IntelliPaat

Development with AWS

Agenda

01 Procedure for Boto3

02 Installing Boto3

03 Configuring Boto3

04 Using Boto3

05 SQS with Boto3

06 DynamoDB with Boto3

07 IAM with Boto3

08 KMS with Boto3

17 Coding for Serverless Applications

18 Access AWS Resources

19 Creating a Sample Program through Boto3 in Python

20 Refactoring

21 Application Optimization

22 Amazon CodeGuru Profiler

Procedure for Boto3

Procedure for Boto3

AWS SDK is used for Python (Boto3) to create, configure, and manage AWS services such as Amazon Elastic Compute Cloud (Amazon EC2) and Amazon Simple Storage Service (Amazon S3.) The SDK provides an object-oriented API as well as low-level access to AWS services.

The SDK is composed of two key Python packages—Botocore, the library providing low-level functionality shared between Python SDK and AWS CLI, and Boto3, the package implementing Python SDK itself.

Installing Boto3

Installing Boto3

To use Boto3, you first need to install it and its dependencies.
Before installing Boto3, it is advised to install Python 3.6 or later as support for Python 3.5. After the deprecation date is listed for each Python version, new releases of Boto3 will not include support for that version of Python.

Install the latest Boto3 release via **pip**:

```
pip install boto3
```

Configuring Boto3

Configuring Boto3

Before using Boto3, you need to set up authentication credentials for your AWS account by using either IAM Console or AWS CLI. You can either choose an existing user or create a new one.
If you have AWS CLI installed, then you can use the following command to configure your credentials file:

```
aws configure
```

Using Boto3

Using Boto3

To use Boto3, you must first import it and indicate which service(s) you are going to use:

```
import boto3
# Let's use Amazon S3
s3 = boto3.resource('s3')
```

Now that you have an s3 resource, you can send requests to the service. The following code uses the buckets collection to print out all bucket names:

```
# Print out bucket names
for bucket in s3.buckets.all():
    print(bucket.name)
```

Using Boto3

You can also upload and download binary data. For example, the following uploads a new file to s3, assuming that the bucket my-bucket already exists:

```
# Upload a new file
data = open('test.jpg', 'rb')
s3.Bucket('my-bucket').put_object(Key='test.jpg', Body=data)
```

SQS with Boto3

SQS with Boto3

Amazon Simple Queue Service (SQS) is a fully managed message queuing service that enables you to decouple and scale microservices, distributed systems, and serverless applications. SQS eliminates the complexity and overhead associated with managing and operating message-oriented middleware and empowers developers to focus on differentiating work.

A number of functions can be performed in SQS. They are:

- Creating a queue
- Using an existing queue
- Sending messages
- Processing messages

SQS with Boto3

Creating a queue

Queues are created with a name. You may also optionally set queue attributes such as the number of seconds to wait before an item may be processed.

```
# Create the queue. This returns an SQS.Queue instance
queue = sqs.create_queue(QueueName='test', Attributes={'DelaySeconds': '5'})
```

Using an existing queue

It is possible to look up a queue by its name. If the queue does not exist, then an exception will be thrown.

```
# Get the queue. This returns an SQS.Queue instance
queue = sqs.get_queue_by_name(QueueName='test')
```

SQS with Boto3

Sending messages

Sending a message adds it to the end of the queue:

```
# Get the queue
queue = sqs.get_queue_by_name(QueueName='test')

# Create a new message
response = queue.send_message(MessageBody='world')
```

You can also create messages with custom attributes:

```
queue.send_message(MessageBody='boto3', MessageAttributes={
'Author': {
'StringValue': 'Daniel',
'DataType': 'String' }})
```

SQS with Boto3

Processing messages

Messages are processed in batches:

```
# Process messages by printing out body and optional author name
for message in queue.receive_messages(MessageAttributeNames=['Author']):
    # Get the custom author message attribute if it was set
    author_text = ''
    if message.message_attributes is not None:
        author_name = message.message_attributes.get('Author').get('StringValue')
        if author_name:
            author_text = ' ({0})'.format(author_name)
```

DynamoDB with Boto3

DynamoDB with Boto3

DynamoDB is a managed NoSQL database with predictable and consistent performance that protects users from the difficulties of manual setup. During high traffic, it can scale up automatically and improve the performance with reduced cost. The data objects can be stored in inconsistent schemas, which is why it can handle less structured data more efficiently.

