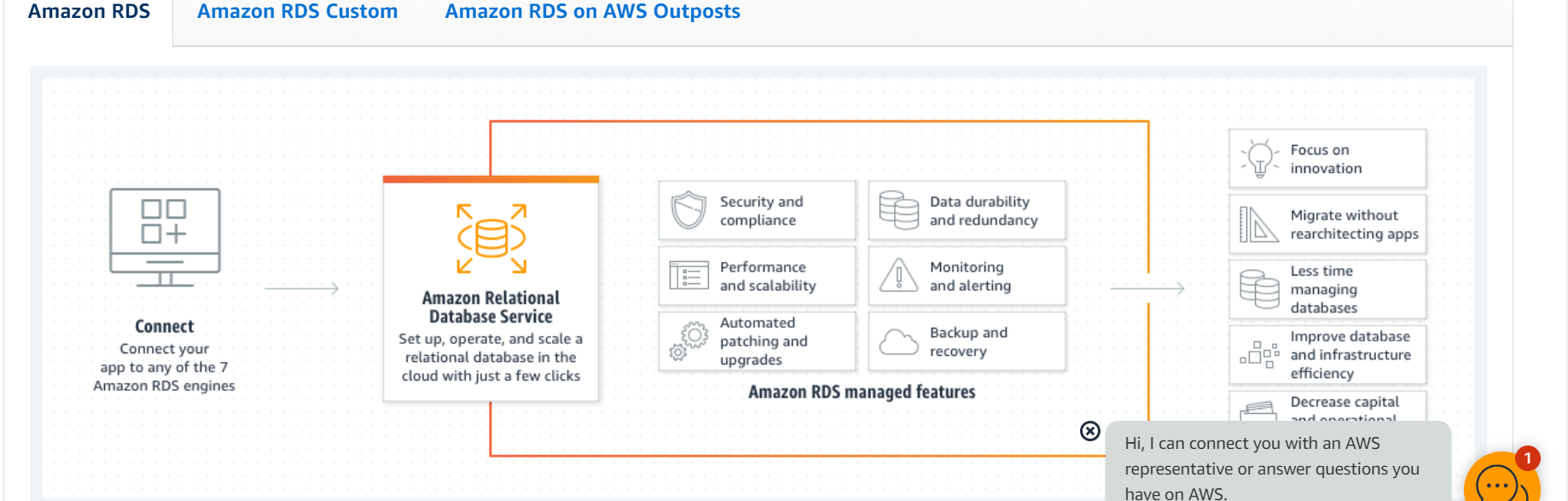
**AWS RDS**

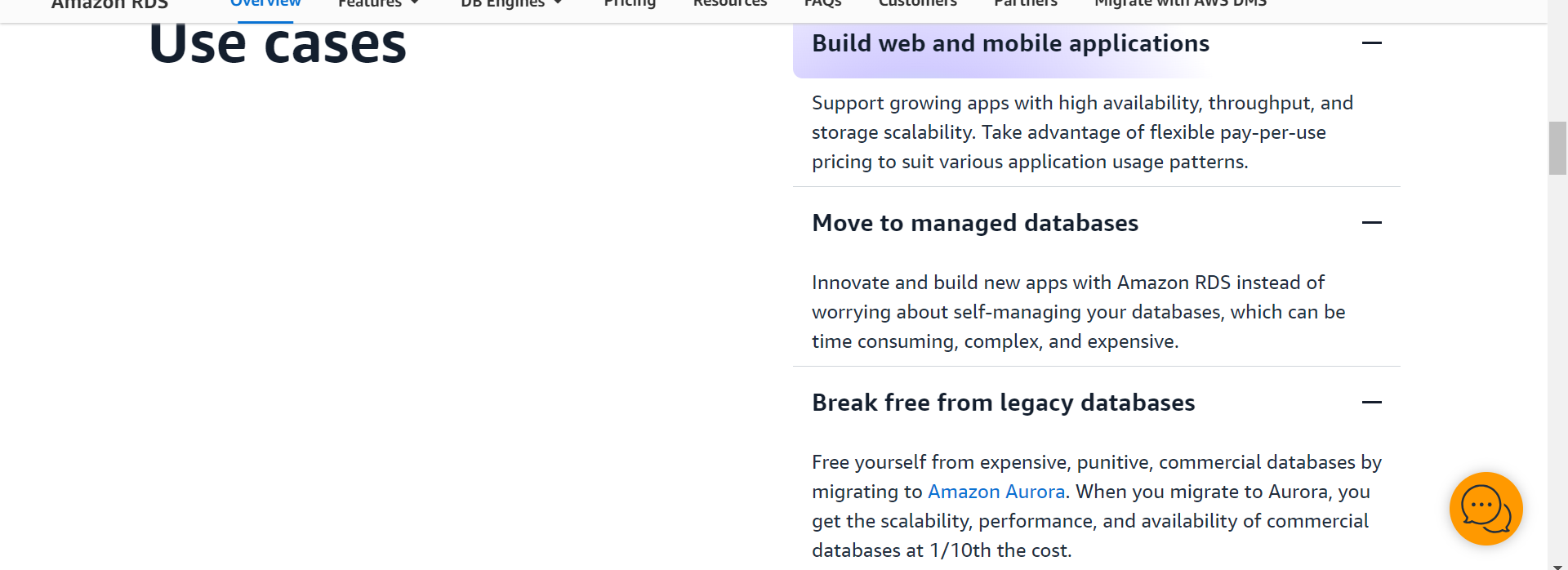
Amazon Relational Database Service (RDS) is a cloud-based service offered by Amazon Web Services (AWS) that lets you set up, operate, and scale relational databases. It simplifies database management by automating tasks like provisioning, patching, backups, and scaling.

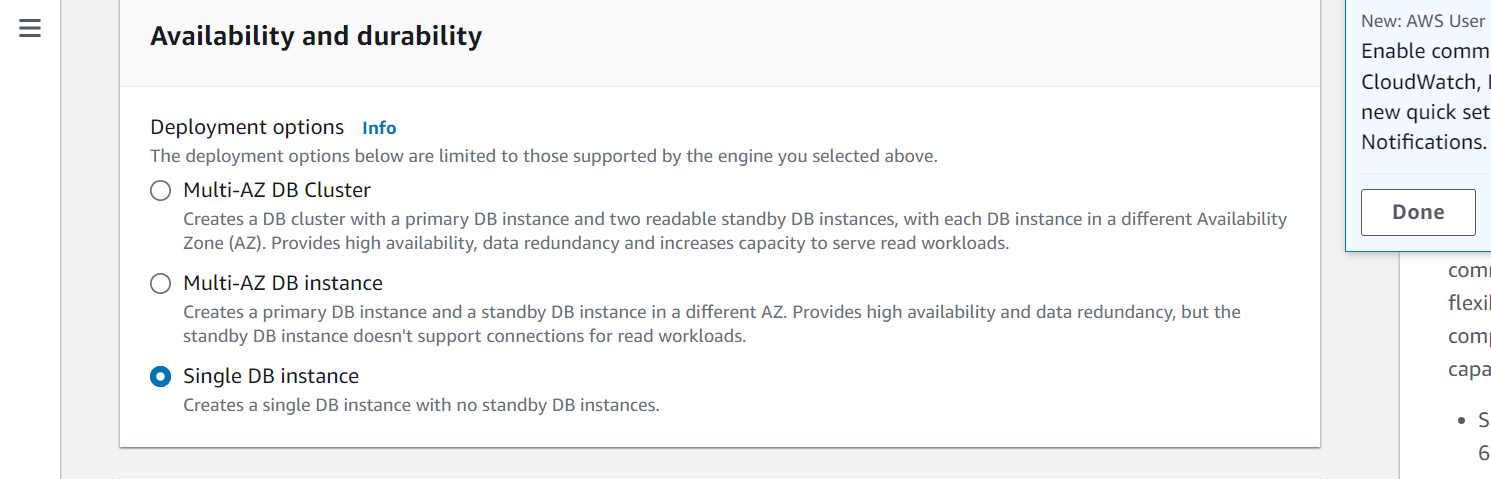
Here are some of the key features of Amazon RDS:

