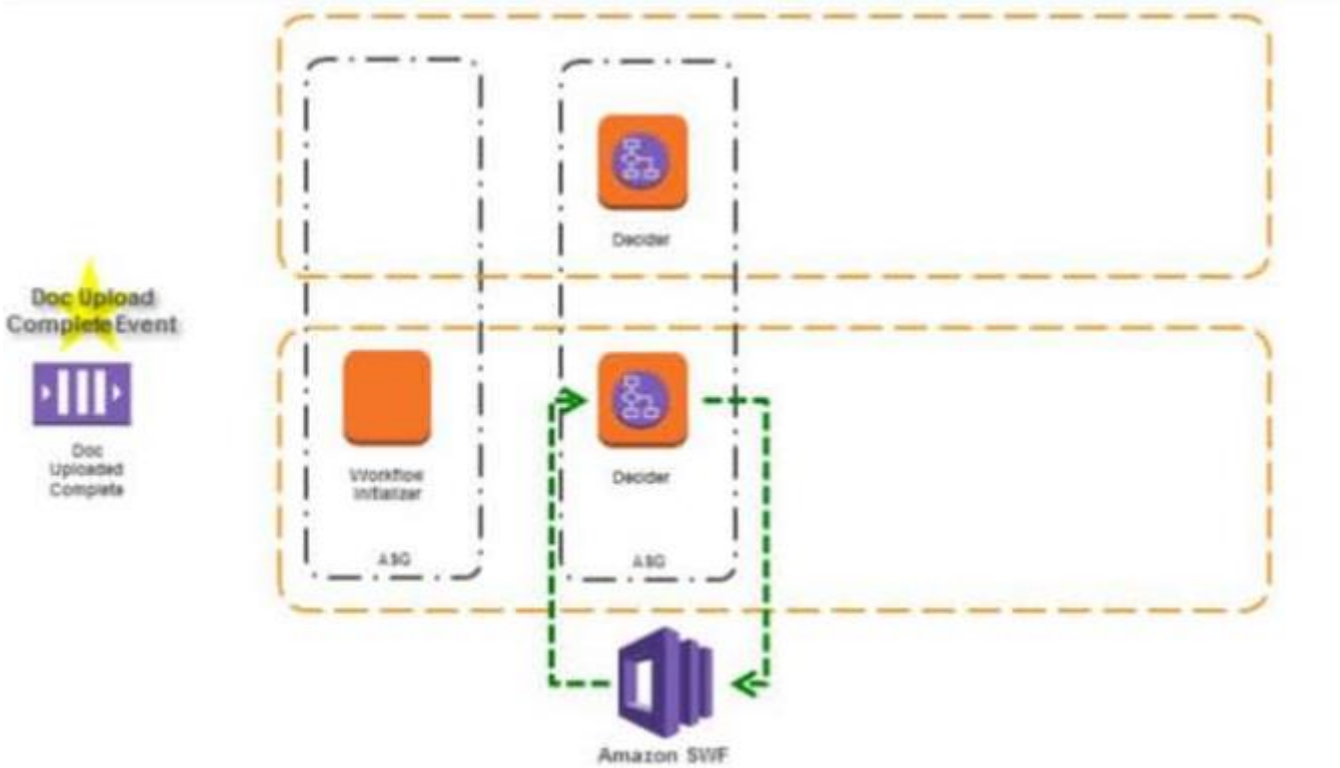
DOCSTORE



amazon web services

Doc Uploaded Initiated Event: A decider schedules the appropriate activity based on the document type

