



# Technical Whitepaper

Totogi Charging-as-a-Service on AWS

Version 1.0

May 26, 2023

## Table of Contents

1. Introduction	3
2. Integrating Totogi Charging-as-a-Service	4
3. Lifecycle of a Charging Request	7
4. Evaluating Charging Data Requests	8
5. Configuring Rate Plans	13
6. The Optimal Database for Charging	14
7. Privacy: Data Isolation & Multi-Tenancy	16
8. Resiliency	20
9. Security	21
10. Latency	24
11. Conclusion	27
12. Appendix	29

# 1. Introduction

Communications service providers (CSPs) deploy standardized network components to deliver voice, data and messaging services to their subscribers. A network function called Online Charging System (OCS) for 4G and Converged Charging System (CCS) for 5G is responsible for metering, rating and charging of network services and maintaining records of service usage for each subscriber. It provides signals to other network functions like the Session Management Function (SMF) to prevent subscribers from consuming services without authorization or exceeding the limits of contracted quota. The usage data that is metered in the charging system is used to deduct balance in real-time and generate reliable usage records for the CSP. The usage records are important inputs for generating invoices and providing marketing intelligence to help CSPs optimize their revenue streams.

**Totogi Charging-as-a-Service** implements the 3GPP standard interfaces for OCS and CCS and enables CSPs to incorporate the service into their core network without installing or managing servers, software or databases on their own. Totogi is a modern, multi-tenant SaaS charging function that provides the flexibility CSPs need to launch new digital brands, empower new MVNOs and replace legacy charging systems they have outgrown or are too costly to operate and maintain.

Charging systems are the heart of a CSPs network and are carefully selected to meter and monetize their core assets. Existing CSPs will be reluctant to perform a heart transplant without careful consideration of the outsized benefits of shifting to something new. Both new and established CSPs need to vet the technology and understand any risks associated with adopting the world's first multi-tenant SaaS charging system. In this paper, we deep dive into Totogi's technical architecture, explain how it is securely incorporated into the CSPs core network and articulate important technical decisions that uniquely position Totogi Charging-as-a-Service.

## 2. Integrating Totogi Charging-as-a-Service

### Southbound Interfaces

Totogi Charging-as-a-Service implements the standard 3GPP OCS and CCS defined in Release 16/17 interfaces that Network Functions like the IMS, SMSC, P-GW and SMF use to verify that subscribers are authorized to consume their services and to subsequently record the units of service consumed during a session. The units are service dependent, and can reflect time (voice minutes) events (text messages) or content based charging of data services (WhatsApp megabytes vs general megabytes). 4G LTE networks use the Diameter protocol to exchange messages with the OCS and 5G networks use JSON messages over HTTP/2. By implementing the 3GPP standards, Totogi is interoperable with core network functions that implement these same protocols. These 3GPP interfaces to the core network functions are referred to as “southbound” interfaces.

While 3GPP defines the standard, there are aspects of the protocol like vendor defined attribute-value pairs (AVPs) that are open to interpretation and extension. Totogi has built-in configurable field mapping which eliminates the need for middleware and directly adapts the 4G and 5G interfaces to the OCS clients in each operator’s core network.

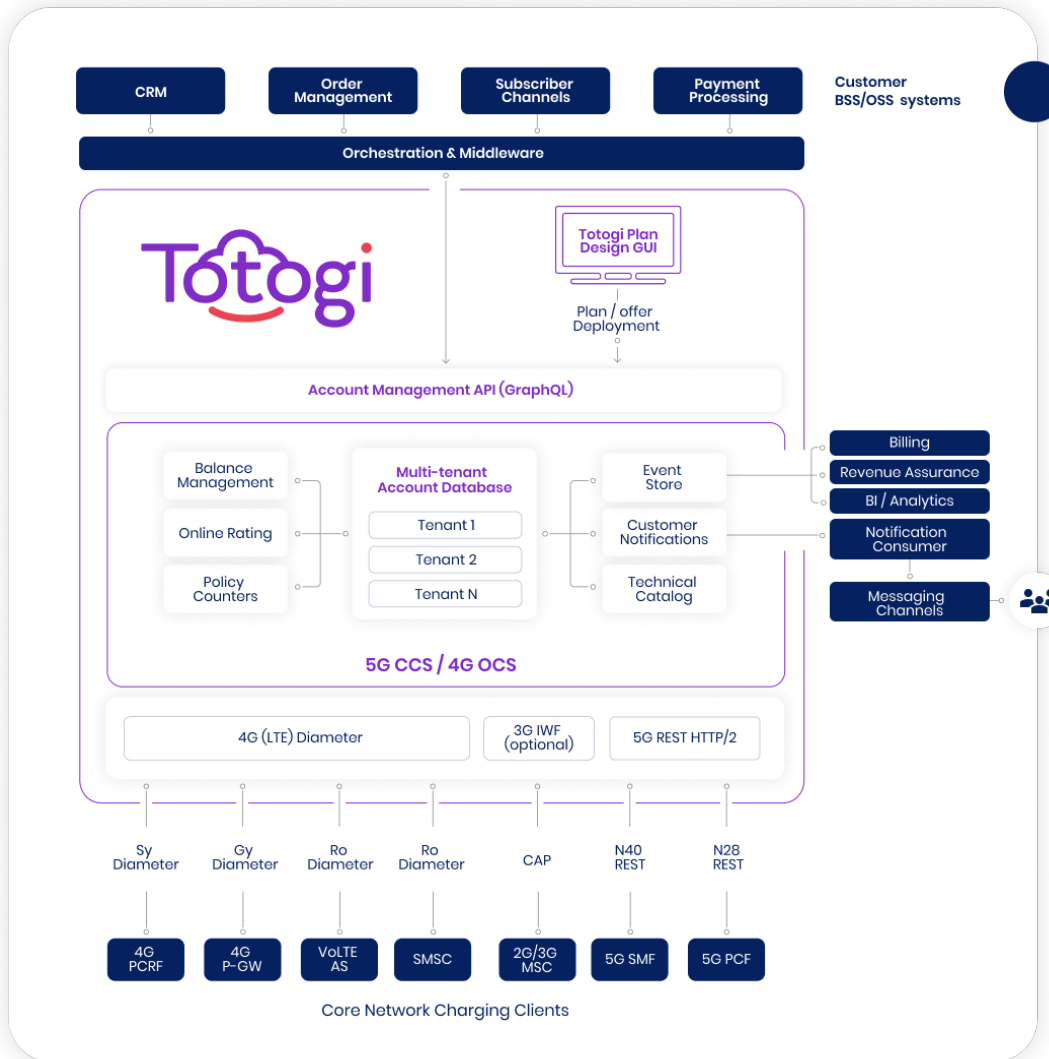


Figure 1: Functional Architecture

## Northbound Interfaces

Totogi’s “northbound” Account Management API enables BSS and OSS systems to accomplish tasks like:

- Create plans dynamically;
- Override existing plans;
- Query balances;
- Obtain usage details;
- Provision subscribers;
- Maintain the state of subscriber;
- Establish or topup balances; and

- Assign rate plans to subscribers.