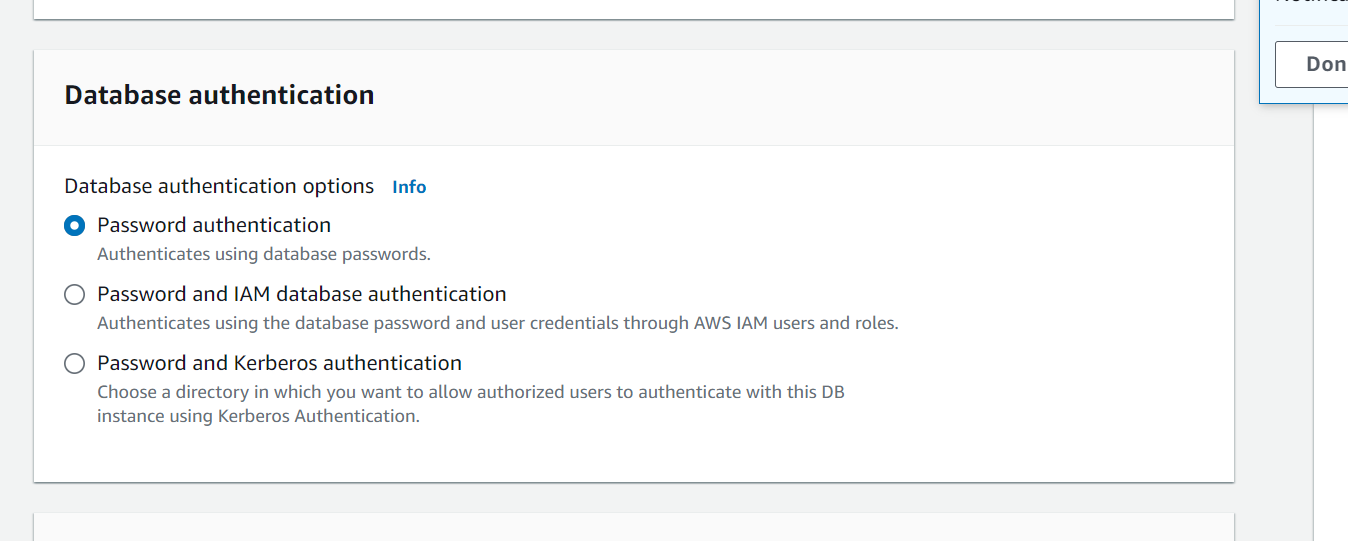
* Supported Engines: It offers a variety of popular relational database engines including MySQL, PostgreSQL, Oracle, SQL Server, and Aurora (MySQL and PostgreSQL compatible versions from Amazon).
* Ease of Use: You can create a new database instance in minutes and manage it through the AWS console, CLI, or APIs.
* Scalability: You can easily scale the storage and compute resources of your database instance to meet your application's needs.
* Cost-Effectiveness: There are two pricing models: on-demand instances for pay-as-you-go use and reserved instances for a fixed cost over a commitment period.
* High Availability: You can configure your database instance for high availability with Multi-AZ deployments or read replicas.
* Security: RDS offers a variety of security features like encryption at rest and in transit, IAM database authentication, and security groups.

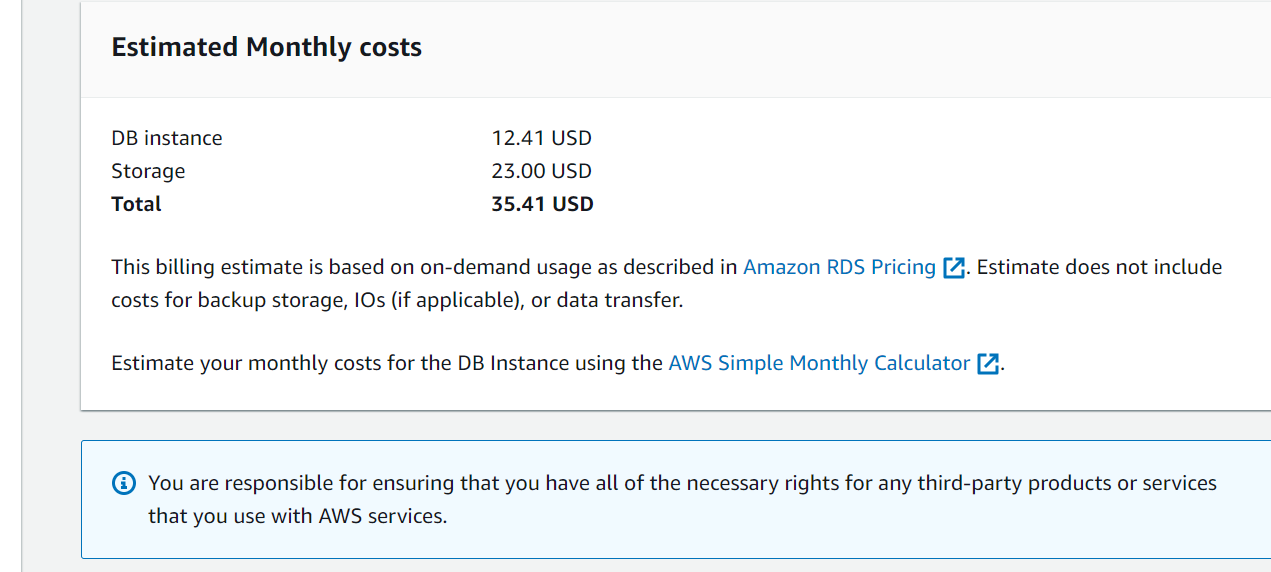
Overall, Amazon RDS is a powerful and easy-to-use service that can help you manage your relational databases in the cloud. If you're looking for a way to simplify database management and scale your applications, then Amazon RDS is a great option to consider.