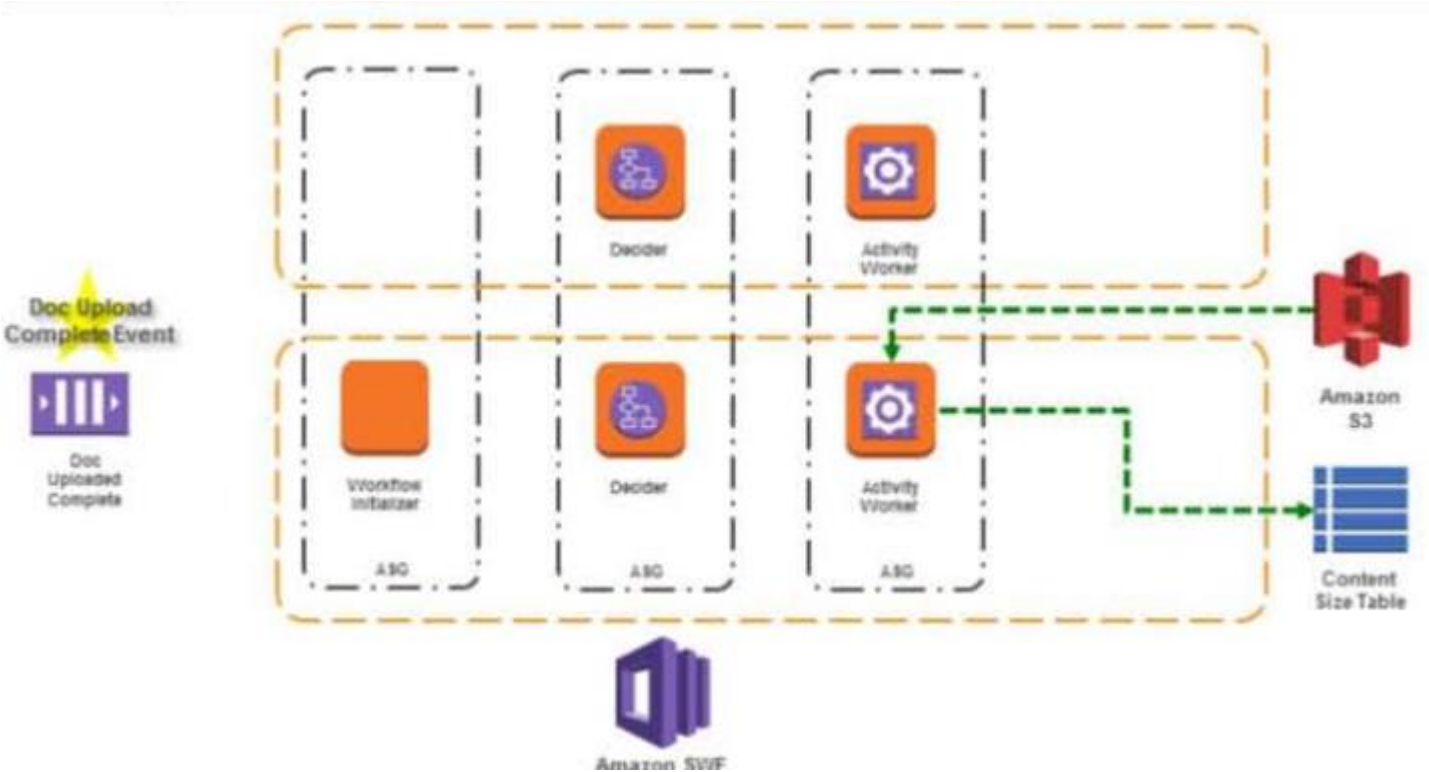
DOCSTORE



amazon web services

Doc Uploaded Initiated Event: the content-length metadata property of every object should be retrieved from S3 and used to increment a counter tracking the total size of all uploaded objects