The northbound Account Management API is provided as a suite of GraphQL queries and mutations, which is not only compatible with most software development languages and toolkits, but also more developer friendly because it enables them to request the specific data they require for their use case. Using the APIs, CSPs implement interfaces with CRM systems, order management systems and various subscriber channels such as self-service ecommerce and in-store POS systems.

## EDR Interface

In addition to maintaining accurate real-time balances for each service that a subscriber is entitled to, the charging system also generates Event Detail Records (EDRs) which provide an audit trail accounting of each subscriber's consumption of network services. While these EDRs are available by API at an account or device level, they are also delivered as files in an open binary format ([ORC](#)) for efficient processing. The files are partitioned by date and by event type:

- **Charging EDRs** are produced for any event received by the charging engine, e.g. CCR Init, CCR Update (4G).
- **Billing EDRs** are produced at the end of a session (CCR Terminate) deducting the balance and totalling the units used in the session from init to terminate.
- **Audit EDRs** are produced when changes affect the subscriber profile, such as when an account is subscribed to a plan or when a low balance warning is published to a subscriber.

Each Totogi customer configures the Amazon S3 bucket to receive their EDR files in. EDRs are buffered and streamed into files every 15 minutes - or more frequently with higher volumes. Customers can configure S3 events to trigger processing as files are created, enabling streaming of records into Kafka or ingesting them into data processing pipelines.

## Notifications Interface

The charging system generates notifications that are often of interest to CRM, self-care systems and end-subscribers. Notifications are generated for events

including but not limited to low balance warnings and plan expiry warnings. Each Totogi customer configures the Amazon EventBridge Event Bus to receive notifications on. Customers then configure event bus rules to trigger different behaviors in response to event types and other event characteristics.

### 3.Lifecycle of a Charging Request

3GPP defines a set of standard message passing interfaces for wireless networks and a subset of those interfaces relate to charging. The call flows vary slightly by release (4G vs 5G) and by interface (Ro, Gy, N40, etc) but it is useful to understand the general pattern by reviewing the 5G call flow. For the nitty gritty, see 3GPP spec [TS 32.255](#).

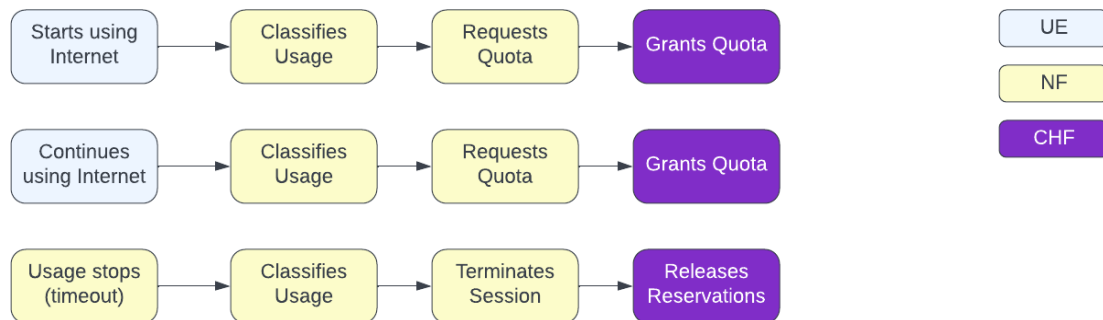


Figure 2: Charging Requests

- First, the subscriber uses a network function. Those functions could include
  - ◆ Sending an SMS message
  - ◆ Receiving an SMS message
  - ◆ Initiating a voice call
  - ◆ Uploading or downloading internet data
- The NF classifies usage into a rating group (RG).
  - ◆ A rating group is an integer that represents a category of network usage.
  - ◆ Similar to how an accounting system has a chart of accounts, the charging network has a hierarchy of rating groups.
    - 100 could represent voice and 110 could represent “HD” voice.
    - 300 could represent data and 310 could represent “WhatsApp Data”.

- The NF sends a Charging Data Request [Initial] message to the CHF. This message includes, among other things
  - ◆ Subscriber identifier (e.g. Device ID)
  - ◆ Rating Group
  - ◆ Subscriber location
  - ◆ Number and type of units requested
  - ◆ Number and type of units used
- The CHF evaluates the request and determines how much quota to grant. It replies with a Charging Data Response message back to the NF. This message includes:
  - ◆ Amount of quota granted (granted units)
  - ◆ Amount of time the quota grant is valid for (quota validity time or QVT)
- Based on the response, the NF can compute when it needs to go back to the CHF to reauthorize the session and request more quota. It will do this when
  - ◆ The quota validity time expires
  - ◆ The UE consumes quota units granted
- When the NF determines it needs more quota or that the QVT has expired, it sends a Charging Data Request [Update] message to the CHF similar in structure to the Initial request. It repeats this process throughout the subscriber's session.
- When the session completes, the NF sends the CHF a Charging Data Request [Terminate] message which informs the CHF to close out the session, release reserved balances and update the ending balance to reflect the units used. The [Terminate] message structure is similar to the [Update].

## 4. Evaluating Charging Data Requests

Now that we all understand the purpose of a charging system and how it fits into the landscape of a communication service provider's enterprise architecture, we can deep dive into **how** Totogi utilizes the power of AWS to satisfy real-time requests for quota servicing billions of devices mapped to thousands of unique rate plans from hundreds of CSPs across the globe.



In this section, we'll peel back the onion and explore how the Charging System handles those Charging Data Request messages.

## Data Structures

Totogi utilizes several data stores in the charging system:

- (a) **Neptune is the plan database.** Its schemaless graph architecture provides the flexibility to model rate plans as a directed acyclic graph of rules, each attached to a collection of configuration parameters.
- (b) **DynamoDB is the runtime workhorse.** Its key/value, serverless architecture enables it to respond to requests consistently under 5ms, regardless of the number of objects stored or concurrent requests in-flight.
- (c) **S3 provides long-term storage of EDRs** in [ORC format](#). EDRs are generated in large quantities - at least one per network charging request - and need to be retained for months or years. S3 enables access through Athena, Redshift Spectrum, or the customer's choice of BI or other software. S3's retention, immutability and access policies can also be used to enforce regulatory compliance.