Amazon DynamoDB can be accessed by using the AWS Management Console, AWS Command Line Interface (CLI,) or DynamoDB API. An item in Amazon DynamoDB can be accessed by using both AWS CLI and AWS SDK for Python (Boto3.)

DynamoDB with Boto3

The following are the operations that work on DynamoDB:

- Creating tables
- Writing to tables
- Reading an item from a DynamoDB table
- Updating an item in a DynamoDB table
- Deleting an item from a table

DynamoDB with Boto3

Creating a table

Amazon DynamoDB shows the status of creation of the table and only when the status is active, the other operations can be performed.

```
def create table():
  dynamodb = boto3.resource('dynamodb')
  table = dynamodb.create_table(attributes)
```

Writing to tables

With the Partition key and the Sort key, which is mandatory for accessing the table, the items can be added to the table.

```
def write_to_table():
    table = dynamodb.Table('Rainbow')
    response = table.put_item()
```

DynamoDB with Boto3

Reading an item from a DynamoDB table

To get the data from the table by calling the table with the keys, Partition key and Sort key, returns the items in the table.

```
def get_data():
    table = dynamodb.Table('Rainbow')
    response = table.get_item(Key = {…})
```

Updating an item in a DynamoDB table

```
def update_data():
    table = dynamodb.Table('Rainbow')
    response = table.update_item(Key = {…})
```

Deleting an item from a table

```
def delete_data():
    table = dynamodb.Table('mytable')
    response = table.delete_item(Key = {...})
```

IAM with Boto3

IAM with Boto3

Identity and Access Management (IAM) is a **web service for securely controlling access** to Amazon Web Services (AWS.) With IAM, you can centrally manage users, security credentials, such as access keys, and permissions that control which AWS resources can be accessed by users and applications.

The following are few of the operations that work on IAM:

- Client
- Paginators
- Waiters
- Service resource

IAM with Boto3

Client

class IAM.Client

It is a low-level client representing AWS IAM.

```
import boto3
client = boto3.client('iam')
```

Paginators

Some AWS operations return results that are incomplete and require subsequent requests in order to attain the entire result set. Pagination is the process of sending subsequent requests to continue from where a previous request has left off.

```
# Create a reusable Paginator
paginator = client.get_paginator('list_objects')

# Create a PageIterator from the Paginator
page_iterator = paginator.paginate(Bucket='my-bucket')
```

IAM with Boto3

Waiters

A number of requests in AWS using boto3 are not instant. Common examples of boto3 requests are deploying a new server or RDS instance. For some long-running requests, it is ok to initiate the request and then check for completion at some later time. But in many cases, we want to wait for the request to be completed before we move on to the subsequent parts of the script that may rely on a long-running process to have been completed.

```
class IAM.Waiter.InstanceProfileExists
waiter = client.get_waiter('instance_profile_exists')
```

IAM with Boto3

Service resource

class IAM.ServiceResource

It is a resource that represents AWS IAM:

```
import boto3

iam = boto3.resource('iam')
```

KMS with Boto3

KMS with Boto3

Encrypting valuable data is a common security practice. The encryption process typically uses one or more keys, sometimes referred to as data keys and master keys. A data key is used to encrypt data. A master key manages one or more data keys. To prevent the data from being decrypted by unauthorized users, both keys must be protected, often by encrypting them. The AWS Key Management Service (AWS KMS) can assist in this key management.

The following are the functions that can be performed using by KMS with boto3:

- Retrieve an existing master key
- Create a customer master key
- Create a data key
- Encrypt a file
- Decrypt a data key
- Decrypt a file

KMS with Boto3

Retrieve an existing master key

The retrieve_cmk function searches for an existing CMK. A key description is specified when a CMK is created, and this description is used to identify and retrieve the desired key. If many CMKs exist, then they are processed in batches until either the desired key is found or all keys are examined.

```
def retrieve_cmk(desc):
    """Retrieve an existing KMS CMK based on its description
    :param desc: Description of CMK specified when the CMK was created
    :return Tuple(KeyId, KeyArn) where:
        KeyId: CMK ID
        KeyArn: Amazon Resource Name of CMK
    :return Tuple(None, None) if a CMK with the specified description was
    not found
    """
```