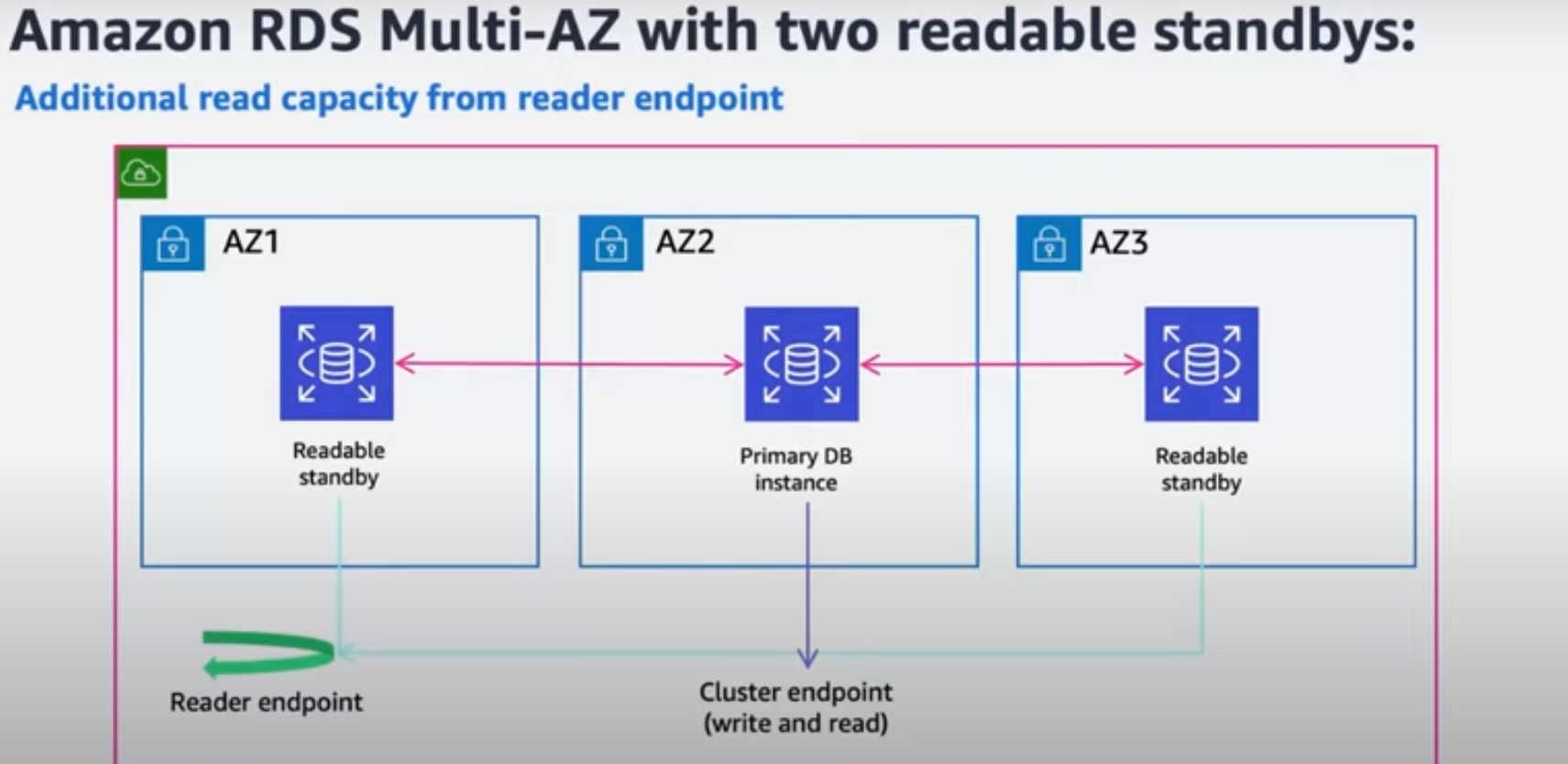


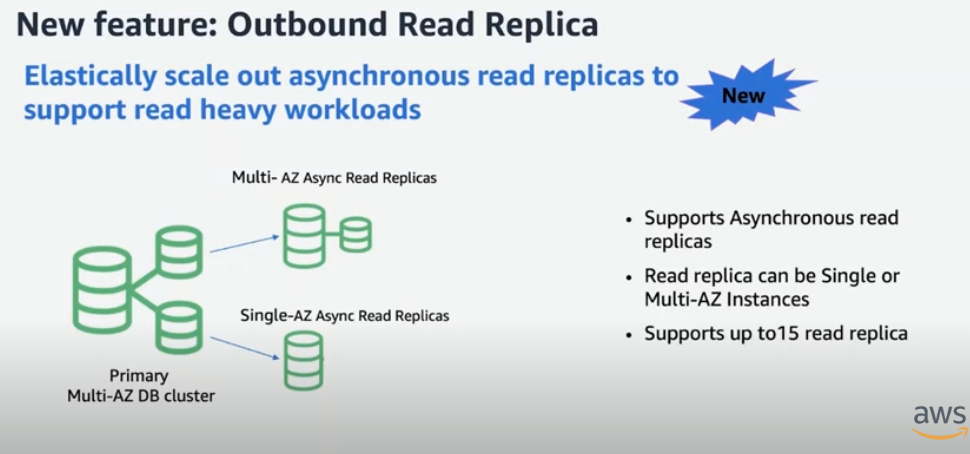


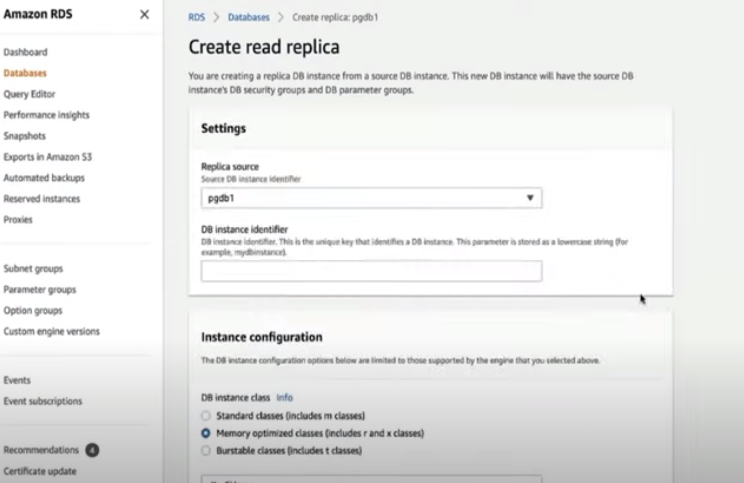


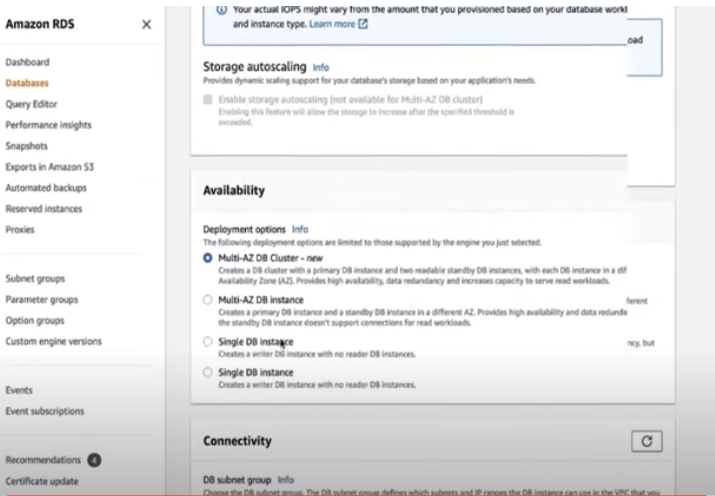


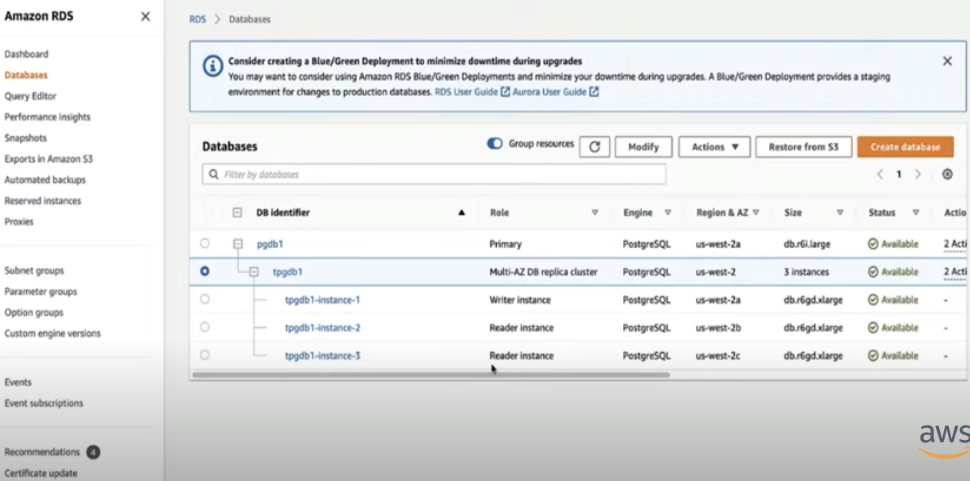


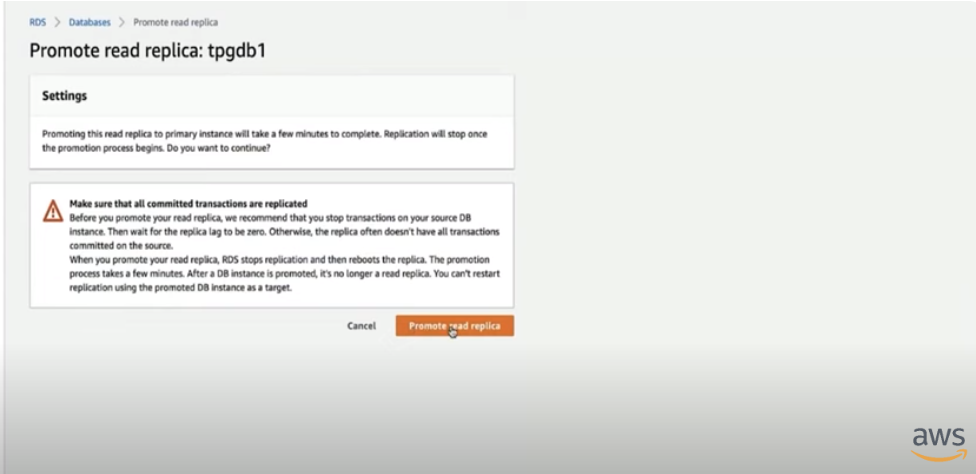












Within an AWS RDS database cluster, there are two main configurations for utilizing multiple instances:

