WHITE PAPER

BUILDING COST EFFECTIVE DATA ARCHITECTURES FOR IOT PLATFORMS

Infosys®
Navigate your next

Introduction

Public cloud platforms are the backbone of many modern systems. Public cloud providers offer a wide range of services, from pure compute and storage to rich platform services such as event streams, databases, cache and container platforms. Cloud platforms are meant to replace private datacenters because of the complexity and high cost of running datacenters. Most system architectures, simple or complex, are built using cloud services, giving rise to the term Cloud Native Engineering or the practice of designing systems using cloud native services. However, if a system is not designed carefully, cloud costs can soon spiral out of control.

In large-scale cloud engagements, platform cost has emerged as a chief consideration, and rightly so. Most customers expect short to medium term cost projections at the start to plan their budgets. Architects need to understand the pricing of various services while designing systems on the cloud. In this article, we discuss techniques that can help control cloud costs when designing large systems.

Cloud providers such as AWS publish pricing for their services. Consider an IoT set up with millions of devices connected to the cloud and send events at frequent intervals. The platform must be able to ingest all events, store and process them. As with typical data architecture, this architecture should follow a hot path for real-time data processing and notifications and a cold path for later use, such as data science and analytics.

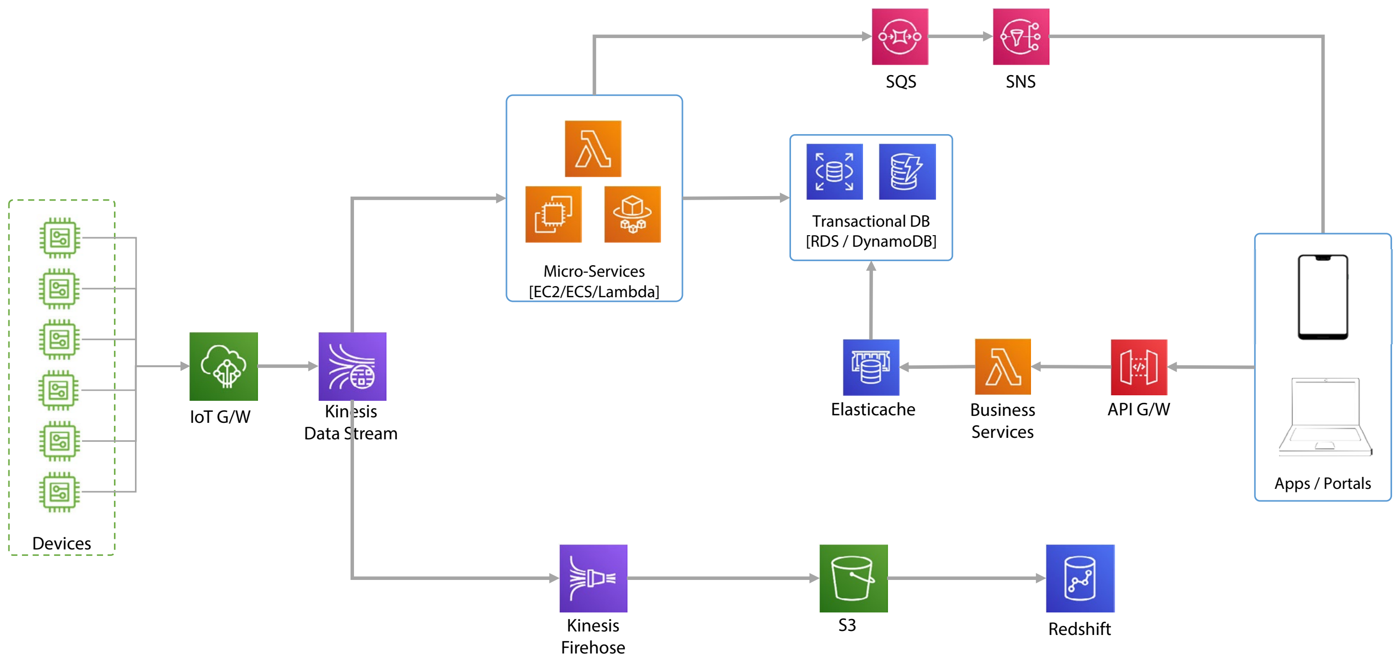

Fig 1: High-level IoT platform architecture using AWS services

Building a platform on AWS would typically involve the following services:

Service	Usage & Pricing
IoT Gateway	Ingests event data from IoT devices and supports multiple protocols. We can apply rules and take actions on incoming messages.
Kinesis Data Streams	High throughput persistent data stream for incoming data. Couple it with Kinesis Firehose to transfer data directly to S3 without custom code
Kinesis Data Firehose	Deliver incoming data to a cold storage such as S3 or Redshift
S3	Object storage for event data, used for storing raw data from all sources.
RDS / DynamoDB	Database for processed data. It serves as a transactional database for front-facing business services.
EC2 / ECS / EKS / Lambda	Various options for hosting micro-services for business APIs
SNS / Pinpoint	Sending notifications
IAM / Cognito / WAF / Shield	Access and security considerations
Redshift	Data warehouse for business analytics
API Gateway	API management

In addition, we need other services such as networking, security, KMS and Parameter Store to design an enterprise level system.

While the pricing of cloud services is not significant, the costs can pile up quickly depending on the volume of data flowing in and processed by the platform.

Considerations for Cloud Cost Optimization

This section discusses a few cloud services in detail, provides an understanding of their pricing and explains how to optimize costs by making informed architectural decisions.

Kinesis Data Firehose

This service can ingest an incoming stream of data and deliver it directly to certain AWS services such as S3 and Redshift without custom code. It is a highly durable service and can buffer incoming messages for a certain duration before delivering to S3.

Pricing: Firehose Pricing

Consider a case where Kinesis Firehose will write to S3. Here, the charges will be for both Firehose and S3. Firehose charges are based on data volume ingested, and there are additional charges for format conversion and data delivery to a different virtual private cloud (VPC). S3 charges are based on the data volume (storage needed), and each read and write.

Writing each incoming message to S3 will incur higher charges than buffering the messages and restricting write operations to S3.

Cost Optimization

While it is not possible to save on Firehose's charges, it is possible to save on S3 bills. By default, Firehose writes each incoming message to S3 independently, which would count towards the number of write operations in S3. If the business case permits, we can configure Firehose to buffer incoming messages and write to S3 only periodically. This leads to reduced write operations in S3 and saves write charges.

Firehose service provides two settings – buffer size and buffer interval. For S3, it can buffer up to 128MiBs and buffer interval can be from 1 min to 15 min (AWS reference). In an IoT scenario, the number of devices and the number of messages are usually high, whereas the message size is quite small. In such a scenario, buffering messages in Firehose can save costs significantly over a period.

S3: Simple Storage Service

A widely used object storage service, S3 is an extremely cheap and highly available storage service. Within S3, AWS offers various tiers or storage classes with different durability levels and retrieval times, and the prices of these tiers vary accordingly. S3 standard tier, which is most durable and highly available, is priced highest, whereas Deep Archive, which has retrieval times of up to 48 hrs, is priced lowest. Read more about S3 storage classes here.

Pricing: S3 Pricing

Storage costs in S3 depend on multiple factors such as data volume, storage tier, duration of storage and number of reads/writes. Costs can increase if these configurations are not set up appropriately.

Cost Optimization

The basic rule is – One size does not fit all. Data storage should differ based on the use cases and types of data . Some of the techniques to control storage cost are:

a. Choose the right storage class for your objects. The decision should be based on durability and availability requirements, and the retrieval times that the business can accommodate.

b. S3 provides lifecycle policies that can automatically move data between tiers based on predefined conditions in the policy. By effectively using lifecycle policies, an organization can control S3 charges by moving data to lower tiers for long term archival. Fig 2 gives a view of supported transitions between different storage classes. Read about these transitions in detail here.

c. Compression: In large enterprises, S3 stores raw data from multiple disparate sources and data volume can be significantly high. Users can save substantially by compressing data before uploading it to S3. The user must analyze data sources and identify good candidates for compression. Kinesis Data Firehose supports few compression formats natively, which can be applied in Firehose itself. There are other popular formats like Parquet that offer several advantages. AWS offers S3 Select service for querying Parquet format data from S3 natively.