**Logically**, the data is organized into this structure:

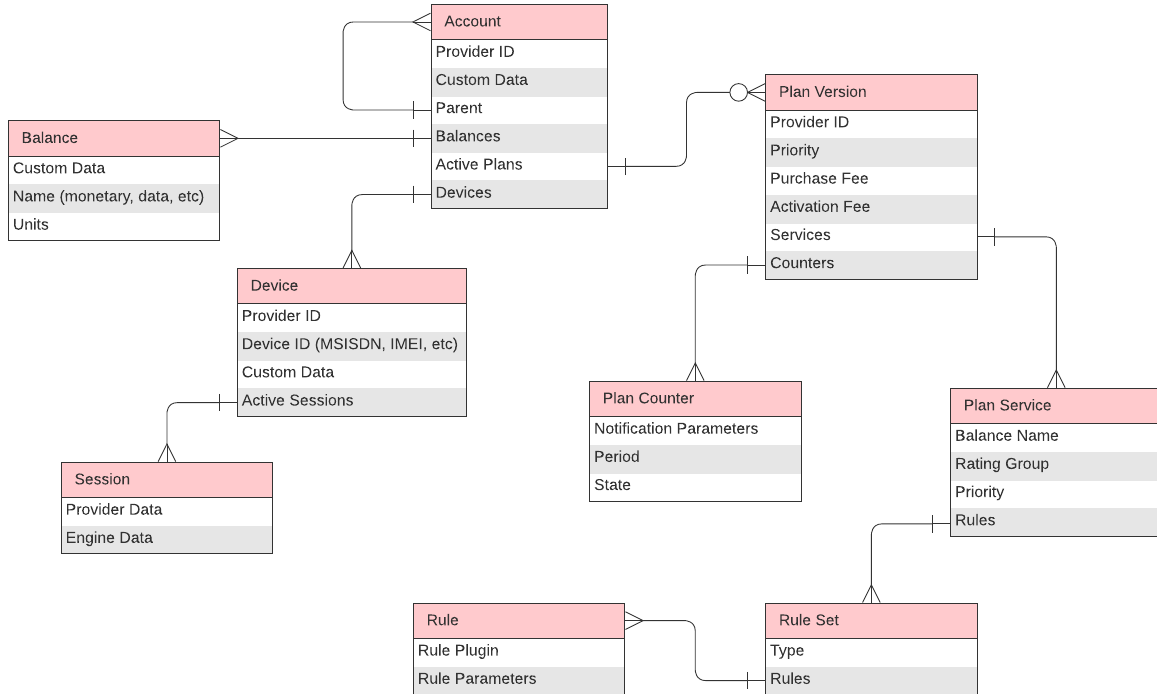


Figure 3: Data Structures

## Accounts

Accounts reflect subscribers in Totogi. They all carry a monetary balance which is debited when subscribing to plans. When subscribed to a plan, balances are created for each plan service.

Accounts may be organized hierarchically to support shared plans for enterprise, family and other use cases.

## Devices

Subscribers access the CSPs network using a device and the ID of the device is presented with each credit control request. Devices provide a link from the network to the account, providing access to the balances and rating rules to apply to each request. Devices also link to session data, as there are things like quota reservations to persist temporarily as subscribers use the system.

## Plans

Plans encapsulate a set of business rules that determine how a subscriber is charged for services. The rule sets are versioned so the entity linked to an account is a **Plan Version**. Plans have a set of services, with examples including off net voice, on-net text, generic data and WhatsApp data.

## Extensibility

Most nodes in the database include a “Custom Data” attribute that enables customers and plugins to augment the stock data structures with additional properties.

## SaaS Architecture

Totogi makes use of a small but powerful collection of services largely exclusive to AWS to deliver a revolutionary charging system that scales automatically as Totogi adds customers and customers add subscribers. The architecture is resilient, with all components distributed across availability zones and fully managed by the hyperscaler to ensure security, durability and performance (more on those topics later in the paper).

There are four layers to Totogi’s architecture:

- **Access Layer** is the bridge between a CSPs system and Totogi. Network functions connect via Diameter or HTTP/2 and BSS/OSS systems connect via GraphQL Northbound (NB) APIs. All connections are encrypted with TLS and secured by Cognito.
- **Decision Engine** is the flexible and scalable compute layer where charging rules are evaluated and balance and session state transitions are calculated. Network inputs are supplemented with device, account and plan data from DynamoDB. EDRs, session and balance updates are transactionally committed to DynamoDB before responses are returned.
- AWS fully **managed storage** is utilized in a plan database (Neptune) and a real-time subscriber database (DynamoDB) and is one of the essential enablers of Totogi’s ability to offer a charging system at a pay-per-use price

point that is 50–80% lower than other charging solutions. We’ll dive deeper into the rationale behind the database choices in [The Optimal Database for Charging](#) section.

- The **Client Services** layer is where customers link their own S3 data lake and EventBridge enterprise service bus to Totogi. This seamless handoff enables clients to configure event handlers and EDR processing simply on their own cloud stack.



Figure 4: AWS Service Architecture

## Charging Request Handling

An auto scaling group of EC2 instances run Totogi’s proprietary decision engine software. Totogi decided **not** to use EKS (docker containers) because EC2 is simpler to manage and as a SaaS operator, simplicity is preferred to portability. An Application Load Balancer provides the endpoint that clients connect to and

distributes requests across engines in the group. Requests must be accompanied by a valid JWT token that contains the tenant ID as one of the claims.

Given the tenant ID in the JWT token and the `subscriberIdentifier` (device ID) in the message body, the engine can retrieve the Account and Session records from DynamoDB, including the current balance(s) and associated charging rules.

After service requests are rated given the plan configurations, the engine transactionally persists balance updates, session updates and EDRs to DynamoDB.

## Credit Control Request Handling - Diameter

A different auto scaling group of adapters handles 4G Diameter requests by converting them into 5G REST requests and propagating them to the 5G CHF endpoint. The adapter obtains a JWT on behalf of the client using either the client side SSL certificate or the originating VPC endpoint ID as the lookup key.

## 5. Configuring Rate Plans

Totogi has a flexible API for registering rate plans, but its real super power is enabling non-technical users in marketing roles to create tariffs and rate plans in its technical catalog without assistance from IT, in a manner of minutes.

The Plan Design UI has been designed with marketers in mind and tested on marketers to ensure it is simple to use and enables them to craft the types of plans they need to compete and innovate in their markets. The system performs checks to ensure that every configuration is intentional and that the plan is compliant with guidelines the telco establishes.

When rate plans are associated with subscribers, the APIs provide a mechanism for overriding attributes of the plan. For example, price, period and allowance can be overridden by parameters defined in the commercial catalog, enabling Totogi plans to be easily reused in different contexts and supporting environments with established enterprise product catalogs.