1. **Multi-AZ Deployments:** This option focuses on high availability and disaster recovery.expand\_more  
   * It consists of one **writer** instance and typically two **reader** instances spread across different Availability Zones (AZs) within the same region.expand\_more
   * Data is replicated from the writer instance to the reader instances using the database engine's built-in replication functionality (semi-synchronous replication).
   * If the writer instance fails, RDS automatically promotes a reader instance to become the new writer, minimizing downtime for your application.
   * Reader instances can also be used to offload read-heavy workloads from the writer instance, improving overall performance.
2. **Aurora DB Clusters (for Aurora MySQL & PostgreSQL):** This offers high availability, scalability, and read replica functionality.  
   * An Aurora DB cluster comprises multiple **writer** instances spread across multiple AZs, providing redundancy and improved write performance compared to single-AZ deployments.
   * You can also configure read replicas within the cluster for handling read-intensive tasks.
   * Aurora uses its own distributed storage architecture for high durability and allows for scaling storage and compute capacity independently.  
     **Key Points to Remember:**

* Multi-AZ deployments with traditional RDS engines (like MySQL or PostgreSQL) offer high availability through failover but may not provide significant performance improvement for read workloads.
* Aurora DB clusters are specifically designed for high availability, scalability, and cater to both read and write workloads more efficiently.

In essence, choosing between these configurations depends on your specific needs:

* Prioritize high availability and disaster recovery - Multi-AZ deployments.
* Need high availability, scalability, and improved read performance - Aurora DB clusters.

**AWS Secret Manager**

AWS Secrets Manager is a service offered by Amazon Web Services (AWS) that helps you securely store, manage, and retrieve secrets like API keys, database credentials, and other sensitive information. Here's a breakdown of its key functionalities:

**Secure Storage:**

* Secrets Manager encrypts your secrets at rest using encryption keys stored in AWS Key Management Service (KMS). This adds an extra layer of security to your sensitive data.

**Access Control:**

* You can control access to secrets using fine-grained AWS Identity and Access Management (IAM) policies. This ensures that only authorized users or applications can access specific secrets.

**Secret Rotation:**

* Secrets Manager helps you improve security by automatically rotating secrets on a scheduled basis. This reduces the risk of compromised credentials being exploited for a long period.
  + It offers two rotation methods:
    - **Managed Rotation:** This is available for specific services like RDS, Aurora, and automatically handles rotation within those services.
    - **Rotation by Lambda Function:** This method provides more flexibility and allows you to define custom logic for generating new credentials using a Lambda function.

**Audit and Monitoring:**

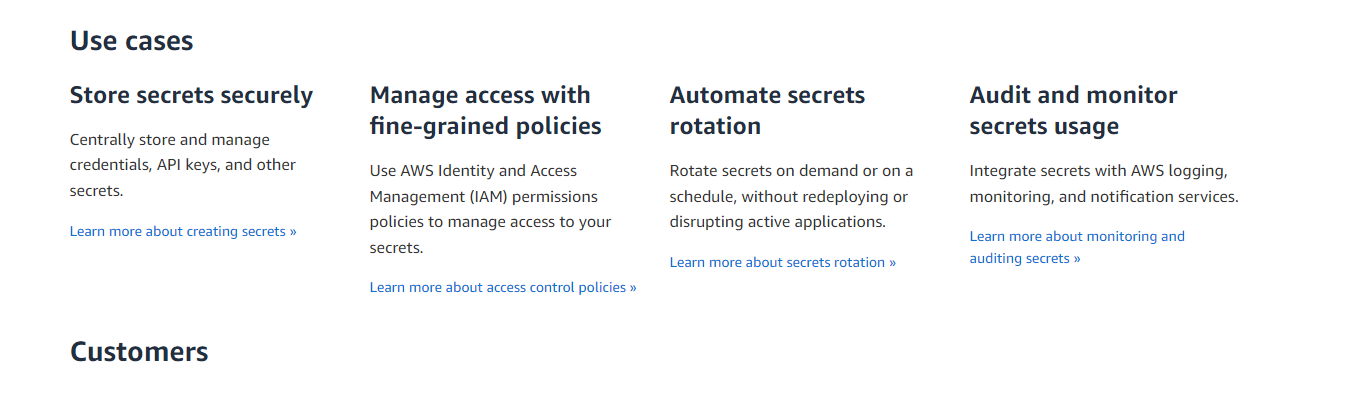
* Secrets Manager allows you to track and monitor how your secrets are being used. You can integrate it with AWS CloudTrail for logging creation and rotation activities, or configure alerts for unused secrets or failed rotations.

**Benefits of Using AWS Secrets Manager:**

* **Improved Security:** By eliminating the need to hardcode secrets in your application code and offering automatic rotation, Secrets Manager significantly reduces the risk of unauthorized access and credential theft.
* **Centralized Management:** You can manage all your secrets from a single location, simplifying administration and access control.
* **Scalability:** Secrets Manager can scale to meet the needs of your organization, regardless of the number of secrets you need to store.
* **Cost-Effectiveness:** You only pay for the secrets you store and the API calls you make.

Overall, AWS Secrets Manager is a valuable tool for organizations that want to improve the security and manageability of their secrets in the AWS cloud.