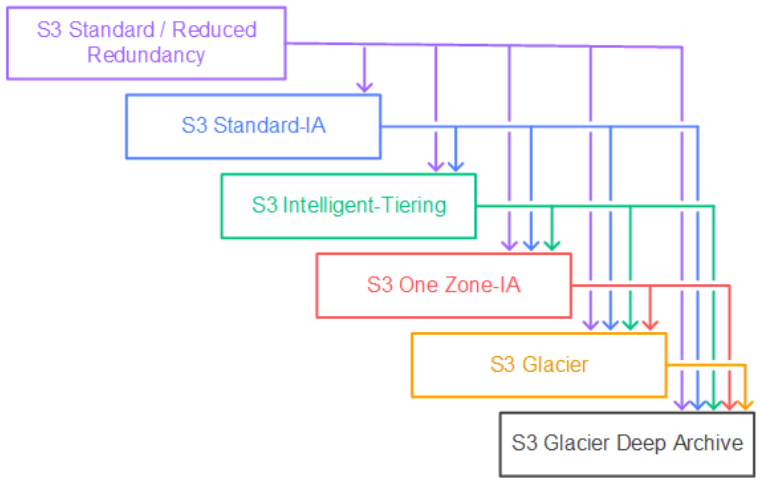

Fig 2: Source: AWS

Lambda Functions

Here is another popular service used in serverless architectures. Enterprises cannot control the number of requests, and hence memory allocation and execution time can control Lambda costs.

Pricing: Lambda Pricing

Charges for Lambda functions depend on the number of function invocations, memory allocated, and the time the function takes to execute.

Cost Optimization

When creating the Lambda function, the memory must be carefully configured. Allocating lesser memory can increase execution time, whereas configuring maximum available memory may not yield desired results. The right approach is to go with an estimated baseline and then fine-tune the memory after running a few tests. This exercise will help strike a balance between the memory needed and the time of execution.

EC2: Elastic Compute

A pure compute service that is offered through multiple hardware configurations. AWS offers various instance types such as general purpose, compute optimized and memory optimized. These types are also known as instance families.

Instances are offered under five major categories: On-Demand, Spot, Reserved, Dedicated Instance and Dedicated Host. The charges for each are based on both the instance family as well as the category.

On-Demand	Reserved	Spot	Dedicated Instance	Dedicated Host
These instances are a great fit for general workload scenarios where the usage is unpredictable, and load can change randomly.	Reserved instances are suitable for scenarios where the application load is predictable, and you know what you need.	Spot is a good option if an instance stops anytime, then the system architecture ensures there is no data loss when the instance is lost.	These instances run on hardware dedicated to a customer. Hence, they are isolated from other AWS accounts at the hardware level.	These are physical servers dedicated to a customer and suitable for scenarios where the customer brings their own software licenses to run on dedicated servers. Customers also get more hardware control in this category.

Pricing: EC2 Pricing

Charges vary based on the instance family, instance categories and usage.

On-Demand	Reserved	Spot	Dedicated Instance	Dedicated Host
On-demand option is more expensive than others, but it does not mandate a commitment. You pay for what you use.	These instances provide up to 75% discount compared to on-demand instances. However, they need to be purchased for the long term.	Spot instances are the cheapest among all compute instances. They can offer up to 90% discount compared to on-demand instances, but their prices can vary.	These instances have two pricing components – per region monthly fee and a usage charge that varies depending on whether the usage is On-demand or Reserved.	These servers are priced depending on the instance family. Prices vary between On-demand, Spot and Reserved. Spot and Reserved hosts are cheaper than On-demand.

Cost Optimization

We can control EC2 costs in multiple ways:

a. Consider the requirements and select the appropriate category of instances. Depending on the use case, a suitable candidate can be On-demand, Spot or Reserved. It is practical to have different instance categories for different applications in the same AWS account as the use cases may vary.

b. Carefully choose instance family, i.e., hardware configuration, as each family type offers a particular combination of CPU and memory and directly impacts the price.

- For general workloads, use general purpose (T series)
- For memory intensive workloads, use memory optimized instances (R, X, Z series)
- For CPU intensive workloads, use CPU optimized (C series)

Start with a smaller configuration, monitor usage and upgrade as needed. Read more on instance types and their differences here.

c. In addition, we can employ techniques like scheduled Start and Stop instances using CloudWatch alarms and actions. For available workload schedules, we can stop them using CloudWatch when they are no longer needed, saving unnecessary usage and charges.

ECS: Elastic Container Service

It is a container orchestration service where application containers are managed by an AWS managed service for running Docker containers. It comes with an option to execute containers in a serverless model known as Fargate.