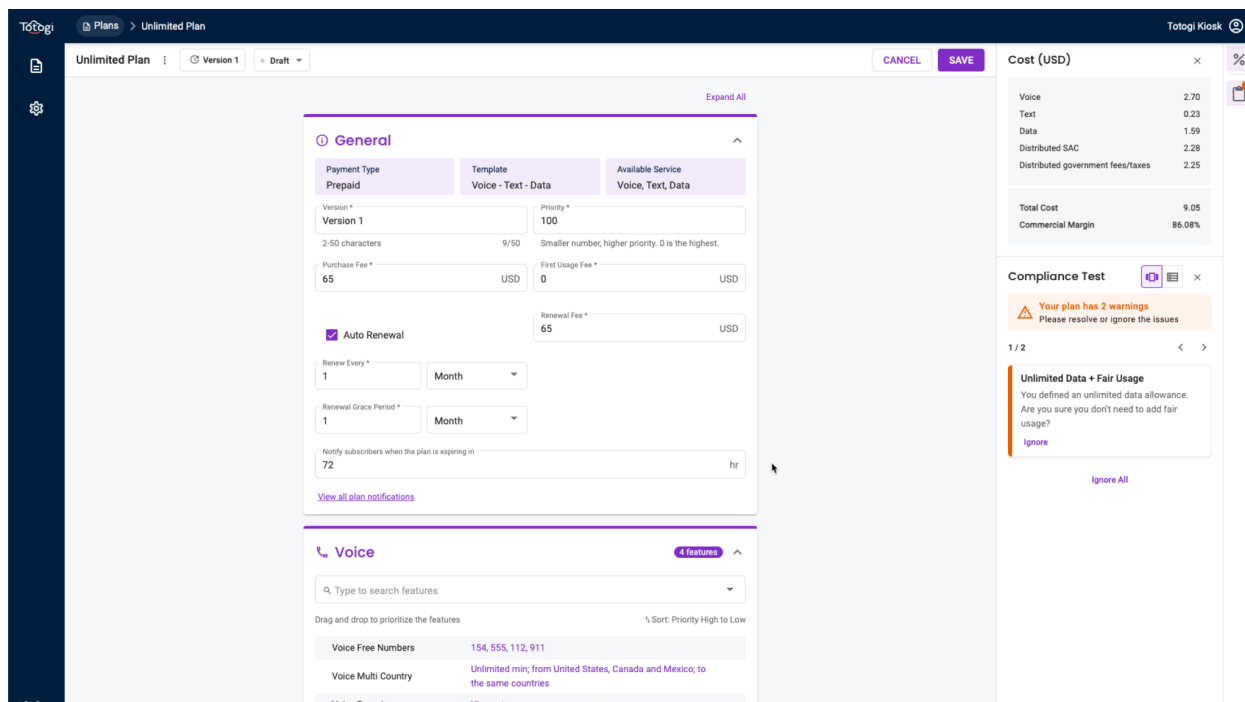


Figure 5: Plan Design makes it simple for Marketers to create and deploy plans

## 6. The Optimal Database for Charging

Why does Totogi select DynamoDB for its core database? Why not choose a relational database like Aurora instead? You'd still receive all the benefits of a managed system, and it would be easier to port to other hyperscalers or even run on-prem. Why choose a database that only works on AWS?

DynamoDB is a key-value database with a track record of success enabling high throughput transactional applications like Amazon.com, which processes north of 100m transactions per second on "Prime Day" annually. Unlike other key-value stores which only provide "eventually consistent" guarantees, Totogi utilized DynamoDB in a mode that provides strongly consistent reads, with writes synchronously recorded in two availability zones and asynchronously replicated to a third (typically within milliseconds). Reads and writes complete reliably in 5ms, regardless of the number of records in the table, which is important for a planet-scale charging system that needs to satisfy quota requests for billions of devices simultaneously.

The key-value pattern is ideally suited to Totogi's Charging-as-a-Service use case. The query patterns are well defined and narrow in scope. The prime examples are accessing session and account records by tenant-id + device-id. The single table design pattern both reduces the number of database calls required to satisfy each request and ensures that all data required for a transaction is co-located on the same partition, improving performance. Internally, DynamoDB is able to provide consistent response times and manage storage automatically using a [logical partitioning scheme](#). Using a relational store to achieve similarly consistent behavior requires extensive tuning and maintenance as well as pre-provisioned capacity.

DynamoDB is provided as a fully managed service. Unlike a relational database, there is no storage to attach, no partitioning schemes or table spaces to manage and no OS or DBMS software to patch. Totogi pays for usage and focuses on table design and domain modeling. There is some art that accompanies science when configuring auto-scaling policies for DynamoDB and other services. When you use them, you need to give the auto-scaling algorithm parameters to work with to ensure that scaling up and down happens without impact on your end-users. DynamoDB auto-scaling RCUs, for example, requires a minimum level, maximum level and utilization target.

API calls for charging systems typically reflect a cyclical pattern that is a function of the number of active subscribers. While the demand curve is generally smooth, it can experience step-wise changes during large migrations onto the platform or as a result of specific events such as system migrations, marketing promotions or significant localized events and emergencies. As telco operators, you plan for these traffic bursts by over-provisioning capacity well above high water marks, in-turn using much more energy than you need 95% of the time.

With a massively multi-tenant SaaS platform, Totogi is able to configure auto-scaling policies that have much higher utilization targets than you would be able to apply for single-tenant solutions because resources are distributed efficiently among tenants. With a diversified portfolio of operators using Totogi's service, the likelihood that there is a "spike" occurring somewhere at any given time is high, so traffic spikes are simply business-as-usual and the usage curve appears much smoother - well-matched for auto-scaling responsiveness.

## 7. Privacy: Data Isolation & Multi-Tenancy

### Isolation Strategy

Totogi is implemented using the [Pool Isolation Strategy](#) of the AWS Well-Architected Framework. The product utilizes pooled resources for compute and storage with separate storage partitions per tenant. Read and write access to a partition is governed by IAM permissions and Cognito Identities, ensuring that each API and end-user client can only operate on their own data.