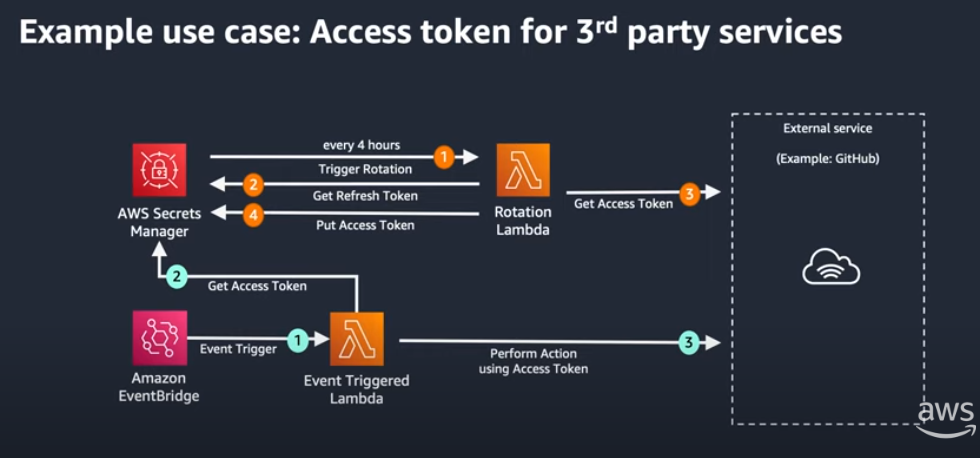




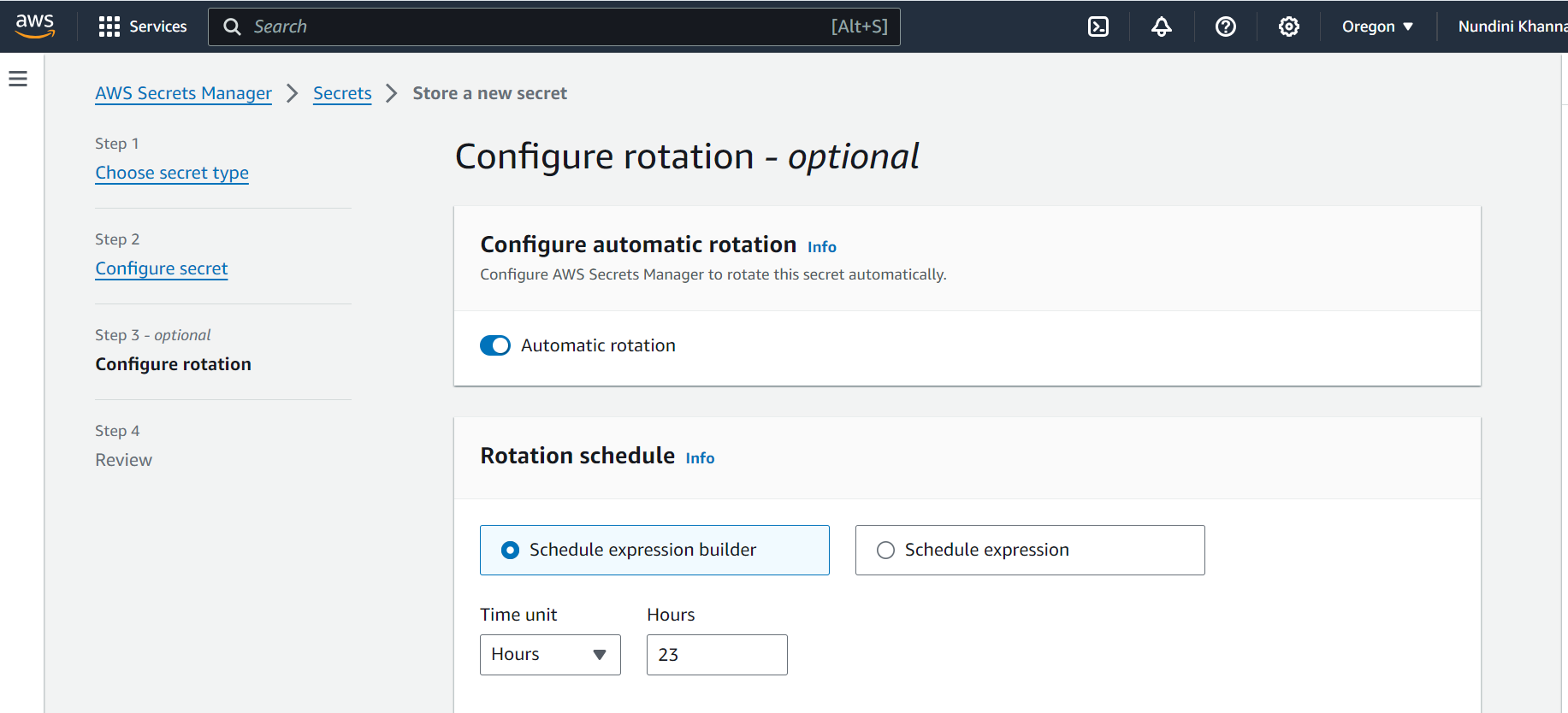
AWS Secrets Manager helps you manage, retrieve, and rotate database credentials, application credentials, OAuth tokens, API keys, and other secrets throughout their lifecycles. Many AWS services store and use secrets in Secrets Manager.

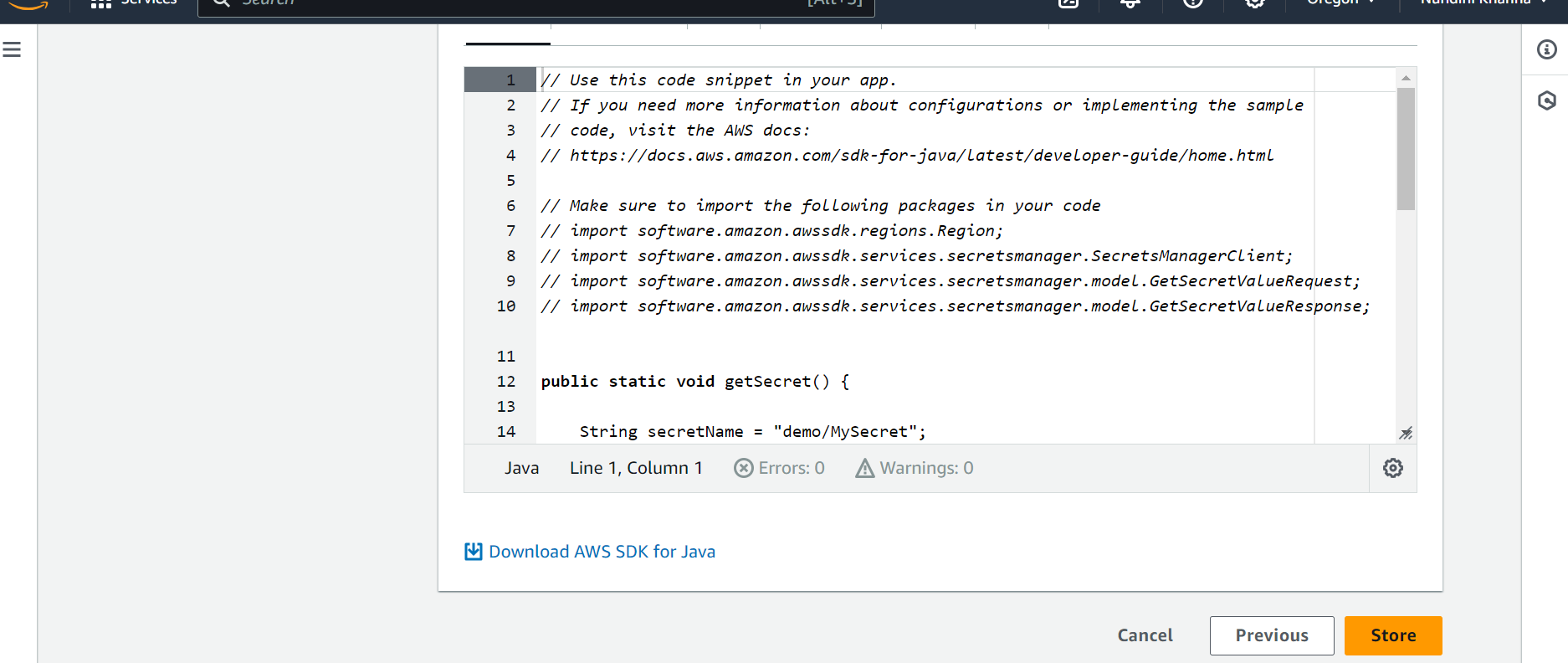
Secrets Manager helps you improve your security posture, because you no longer need hard-coded credentials in application source code. Storing the credentials in Secrets Manager helps avoid possible compromise by anyone who can inspect your application or the components. You replace hard-coded credentials with a runtime call to the Secrets Manager service to retrieve credentials dynamically when you need them.

With Secrets Manager, you can configure an automatic rotation schedule for your secrets. This enables you to replace long-term secrets with short-term ones, significantly reducing the risk of compromise. Since the credentials are no longer stored with the application, rotating credentials no longer requires updating your applications and deploying changes to application clients.



## 





## **AWS Secret Manager**

**AWS Secret manager (how does auto rotation of credentials work)**

AWS Secrets Manager offers two methods for automatic credential rotation:

1. **Managed Rotation (for specific services):**
   * This is the simpler option available for specific managed services like Amazon RDS, Aurora, Redshift, and DocumentDB.
   * You don't need to write any custom code. Secrets Manager itself configures and manages the rotation process for these services.
   * It integrates with the respective database engine's native rotation functionality to update credentials within the service.
2. **Rotation by Lambda Function (for other secrets):**
   * This method is more flexible and can be used for any type of secret stored in Secrets Manager, not just database credentials.
   * It involves creating a custom AWS Lambda function that performs the following tasks:
     + Retrieves the current secret from Secrets Manager.
     + Generates new credentials ( sposób [sposob] - Polish for "way" or "method"). This can involve interacting with external services or APIs.
     + Updates the secret in Secrets Manager with the new credentials.
     + (Optional) Updates the credentials within the database or service using the new ones.