Pricing: ECS Pricing

It can be used in two modes – Fargate or EC2. The former is a serverless model that charges on vcpu and memory selected, whereas the latter charges on the instance type selected. Both options have their pros and cons, but typically, EC2 costs lesser than Fargate for a similar configuration.

Cost Optimization

The mechanisms to control costs in ECS are similar to EC2, as listed below:

a. Study the requirements carefully and decide whether you need the EC2 or Fargate model. For Fargate, calibrate the vcpu and memory required for running tasks. You may need to run a few tests and check usage metrics to fine-tune vcpu and memory.

b. Ensure you have running tasks that run for long (24/7) only if needed - shutdown the tasks when not required.

c. Capacity planning: Spin off only the required tasks as each will be charged. You need to consider the overall architecture and use case. If the requests are buffered and can be served near real-time, the system can work with fewer tasks. On the other hand, if the requests expect real-time responses, such as UI requests, capacity planning comes into the picture. Always run a smaller number of tasks and add scaling policies to scale out based on the load.

RDS: Relational Database Service

It is a platform managed database service that offers a broad range of relational database engines.

Pricing: RDS Pricing

It varies based on the engine type selected and the instance types. In general, there are two options to choose from: On-demand instances and Reserved instances. As observed in EC2, Reserved instances always offer a better price if there is a commitment of at least a year.

Cost Optimization

The levers that can help in cost control include:

a. a.   Right database engine: Licensed engines such as Oracle and SQL server are expensive, while MySQL and PostgreSQL are cheap. AWS also implements MySQL and PostgreSQL engines through RDS Aurora, which is cheaper and faster.

b. Capacity planning: Estimate the optimal instance class for DB nodes. Getting to a good approximation being difficult, choose an instance class, monitor usage, and change the instance class as needed.

c. Scaling policies: In RDS, write instance can only scale up but read instances can scale out. Start with one reader node and add an auto-scaling policy to add read-replicas when the read traffic increases. Enable scale-in so that replicas are removed when the request load comes down.

Cluster endpoint for writer services and reader endpoint for read-only services: When endpoints are used incorrectly, it can mislead on DB capacity – for example, it can wrongly signal that DB capacity is insufficient during peak load. Moreover, when a scaling event occurs, the reader endpoint load-balances traffic between read replicas.

DynamoDB: NoSQL Database Service

This is a popular NoSQL database service as it provides extremely high throughput, high availability and several useful features. At the same time, its pricing is complex and can significantly increase your AWS bills. It is critical to understand how AWS charges for DynamoDB and carefully design the database, indexes and queries.

Pricing: DynamoDB Pricing

DynamoDB is charged based on multiple factors, including but not limited to storage, read/write requests, indexes, scans and backups. As in EC2, DynamoDB offers On-Demand capacity and Provisioned capacity. The former is suitable for sporadic workloads, and the latter is useful for predictable workloads where performance guarantees are needed. Provisioned capacity offers a reserved mode that is significantly cheaper, but the user has to pay a one-time cost at the start along with an hourly price.

Cost Optimization

Some of the common and important pricing levers and the techniques that can help bring down the costs are:

a. On-Demand vs. Provisioned: On-demand charges are generally higher than provisioned. In the case of provisioned, payment must be made for the provisioned capacity irrespective of whether it is used or not. Therefore, it is a balancing act between the two. A good way is to project the workload and provision capacity equal to the lower end of the projected range. This approach will generate savings for minimum traffic, and as the workload increases, auto-scaling can increase capacity at run-time.

b. Read & Write Requests: This is the core of DynamoDB pricing. All Read and Write requests consume RCUs (read capacity units) and WCUs (write capacity units), which are the basic units for pricing calculations. Depending on the data size, one read/write operation can consume one or more RCUs/WCUs, respectively. There are two ways to reduce read operations on the DB:

- Optimize queries to read only the required data rather than fetching all the attributes for a key. RCUs depend on the amount of data returned. Hence a lesser number of attributes are returned, leading to lesser charges.
- Use cache to access frequently used data. This approach requires mechanisms to refresh the cache, keep it consistent with the DB and pay for the cache, but it will soon offset the RCU charges incurred on DynamoDB.

c. DAX: A DynamoDB cache, it is managed and offered by AWS. DAX is charged based on the type of node used and for the duration used. For read-heavy use cases, DAX can be a good fit to save RCUs.

d. Global Secondary Indexes: Indexes also contribute to WCU and RCU. DynamoDB creates each index as a separate table. When creating indexes, address these points:

- Add attribute projections to indexes to reduce WCUs. DynamoDB maintains separate tables for indexes, reducing the number of attributes.
- Remove unused indexes.

