**Top Gear Challenge:**

**Monitoring Cloud Network**

Cisco team get the utilization cost of Available services across all the regions,  
Here data will collect from AWS API by connecting. So no of trips between the scripts of end users and AWS APi is high with the Specified Regions. So, it’s a challenge to reduce the no of networking trips between the AWS and End user.

**Explanation:**

Cisco wants to get the utilization cost of available services across all the regions

Connection to AWS  
by passing access key id and secret access key

List of Regions

$ python3 so\_regions2.py

Using region: **us-east-1**

region\_id: **eu-north-1** region\_name: EU (Stockholm)

region\_id: **ap-south-**1 region\_name: Asia Pacific (Mumbai)

region\_id: **eu-west-3** region\_name: EU (Paris)

region\_id: **eu-west-2** region\_name: EU (London)

region\_id: **eu-west-1** region\_name: EU (Ireland)

region\_id: **ap-northeast-2** region\_name: Asia Pacific (Seoul)

region\_id: **ap-northeast-1** region\_name: Asia Pacific (Tokyo)

region\_id: **sa-east-1** region\_name: South America (Sao Paulo)

region\_id: **ca-central-1** region\_name: Canada (Central)

region\_id: **ap-southeast-1** region\_name: Asia Pacific (Singapore)

region\_id: **ap-southeast-2** region\_name: Asia Pacific (Sydney)

region\_id: **eu-central-1** region\_name: EU (Frankfurt)

region\_id: **us-east-1** region\_name: US East (N. Virginia)

region\_id: **us-east-2** region\_name: US East (Ohio)

region\_id: **us-west-1** region\_name: US West (N. California)

region\_id: **us-west-2** region\_name: US West (Oregon)

