Role -- Which is used to provide access to any other aws services. Other aws services,Other aws accounts,cognito and saml

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| cidr = 2 ^ 32-range | 2^32-26 | | 2^6 | 64 |  |  |
|  | 16 \* 4 = 64 | | 4 subnets ranges have been divided | | |  |
|  | 2^32-28 | | 2^4 | 16 per subnet is used define 0-15 subnets addresses | | |
| 10.0.0.0/28 (16addresses) | |
| 10.0.0.16/28 | |
| 10.0.0.32/28 | |

Only 11 will be available remaining 5 ip's are available for internal researches .i.e the first four and last one ip address

Create two route tables:

1. Public RT : Used to connect/interact with public subnets
2. Private Rt : Used to connect/interact with private subnets

Along with VPC main route table will be created

One subnet can be associated with only one route table at a time

But one route table can have any no of subnets associated with it self

Public : Having internet connectivity

Internet gateway : Connecting internet with vpc to outside.IG is attached with VPC.

Route Tables : Associate IG with public route table

NAT Gateway : It should be associated with Public subnets only

**Route Tables :**

1. **Public Route Table :**

**Subnet Association :** publicsubnets define here

**Route** : 0.0.0.0/0 with IGW

1. **Private** **Route Table**:

**Subnet Association :** privatesubnets define here

**Route** : 0.0.0.0/0 with NAT Gateway

**Now, every instance type has a set of instances which are optimized for different workloads:**

* General Instances
  + t2
  + m4
  + m3
* Compute Instances
  + c4
  + c3
* Memory Instances
  + r3
  + x1
* Storage Instances
  + i2
  + d2
* GPU Instances
  + g2
* **Provisioned IOPS**: This category is for workloads which are mission critical, it provides high IOPS rates.
* **General Purpose:** It is for workloads which need a performance and cost balance.
* **Magnetic:** It is for data which is accessed less frequently, and also retrieval time is more.

No. of instances- you can provision up to 20 instances at a time.

**RDS AWS Components:**

* DB Instances
* Regions and Availability Zones
* Security Groups
* DB Parameter Groups
* DB Option Groups

**DB Parameter groups**

* It contains the engine configuration values that can be applied to one or more DB Instances of the same instance type.
* If you don’t apply a DB Parameter group to your instance, you are assigned a default Parameter group which has the default values.

**DB Option groups**

* Some DB engines offer tools that simplify managing your databases.
* RDS makes these tools available with the use of Option groups.

**RDS AWS Advantages**

Let’s talk about some interesting advantages that you get when you are using RDS AWS,

* So usually when you talk about database services, the CPU, memory, storage, IOs is bundled together, i.e. you cannot control them individually, but with AWS RDS, each of these parameters can be tweaked individually.
* Like we discussed earlier, it manages your servers, updates them to the latest software configuration, takes backup, everything automatically.
* The backups can be taken in two ways
  + The automated backups where in you set a time for your backup to be done.
  + DB Snapshots, where in you manually take a backup of your DB, you can take snapshots as frequently as you want.
* It automatically creates a secondary instance for a failover, therefore provides high availability.
* RDS AWS supports **read replicas**i.e. snapshots are created from a source DB and all the read traffic to the source database is distributed among the read replicas, this reduces the overall overhead on the source DB.
* RDS AWS can be integrated with IAM, for giving customized access to your users who will be working on that database.

The updates to your database in RDS AWS are applied in a **maintenance window**. This maintenance window is defined during the creation of your DB Instance, the way it functions is like this:

* When an update is available for your DB you get a notification in your RDS Console you can take one of the following actions
  + Defer the maintenance items.
  + Apply maintenance items immediately.
  + Schedule a time for those maintenance items.
* Once maintenance starts, your instance has to be taken offline for updating it, if your instance is running in Multi-AZ, in that case the standby instance is updated first, it is then promoted to be a primary instance, and the primary instance is then taken offline for updating, this way your application does not experience a downtime.
* If you want to scale your DB instance, the changes that make to your DB instance also happen during the maintenance window, you can also apply them immediately, but then your application will experience a downtime if its in a Single-AZ.
* The next step is to export the application database. You can either do this using the command-line mysqldump tool or the browser-based phpMyAdmin application.

Make a backup of the existing database by entering the following command:

*mysqldump -u*username*-p -h*hostnamedatabase\_name*> backup.sql*

Using PhpMyAdmin

You can also accomplish this with the browser-based phpMyAdmin application included by default with Bitnami stacks.

* Select the application database in the left navigation menu. In this example, the database is named *bitnami\_wordpress*.
* Select the “Export” menu item.
* On the resulting page, select the “Quick” export method and the “SQL” output format.

**Step 5: Import The Application Database To Amazon RDS**

* The next step is to import the application database to your RDS database instance. Again, you can accomplish this either using the command-line tool or the browser-based phpMyAdmin application.
* Using The Command Line
* Use the mysql command-line client to import the backup from the previous step into your Amazon RDS database instance as follows:

Amazon S3 actually has more in common with your own server (a single physical location) than a CDN (lots of locations around the globe).

**Amazon S3**: While you can definitely serve files from S3 to your visitors, S3’s **primary focus is storage.**

**CDN**: A CDN is focused **on delivering files as quickly as possible**, rather than affordably storing them (like S3).

**Cross-region Replication:**

As the name suggests, Cross-region Replication enables user to either replicate or transfer data to some other location without any hassle.