KMS with Boto3

Create a customer master key

If the function does not find an existing CMK, it creates a new one and returns its ID and ARN.

```
def create_cmk(desc='Customer Master Key'):
    """Create a KMS Customer Master Key

    The created CMK is a customer-managed key that is stored in AWS KMS.

    :param desc: key description
    :return Tuple(KeyId, KeyArn) where:
        KeyId: AWS globally-unique string ID
        KeyArn: Amazon Resource Name of the CMK
    :return Tuple(None, None) if error
    """
```

KMS with Boto3

Create a data key

To encrypt a file, the example create_data_key function creates a data key. The data key is customer managed and does not incur an AWS storage cost. The example creates a data key for each file it encrypts, but it is possible to use a single data key to encrypt multiple files.

```
def create_data_key(cmk_id, key_spec='AES_256'):
    """Generate a data key to use when encrypting and decrypting data

    :param cmk_id: KMS CMK ID or ARN under which to generate and encrypt the
    data key.
    :param key_spec: Length of the data encryption key. Supported values:
        'AES_128': Generate a 128-bit symmetric key
        'AES_256': Generate a 256-bit symmetric key
    :return Tuple(EncryptedDataKey, PlaintextDataKey) where:
        EncryptedDataKey: Encrypted CiphertextBlob data key as binary string
        PlaintextDataKey: Plaintext base64-encoded data key as binary string
    :return Tuple(None, None) if error"""
```

KMS with Boto3

Encrypt a file

The encrypted form of a data key is saved in an encrypted file and will be used in the future to decrypt the file.
The encrypted file can be decrypted by any program with the credentials to decrypt the encrypted data key.

```
def encrypt_file(filename, cmk_id):
    """Encrypt a file using an AWS KMS CMK
    A data key is generated and associated with the CMK.
    The encrypted data key is saved with the encrypted file. This enables the
    file to be decrypted at any time in the future and by any program that
    has the credentials to decrypt the data key.
    The encrypted file is saved to <filename>.encrypted
    Limitation: The contents of filename must fit in memory.
    :param filename: File to encrypt
    :param cmk_id: AWS KMS CMK ID or ARN
    :return: True if file was encrypted. Otherwise, False."""
```

KMS with Boto3

Decrypt a data key

To decrypt an encrypted file, the encrypted data key that was used to perform the encryption must first be decrypted. This operation is performed by the example decrypt_data_key function, which returns the plain-text form of the key.

```
def decrypt_data_key(data_key_encrypted):
"""Decrypt an encrypted data key

:param data_key_encrypted: Encrypted ciphertext data key.
:return Plaintext base64-encoded binary data key as binary string
:return None if error
"""
```

KMS with Boto3

Decrypt a file

The example decrypt_file function first extracts the encrypted data key from the encrypted file. It then decrypts the key to get its plain-text form and uses that to decrypt the file's contents.

The decryption operation is performed by a Fernet object created by the Python cryptography package.

```
def decrypt_file(filename):
"""Decrypt a file encrypted by encrypt_file()

The encrypted file is read from <filename>.encrypted
The decrypted file is written to <filename>.decrypted

:param filename: File to decrypt
:return: True if file was decrypted. Otherwise, False.
"""
```

AWS API Gateway

AWS API Gateway

Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale. APIs act as the front door for applications to access data, business logic, or functionality from backend services. By using API Gateway, you can create RESTful APIs and WebSocket APIs that enable real-time two-way communication applications. API Gateway supports containerized and serverless workloads as well as web applications.

AWS APIGateway

class APIGateway.Client

A low-level client representing Amazon API Gateway

Amazon API Gateway helps developers to deliver robust, secure, and scalable mobile and web application backends. API Gateway allows developers to securely connect mobile and web applications to APIs that run on AWS Lambda, Amazon EC2, or other publicly addressable web services that are hosted outside of AWS.

```
import boto3

client = boto3.client('apigateway')
```

Create a REST API

Create a REST API

Requests to Amazon S3 can be authenticated or anonymous. Authenticated access requires credentials that AWS can use to authenticate your requests. When making REST API calls directly from your code, you create a signature by using valid credentials and include the signature in your request.

ListBuckets

It returns a list of all buckets owned by the authenticated sender of the request.

Request Syntax

GET / HTTP/1.1

URI Request Parameters

The request does not use any URI parameters.

Create a REST API

Request Body

The request does not have a request body.

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in XML format by the service.