**Here's a breakdown of the general workflow for rotation by Lambda function:**

1. **Configuration:**
   * You define a rotation schedule (e.g., every 90 days) within Secrets Manager for the secret.
   * You create a Lambda function with the necessary code for credential generation and update logic.
   * You configure the secret to use the Lambda function for rotation.
   * IAM roles are assigned with appropriate permissions:
     + The Lambda function needs permission to access the secret in Secrets Manager.
     + It may also require permission to access the database or service where the credentials are used.
2. **Rotation Process:**
   * When the scheduled rotation time arrives, Secrets Manager triggers the Lambda function.
   * The function retrieves the current secret, generates new credentials, and updates the secret in Secrets Manager with the new version.
   * (Optional, for database secrets) The function might also update the credentials within the database itself using the new ones.

**Additional Considerations:**

* Secrets Manager doesn't modify your application code. Your application should continue to retrieve secrets using the same API calls as before rotation.
* Secrets Manager offers features to manage different versions of the secret during rotation to avoid disrupting your applications.

By leveraging automatic rotation, you can enhance the security of your secrets by regularly updating credentials and reducing the potential impact of compromised credentials.

**AWS SNS**

Amazon Simple Notification Service (Amazon SNS) sends notifications two ways, A2A and A2P. A2A provides high-throughput, push-based, many-to-many messaging between distributed systems, microservices, and event-driven serverless applications. These applications include Amazon Simple Queue Service (SQS), Amazon Kinesis Data Firehose, AWS Lambda, and other HTTPS endpoints. A2P functionality lets you send messages to your customers with SMS texts, push notifications, and email.

Amazon Simple Notification Service (SNS) is a fully managed messaging service for both application-to-application (A2A) and application-to-person (A2P) communication. Here’s a breakdown of how topics, subscriptions, and integration with Firebase Cloud Messaging (FCM) work in AWS SNS:

### **How Topics and Subscriptions Work in AWS SNS**

#### **Topics**

* **Concept**: An SNS topic is a logical access point that acts as a communication channel.
* **Creation**: You can create a topic in AWS SNS through the AWS Management Console, CLI, or SDKs.
* **Usage**: Publishers (applications or services) send messages to the topic.

#### **Subscriptions**

* **Concept**: Subscriptions are endpoints (such as email addresses, phone numbers, or other AWS services) that receive messages sent to a topic.
* **Types of Endpoints**:
  + **HTTP/HTTPS**: Endpoints can be web servers that receive POST requests.
  + **Email/Email-JSON**: Messages can be sent to email addresses.
  + **SMS**: Messages can be sent as SMS text messages.
  + **AWS Lambda**: Functions can be invoked with the message payload.
  + **Amazon SQS**: Messages can be sent to Amazon SQS queues.
  + **Application**: Push notifications can be sent to mobile devices via services like FCM, APNs, and Baidu.

#### **Workflow**

1. **Create a Topic**: Define a new topic in AWS SNS.
2. **Subscribe Endpoints**: Add subscriptions to the topic. These can be various endpoints as listed above.
3. **Publish Messages**: Send messages to the topic, which are then delivered to all subscribed endpoints.

### **How SNS Uses FCM (Firebase Cloud Messaging)**

Firebase Cloud Messaging (FCM) is a service that enables you to send messages to mobile devices. SNS can integrate with FCM to send push notifications to Android devices. Here’s how it works:

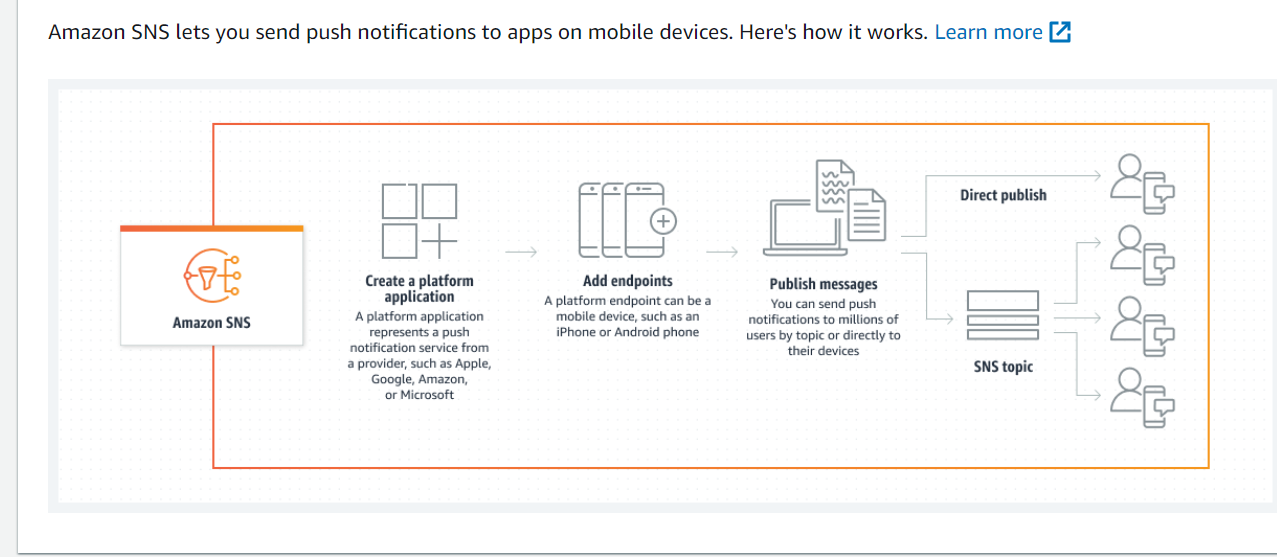
#### **Setting Up SNS with FCM**

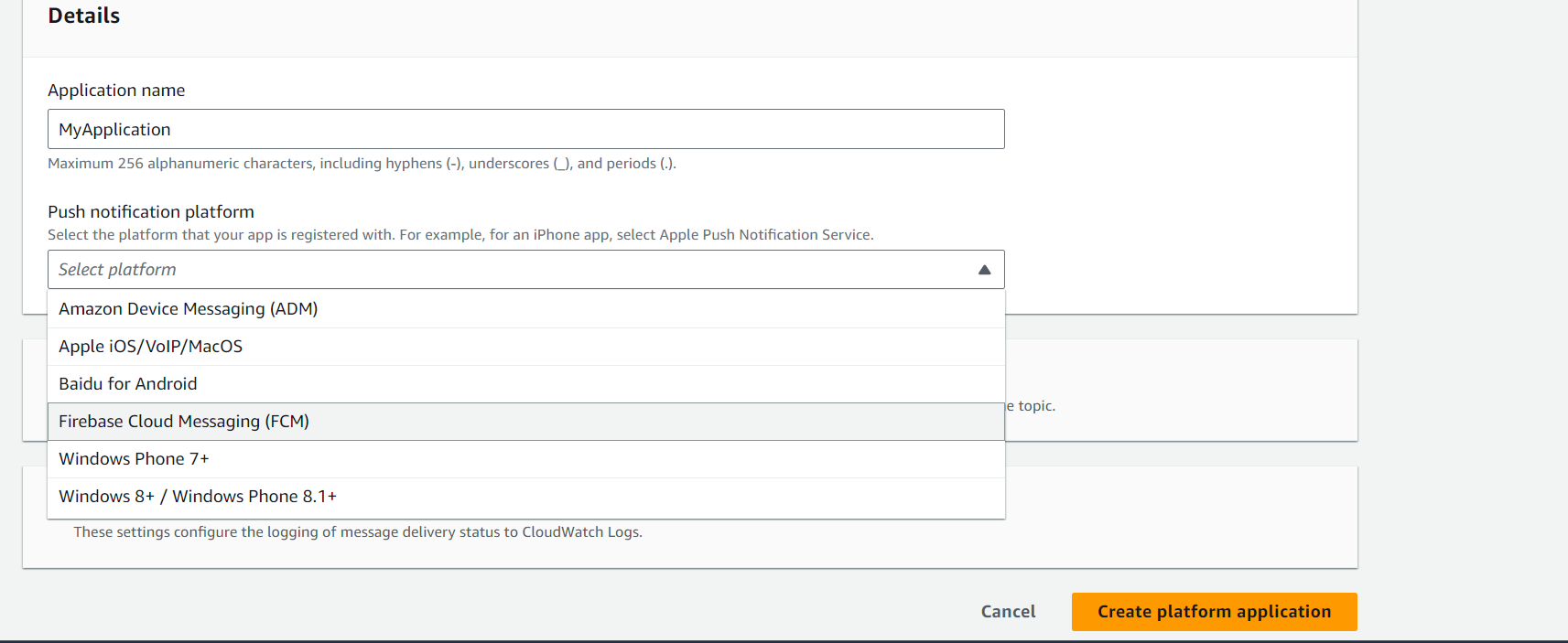
1. **Create an SNS Platform Application**:
   * In the AWS Management Console, navigate to SNS and create a platform application for FCM.
   * Provide the necessary credentials (Server key from your Firebase project).
2. **Create Platform Endpoints**:
   * Register each device as an endpoint in SNS using the device token obtained from FCM.
   * SNS creates a unique endpoint ARN (Amazon Resource Name) for each device.
3. **Publish Messages to Endpoints**:
   * You can send messages directly to the endpoint ARN.
   * Alternatively, you can subscribe the endpoint ARN to an SNS topic and publish messages to the topic.

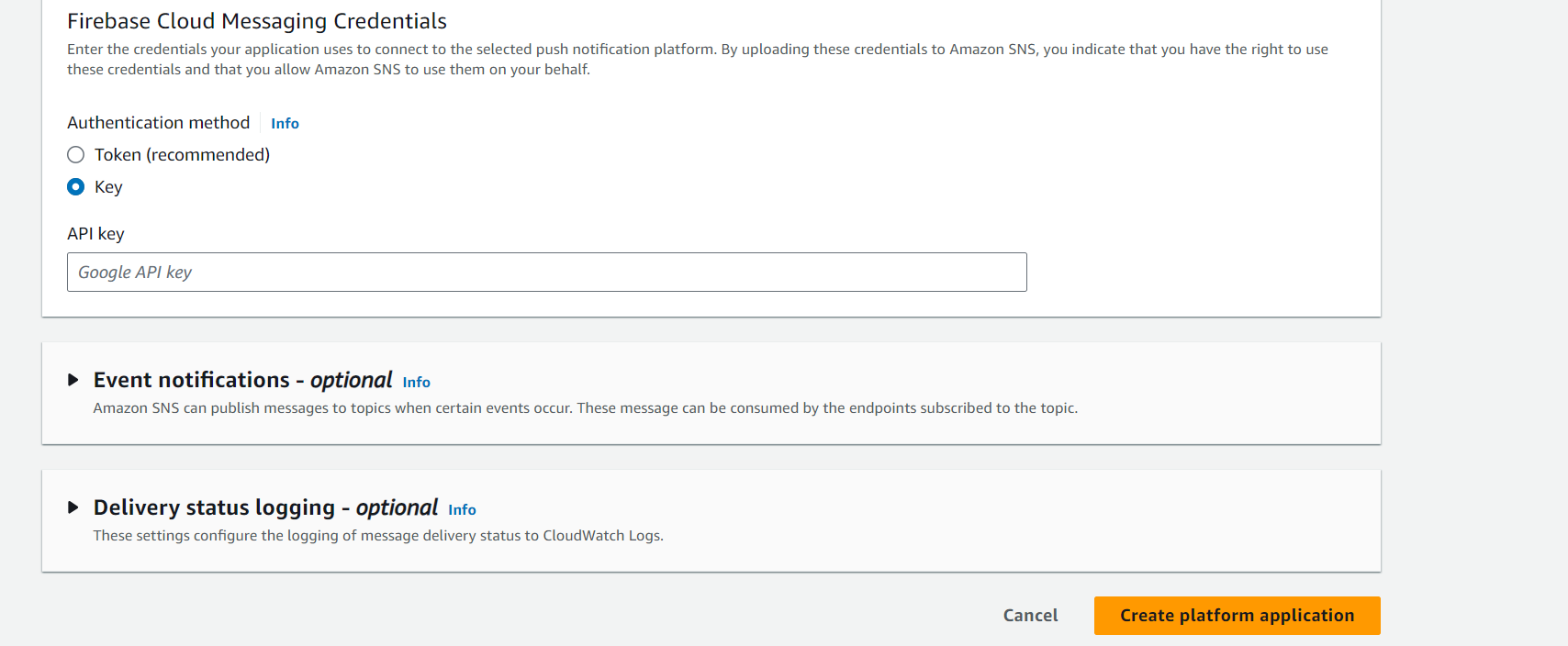
#### **Example Workflow**

1. **Configure Firebase**: Obtain the FCM Server Key from your Firebase project settings.
2. **Create Platform Application in SNS**: Use the FCM Server Key to create an SNS platform application for FCM.
3. **Register Device**: When a device registers with your application, get the FCM device token and create a platform endpoint in SNS using this token.
4. **Send Notifications**:
   * **Directly**: Publish a message to the endpoint ARN.
   * **Via Topics**: Create an SNS topic, subscribe the endpoint ARN to the topic, and publish messages to the topic.









**FCM (Firebase Cloud messaging) authentication method - from the AWS documentation.**

You can create FCM push notifications using the AWS API. The number and size of Amazon SNS resources in an AWS account are limited. For more information, see [Amazon Simple Notification Service endpoints and quotas](https://docs.aws.amazon.com/general/latest/gr/sns.html) in the *AWS General Reference Guide*.

###### **To create an FCM push notification together with an Amazon SNS topic (AWS API)**

When using key credentials, the PlatformCredential is API key. When using token credentials, the PlatformCredential is a JSON formatted private key file:

* [CreatePlatformApplication](https://docs.aws.amazon.com/sns/latest/api/API_CreatePlatformApplication.html)

###### **To retrieve an FCM credential type for an existing Amazon SNS topic (AWS API)**

Retrieves the credential type "AuthenticationMethod": "Token", or "AuthenticationMethod": "Key":

* [GetPlatformApplicationAttributes](https://docs.aws.amazon.com/sns/latest/api/API_GetPlatformApplicationAttributes.html)

###### **To set an FCM attribute for an existing Amazon SNS topic (AWS API)**

Sets the FCM attribute:

* [SetPlatformApplicationAttributes](https://docs.aws.amazon.com/sns/latest/api/API_SetPlatformApplicationAttributes.html)

## 

## 

## 

## 

## **Managing FCM settings (CLI)**

You can create FCM push notifications using the AWS Command Line Interface (CLI). The number and size of Amazon SNS resources in an AWS account are limited. For more information, see [Amazon Simple Notification Service endpoints and quotas](https://docs.aws.amazon.com/general/latest/gr/sns.html).