This obviously has a cost to it which has been discussed further in this article.

**Cloud Front** is a caching service by AWS, in which the data from client site gets transferred to the nearest edge location and from there the data is routed to your AWS S3 bucket over an optimized network path.

AWS Storage Gateway is a hybrid storage solution from Amazon that allows your on premise applications to use AWS-backed storage, such as EBS and S3.

Select gateway type: File Gateway, Volume gateway

Host platform: VMware esxi/ova platform

Go back to AWS console. Click next, key in the Gateway IP address, then click Connect to gateway.

You will be prompted to set Gateway time zone and name. Do so accordingly, then click Activate gateway.

On the host platform select where you’ll place the storage gateway. You can download an OVA for VMware, or spin up an EC2 instance. For this post I used the EC2 instance. Click the “Launch instance” button to take you to that wizard.

Now we walk through setting up an EC2 instance. The recommended starting size is m4.large but you can select what makes sense for your environment. Click Next.

On the storage page, add a new EBS volume for your caching. I used a 100 GB drive but this all depends on how much data you want to cache for your S3 bucket. Obviously, the larger the drive, the more data you can cache for the S3 bucket, but that comes at the cost of EBS storage.

In a moment you’ll see your new instance starting up in the EC2 console.

Once the EC2 instance has had time to spin up, we can go back to our storage gateway wizard where we jumped out of to create our EC2 instance. Enter in the IP Address of the new EC2 instance. I used the public IP Address but you could use a private IP if you have connectivity from your workstation to the storage gateway. Clicking the “Connect to gateway” button just redirects your browser to activate the gateway.

**Create gateway:**

Connect to gateway

Activate gateway

Configure local disks

Configure share: create file share

With **Elastic Beanstalk,** you can **deploy**, **monitor**, and **scale** an application quickly and easily.

Checks the health of an application

End point will be available for implementation

Dashboard will displays logs,monitoring,health status

**Systems Manager : (Parameter Store)**

The Parameter Store offers the ability to store 3 different types of data, which can then be programmatically accessed via the SSM API. The 3 types of data are: String, String List, and Secure String.

When you store a secure string in the EC2 Parameter Store, the data is encrypted via a KMS key associated with your account. You can either choose the default KMS key, or create your own. By using KMS, we can easily secure these strings of data within the AWS ecosystem, utilizing all of the benefits KMS gives.

1 creating our own KMS key which we can later use with the EC2 Parameter Store:

2 . Storing a Secure String

There are 2 main ways to add a secure string to the EC2 Systems Manager for retrieval. Keys can be added via the console, under the EC2 section, or they can be created programmatically via API/SDK/CLI.

cannot choose which KMS key to use when creating a secure sting in the console. That has since changed and you can now select a custom KMS key to use for encrypting secure strings directly within the AWS console!

To add a secure string via the AWS CLI, use the ssm put-parameter command:

aws --region=us-east-2 ssm put-parameter --name "<string-name>" --value '<secure data>' --type SecureString --key-id alias/<kms-key-name>

aws --region=us-east-2 ssm put-parameter --name "secret-password" --value 'password1234' --type SecureString --key-id alias/secret-strings

By using a JSON string, you can programmatically access that data and parse the string into a JSON object. This is useful for storing perhaps an IAM Access Key and Secret Access Key:

{ "AccessKey": "ABCDE43235132D", "SecretKey": "ACBDEFGHIJKL123456789ABCDEF" }

**Accessing Values via CLI**

Via the CLI, you can access secure strings using the ssm get-parameterscommand:

**Format:**

aws --region=us-east-2 ssm get-parameters --names "<string-name>" --with-decryption

aws --region=us-east-2 ssm get-parameters --names "secret-password" --with-decryption

**Accessing Values via Python SDK**

The official AWS SDKs allow you to access EC2 Parameters.

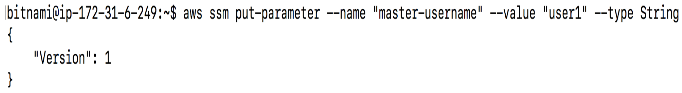
**Parameter Store usage :** **https://mahadevops.com/aws/aws-ssm-parameter-store-step-by-step-tutorial/**

Step 1 : First of all install AWS CLI on your ec2 if you don’t have it:

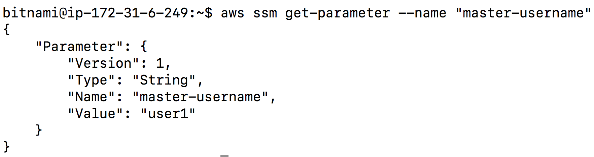
Step 2 : Create a role with full ssm access.

Step 3**:** Attach the role to EC2

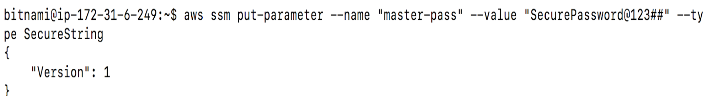
Step 4:Run the command to store a ssm parameter from ec2



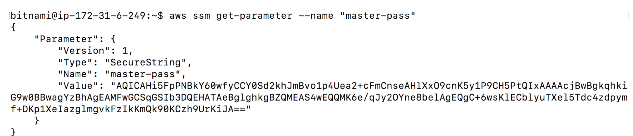
Step 5**:** You stored a parameter which is not secure. Now retrieve the parameter with this command. You will see the output return your value for the key (name) specified.