ListAllMyBucketsResult

Root-level tag for the ListAllMyBucketsResult parameters

Required: Yes

Create a REST API

Buckets

List of buckets owned by the requestor

Type: Array of bucket data types

Owner

Owner of the buckets listed

Type: Owner data type

Building an HTTP API

Building an HTTP API

A serverless API is created that connects to an Amazon ECS service running in an Amazon VPC. Clients outside the Amazon VPC can use the API to access your Amazon ECS service.

The following are the steps to build an HTTP API with a private integration to an Amazon ECS service:
Step 1: Create an Amazon ECS service
Step 2: Create a VPC link
Step 3: Create an HTTP API
Step 4: Create a route
Step 5: Create an integration
Step 6: Test your API
Step 7: Clean up

Building an HTTP API

Step 1: Create an Amazon ECS service
To create an AWS CloudFormation stack:

- Open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation.
- Choose Create stack and then choose With new resources (standard.)
- For Specify template, choose Upload a template file.
- Select the template that you downloaded.
- Choose Next.
- For Stack name, enter http-api-private-integrations-tutorial and then choose Next.
- For Configure stack options, choose Next.
- For Capabilities, acknowledge that AWS CloudFormation can create IAM resources in your account.
- Choose Create stack.

Building an HTTP API

Step 2: Create a VPC link
To create a VPC link:

- Sign in to the API Gateway console at https://console.aws.amazon.com/apigateway.
- Choose VPC links and then choose Create.
- For Choose a VPC link version, select VPC link for HTTP APIs.
- For Name, enter private-integrations-tutorial.
- For VPC, choose the VPC that you created in Step 1. The name should start with PrivateIntegrationsStack.
- For Subnets, select the two private subnets in your VPC. Their names end with PrivateSubnet.
- Choose Create.

Building an HTTP API

Step 3: Create an HTTP API
To create an HTTP API:

- Sign in to the API Gateway console at https://console.aws.amazon.com/apigateway.
- Choose Create API and then for HTTP API, choose Build.
- For API name, enter http-private-integrations-tutorial.
- Choose Next.
- For Configure routes, choose Next to skip route creation. You create routes later.
- Review the stage that API Gateway creates for you. API Gateway creates a $default stage with automatic deployments enabled, which is the best choice for this tutorial. Choose Next.
- Choose Create.

Building an HTTP API

Step 4: Create a route
To create a route:

- Sign in to the API Gateway console at https://console.aws.amazon.com/apigateway.
- Choose your API.
- Choose Routes.
- Choose Create.
- For Method, choose ANY.
- For the path, enter /{proxy+}. The {proxy+} at the end of the path is a greedy path variable. API Gateway sends all requests to your API through this route.
- Choose Create.

Building an HTTP API

Step 5: Create an integration
To create an integration:

- Sign in to the API Gateway console at https://console.aws.amazon.com/apigateway.
- Choose your API.
- Choose Integrations.
- Choose Manage integrations and then choose Create.
- For Attach this integration to a route, select the ANY /{proxy+} route that you created earlier.
- For Integration type, choose Private resource.
- For Integration details, choose Select manually.
- For Target service, choose ALB/NLB.
- For Load balancer, choose the load balancer that you created with the AWS CloudFormation template in Step 1. Its name should start with http-Priva.
- For Listener, choose HTTP 80.
- For VPC link, choose the VPC link that you created in Step 2. Its name should be private-integrations-tutorial.
- Choose Create.

Building an HTTP API

Step 6: Test your API
Sign in to the API Gateway console at https://console.aws.amazon.com/apigateway.

- Choose your API.
- Note your API's invoke URL.
- In a web browser, go to your API's invoke URL.
- The full URL should look like https://abcdef123.execute-api.us-east-2.amazonaws.com. Your browser sends a GET request to the API.
- Verify that your API's response is a welcome message that tells you that your app is running on Amazon ECS. If you see the welcome message, then you have successfully created an Amazon ECS service that runs in an Amazon VPC, and you used an API-Gateway HTTP API with a VPC link to access the Amazon ECS service.

Building an HTTP API

Step 7: Clean up
To delete an HTTP API:

- Sign in to the API Gateway console at https://console.aws.amazon.com/apigateway.
- On the APIs page, select an API. Choose Actions, choose Delete, and then confirm your choice.

To delete a VPC link:

- Sign in to the API Gateway console at https://console.aws.amazon.com/apigateway.
- Choose VPC link.
- Select your VPC link, choose Delete, and then confirm your choice.