### Compute Isolation

#### Context

At a high level, charging requests originate from a network function like a Packet Gateway, IMS AS (VoLTE) or SMS-C. The request messages carry information that identifies the authenticated device (e.g MSISDN), type of units used (e.g. data octets, voice minutes, etc) and the quantity of units used or requested. The charging function:

- 1) Looks up configuration data and balances related to the device
- 2) Computes the impact on balances
- 3) Computes an accept or reject decision
- 4) Persists the new balances and event detail records
- 5) Responds with a message containing the decision

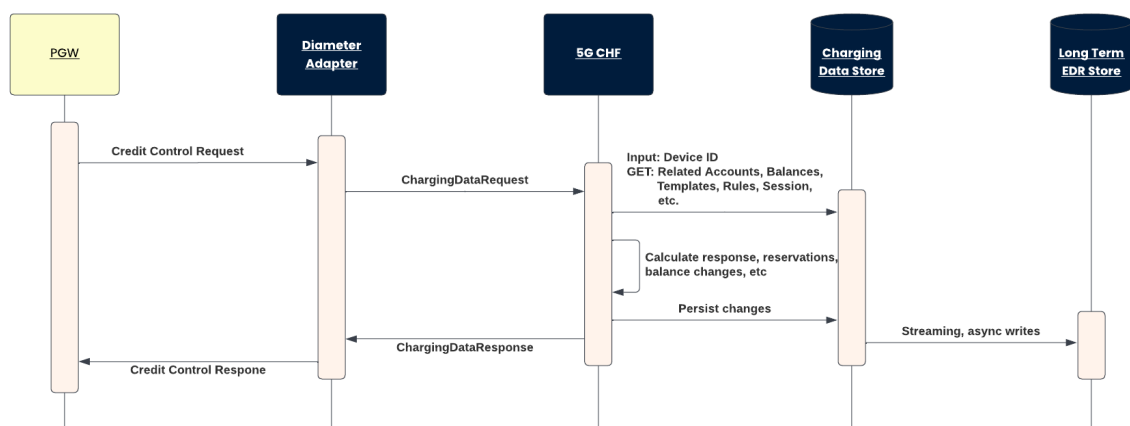


Figure 7: Simplified Charging Message Flow



## Tenant Authentication

For a 5G core, the network functions interface with the charging function using REST calls over HTTP/2. On 4G cores, the interfaces use the Diameter protocol over TLS/TCP. Totogi charger implements the 5G interface natively and includes an adapter that bidirectionally converts the Diameter protocol charging messages to REST.

On 5G, REST calls are authenticated using OAuth 2.0. Access tokens embedded in each request incorporate claims that identify the tenant account the network function belongs to. The tenant account is a component of the key used for each database read or write.

The Diameter adapter relies on mutual authentication in the TLS protocol. The adapter securely maps the CN of the client to a 5G identity when translating the request to the 5G NCHF calls. Client certificates must be signed by a trusted CA. Requests that fail to meet that criteria are dropped at the TLS layer.

## Stateless Compute Layer

All charging requests are load-balanced across an auto-scaling group of charging engines, responsible for handling charging data requests. There is no state in this layer. Each request uses inputs in the authentication headers and the request message to retrieve data required for handling the request from the database.

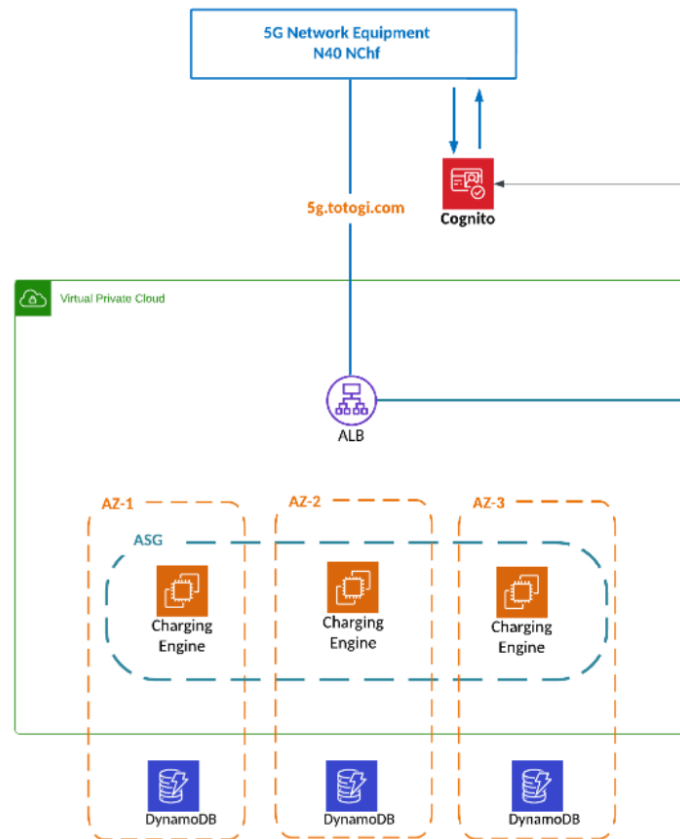


Figure 8: Auto-scaling Compute

## Storage Isolation

Totogi Charging-as-a-Service makes use of 3 data stores:

### Design Time Configuration Data

Rate plans in Totogi are created and maintained as graphs of objects in [Amazon Neptune](#), prior to being versioned and published to the runtime configuration data store in DynamoDB. Totogi incorporates the tenantID into every node ID in the graph and each graph begins with a root node identified by the tenant ID. Every graph query or update incorporates the tenant ID, ensuring that each customer's data is segregated.

## Runtime Configuration Data

Totogi uses [Amazon DynamoDB](#) for its runtime storage layer. DynamoDB is the Key-Value database originally created for Amazon's retail business. It scales because records are distributed across partitions identified by each record's partition key. To read or write data in DynamoDB, applications must include the partition key and are only able to operate on a single partition at a time.

Totogi incorporates the tenant ID into the partition key of each record, ensuring that every data access is scoped to a single tenant. Examples of configuration data persisted to this runtime store include:

- Balances
- Accounts
- Device → Account mappings
- Account → Plan mappings
- Plan charging rules

## Event Detail Records

Event Detail Records (EDRs) are initially stored in DynamoDB, using the same partitioning scheme as runtime configuration data. These records are then asynchronously streamed out to S3 where they can be downloaded or analyzed.

Files are organized in S3 such that IAM policies independently restrict access to each tenant's EDRs.

## **Mitigating Drawbacks of Pool Isolation**

### Noisy Neighbor

When resources are shared there is the potential for one tenant to impact the experience of another. For example, what happens when a CSP that uses Totogi acquires another CSP and adds 5 million new subscribers in one day? Can the increased usage affect the performance and stability of other Totogi customers?

Totogi's architecture contains the impacts of one customer's actions against others:

1. The runtime storage layer partitions are fully dedicated to individual tenants
2. The stateless compute layer dynamically adds capacity as demand increases

## 8. Resiliency

The OCS/CCS is a critical and core component of an operator's network and Totogi has been architected to provide carrier grade resiliency and reliability. Redundancy and failover are built into each layer of the system.