**2.**  By Passing the Region and get the available services

get regions

Get available services

We have nearly **330 plus** available services are in AWS

**AWS Account Management, AWS Activate, Alexa for Business, AmazonMediaImport, AWS Amplify, AWS Amplify Admin, AWS Amplify UI Builder, Apache Kafka APIs for Amazon MSK clusters, Amazon API Gateway, Amazon API Gateway Management, Amazon API Gateway Management V2, AWS App Mesh, AWS App Mesh Preview, AWS App Runner, AWS App2Container, AWS AppConfig, Amazon AppFlow, Amazon AppIntegrations, AWS Application Auto Scaling, AWS Application Cost Profiler Service, Application Discovery Arsenal, AWS Application Discovery Service, AWS Application Migration Service, Amazon AppStream 2.0, AWS AppSync, AWS Artifact, Amazon Athena, AWS Audit Manager, AWS Auto Scaling, AWS Backup, AWS Backup Gateway, AWS Backup storage, AWS Batch, AWS Billing and Cost Management, AWS Billing Conductor, AWS Billing and Cost Management Console, Amazon Braket, AWS Budget Service, AWS BugBust, AWS Certificate Manager, AWS Chatbot, Amazon Chime, AWS Cloud Control API, Amazon Cloud Directory, AWS Cloud Map, AWS Cloud9, AWS CloudFormation, Amazon CloudFront, AWS CloudHSM, Amazon CloudSearch, AWS CloudShell, AWS CloudTrail, Amazon CloudWatch, Amazon CloudWatch Application Insights, Amazon CloudWatch Evidently, Amazon CloudWatch Logs, AWS CloudWatch RUM, Amazon CloudWatch Synthetics, AWS CodeArtifact, AWS CodeBuild, AWS CodeCommit, AWS CodeDeploy, AWS CodeDeploy secure host commands service, Amazon CodeGuru, Amazon CodeGuru Profiler, Amazon CodeGuru Reviewer, AWS CodePipeline, AWS CodeStar, AWS CodeStar Connections, AWS CodeStar Notifications, Amazon Cognito Identity, Amazon Cognito Sync, Amazon Cognito User Pools, Amazon Comprehend, Amazon Comprehend Medical, AWS Compute Optimizer, AWS Config, Amazon Connect, Amazon Connect Cases, Amazon Connect Customer Profiles, Amazon Connect Voice ID, Amazon Connect Wisdom, AWS Connector Service, AWS Control Tower, AWS Cost and Usage Report, AWS Cost Explorer Service, AWS Data Exchange, Amazon Data Lifecycle Manager, AWS Data Pipeline, AWS Database Migration Service, Database Query Metadata Service, AWS DataSync, AWS DeepComposer, AWS DeepLens, AWS DeepRacer, Amazon Detective, AWS Device Farm, Amazon DevOps Guru, AWS Direct Connect, AWS Directory Service, Amazon DynamoDB, Amazon DynamoDB Accelerator (DAX), Amazon EC2, Amazon EC2 Auto Scaling, Amazon EC2 Image Builder, Amazon EC2 Instance Connect, AWS Elastic Beanstalk, Amazon Elastic Block Store, Amazon Elastic Container Registry, Amazon Elastic Container Registry Public, Amazon Elastic Container Service, AWS Elastic Disaster Recovery, Amazon Elastic File System, Amazon Elastic Inference, Amazon Elastic Kubernetes Service, Elastic Load Balancing, Elastic Load Balancing V2, Amazon Elastic MapReduce, Amazon Elastic Transcoder, Amazon ElastiCache, AWS Elemental Appliances and Software, AWS Elemental Appliances and Software Activation Service, AWS Elemental MediaConnect, AWS Elemental MediaConvert, AWS Elemental MediaLive, AWS Elemental MediaPackage, AWS Elemental MediaPackage VOD, AWS Elemental MediaStore, AWS Elemental MediaTailor, AWS Elemental Support Cases, AWS Elemental Support Content, Amazon EMR on EKS (EMR Containers), Amazon EMR Serverless, Amazon EventBridge, Amazon EventBridge Scheduler, Amazon EventBridge Schemas, AWS Fault Injection Simulator, Amazon FinSpace, AWS Firewall Manager, Amazon Forecast, Amazon Fraud Detector, Amazon FreeRTOS, Amazon FSx, Amazon GameLift, Amazon GameSparks, AWS Global Accelerator, AWS Glue, AWS Glue DataBrew, AWS Ground Station, Amazon GroundTruth Labeling, Amazon GuardDuty, AWS Health APIs and Notifications, Amazon HealthLake, High-volume outbound communications, Amazon Honeycode, AWS IAM Access Analyzer, AWS IAM Identity Center (successor to AWS Single Sign-On), AWS IAM Identity Center (successor to AWS Single Sign-On) directory, Identity And Access Management, AWS Identity and Access Management Roles Anywhere, AWS Identity Store, AWS Identity Store Auth, AWS Identity Sync, AWS Import Export Disk Service, Amazon Inspector, Amazon Inspector2, Amazon Interactive