Step 6: Now let’s try yo store a secure string on SSM



Step 7: Retrieve the securestring with this command



**Amazon SQS**

**Create New Queue** page, ensure that you're in the correct region and then type the **Queue Name**.

The name of a FIFO queue must end with the .fifo suffix

**Standard** is selected by default. C

**ontent-Based Deduplication** column displays whether you have enabled [exactly-once processing](https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/FIFO-queues.html#FIFO-queues-exactly-once-processing)

## Step 2: Send a Message

After you create your queue, you can send a message to it. The following example shows sending a message to an existing queue.

From **Queue Actions**, select **Send a Message**.

## Step 3: Receive and Delete Your Message

After you send a message into a queue, you can consume it (retrieve it from the queue). When you request a message from a queue, you can't specify which message to get. Instead, you specify the maximum number of messages (up to 10) that you want to get.

Amazon SQS begins to poll the messages in the queue. The dialog box displays a message from the queue. A progress bar at the bottom of the dialog box displays the status of the message's visibility timeout.

## Step 4: Delete Your Queue

If you don't use an Amazon SQS queue (and don't foresee using it in the near future), it is a best practice to delete it from Amazon SQS. In this tutorial you'll learn how to delete a queue.

IOT Core

-----------

Manage -- > Things -- > Register a thing

An IoT thing is a representation and record of your phyisical device in the cloud. Any physical device needs a thing record in order to work with AWS IoT

# Register a single AWS IoT thing

Create a thing in your registry

# Bulk register many AWS IoT things

Create things in your registry for a large number of devices already using AWS IoT, or register devices so they are ready to connect to AWS IoT.

**Add Device to the thing registry :**

Name: DTH11

Apply a type to this thing: Create Type:

This will help you organize, categorize, and search for your things.

Name : DHT

# Set searchable thing attributes

You can define up to three attributes for a thing type. Things associated with this type can be searched by using these fields.

If we have different types of devices, such as motion sensors, door sensors, or DHT11 sensors, we can create a Type to easily group our nodes.

# Add this thing to a group

Adding your thing to a group allows you to manage devices remotely using jobs.

Parent Group: Groups/

**Set Group Attributes:**

Enter a value for one or more of these attributes

Device : RasberryPi3

Sensor : DTH11

# 3 Add a certificate for your thing

A certificate is used to authenticate your device's connection to AWS IoT.

One-click certificate creation (recommended)

This will generate a certificate, public key, and private key using AWS IoT's certificate authority.

Create with CSR

Upload your own certificate signing request (CSR) based on a private key you own.

Use my certificate

Register your CA certificate and use your own certificates for one or many devices.

Skip certificate and create thing

You will need to add a certificate to your thing later before your device can connect to AWS IoT.

**Once the certificates are created,** download the following:

Client certificate: db80b0f635.cert.pemPublic Key: db80b0f635.public.keyPrivate Key: db80b0f635.private.keyRoot CA: From this URL you can download or copy the text:

Once you have downloaded the keys, click on the Activate button.

Once the activation is successful, click on Attach a policy. Since we did not create any policies, you will see a screen similar to as what is shown here:

 We will create a policy manually and associate it with this certificate in a moment.

Finally, click on the Register Thing button and a new Thing named Pi3-DHT11-Node will be created. Click on Pi3-DHT11-Node and you should see something like this:

We are not done with the setup yet. We still need to create a policy and attach it with a certificate to proceed.

Go to **Secure 🡪 Policies**

AWS IoT policies give things permission to access AWS IoT resources (like other things, MQTT topics, or thing shadows).

# Create a policy

Create a policy to define a set of authorized actions. You can authorize actions on one or more resources (things, topics, topic filters). To learn more about IoT policies

Add statements

Policy statements define the types of actions that can be performed by a resource.

Action : IOTCore\*

Resource ARN : arn:aws:iot:us-east-2:907288793553:topic/replaceWithATopic

we are going to attach this policy to a certificate.

Setting up Raspberry Pi 3 on the DHT11 node

Now that we have our Thing set up in AWS IoT, we are going to complete the remaining operation in Raspberry Pi to send data.

Things needed

You will need the following hardware to set up Raspberry Pi 3 on the DHT11 node:

* One Raspberry Pi 3: <https://www.amazon.com/Raspberry-Pi-Desktop-Starter-White/dp/B01CI58722>
* One breadboard: <https://www.amazon.com/Solderless-Breadboard-Circuit-Circboard-Prototyping/dp/B01DDI54II/>
* One DHT11 sensor: <https://www.amazon.com/HiLetgo-Temperature-Humidity-Arduino-Raspberry/dp/B01DKC2GQ0>
* Three male-to-female jumper cables: <https://www.amazon.com/RGBZONE-120pcs-Multicolored-Dupont-Breadboard/dp/B01M1IEUAF/>

The **AWS IOT service stands for Amazon Web Service Internet of Things**. It allows you to connect your things (devices) to the internet to exchange data securely, process it and act upon it.

Testing the Thing using the AWS:

**Step 1:**Go to the main screen and select the *Test*option. This will load the MQTT client which can be used to test our thing.

**Step 2:** On the left side you can see two options Subscribe to a topic and Publish to a topic. First you have to subscribe to the thing we just created by entering the name which in my case is *circuitdigest*.