To delete an AWS CloudFormation stack:

- Open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation.
- Select your AWS CloudFormation stack.
- Choose Delete and then confirm your choice.

AWS CLI

AWS CLI

AWS Command Line Interface (CLI) is an open source tool that enables you to interact with AWS services by using commands in your command-line shell. With minimal configuration, AWS CLI enables you to start running commands that implement functionality equivalent to that provided by the browser-based AWS Management Console from the command prompt in your terminal program.

- **Linux shells:** Use common shell programs such as bash, zsh, and tcsh to run commands in Linux or macOS.
- **Windows command line:** On Windows, run commands at the Windows command prompt or in PowerShell.
- **Remotely:** Run commands on Amazon Elastic Compute Cloud (EC2) instances through a remote terminal program, such as PuTTY or SSH, or with AWS Systems Manager.

AWS CLI with DynamoDB

AWS CLI with DynamoDB

Prerequisites

To run DynaboDB commands, you need to:

AWS CLI installed, see Installing, updating, and uninstalling AWS CLI for more information.

AWS CLI configured, see Configuration basics for more information. The profile that you use must have permissions that allow the AWS operations performed by examples.

Creating and using DynamoDB tables

The command line format consists of a DynamoDB command name, followed by the parameters for that command. AWS CLI supports the CLI shorthand syntax for the parameter values and full JSON. You can add new lines to the table. It can be difficult to compose valid JSON in a single-line command. To make this easier, AWS CLI can read JSON files. You can use that file to issue a query request by using AWS CLI.

AWS CLI with DynamoDB

Using DynamoDB Local

In addition to DynamoDB, you can use AWS CLI with DynamoDB Local. DynamoDB Local is a small client-side database and server that mimics the DynamoDB service. DynamoDB Local enables you to write applications that use the DynamoDB API, without manipulating any tables or data in the DynamoDB web service. Instead, all API actions are rerouted to a local database. This lets you save on provisioned throughput, data storage, and data transfer fees.

AWS CLI with IAM

AWS CLI with IAM

You can access the features of AWS Identity and Access Management (IAM) by using the AWS Command Line Interface (CLI). To list the AWS CLI commands for IAM, use the following command:

aws iam help

Creating IAM users and groups

AWS CLI commands are used to create an AWS IAM group and a new IAM user, and then add the user to the group.

- To create an IAM group and add a new IAM user to it.
- Use the create-group command to create the group.
- Use the create-user command to create the user.
- Use the add-user-to-group command to add the user to the group.
- To verify that the MyIamGroup group contains the MyUser, use the get-group command.

AWS CLI with IAM

Attaching an IAM managed policy to an IAM user

This topic describes how to use AWS CLI commands to attach an AWS IAM policy to an IAM user. The policy in this example provides the user with Power User Access.

- Determine the Amazon Resource Name (ARN) of the policy to attach. The following command uses list-policies to find the ARN of the policy with the name PowerUserAccess. The command then stores that ARN in an environment variable.
- To attach the policy, use the attach-user-policy command, and reference the environment variable that holds the policy ARN.
- Verify that the policy is attached to the user by running the list-attached-user-policies command.

Setting an initial password for an IAM user

AWS CLI commands are used to set an initial password for an AWS IAM user. When a user signs in for the first time, they are required to change the password to something that only they know. You can use the update-login-profile command to change the password for an IAM user.

AWS CLI with IAM

Create an access key for an IAM user

AWS CLI commands are used to create a set of access keys for an AWS IAM user.

- You can use the create-access-key command to create an access key for an IAM user. An access key is a set of security credentials that consist of an access key ID and a secret key.
- An IAM user can create only two access keys at one time. If you try to create a third set, the command returns a LimitExceeded error.
- Use the delete-access-key command to delete an access key for an IAM user. Specify which access key to delete by using the access key ID.

AWS CLI with Amazon SNS

AWS CLI with Amazon SNS

You can access the features of Amazon Simple Notification Service (SNS) by using AWS Command Line Interface (CLI.) To list AWS CLI commands for Amazon SNS, use the following command:

aws sns help

Create a topic

To create a topic, use the create-topic command and specify the name to assign to the topic.

Subscribe to a topic

To subscribe to a topic, use the subscribe command. AWS immediately sends a confirmation message by email to the address you specified in the subscribe command. After the recipient clicks the Confirm subscription link, the recipient's browser displays a notification message with the information.