### High Availability

Totogi is deployed on AWS infrastructure in an AWS region. Within the region, Totogi achieves high availability by deploying services redundantly across two or more availability zones within a single VPC. Pairs of Diameter interfaces are provided to operators to enable signaling traffic to flow to the secondary Diameter interface if the primary is unresponsive.

HTTP/2 interfaces for Nchf are serviced by highly available AWS Application and Network load balancers, which use round-robin DNS to failover between load balancer instances and which front end pools of worker nodes spread across AZs. GraphQL APIs are provided by AppSync, an AWS service that provides resilience across multiple AZs.

Data is also stored redundantly, with DynamoDB, Neptune and S3 all providing fully redundant data storage as part of the service.

### Geo-Redundancy

Totogi provides a geo-redundancy option that uses AWS service capabilities to support disaster recovery scenarios where an AWS region becomes unavailable. DynamoDB Global Tables replicate state to the secondary region and S3's

Cross-Region-Replication feature is enabled to mirror the EDRs. The Decision Engine's stateless services run in both regions and DNS updates result in the NFs switching to the secondary region.

## Network Resiliency

Totogi customers ensure that they have resilient network routes to AWS. Those that use DirectConnect or VPN connections utilize AWS best-practices for HA connections with those services.

## 9.Security

Totogi has followed the guidance of the [security pillar of the AWS Well-Architected Framework](#) as a guide for developing its secure Charging System and adhering to the best practices and AWS services recommendations to achieve security excellence for the platform.

### Data Security

The Charging System reads and writes data on AWS S3, EBS, DynamoDB, Neptune, and CloudWatch services. All of these services encrypt data at rest leveraging cryptographic keys stored and managed in the AWS Key Management Service.

### Authentication and Access Control

Totogi leverages [Amazon Cognito](#) for sign-in and authentication. Users and machines authenticate with the Cognito service using open standard OAuth2 call flows and obtain bearer tokens that must be presented to each Totogi API call. Totogi's services check the authenticity and validity of the tokens with Cognito and use the grants encoded in the token to restrict actions to the Totogi account or tenant the token was issued for, along with the scope of entitlements the user has for that tenant. Human and machine users are classified into roles like Plan Designer, Plan Publisher and Network Operator, which are each mapped to a subset of functions they are able to perform.

## Audit

[AWS CloudTrail](#) provides a history of AWS API calls, allowing for identification of source IPs for attempted AWS services access and to verify that only authorized services and parties are accessing the systems. [CloudWatch](#) logs capture data access and changes. Between CloudTrail and CloudWatch logs, Totogi maintains comprehensive forensic data that are used in incident management processes.

## Connectivity

Totogi provides 2 options for IP connectivity from a CSP's core network:

- 1) TLS encrypted public internet links
- 2) Dedicated private internet addresses enabled by [AWS PrivateLink VPC endpoints](#) in customer VPCs.

All interfaces to the charging system are limited to protocols that provide encryption in transit. This includes TLS 1.3 for Diameter connections, 5G HTTP/2 connections and HTTPS GraphQL APIs. Customers that use VPC endpoints route to them using AWS standard networking services like DirectConnect and VPN.

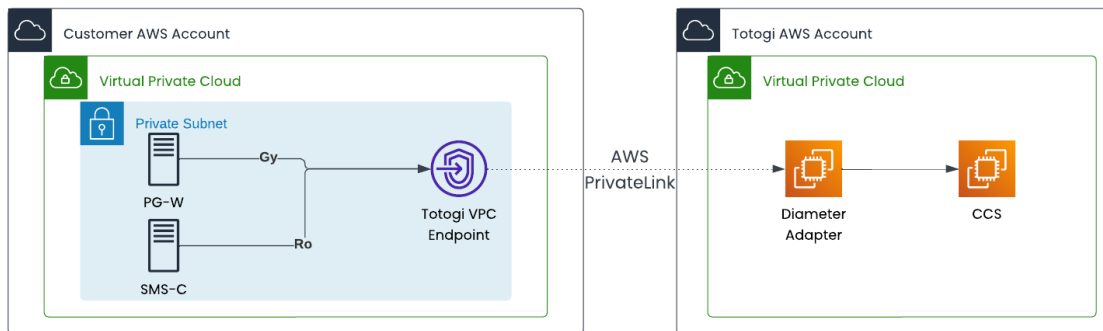


Figure 9: Accessing Totogi from AWS deployed Core Network

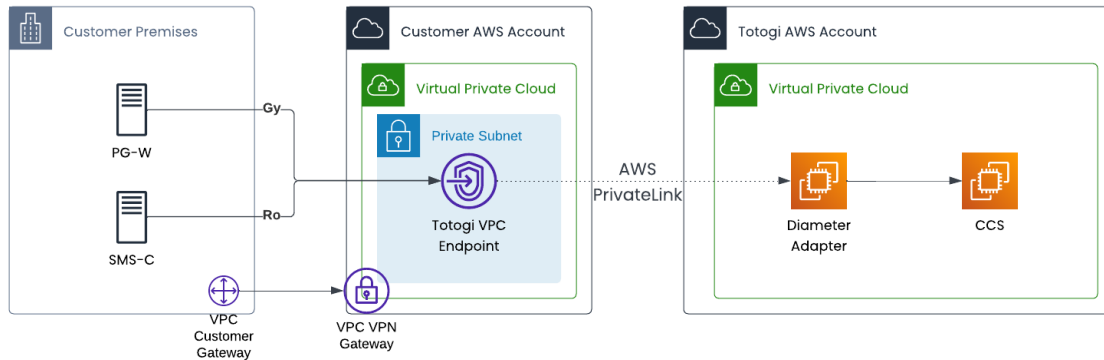


Figure 10 Accessing Totogi via VPN

For pilot and trial implementations, Totogi has found internet connectivity to be secure, effective and fast to setup. In addition to internet routing, customers have two other options for securely routing traffic to Totogi:

### AWS Direct Connect

Many Totogi customers already have [AWS Direct Connect](#) private leased line connections to AWS regions where Totogi Charging-as-a-Service runs.

### AWS VPN

Customers that do not have Direct Connect may opt to configure site-to-site VPN connectivity from their local network to their AWS VPC using [AWS VPN](#).

## Network Boundaries

Totogi uses Amazon VPC to establish private networks and control access to the Charging System using subnets, security groups that are stateful and Network Access Control Lists (NACL) that are stateless. This enables the isolation of Totogi Charging-as-a-Service from other Totogi applications and customer applications and network elements, ensuring only specific access is allowed.