Click on *publish to topic* and you should see the message reflected in your thing as shown below

Using MQTT.fx with AWS IOT:

MQTT.fx is an application which can be used as a client to test and debug IOT devices. In the following steps we will learn how we can connect the MQTT.fx with the thing that we just created.

Step 1: Download the MQTT.fx file form [this download link](http://mqttfx.jensd.de/index.php/download). Make sure you select the correct operating system of your machine.

Step 2: Open the application and click on the settings icon to configure the MQTT as client. The settings icon is shown in the picture below:

**Step 3:**A new window called *Edit Connection Profiles* will pop up. Here we have to create the profile for the thing that we just created using Amazon AWS. In the Profile name give a name of your choice I have given “*MQTT\_Sample*”. Then for the broker address paste the address that we got in previous section of “**Getting your AWS thing Details”**section. The Broker Port for AWS IOT is 8883 for all users so enter the same. Then select SSL/TLS

**Step 4:**Now we have to select Self-signed certificates and link the certificates that we downloaded in step 8. Also make sure PEM formatted is checked. Follow the image below to know which keys you should choose

**Step 5:**Finally click on OK and then you will taken back to the main window. Now click on the connect button. IF everything is working properly then MQTT should be able to connect to our thing and the following screen will be displayed. Check for the green circle on the top right corner (en-circled)

**Step 6:**Now that we are connected to the thing we can try testing it subscribing to a name. Click on *Subscribe* tab and give any random name and then click on subscribe. Here I have selected bingo as my name. After subscribing you will get the following name.

**Step 7:** Now go back to the*Publish* screen tab and publish a message and check if we are getting it on our subscribed channel .To publish a message use the same name. Here I have used the same “bingo” as my name and my message is “*Hola! Amingo*”. Click on the publish button

**Step 8:**Now go back to *Subscribe* tab and you should be able to see the message that we just published as shown below

We are all done with creating and testing our thing that was created using AWS IOT. Since we have also linked it with MQTT as client it should be easy for us to monitor and debug the thing in future. You can also **follow the video** at the bottom to get a better understanding.

In our next tutorials, we will learn how we can use actual hardware things like ESP8266, ESP12, Raspberry Pi etc to use this thing to send/receive information.

# Greengrass

# AWS IoT Greengrass is software that extends cloud capabilities to local devices. This enables devices to collect and analyze data closer to the source of information, react autonomously to local events, and communicate securely with each other on local networks.

# 

# In AWS IoT Greengrass, devices securely communicate on a local network and exchange messages with each other without having to connect to the cloud. AWS IoT Greengrass provides a local pub/sub message manager that can intelligently buffer messages if connectivity is lost so that inbound and outbound messages to the cloud are preserved.

AWS IoT Greengrass provides secure, over-the-air software updates of Lambda functions.

AWS IoT Greengrass consists of:

* Software distributions
  + AWS IoT Greengrass core software
  + AWS IoT Greengrass core SDK
* Cloud service
  + AWS IoT Greengrass API
* Features
  + Lambda runtime
  + Shadows implementation
  + Message manager
  + Group management
  + Discovery service
  + Over-the-air update agent
  + Local resource access
  + Machine learning inference
  + Local secrets manager

# An AWS IoT Greengrass group is a collection of settings and components, such as an AWS IoT Greengrass core, devices, and subscriptions. Groups are used to define a scope of interaction.

AWS IoT Greengrass cores

A Greengrass core is an AWS IoT device that runs the AWS IoT Greengrass core software, which enables it to communicate directly with the AWS IoT and AWS IoT Greengrass cloud services. A core has its own certificate used for authenticating with AWS IoT.

# LAB

# ====

# Manage 🡪 GreenGrass 🡪 grren gross group 🡪

# Set up your Greengrass Group :

# Easy Group creation (recommended)

# This process will automatically provision a Core in the registry, use default settings to generate a new Group, and provide your Core with a new certificate and a key pair

SET UP YOUR GREENGRASS GROUP

Every Group needs a Core to function

Run a scripted easy Group creation

**Download and store your Core's security resources**, choose **Download these resources as a tar.gz** to download the required security resources for your AWS IoT Greengrass core.

# Module 1: Environment Setup for Greengrass

Setting Up a Raspberry Pi

sudo raspi-config

sudo reboot

hostname -I

ssh pi@IP-address

If you are using Windows, you need to install and configure [PuTTY](https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html" \t "_blank). Expand Connection, choose Data, and make sure that Prompt is selected:

You are now ready to set up the Raspberry Pi for AWS IoT Greengrass. First, run the following commands from a local Raspberry Pi terminal window or an SSH terminal window:

sudo adduser --system ggc\_user

sudo addgroup --system ggc\_group

Your Raspberry Pi should now be ready for AWS IoT Greengrass.

To make sure that you have all required dependencies, download and run the Greengrass dependency checker

# Module 2: Installing the Greengrass Core Software

You can download the core software from the Software page in the AWS IoT Core console or from the [AWS IoT Greengrass Core Software](https://docs.aws.amazon.com/greengrass/latest/developerguide/what-is-gg.html#gg-core-download-tab) downloads.

DynamoDB

=========

Amazon DynamoDB is the latest NoSQL offering from AWS. It is a managed, scalable and on-demand database with provisioned throughput. Provisioned Throughput will let you state your read and write throughput requirements upfront based on which the cost will be calculated

 Amazon DynamoDB is a fully managed non-relational database service that provides fast and predictable performance with seamless scalability.