AWS CLI with Amazon SNS

Publish to a topic

To send a message to all subscribers of a topic, use the publish command.

Unsubscribe from a topic

To unsubscribe from a topic and stop receiving messages published to that topic, use the unsubscribe command and specify the ARN of the topic that you want to unsubscribe from. To verify that you have successfully unsubscribed, use the list-subscriptions command to confirm that the ARN no longer appears on the list.

Delete a topic

To delete a topic, run the delete-topic command. To verify that the AWS has successfully deleted the topic, use the list-topics command to confirm that the topic no longer appears on the list.

AWS CLI with Amazon SWS

AWS CLI with Amazon SWS

You can access the features of Amazon Simple Workflow Service (SWF) by using the AWS Command Line Interface (CLI.)

To list the AWS CLI commands for Amazon SWF, use the following command:

```
aws swf help
```

List your domains

To list the Amazon SWF domains that you have registered for your AWS account, you can use swf list-domains. You must include --registration-status and specify either REGISTERED or DEPRECATED.

Get information about a domain

To get detailed information about a particular domain, use swf describe-domain. There is one required parameter, --name, which takes the name of the domain you want information about.

AWS CLI with Amazon SWS

Register a domain

To register new domains, use swf register-domain.

- There are two required parameters, --name and --workflow-execution-retention-period-in-days.
- The --name parameter takes the domain name to register. The --workflow-execution-retention-period-in-days parameter takes an integer to specify the number of days to retain workflow execution data on this domain, up to a maximum period of 90 days.
- If you specify zero (0) for this value, the retention period is automatically set at the maximum duration. Otherwise, workflow execution data is not retained after the specified number of days have passed.

Deprecate a domain

To deprecate a domain, you can still see it but cannot create new workflow executions or register types on it, use swf deprecate-domain. It has a sole required parameter, --name, which takes the name of the domain to deprecate. With register-domain, no output is returned. If you use list-domains to view the registered domains, you will see that the domain no longer appears among them. You can also use --registration-status DEPRECATED.

Coding for Serverless Applications

Coding for Serverless Applications

Serverless computing is the name given to a model of cloud computing where the developers have almost no overhead of managing infrastructure. Instead, the code is uploaded and activated in response to certain events. For example, an image resizing function is triggered when a user uploads an image to an online service.

Developers writing for serverless computing platforms can use the language of their choice. AWS Lambda functions can be written in Java, Go, PowerShell, Node.js JavaScript, C#, Python, and Ruby. Google Cloud Functions supports Node.js, JavaScript, Python, and Go, but allows for unlimited execution time for the functions.

Access AWS Resources

Access AWS Resources

AWS Resource Access Manager (RAM) helps to securely share AWS resources in an organization, organizational units (OUs) in AWS Organizations, and with AWS accounts. For supported resource types, you can also share resources with IAM roles and users. If you have multiple AWS accounts, you can create a resource once and use AWS RAM to share that resource across accounts.

AWS RAM offers the following benefits:

- **Reduces operational overhead:** Create resource shares and use AWS RAM to share those resources with other accounts. This eliminates the need to provision duplicate resources in every account, which reduces operational overhead.
- **Provides security and consistency:** Govern consumption of shared resources by using existing policies and permissions to achieve security and control. AWS RAM offers a consistent experience for sharing different types of AWS resources.
- **Provides visibility and auditability:** View usage details for shared resources through integration with Amazon CloudWatch and AWS CloudTrail. AWS RAM provides comprehensive visibility into shared resources and accounts.

Access AWS Resources

Sharing your resources

With AWS RAM, you share resources that you own by creating a resource share. When creating a resource share, you need to specify the following:

- Name for the resource share.
- Type of resources to share; for example, subnets.
- Specific resources for each type; for example, a specific subnet.
- AWS RAM managed permission to associate with each resource type.
- Principals to which to grant access to the share. Principals can be an organization or OUs in AWS Organizations, AWS accounts, IAM roles, and IAM users.

Using shared resources

When the owner of a resource shares it with one's account, one can access the shared resource just as they would if it was owned by their account. One can access the resource by using the respective service's console, AWS CLI commands, and API actions. The actions that principals are allowed to perform vary depending on the resource type. All IAM policies and service control policies configured in one's account apply, which enables one to leverage their existing investments in security and governance controls.