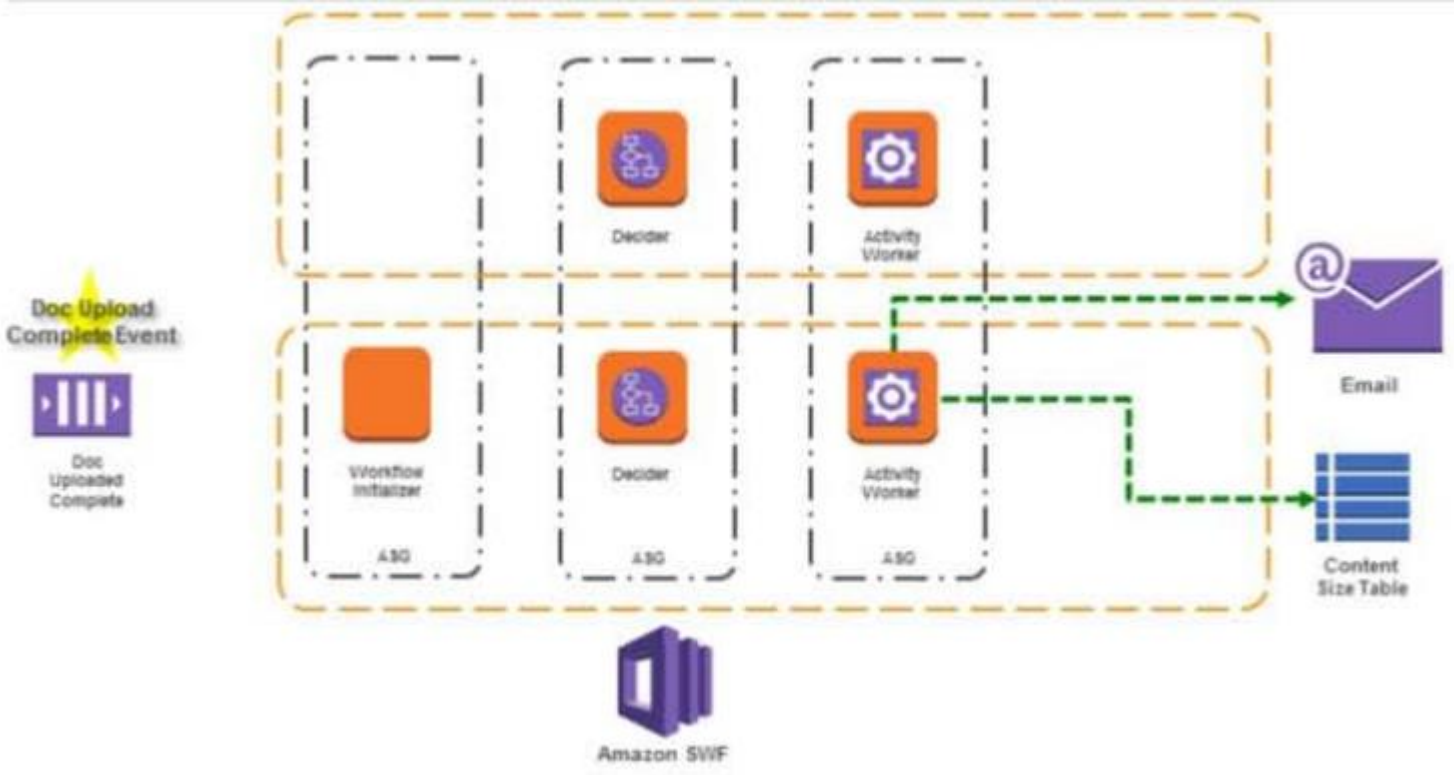
DOCSTORE



amazon web services

Doc Uploaded Initiated Event: a counter tracking the total content size uploaded by each users is also incremented. If the new doc causes the user to exceed his allocated amount, a flag is set and the user will receive an email via SES

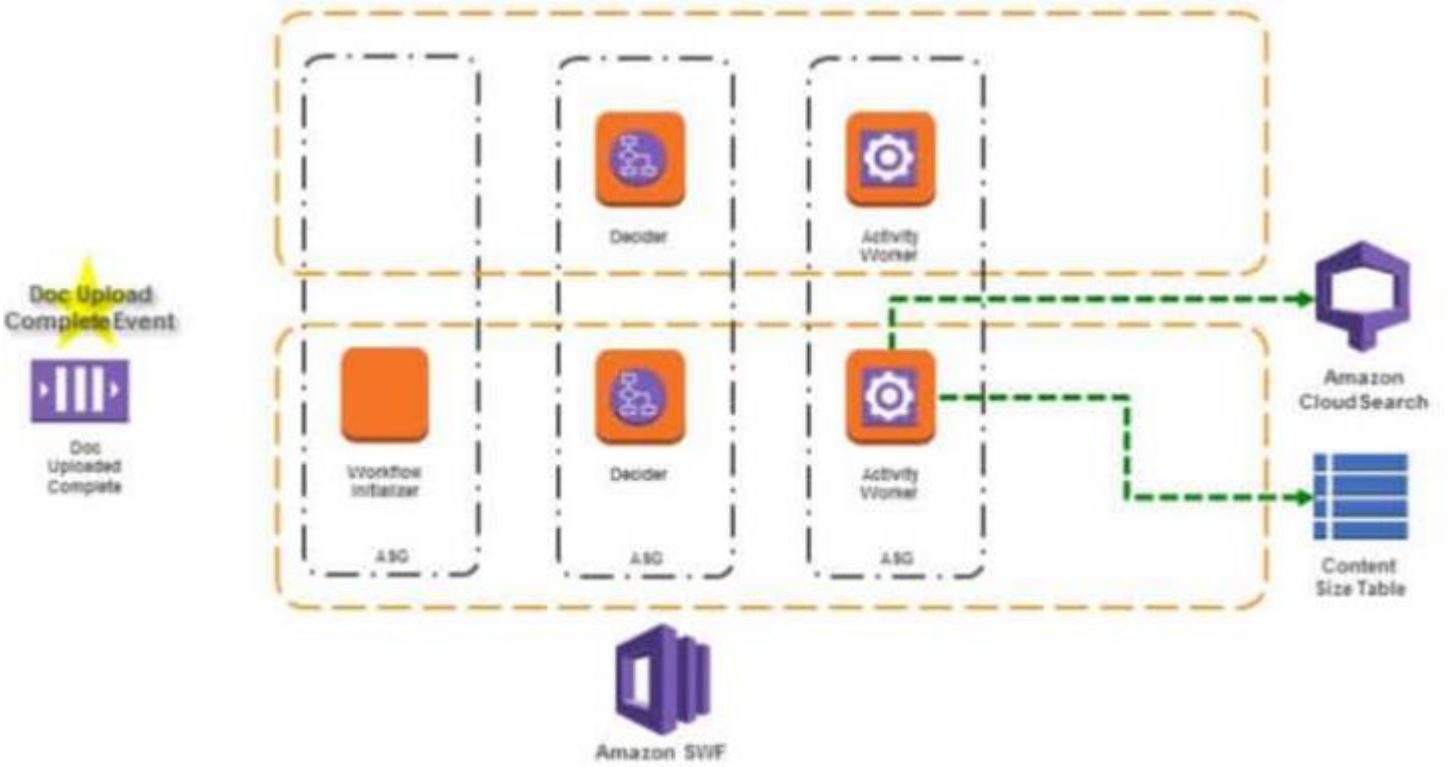
DOCSTORE



amazon web services

Doc Uploaded Initiated Event: if the document was uploaded by a premium user and it has text context, the full-text content will be indexed in CloudSearch

DOCSTORE



Doc Uploaded Initiated Event: Finally, several custom metrics are also put to CloudWatch, allowing us to monitor how many new documents are being uploaded by users type, how big the documents are, etc.

DOCSTORE