DynamoDB is a schema-less database that only requires a table name and primary key. The table’s primary key is made up of one or two attributes that uniquely identify items, partition the data, and sort data within each partition.

*DynamoDB uses port 8000 by default. If the port 8000 is unavailable, you can use****-port****option to assign another port.*

To run DynamoDB on your computer, you must have the Java Runtime Environment (JRE) version 6.x or newer. The application doesn't run on earlier JRE versions.

To access DynamoDB running locally, use the --endpoint-url parameter. For example, use the following command to list DynamoDB tables:

aws dynamodb list-tables --endpoint-url http:*//localhost:8000*

Table settings:

Default settings provide the fastest way to get started with your table. You can modify these default settings now or after your table has been created.

|  |
| --- |
| Read/write capacity mode |

Select on-demand if you want to pay only for the read and writes you perform, with no capacity planning required. Select provisioned to save on throughput costs if you can reliably estimate your application's throughput requirements.

Provisioned capacity

Read capacity units 5 Write capacity units 5

|  |  |
| --- | --- |
| Auto Scaling |  |

Target utilization

Minimum provisioned capacity

Maximum provisioned capacity

To load the ProductCatalog table with data, enter the following command:

aws dynamodb batch-write-item --request-items file://ProductCatalog.json

CloudFormation Template Basics

The main things you need to keep in mind when building a template are

* A CloudFormation template is a JSON-formatted text file that describes your AWS infrastructure.
* Templates can include several major sections:  
  – AWSTemplateFormatVersion  
  – Description  
  – Metadata  
  – Parameters  
  – Mappings  
  – Conditions  
  – Resources  
  – Outputs
* The Resources section is the only section that is actually required.
* The first character in the CloudFormation template must be an open brace ({), and the last character must be a closed brace (}).

### **Create a stack**

AWS CloudFormation allows you to quickly and easily deploy your infrastructure resources and applications on AWS.

A StackSet is a container for AWS CloudFormation stacks that lets you provision stacks across AWS accounts and regions by using a single AWS CloudFormation template.

Design a template

Templates tell AWS CloudFormation which AWS resources to provision and how to provision them. When you create a CloudFormation stack, you must submit a template.

Create a Template from your Existing Resources

If you already have AWS resources running, the **CloudFormer** tool can create a template from your existing resources. This means you can capture and redeploy applications you already have running.

 Rollback Triggers

Rollback triggers enable you to have AWS CloudFormation monitor the state of your application during stack creation and updating, and to rollback that operation if the application breaches the threshold of any of the alarms you've specified.

$ aws cloudformation create-stack --template-body file://templates/single\_instance.yml --stack-name single-instance --parameters ParameterKey=KeyName,ParameterValue=tutorial ParameterKey=InstanceType,ParameterValue=t2.micro

**Amazon Cognito** and Federated Identities. Cognito is the AWS solution for managing user profiles, and Federated Identities help keep track of your users across multiple logins.

Amazon Cognito offers user pools and identity pools. User pools are user directories that provide sign-up and sign-in options for your app users. Identity pools provide AWS credentials to grant your users access to other AWS services.

With Cognito User Pools, you can easily and securely add sign-up and sign-in functionality to your mobile and web apps with a fully-managed service that scales to support hundreds of millions of users.

With Cognito Identity Pools, your app can get temporary credentials to access AWS services for anonymous guest users or for users who have signed in.

* + - * 1. General Settings
        2. App integration
        3. Federation

Now we should note that we already created a UserPool and an appClient, we should be able to take a note of 3 important configuration settings.

**Authentication vs. Authorization**

In the security world, the terms “authentication” and “authorization” have very specific meanings. Authentication is the process of verifying a user’s identity. Most commonly, users authenticate with a username (which identifies the user) and a password (which confirms the user is who he claims). Authorization, in contrast, is the process of granting users access to specific resources after they have been authenticated. For example, users might be placed into one or more groups based on their job title, and the application then determines which features are available to them based on their group membership.

Identity pools are used to store end user identities. To declare a new identity pool, enter a unique name.

### **Authentication providers**

Cognito,aazon,Facebook,Google+,Twitter,OpenID,saml,custom

**Lambda :** lets you run code without thinking about servers

In the case of AWS Lambda Functions, this is called a trigger. Lambda Functions can be triggered in different ways: an HTTP request, a new document upload to S3, a scheduled Job, an AWS Kinesis data stream, or a notification from AWS Simple Notification Service (SNS).

EMR

------

Amazon Elastic MapReduce (Amazon EMR) is a web service that enables businesses, researchers, data analysts, and developers to easily and cost-effectively process vast amounts of data.

Amazon EMR provides a managed platform that makes it easy, fast, and cost-effective to process large-scale data across dynamically scalable Amazon EC2 instances, on which you can run several popular distributed frameworks such as Apache Spark.

Even if according to AWS EMR docs it is supposed to be easy as hell to set up and use, digging into some concepts of the AWS platform to understand what I was doing was a bit time-consuming.