## 10. Latency

After security, latency is the biggest concern most operators have when considering a cloud based charging system – especially if they will be peering it with an on-premise system. The fact is that no matter how quickly Totogi responds to credit control requests, unless the core network functions are deployed in the same AWS region as Totogi, there will be added latency required for network packets to travel to and from AWS.

Depending on which regional Totogi deployment you are connecting to and where the core network resides, that added latency can range from 3 milliseconds to 300 milliseconds. While there are often ways to reduce latency by using AWS Direct Connect or [AWS Global Accelerator](#), light travels at fixed speeds and there are physical minimum latencies that need to be acknowledged. In this section we discuss why this added latency is not actually a problem.

### Control Plane Latency and the Cloud

The **user plane** is the path to which data flows between the user's equipment, the packet gateway (or SMF) and the public internet. The **control plane** is the path for signaling traffic between the packet gateway and the control application (OCS). With the OCS on the Cloud, the control plane latency is what operators worry about.

A quota request from an on-prem P-GW to an on-prem OCS may have a total Round Trip Time of < 50ms (C1 + C2 + OCS Processing illustrated below).



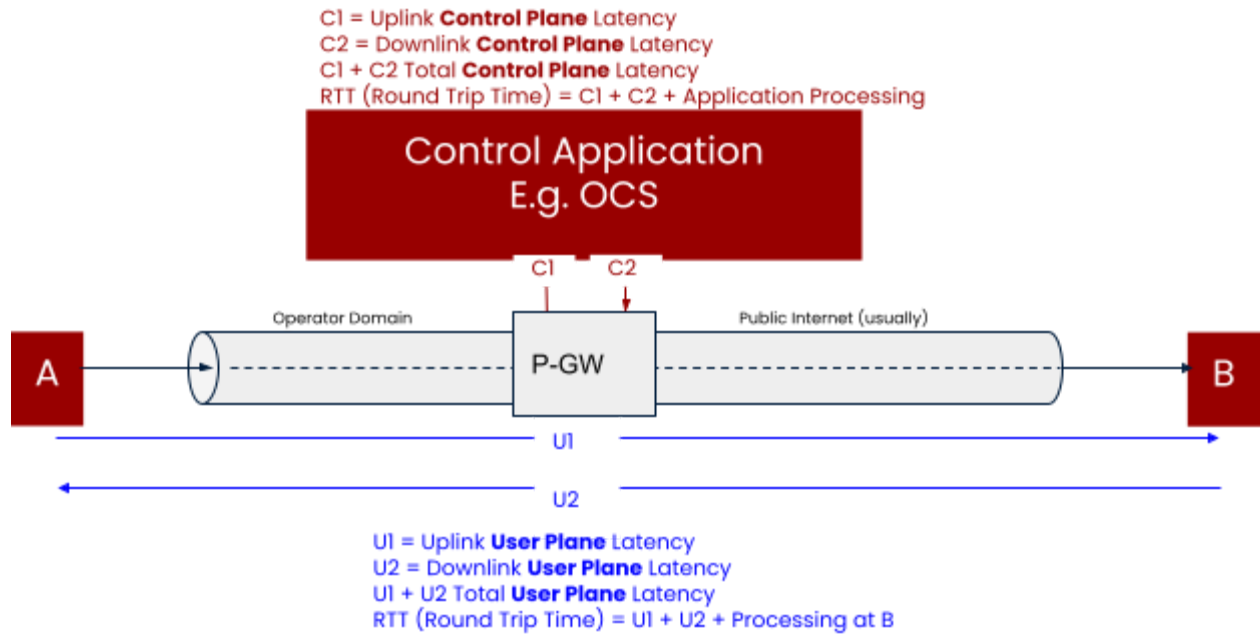


Figure 11: Control Plane Latency

On the cloud, the absolute minimum latency is based on physical distance and speed of light in a fiber (approximately 210 million meters per second). Geographical separation and latency are directly coupled.

### Latency and Round Trip (milliseconds)

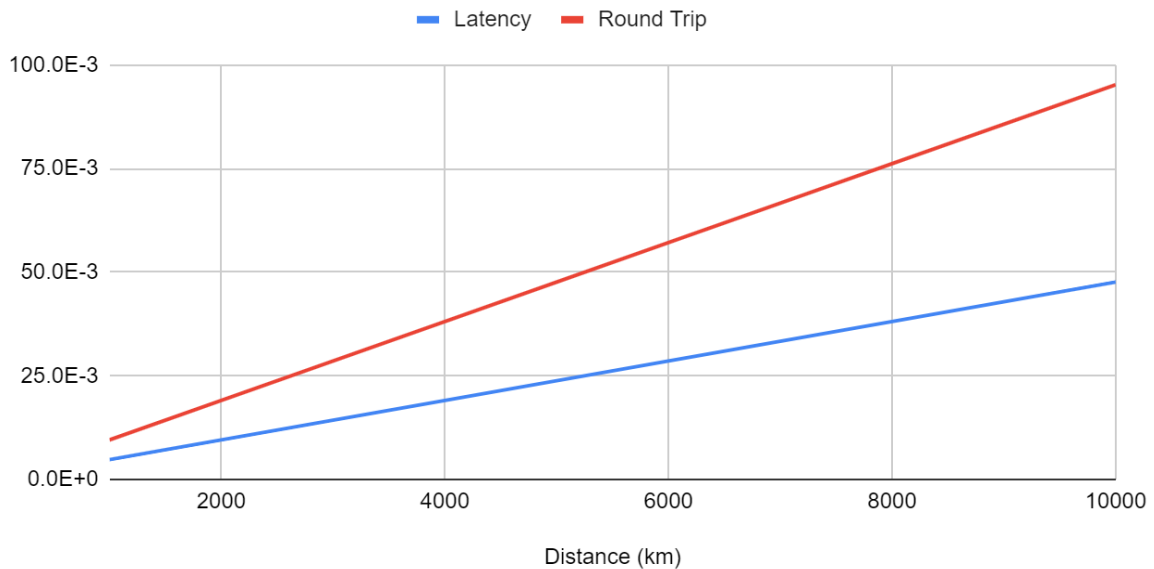


Figure 12: Latency vs Distance

## Control Plane Latency is **Not** necessarily a Problem

While it is simple to argue that lower latency is always better and that sub-50ms response times are important because that's what the legacy on-prem system provides, it is important to consider the entire picture. Remember, the OCS grants quota (megabytes) to the P-GW. When the user has consumed most of this quota, the P-GW requests more from the OCS. The low watermark that triggers the new request is generally configurable.

If the P-GW does not receive a response before all its remaining quota is used (due to Control Plane latency), what should it do? Allow service at risk of no quota or deny service?