Access AWS Resources

Accessing AWS RAM

One can work with AWS RAM in any of the following ways:

AWS RAM console

AWS RAM provides a web-based user interface, the AWS RAM console. If one has signed up for an AWS account, one can access the AWS RAM console by signing into the AWS Management Console and choosing AWS RAM from the console home page.

AWS Command Line Interface (CLI)

AWS CLI provides direct access to the AWS-RAM public API operations. It is supported on Windows, macOS, and Linux.

AWS tools for Windows PowerShell

AWS provides commands for a broad set of AWS products for those who script in the PowerShell environment.

Query API

The AWS RAM HTTPS Query API gives programmatic access to AWS RAM and AWS. AWS RAM API lets you issue HTTPS requests directly to the service. When you use the AWS RAM API, you must include code to digitally sign the requests by using your credentials.

Creating sample program through Boto3 in Python

Creating sample program through Boto3

Creating sample program through Boto3

Creating sample program through Boto3

Refactoring

Refactoring

There are many reasons to move an application to the cloud. In the early days, it was often about reducing infrastructure cost and complexity. The standard migration path was lift and shift. That meant spinning up virtual infrastructure, uploading code as is, and hoping to fix it later. This is still a common approach and, in some cases, a necessary one.

Here are the top five things that you can do at AMS as a part of minimum viable refactoring. Every app is different, but these actions tend to deliver big rewards with minimal risk.

- Update the operating system
- Upgrade the framework
- Improve the security
- Clean up access rights
- Explore database options

Refactoring

Update the operating system

In an on-premises environment, upgrading requires buying new licenses, provisioning capacity, often through complex IT procurement processes, testing the app, and and performing migration. It is labor-intensive and can be costly. Cloud migration is a perfect opportunity to upgrade. You can quickly and cheaply spin up a small instance to test your application.

Upgrade the framework

As with OS versions, framework versions such as .NET or Node.js, typically have an expiration date past which they are no longer patched or officially supported. Companies have logical reasons for keeping frameworks on life support. Upgrading on premises can be costly and require app surgery. The cloud makes it much easier to experiment with framework updates.

Refactoring

Improve the security

There is a false sense of security that comes from having your apps behind a firewall in your own data center. Once you move to the cloud, the hard-coded DNS or IP address that in 2004 seemed harmless running locally, can become an entry point for a hacker in 2020.

Clean up access rights

Identity and access are closely tied to security. It is also an area where technical debt can creep up on you. Most applications start off in a pristine state, with only a small set of privileged users having access. When you are getting ready to migrate, you will see who has access and you can choose who really needs it. Because of the change management process, every access request is made through the command line or automation. There are no standing privileges. Access is granted or denied by using role-based policies. Additionally, IP masking provides another layer of control between admin and instance.

Refactoring

Explore your database options

Moving to a new database can range in difficulty, from trivial to superhuman. Cloud can be an ideal test bed for experimentation and incremental improvement. Many organizations can move to the fully-managed Amazon Relational Database Service (RDS) with few changes. Amazon RDS gives you cost-efficient and resizable capacity while automating time-consuming administration tasks such as hardware provisioning, database setup, patching, and backups. With the availability of PostgreSQL, MySQL, MariaDB, Oracle Database, SQL Server, and Amazon Aurora, you can try one of them with your application. With the AWS Database Migration Service, migration or replication can be surprisingly fast and easy.

Application Optimization

Application Optimization

Application performance optimization employs a number of techniques that are implemented in an organization's IT infrastructure to amplify the functionality of a network, including the monitoring of bandwidth capacity, application protocols, overall network traffic, application coding, network latency, potential attacks on software vulnerabilities, and more.

A few techniques that are implemented to optimize an application are:

- Rightsizing EC2 instances
- Scheduling on or off times
- Purchasing reserved instances and savings plans
- Delete unattached EBS volumes
- Delete obsolete snapshots
- Release unattached Elastic IP addresses
- Upgrade instances to the latest generation
- Purchase reserved nodes for Redshift and ElastiCache services
- Terminate zombie assets
- Move infrequently-accessed data to lower-cost tiers

Application Optimization

Rightsizing EC2 instances

If you double the capacity when you go up one size, then you also half the capacity when you go down one size. Therefore, rightsizing is a worthwhile practice if there are instances whose peak utilization does not exceed ~45 percent. It is still worth analyzing utilization metrics to find opportunities to move workloads to different families, other than General Purpose, that better suit the users' needs.