So here is the plan:

* set up the cluster
* export my input data to a CSV file on S3
* send my Spark job to the cluster
* gather the results somewhere on S3

Steps:

=====

Create Cluster - Quick Options:

General Configuration: Cluster name Launch mode

Software configuration:

Applications: Core Hadoop,HBase,Spark

Hardware configuration: Security and access

$ aws emr create-cluster \ --name "My cluster" \ --instance-type m3.xlarge \ --release-label emr-5.4.0 \ --instance-count 3\ --use-default-roles \ --applications Name=Spark

## Submitting jobs to the cluster

Via the GUI, just click on the Add step button. The following form should pop up:

Select a Spark application and type the path to your Spark script and your arguments. Note that the Spark job script needs to be submitted to the master node (and will then be copied on the slave nodes by the Spark platform). I uploaded the script in an S3 bucket to make it immediately available to the EMR platform.

Application Type

Deploy Mode

Application Location

Arguments

Action on Failure

* Elastic MapReduce (EMR), a managed cluster platform that simplifies running big data frameworks, such as [Apache Hadoop](https://aws.amazon.com/elasticmapreduce/details/hadoop) and [Apache Spark](https://aws.amazon.com/elasticmapreduce/details/spark). It can be view like Hadoop-as-a-Service, you start a cluster with the number of nodes you want, run any job you want and only pay for the time the cluster is actually up.

The aim of this tutorial is to launch the classic word count Spark Job on EMR. The input and output files will be store using S3 storage.

**Create an Amazon S3 Bucket**

In this use case, we will use Amazon S3 bucket to store our Spark application jar, logs, input and output files.

**Upload files on Amazon S3**

Now that our S3 bucket is created, we will upload the Spark application jar and an input file on which we will apply the wordcount.

**Amazon Kinesis**

Easily collect, process, and analyze video and data streams in real time, so you can get timely insights and react quickly to new information.

1 Evolve from batch to real-time analytics : Perform real-time analytics on data that has been traditionally analyzed using batch processing in data warehouse or using Hadoop frameworks.

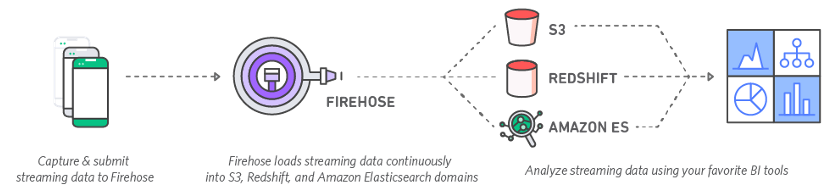
2 Build real-time applications Create visibility into what your customers, applications, and products are doing right now by building real-time applications for application monitoring, fraud detection, and live leader-boards

Analyze IoT device data Processing streaming data from consumer applications, embedded sensors, TV set-top boxes, and other IoT devices. Send real-time alerts or perform other actions programmatically based on data.

Kinesis offer good performances for data processing and it is also easy to integrate with other Amazon services.

the power of Kinesis is also giving the developer the ability to make calls, request data and analyze it while it is passing between the producer (the machine sending the stream) and the data stores (S3, Redshift, Elastic Search ..etc)

This part of loading streamed data to AWS is done using Amazon Kinesis Firehose.



To use Kinesis you will need to create a Kinesis Stream that will collect and stream data for ordered, repayable, real-time processing.

**Shard**

A shard is the base throughput unit of a stream. Each shard ingests up to 1MB/sec and 1000 records/sec, and emits up to 2MB/sec. To accommodate for higher or lower throughput, the number of shards can be modified after the Kinesis stream is created using the API.

You can use 1 or more shards. This depends on how much data are you going to send per second.

A shard is described by:

* ShardId
* EndingHashKey + StartingHashKey
* SequenceNumberRange

**Data Record**

A record is the unit of data stored in a stream.

Inside a record you will find:

* A sequence number
* A partition key
* Data blob (1 megabyte (MB) maximum)

**Partition Key**

* Partition key is used to segregate and route data records to different shards of a stream.
* You can create more than one shard in order to send all the data with a specific partition key to a specific shard.

**Sequence Number**

* A sequence number is a unique identifier for each data record.
* When a developer calls Put Record from the data producer, a sequence number is assigned by Amazon Kinesis Streams to the data sequence

The advantage of using Fire hose in our case is that we can deliver the streaming data to a data store without writing any code. A simple configuration from that AWS console is sufficient.

3 delivery scenarios are possible:

* Simply Delivering to S3 (encryption and compression are possible)
* Delivering to S3 and executing the COPY command to store the same data to Redshift
* Storing to Amazon Elastic Search

### **Direct PUT or other sources**

* After creating the delivery stream, send source records using the Firehose PUT API or the Amazon Kinesis Agent.
* **Firehose PUT APIs**
* Use the Firehose PutRecord() or PutRecordBatch() API to send source records to the delivery stream. [Learn more](https://docs.aws.amazon.com/console/firehose/send-data/sdk)
* **Amazon Kinesis Agent**
* The Amazon Kinesis Agent is a stand-alone Java software application that offers an easy way to collect and send source records to Firehose. [Learn more](https://docs.aws.amazon.com/console/firehose/send-data/agent)
* **AWS IoT**
* Create AWS IoT rules that send data from MQTT messages. [Learn more](https://docs.aws.amazon.com/console/firehose/send-data/iot)
* **CloudWatch Logs**
* Use subscription filters to deliver a real-time stream of log events. [Learn more](https://docs.aws.amazon.com/console/firehose/send-data/cloudWatchLogs)
* **CloudWatch Events**
* Create rules to indicate which events are of interest to your application and what automated action to take when a rule matches an event. [Learn more](https://docs.aws.amazon.com/console/firehose/send-data/cloudWatchEvents)

Process records

Transform source records with AWS Lambda

Convert record format

* Shards are limited to 50 for the following regions only: US East, US West, EU Ireland while all other regions have a default shard limit of 25.