Video Service, Amazon Interactive Video Service Chat, AWS IoT, AWS IoT 1-Click, AWS IoT Analytics, AWS IoT Core Device Advisor, AWS IoT Core for LoRaWAN, AWS IoT Device Tester, AWS IoT Events, AWS IoT Fleet Hub for Device Management, AWS IoT FleetWise, AWS IoT Greengrass, AWS IoT Greengrass V2, AWS IoT Jobs DataPlane, AWS IoT RoboRunner, AWS IoT SiteWise, AWS IoT TwinMaker, AWS IQ, AWS IQ Permissions, Amazon Kendra, AWS Key Management Service, Amazon Keyspaces (for Apache Cassandra), Amazon Kinesis, Amazon Kinesis Analytics, Amazon Kinesis Analytics V2, Amazon Kinesis Firehose, Amazon Kinesis Video Streams, AWS Lake Formation, AWS Lambda, Launch Wizard, Amazon Lex, Amazon Lex V2, AWS License Manager, AWS License Manager User Subscriptions, Amazon Lightsail, Amazon Location, Amazon Lookout for Equipment, Amazon Lookout for Metrics, Amazon Lookout for Vision, Amazon Machine Learning, Amazon Macie, AWS Mainframe Modernization Service, Amazon Managed Blockchain, Amazon Managed Grafana, Amazon Managed Service for Prometheus, Amazon Managed Streaming for Apache Kafka, Amazon Managed Streaming for Kafka Connect, Amazon Managed Workflows for Apache Airflow, AWS Marketplace, AWS Marketplace Catalog, AWS Marketplace Commerce Analytics Service, AWS Marketplace Entitlement Service, AWS Marketplace Image Building Service, AWS Marketplace Management Portal, AWS Marketplace Metering Service, AWS Marketplace Private Marketplace, AWS Marketplace Procurement Systems Integration, AWS Marketplace Vendor Insights, Amazon Mechanical Turk, Amazon MemoryDB, Amazon Message Delivery Service, AWS Microservice Extractor for .NET, AWS Migration Hub, AWS Migration Hub Orchestrator, AWS Migration Hub Refactor Spaces, AWS Migration Hub Strategy Recommendations, Amazon Mobile Analytics, AWS Mobile Hub, Amazon Monitron, Amazon MQ, Amazon Neptune, AWS Network Firewall, AWS Network Manager, Amazon Nimble Studio, Amazon OpenSearch Service, AWS OpsWorks, AWS OpsWorks Configuration Management, AWS Organizations, AWS Outposts, AWS Panorama, AWS Performance Insights, Amazon Personalize, Amazon Pinpoint, Amazon Pinpoint Email Service, Amazon Pinpoint SMS and Voice Service, Amazon Pinpoint SMS Voice V2, Amazon Polly, AWS Price List, AWS Private Certificate Authority, AWS Proton, AWS Purchase Orders Console, Amazon QLDB, Amazon QuickSight, Amazon RDS, Amazon RDS Data API, Amazon RDS IAM Authentication, AWS Recycle Bin, Amazon Redshift, Amazon Redshift Data API, Amazon Redshift Serverless, Amazon Rekognition, AWS Resilience Hub Service, AWS Resource Access Manager, AWS Resource Explorer, Amazon Resource Group Tagging API, AWS Resource Groups, Amazon RHEL Knowledgebase Portal, AWS RoboMaker, Amazon Route 53, Amazon Route 53 Domains, Amazon Route 53 Recovery Cluster, Amazon Route 53 Recovery Controls, Amazon Route 53 Recovery Readiness, Amazon Route 53 Resolver, Amazon S3, Amazon S3 Glacier, Amazon S3 Object Lambda, Amazon S3 on Outposts, Amazon SageMaker, Amazon SageMaker Ground Truth Synthetic, AWS Savings Plans, AWS Secrets Manager, AWS Security Hub, AWS Security Token Service, AWS Server Migration Service, AWS Serverless Application Repository, AWS Service Catalog, AWS service providing managed private networks, Service Quotas, Amazon SES, Amazon Session Manager Message Gateway Service, AWS Shield, AWS Signer, Amazon Simple Email Service v2, Amazon Simple Workflow Service, Amazon SimpleDB, AWS Snow Device Management, AWS Snowball, Amazon SNS, AWS SQL Workbench, Amazon SQS, AWS Step Functions, AWS Storage Gateway, Amazon Sumerian, AWS Support, AWS Support App in Slack, AWS Support Plans, AWS Sustainability, AWS Systems Manager, AWS Systems Manager GUI Connect, AWS Systems Manager Incident Manager, AWS Systems Manager Incident Manager Contacts, AWS Tag Editor, AWS Tax Settings, Amazon Textract, Amazon Timestream, AWS Tiros, Amazon Transcribe, AWS Transfer Family, Amazon Translate, AWS Trusted Advisor, AWS WAF, AWS WAF Regional, AWS WAF V2, AWS Well-Architected Tool, Amazon WorkDocs, Amazon WorkLink, Amazon WorkMail, Amazon WorkMail Message Flow, Amazon WorkSpaces, Amazon WorkSpaces Application Manager, Amazon WorkSpaces Web, AWS X-Ray**