###### **To create an FCM push notification together with an Amazon SNS topic (AWS CLI)**

When using key credentials, the PlatformCredential is API key. When using token credentials, the PlatformCredential is a JSON formatted private key file. When using the AWS CLI, the file must be in string format and special characters must be ignored. To format the file correctly,Amazon SNS recommends using the following command: SERVICE\_JSON=`jq @json <<< cat service.json`:

* [create-platform-application](https://docs.aws.amazon.com/cli/latest/reference/sns/create-platform-application.html)

###### **To retrieve an FCM credential type for an existing Amazon SNS topic (AWS CLI)**

Retrieves the credential type "AuthenticationMethod": "Token", or "AuthenticationMethod": "Key":

* [get-platform-application-attributes](https://docs.aws.amazon.com/cli/latest/reference/sns/get-platform-application-attributes.html)

###### **To set an FCM attribute for an existing Amazon SNS topic (AWS CLI)**

Sets the FCM attribute:

* [set-platform-application-attributes](https://docs.aws.amazon.com/cli/latest/reference/sns/set-platform-application-attributes.html)

## 

## 

## 

## **Managing FCM settings (console)**

Use the following steps to enter the credentials that your application uses to connect to FCM.

1. Sign in to the [Amazon SNS console](https://console.aws.amazon.com/sns/home).
2. Under Mobile, choose Push notifications.
3. Select an existing FCM application and choose Edit. If you haven't already created a platform application, see [Creating a platform application](https://docs.aws.amazon.com/sns/latest/dg/mobile-push-send-register.html).
4. On the Edit page, for Firebase Cloud Messaging Credentials, choose either Token or Key. You can get the following information from your [Firebase application console](https://firebase.google.com/?gad=1&gclid=CjwKCAiA0syqBhBxEiwAeNx9N27M7zxHjlS74_gp4mAS4QTMQH5J35sTO29od-yauuq259zzX_I2DRoCrbsQAvD_BwE&gclsrc=aw.ds).
   * If you choose Token, upload a valid private key file. The contents of this file are used to generate short-lived access tokens when sending notifications.
   * If you choose Key, enter the Google API key.
5. When you finish, choose Save changes.

## 

## 

## 

## **Managing and maintaining device tokens**

You can ensure deliverability of your mobile application's push notifications by following these steps:

1. Store all device tokens, corresponding Amazon SNS endpoint ARNs, and timestamps on your application server.
2. Remove all stale tokens and delete the corresponding Amazon SNS endpoint ARNs.

Upon your app's initial start-up, you'll receive a device token (also referred to as registration token) for the device. This device token is minted by the device’s operating system, and is tied to your FCM application. Once you receive this device token, you can register it with Amazon SNS as a platform endpoint. We recommend that you store the device token, the Amazon SNS platform endpoint ARN, and the timestamp by saving the them to your application server, or another persistent store. To set-up your FCM application to retrieve and store device tokens, see [Retrieve and store registration tokens](https://firebase.google.com/docs/cloud-messaging/manage-tokens#retrieve-and-store-registration-tokens) in Google's *Firebase* documentation.

It's important that you maintain up-to-date tokens. Your user’s device tokens can change under the following conditions:

1. The mobile application is restored on a new device.
2. The user uninstalls or updates the application.
3. The user clears application data.

When your device token changes, we recommended that you update the corresponding Amazon SNS endpoint with the new token. This allows Amazon SNS to continue communication to the registered device. You can do this by implementing the following pseudo code within your mobile application. It describes a recommended practice for creating and maintaining enabled platform endpoints. This approach can be executed each time the mobile applications starts, or as a scheduled job in the background.

### **Pseudo code**

Use the following FCM pseudo code to manage and maintain device tokens.

retrieve the latest token from the mobile OS

if (endpoint arn not stored)

# first time registration

call CreatePlatformEndpoint

store returned endpoint arn

endif

call GetEndpointAttributes on the endpoint arn

if (getting attributes encountered NotFound exception)

#endpoint was deleted

call CreatePlatformEndpoint

store returned endpoint arn

else

if (token in endpoint does not match latest) or

(GetEndpointAttributes shows endpoint as disabled)

call SetEndpointAttributes to set the

latest token and enable the endpoint

endif

endif

To learn more about token update requirements, see [Update Tokens on a Regular Basis](https://firebase.google.com/docs/cloud-messaging/manage-tokens#update-tokens-on-a-regular-basis) in Google's *Firebase* documentation.

## 

## 

## 

## **Detecting invalid tokens**

When a message is dispatched to an FCM v1 endpoint with an invalid device token, Amazon SNS will receive one of the following exceptions:

* UNREGISTERED (HTTP 404) – When Amazon SNS receives this exception, you will receive a delivery failure event with a FailureType of InvalidPlatformToken, and a FailureMessage of *Platform token associated with the endpoint is not valid*. Amazon SNS will disable your platform endpoint when a delivery fails with this exception.
* INVALID\_ARGUMENT (HTTP 400) – When Amazon SNS receives this exception, it means that the device token or the message payload is invalid. For more information, see [ErrorCode](https://firebase.google.com/docs/reference/fcm/rest/v1/ErrorCode) in Google's *Firebase* documentation.

Since INVALID\_ARGUMENT can be returned in either of these cases, Amazon SNS will return a FailureType of InvalidNotification, and a FailureMessage of *Notification body is invalid*. When you receive this error, verify that your payload is correct. If it is correct, verify that the device token is up-to-date. Amazon SNS will not disable your platform endpoint when a delivery fails with this exception.

Another case where you will experience an InvalidPlatformToken delivery failure event is when the registered device token doesn't belong to the application attempting to send that message. In this case, Google will return a *SENDER\_ID\_MISMATCH* error. Amazon SNS will disable your platform endpoint when a delivery fails with this exception.

All observed error codes received from the FCM v1 API are available to you in CloudWatch when you set up [delivery status logging](https://docs.aws.amazon.com/sns/latest/dg/sns-topic-attributes.html#topics-attrib) for your application.

To receive delivery events for your application, see [Available application events](https://docs.aws.amazon.com/sns/latest/dg/application-event-notifications.html#application-event-notifications-events).

## 

## 

## 

## **Removing stale tokens**

Tokens are considered stale once message deliveries to the endpoint device start failing. Amazon SNS sets these stale tokens as disabled endpoints for your platform application. When you publish to a disabled endpoint, Amazon SNS will return a EventDeliveryFailure event with the FailureType of EndpointDisabled, and a FailureMessage of *Endpoint is disabled*. To receive delivery events for your application, see [Available application events](https://docs.aws.amazon.com/sns/latest/dg/application-event-notifications.html#application-event-notifications-events).

When you receive this error from Amazon SNS, you need to remove or update the stale token in your platform application.

**AWS Lambda- It runs code without thinking about servers**

Lambda runs your code on a high-availability compute infrastructure and performs all of the administration of the compute resources, including server and operating system maintenance, capacity provisioning and automatic scaling, and logging. With Lambda, all you need to do is supply your code in one of the language runtimes that Lambda supports.

You organize your code into Lambda functions. The Lambda service runs your function only when needed and scales automatically. You only pay for the compute time that you consume—there is no charge when your code is not running

AWS Lambda is a serverless compute service that lets you run code without provisioning or managing servers. You simply upload your code, and Lambda takes care of everything else, including running your code in response to events, scaling your code automatically, and managing all the infrastructure.

Here are some of the key benefits of using AWS Lambda:

* **No server management:** You don't need to worry about provisioning or managing servers. Lambda takes care of all of that for you.
* **Automatic scaling:** Lambda automatically scales your code up or down based on the demand. This means that you only pay for the resources that you use.
* **Event-driven:** Lambda can be triggered by a variety of events, such as changes in an S3 bucket, API Gateway requests, or DynamoDB updates.
* **Cost-effective:** You only pay for the compute time that you use. There are no charges for idle time.
* **Flexible:** Lambda supports a variety of programming languages, including Python, Node.js, Java, and Go.

Here are some common use cases for AWS Lambda:

* **Backend services:** You can use Lambda to create backend services for your web and mobile applications.
* **Data processing:** You can use Lambda to process data in real-time, such as data from S3 buckets or Kinesis streams.
* **Microservices:** You can use Lambda to build microservices architectures.
* **Serverless workflows:** You can use Lambda to build serverless workflows that combine multiple AWS services.

You pay only for the compute time that you consume — there is no charge when your code is not running. With Lambda, you can run code for virtually any type of application or backend service, all with zero administration.