**Amazon Kinesis** is a managed, scalable, cloud-based service that allows real-time processing of streaming large amount of data per second.

You can use **Amazon Route 53** to register new domains, transfer existing domains, route traffic for your domains to your AWS and external resources, and monitor the health of your resources.

DNS management

If you already have a domain name, such as example.com, Route 53 can tell the Domain Name System (DNS) where on the Internet to find web servers, mail servers, and other resources for your domain.

Traffic management

Route 53 traffic flow provides a visual tool that you can use to create and update sophisticated routing policies to route end users to multiple endpoints for your application.

Availability monitoring

Route 53 can monitor the health and performance of your application as well as your web servers and other resources. Route 53 can also redirect traffic to healthy resources.

Domain registration

If you need a domain name, you can find an available name and register it by using Route 53. You can also make Route 53 the registrar for existing domains that you registered with other registrars.

Amazon Route 53 is an authoritative Domain Name System (DNS) service. DNS is the system that translates human-readable domain names (example.com) into IP addresses (192.0.2.0). With authoritative name servers in data centers all over the world, Route 53 is reliable, scalable, and fast.

Hosted zone for your domain will be created and you will be given four DNS endpoints called delegation set. These endpoints should to be updated in your domain names name server section.

These record sets are used map the domain names to respective aws instances where you host your application.

Go back to route53 console and click “got to record sets” option. By default, you will have two record sets Type NS and SOA.

Let’s say if you have two instances in two different aws regions. Now will have to route your traffic. Click create record set option, fill in the following details and create record set option.

* Name: append www
* Type: A
* TTL: 500
* Alias:

Yes, if you want to point the domain name to any load balancer, s3 or cloud front endpoints.

No, if you want to point the domain name to the servers ip. I my case, am selecting no because am pointing the domain name directly to the servers ip address.

* Value: IP address of your server which is hosting your application
* Routing Policy: latency

There are four routing policies in AWS[simple , latency, weighted and failover.](http://docs.aws.amazon.com/Route53/latest/DeveloperGuide/WeightedResourceRecordSets.html)

A Hosted Zone is created automatically as part of the registration process:

That lets us create a new entry with the Create Record Set button.  Because we want to redirect the default www zone, let’s create a CNAME record for that and directly assign discoposse.com in the Value field:

 click the Alias radio button to Yes, and click the Alias Targetfield to trigger the drop down list where you can select the S3 website endpoint that matches the domain root:

**CloudWatch**

Suppose, you have multiple servers running a simple web application on apache or Nginx and you want to put all the error/access logs on a centralized place so that you can troubleshoot your system in case of any error after getting alert notification configured on your logs .

Amazon CloudWatch now provides us the flexibility to monitor, maintain, store and access our custom log files, log files from EC2 Servers, CloudTrail and other resources. We can also generate alerts on those logs. This will help us in troubleshooting our servers by monitoring all the appplication-specific logs on CloudWatch in real time.

**Step 1 :-**

Firstly we need two policies attached to an IAM role which we will assign to EC2 instances so that the logs from the instances can be pushed to the CloudWatch.

There is a need of an CloudWatch agent which will do the task to push logs onto the CloudWatch.An agent-configuration file is necessary which we can store in our S3 bucket and at the time of launching an instance we will use that agent-configuration file.

Policy 1 :-

This policy will allow your EC2 instance to access the agent-configuration file stored in your S3 bucket.Here you would give the name of your S3 bucket.

Policy 2 :-

This policy will allow your EC2 instance to push the log file stored in your CloudWatch.So here I am assigning all the permissions to my EC2 instance so that it can create log group,log stream and other necessary files.

So create an IAM role and assign two policies to it and at the time of launching EC2 instance you will be assigning this role to your EC2 instance.

<http://www>.tothenew.com/blog/logs-monitoring-using-aws-cloudwatch/

1 Dashboard

Create ,

Alarams : Insufficient,Ok,Alarm

* + - Metric : Select a metric to alarm on.
    - Alarm details : Provide the details and threshold for your alarm. Use the graph to help set the appropriate threshold.

Events : Amazon CloudWatch Events delivers a near real-time stream of system events that describe changes in Amazon Web Services (AWS) resources.

Step 1: Create rule

Create rules to invoke Targets based on Events happening in your AWS environment.

Targets

Select Target to invoke when an event matches your Event Pattern or when schedule is triggered.

Event Buses

Default event bus accepts events from AWS services, PutEvents API calls, and other authorized accounts. You can manage permissions on the default event bus to authorize other accounts. Other AWS accounts can share their events with you by adding your default event bus as a target to their rules

Logs

CloudWatch Logs Insights is now available!

Log Groups

Log Streams

Metrics

All Resources

**Amazon Elasticsearch Service**

Amazon Elasticsearch Service (Amazon Elasticsearch Service) makes it easy to set up, operate, and scale an Elasticsearch cluster in the cloud.

Elasticsearch has REST API operations for everything—including its indexing capabilities. Besides the REST API, there are AWS SDKs for the most popular development languages. In this guide,

Indexing is the core of Elasticsearch. It’s what allows you to perform blazing-fast searches across terabytes of data. But you can’t search data that doesn’t exist. So, in this post, I go over how to create indexes, put data into Elasticsearch, and then search with Elasticsearch using [Amazon Elasticsearch Service](https://aws.amazon.com/elasticsearch-service/).