Scheduling on or off times

It is worth scheduling on or off times for non-production instances such as those used for developing, staging, testing, and QA. More aggressive schedules can be applied by analyzing utilization metrics to determine when the instances are most frequently used or to apply an always stopped schedule, which can be interrupted when access to the instances is required.

Application Optimization

Purchasing reserved instances and savings plans

Purchasing reserved instances is an easy way to reduce AWS costs. It can also be an easy way to increase AWS costs if you do not utilize the reserved instance as much as you expected to, purchase the wrong type of reserved instance, or purchase a standard reserved instance only to find AWS prices fall over the term of your reservation by more than what the reservation saves.

Delete unattached EBS volumes

Returning to Elastic Block Storage (EBS,) when you launch an EC2 instance, an EBS volume is attached to the instance to act as its local block storage. When you terminate the EC2 instance, the EBS volume is only deleted if you checked the delete-on-termination box when the instance was launched. Depending on how long your business has been operating in the cloud and the number of instances launched without the delete box being checked, there could be thousands of unattached EBS volumes in your AWS cloud.

Application Optimization

Delete obsolete snapshots

Snapshots are an efficient way to back up data on an EBS volume to an S3 storage bucket because they only back up data that is changed since the last snapshot to prevent duplications in the S3 bucket. Consequently, each snapshot contains information that is needed to restore your data, from the moment when the snapshot was taken to a new EBS volume. Although snapshots do not cost very much individually, you could save thousands of dollars by deleting those that are no longer needed.

Release unattached Elastic IP addresses

Elastic IP addresses are public IPv4 addresses from Amazon's pool of IP addresses that are allocated to an instance so it can be reached via the internet. Businesses are allowed a maximum of five Elastic IP addresses per account because Amazon does not have an unlimited pool of IP addresses. However, they are free of change when attached to a running service.

Application Optimization

Upgrade instances to the latest generation

When AWS releases a new generation of instances, they tend to have improved performance and functionality compared to their predecessors. This means that you can either upgrade existing instances to the latest generation or downsize existing instances with borderline utilization metrics in order to benefit from the same level of performance at a lower cost.

Purchase reserved nodes for Redshift and ElastiCache services

Reserved nodes can be purchased for the services of Redshift, ElastiCache, Redis, and Memcached for one-year or three-year terms, with the option of paying the full amount upfront, partially upfront, or monthly. One important note is that in order to take the advantage of reservations on ElastiCache, you must first upgrade the nodes to the latest generation.

Application Optimization

Terminate zombie assets

Zombie assets is most often used to describe any unused asset that contributes to the cost of operating in the AWS cloud—many typical zombie assets have already been mentioned such as unattached EBS volumes, obsolete snapshots, etc. Other assets that fall into this category include unused elastic load balancers and components of instances that were activated when an instance failed to launch.

Move infrequently-accessed data to lower-cost tiers

AWS currently offers six tiers of storage at different price points. Determining which storage tier is most suitable for data will depend on factors such as how often data is accessed, as retrieval fees apply to the lower tiers, and how quickly would a business need to retrieve data in the event of a disaster, as it may take hours to retrieve data from a lower tier.

Amazon CodeGuru Profiler

Amazon CodeGuru Profiler

CodeGuru Profiler provides insights into your application's runtime performance with a continuous, always-running production profiler. It collects data from your application's runtime on what threads are currently running and what code those threads are executing. CodeGuru Profiler provides visualizations to help you to understand your application's execution and automatic recommendations on how to optimize it. These recommendations focus on reducing CPU utilization and application latency to reduce infrastructure costs and give your customers a snappier experience.

These optimizations can result in significant cost savings when you apply them to large fleets. Within Amazon, CodeGuru Profiler is running in over 80,000 applications and has resulted in tens of millions of dollars of savings with the suggested optimizations. It is also consistently used to analyze operational issues and their impact on performance.

Amazon CodeGuru Profiler

CodeGuru Profiler consists of three main components:

- **Agent:** Runs inside your application polling its current runtime state and submits this data to CodeGuru Profiler
- **Console:** Provides visualizations into your data and recommendations for potential improvements
- **API:** Allows management of profiling groups and retrieving profiles and recommendations

India: +91-7022374614

US: 1-800-216-8930 (toll-free)

support@intellipaat.com

24/7 Chat with our Course Advisors