### Options for managing control plane latency

**Option 1: "Good customer experience"** - allow usage to continue but the P-GW keeps count of usage in order to deduct it from the new quota when it finally arrives. But what if the OCS balance is fully depleted (no more quota can be provided, or at least less than the amount used while waiting for the response).

It is a **Revenue loss** - data was used that cannot be paid for. More latency means more revenue loss, but this only happens when the balance is low on the very final request-response. The chart below illustrates the extent of the revenue loss and when considered against the broader business and capability benefits of moving the OCS to the cloud, **the exposure should not be material.**

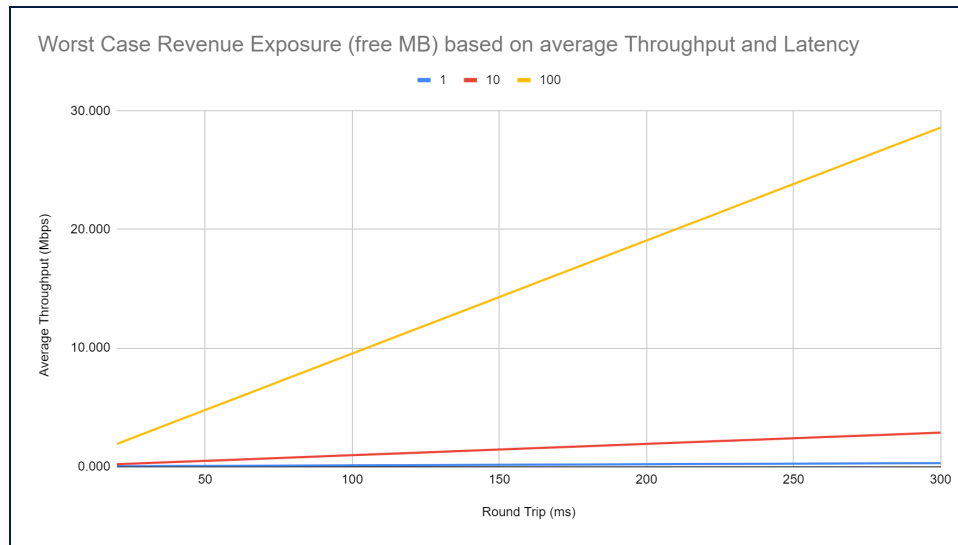


Figure 13: Latency vs Revenue Exposure

**Option 2: “Bad customer experience”** – deny usage until the OCS response is received. Some services like email and web-surfing will be resilient to a small delay and the user is unlikely to notice. On the contrary, higher value, delay sensitive services like streaming and other TCP protocols can be highly sensitive to even small user plane disruption.

### How do Totogi customers manage control plane latency?

The best way to manage the latency and minimize revenue loss is to simply configure the low-water-mark re-authorization level to be higher than the amount of data that will be consumed during the expected OCS latency period.

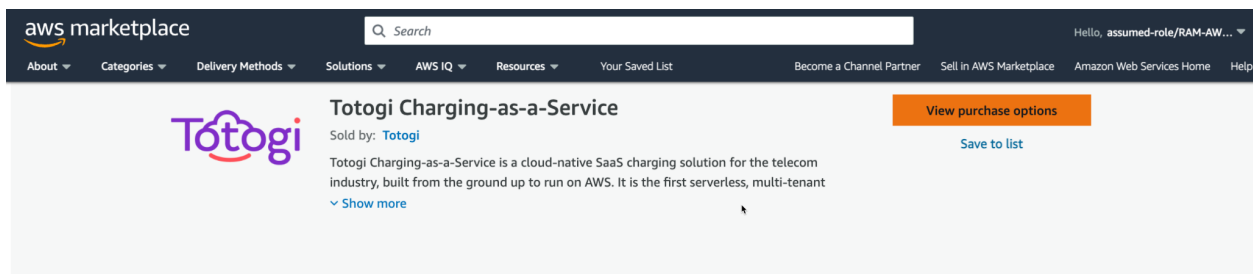
## 11. Conclusion

Totogi Charging-as-a-Service is a 3GPP OCS / CCS that is always-on and available to incorporate into carriers new or existing networks, providing real-time charging for pre-paid and post-paid service. As a true, cloud-native SaaS platform, Totogi Charging-as-a-Service unlocks pricing based on usage - with no hidden charges. There is no software or infrastructure to install - operators can simply subscribe to the service in the [AWS Marketplace](#) and configure their 4G and/or 5G SA network functions to use the Totogi charging endpoints. Totogi’s intuitive Plan Design interface allows non-technical marketing users to configure plans, gain competitive

insights and understand commercial margins in just a few clicks. Totogi Charging-as-a-Service APIs make it simple for developers to implement self-service and onboarding systems with ease.

By adopting Totogi Charging-as-a-Service, operators can reduce the time to market for new products and empower digital sub-brands and MVNOs with their own charging accounts, enabling them to move quickly and reducing the time and resources required to service them. Operators can safely try out Totogi on test networks or on small segments of their subscriber base because it is so simple to get started, while knowing that AWS has completed a [Foundational Technical Review](#) of the software and operational systems.

Are you ready to experiment with Totogi Charging-as-a-Service? You can subscribe in AWS Marketplace today. Just visit <https://aws.amazon.com/marketplace/> and search for Totogi.



The screenshot shows the AWS Marketplace interface. At the top, there is a search bar with the text "Search" and a navigation menu with items like "About", "Categories", "Delivery Methods", "Solutions", "AWS IQ", "Resources", "Your Saved List", "Become a Channel Partner", "Sell in AWS Marketplace", "Amazon Web Services Home", and "Help". The main content area displays the product listing for "Totogi Charging-as-a-Service". The listing includes the Totogi logo, the product name, and a description: "Totogi Charging-as-a-Service is a cloud-native SaaS charging solution for the telecom industry, built from the ground up to run on AWS. It is the first serverless, multi-tenant". There are two buttons: "View purchase options" (orange) and "Save to list" (blue). A "Show more" link is also present.

## 12. Appendix

### Contributors

Contributors to this document include:

- Marc Breslow, Field CTO, Totogi
- Amanveer Singh, Sr. Solutions Architect Telecom, Amazon Web Services
- Visu Sontam, SA Leader, Telecom OSS/BSS, Amazon Web Services

### Glossary

BSS - Business Support Systems

CCS - Converged Charging System (5G)

CHF - Charging Function (5G)

CSP - Communications Service Provider

IMS - IP Multimedia System

NF - Network Function

OCS - Online Charging System (4G)

OSS - Operational Support Systems

P-GW - Packet Gateway (4G)

QVT - Quota Validity Time

SMF - Session Management Function (5G)

SMSC - Short Message Service Center

UE - User Equipment