When your account is ready, create an Amazon Elasticsearch Service domain (cluster with config).

There are only a few basic steps to getting an Amazon Elasticsearch Service domain up and running:

1. Define your domain
2. Configure your cluster
3. Set up access
4. Review

Then launch [Kibana](https://aws.amazon.com/elasticsearch-service/kibana/" \t "_blank) so that you can follow along. Kibana is available via a link in your domain overview. To access it, you need to set up the appropriate permissions. To use Amazon Cognito for granting access, see [Amazon Cognito Authentication for Kibana](https://docs.aws.amazon.com/elasticsearch-service/latest/developerguide/es-cognito-auth.html).

## How to get data into Amazon Elasticsearch Service

In Elasticsearch, data is put into an index as a JSON document. You could explicitly create an index, but there’s no real need for that. Amazon Elasticsearch Service creates an index around the first document you add. This makes it possible to put a document into an index without knowing whether it exists.

**Putting a document into an index**

PUT /vegetables/\_doc/1

{

"name":"carrot",

"color":"orange"

}

You can use any HTTP tool, such as [Postman](https://www.getpostman.com/), curl, or the dev console in Kibana.

Amazon Elasticsearch Service generate an ID for you, like some other JSON repositories

**Auto-generated IDs**

It’s simple to have Amazon Elasticsearch Service generate an ID for your documents. All you have to do is use a POSTinstead of a PUT.

**Updating a document with a post**

Use an HTTP POST with the identifier to update an existing document.

Let’s recap the commands so far:

* PUT creates a document with a specified ID.
* POST updates the document with the specified ID.
* POST also creates a document with an auto-generated ID when you don’t provide one.

### Bulk actions

* Using the \_bulk API operation, you can perform many actions on one or more indexes in one call. Performing several create, update, and delete actions in a single call speeds up your operations. Here’s the basic formula:
* POST /\_bulk
* <action\_meta>\n
* <action\_data>\n
* <action\_meta>\n
* <action\_data>\n

POST /\_bulk

{ "create" : { "\_index" : "veggies", "\_type" : "\_doc", "\_id" : "7" } }

{ "name":"kale", "color":"green", "classification":"leafy-green" }

**How to search with Amazon Elasticsearch Service**

Searching is the main event when it comes to Elasticsearch! Having a lot of data is great, but what good does it do until you actually put it to use? And what better way to start using your data than to search for specific values?

Are you looking for all the root vegetables? Do you need a count of all leafy greens? How about the number of errors logged per hour? The answers all start with an index search.

Let’s take a look at a basic search. Then you can move on to some more advanced searching.

**Basic searches**

Your basic search looks like the following:

GET /veggies/\_search?q=name:l\*

**Advanced searches**

You can do some advanced searching by providing the query options as JSON in the request body. Try the following:

GET /veggies/\_search

{

"query": {

"term": {

"name": "lettuce"

}

}

}

**How to put bulk and streaming data into Amazon Elasticsearch Service**

Now that you know how to search your data, you probably want to try working with massive amounts of your own data. I’m sure you can think of many uses for searching and aggregating your own data. Think of your logs and all the events that occur in your system. Do you have event logs? Event streams? What about data coming in from IoT devices?

This section covers different ways to load streaming data into Amazon Elasticsearch Service. After the data is in, you can start pulling together valuable insights using the search and query APIs that you have already learned about.

We already covered the bulk API, but there’s another way to get data into your [Amazon Elasticsearch Service](https://aws.amazon.com/elasticsearch-service/getting-started/) domain: you can connect a stream data source to it. Here’s how that works.

**Stream data connections**

When you’re running on AWS, you can use your existing data pipelines to feed data into Amazon Elasticsearch Service. There’s a basic pattern for connecting [Amazon S3](https://aws.amazon.com/s3/), [Amazon Kinesis Data Streams](https://aws.amazon.com/kinesis/data-streams/), and [Amazon DynamoDB](https://aws.amazon.com/dynamodb/). You use an [AWS Lambda](https://aws.amazon.com/lambda/) function to connect to the source and put the data into Amazon Elasticsearch Service.

[Kinesis Data Firehose, Amazon CloudWatch, and AWS IoT](https://aws.amazon.com/elasticsearch-service/data-ingestion/) have more integrated solutions. Amazon Elasticsearch Service is a destination for these three streams. For example, you would use a rule action to send IoT stream data to an Amazon Elasticsearch Service domain.

**Amazon Cloud-front**

It is responsible for content delivery, i.e. used to deliver website. It may contain dynamic, static, and streaming content using a global network of edge locations. Requests for content at the user's end are automatically routed to the nearest edge location, which improves the performance.

Amazon Cloud-front is optimized to work with other Amazon Web Services, like Amazon S3 and Amazon EC2. It also works fine with any non-AWS origin server and stores the original files in a similar manner.

In Amazon Web Services, there are no contracts or monthly commitments. We pay only for as much or as little content as we deliver through the service.

**Elastic Caches**

Amazon Elastic Cache is a web service that manages the memory cache in the cloud. In memory management, cache has a very important role and helps to reduce the load on the services, improves the performance and scalability on the database tier by caching frequently used information.