**3.** Get the utilization cost of each service on **DAILY, MONTHLY, YEARLY** Basis based on requirements

Get the cost and maintain in csv file

Get usage cost of usage services

get available services

4. in this way, by calling the python script every time to check the cost of usage of each service, it may cause the network trip between the Aws and end user. So, it not good for the health of AWS and end user for every time a numerous number of end user may trigger to AWS   
  
a. Every time need connect the AWS for every trigger based on the requirement

b. No. of triggers increase from end user side to get the connection with AWS and get the response

c. increases the time taken to get the response from AWS rest Api’s

d. network traffic is also increased

e. if the same service has utilization cost of yesterday, for some have daily, for someone have monthly /yearly.

**Request r1**

List of Region

End User Request

**r2,r3**

**r4**

List of services

Daily Cost

Yesterday Cost

Monthly Cost

Yearly Cost

See the above Algorithm, how the flow will go and execute it  
  
Example: **Request r1, r2, r3, r4** wants to connect the AWS Rest Api

1. **r1** wants to get the usage cost of **Daily cost** of one specified region having available services
2. **r2** wants to get the usage cost of **yesterday cost** of one specified region having available services
3. **r3** wants to get the usage cost of **Monthly cost** of one specified region having available services
4. **r4** wants to get the usage cost of **Yearly cost** of one specified region having available services
5. **r5** wants to get the usage cost of **Daily cost** in all regions having available services
6. **r6** wants to get the usage cost of **Yesterday** **cost** in all regions having available services
7. **r7** wants to get the usage cost of **Monthly cost** in all regions having available services
8. **r8** wants to get the usage cost of **Yearly cost** in all regions having available services
9. **r9, r10, r11, r12, ………** based on requirement a new request trigger the AWS
10. we have nearly **300 plus** available services in all regions in AWS

See, just imagine if a 1 end user wants to get the data from AWS

1. How many ways end user wants the response data based on the requirement
2. Every time end user wants to connect AWS
3. Just imagine in an organization, how many end users are there and how many times they want to get the usage cost of date range with respect to services and regions
4. See how the network traffic is high for every request it’s a new network trigger
5. If in case, any changes happened in AWS, or outage of AWS Server down, in middle its failed

So, after exploring the AWS Documents and work on it, I found these drawbacks and I have a solution to overcome these drawbacks above mentioned.

**Solution:**

To Resolve these above issues, I came up with a solution and get the response as expected

1. Write a one script as a job and run it once in day and get the required response and store it in DB in table
2. When we call the job and maintain in DB daily, then no need to call the separate AWS call for Yesterday, Monthly and Yearly data
3. We need to fetch the data from DB, by passing the yesterday date / month dates / year dates in our API
4. So here no of trips between AWS API will mostly Decreased
5. Here no. of Db calls also reduced if we used Hibernate cache

**`request r1,r2,r3,…**

End user

Get regions

Cost data available in DB

get all available services

**(if no)**

Daily Cost

Yesterday cost

Monthly Cost

Yearly cost

**response R1,R2…**

**(if yes)**

HB

Database

Dashboard

See, the above flow and observe, how it going and useful to us in environment

1. No of trips are reduced as much if we follow this flow
2. Response also generated fast compared to previous one
3. If AWS server will go to outage or server down, no need to worry, get the data from Data base
4. Network Traffic is also reduced.
5. Db Calls also reduced if we are used Hibernate Cache.

See Here, so many no. of trips reduced between AWS and End user to get their response

Example:

1. If Request r1 wants to get the usage cost of yesterday cost, then first request checking in DB using Cache Level 1 and Level 2. if not, data is present in DB, then it connects to AWS
2. If Request r2 wants to get the usage cost of Daily cost, then first request checking in DB using Cache Level 1 and Level 2. if not, data is present in DB, then it connects to AWS
3. If Request r3 wants to get the usage cost of Monthly cost, then first request checking in DB using Cache Level 1 and Level 2. if not, data is present in DB, then it connects to AWS
4. If Request r2 wants to get the usage cost of Yearly cost, then first request checking in DB using Cache Level 1 and Level 2. if not, data is present in DB, then it connects to AWS

**This script is running only once and get the daily cost of all available regions of their available services and store in Data Base**   
  
if we follow this process, we have **benefits** like this

1. If End user wants to check the **Yesterday cost, Monthly Cost, Yearly cost or any specified Date range,** no need to connect with AWS for every request, directly we can connect with DB for same result which is store before.
2. If end user wants to check the Daily cost of available service in specified region, it will check in Db if, already script job in run in that day and it stores in DB, so no need to connect again AWS
3. This way we can avoid much traffic between AWS and end user, but we find much traffic between end user and Db
4. To reduce the Db calls we are using Hibernate with Cache levels, so that we must avoid as much as of less DB calls and get response
5. Response time is low and very Fast also.

In diagram **HB means Hibernate**

we can also maintain data in Dashboard, so that in an organization every one can use it and no need to connect AWS, and run the script every time and also, we need to get the specified date range also with this way of planning structure.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* END\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*