This is a good reference to understand how global secondary indexes work.

e. Table Scans: A scan operation is where the DB engine parses all the data in a table because it is not passed on a partition key. It is charged based on the volume of data scanned, and not the volume of data returned. Scan operations can be expensive depending on the table size and number of operations. They must be avoided where possible.

In addition, DynamoDB charges for Global tables, Backups and Streams, but they are not covered in this paper

Redshift: Data Warehouse

This is platform managed data warehouse service, used by data scientists to cleanse and prepare data for analysis purposes. Redshift needs compute and storage to load the data and run queries. Traditionally, one needs to provision Redshift clusters and load data from supported sources. Users can then run complex SQL queries to filter and analyze data. Redshift uses efficient columnar format for data which improves query execution efficiency since it only scans the columns needed for the query.

Pricing: Redshift Pricing

Redshift pricing depends on the compute nodes and disks provisioned in the cluster. It offers pay-as-you-go pricing model where one pays based on usage of provisioned hardware.

Cost Optimization

There are broadly two ways to save costs – go for long-term commitment or avoid loading all the data in Redshift:

a. Reserved Instance: Redshift offers pay-as-you-go and reserved instance models. The traditional approach of loading full data in Redshift can turn out to be quite expensive depending on cluster size and capacity. Like EC2, opting for reserved instances with long-term commitment is better than pay-as-you-go.

b. Redshift Spectrum: With this service, one need not load the data in Redshift servers. Spectrum allows users to run SQL queries directly on S3 without loading the data in Redshift, and so is a potential lever to save money. With Spectrum, one pays for data scanned. Thus, if the data is stored in an efficient columnar format in S3, it would reduce the amount of data scanned and thus reduce the cost further.

c. Use Distribution & Sort Keys: Absence or incorrect use of these keys can significantly impact how the data is stored and processed. To explain briefly, Distribution key is like the primary partition key. If the queries are generally executed based on a specific column value, eg. an identifier, then specifying it as the distribution key will ensure that data is partitioned on that column. In this way, all the matching data for a particular value of distribution key column, will be stored together on 1 node. This increases query response time as all the query data is co-located. On the other hand, if the data size for a distribution key is huge, query performance may be better if data is distributed across all nodes, as Redshift will process the data on all nodes in parallel and aggregate the results, which will increase the efficiency.

The sort key is used to sort data on a specific column. For eg. if the query results are to be sorted on time, and if time column is configured as sort key, Redshift will store the data in a sorted fashion. If it is used in conjunction with distribution key, then data will be partitioned and sorted. Both these are very powerful tools in determining the query plan and its efficiency.

Conclusion

Managing cloud platform costs is a continuous process. The techniques covered in this article have potential to save hundreds of thousands of dollars in cloud spend over time. These techniques are helpful in making right architectural decisions but once the platform is developed and goes live, it is important to keep an eye on the bills and monitor usage. AWS provides services such as Cost Explorer and Budgets to analyze service costs and to create a budget respectively. When starting out, one should study the costs over a period and create baseline budgets for one or more services. Budgets can be created for cost or usage and they can send alerts when they crosses a defined threshold.

Cost efficiency is one of the core pillars of well-architected framework and architects must pay equal attention to it along with other pillars while designing platforms. Although this article is based on AWS services, similar thought process should be applicable to other cloud providers as well. The idea is to think intricately about each cloud services' pricing and look for windows of optimization.

***Disclaimer:** This article is based on AWS based IoT platform implementations for multiple clients and technical research.*

About the author

Ruchin Goel, *Senior Architect, Engineering Services, Infosys*

Ruchin Goel is a Senior Technology Architect with the IoT group of Infosys Engineering Services. He has over 20 yrs. of experience in design and development of enterprise products & platforms in Infosec, Telematics and IoT space. He has experience on multiple cloud platforms such as AWS and Azure.

Ruchin works on large green-field and transformation programs for IoT and telematics clients across geographies. His primary responsibilities include designing end-to-end platform architectures with focus on data and security architectures.

For more information, contact askus@infosys.com

Infosys®
Navigate your next

Infosys.com | NYSE: INFY

Stay Connected