Overview

CPS Architecture Overview

The Cisco Policy Suite (CPS) solution utilizes a three-tier virtual architecture for scalability, system resilience, and robustness consisting of an I/O Management, Application, and Persistence layers.

The main architectural layout of CPS is split into two main parts:

- Operations, Administration and Management (OAM)
- Three-tier Processing Architecture

Operations, Administration and Management (OAM)

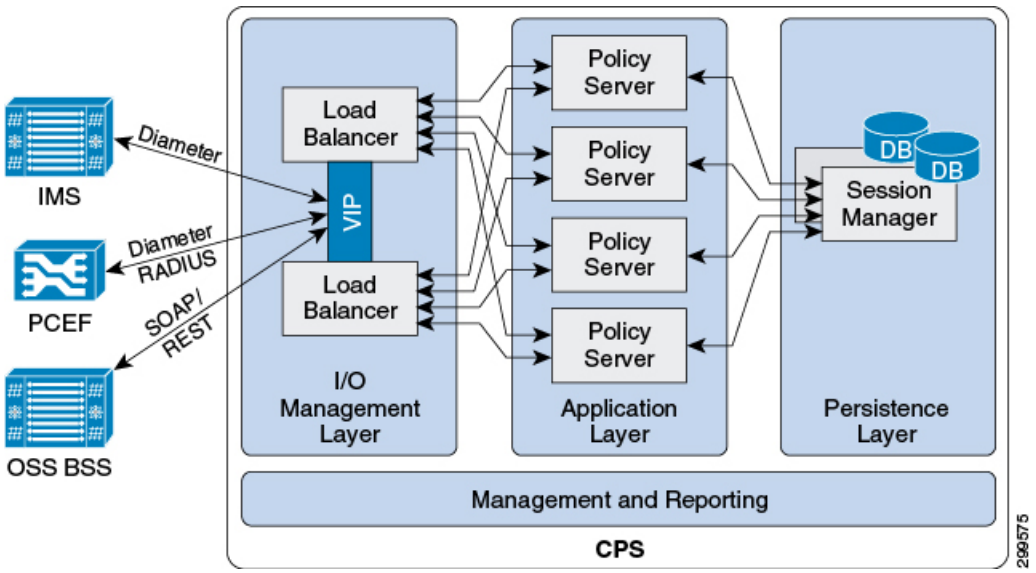
The OAM contains the CPS functions related to the initial configuration and administration of CPS.

- Operators use the Policy Builder GUI to define the initial network configuration and deploy customizations.
- Operators use the Control Center GUI or functionality that is provided through the Unified API to monitor day-to-day operations and manage the network. One of the primary management functions of Control Center is to manage subscribers. This aspect of Control Center is also referred to as the Unified Subscriber Manager (USuM).

Three-tier Processing Architecture

This section describes the three-tier architecture.

Figure 1: 3 Tier Architecture



The three-tier architecture defines how CPS handles network messages and gives CPS the ability to scale. The three processing tiers are:

- I/O Management Layer

  The I/O Management Layer handles I/O management and distribution within the platform. Load Balancer (LB) VMs, also referred to as Policy Director (PD) VMs implements I/O management layer functions. This layer supports internal load balancers to load balance requests to the relevant modules.

- Application Layer

  The Application Layer handles the transaction workload and does not maintain subscriber session state information. The main module of the Application Layer is a high performance rules engine.

- Persistence Layer

  The Persistence Layer consists of the Session Manager - a document-oriented database used to store session, subscriber information, and balance data (if and when applicable). Session Manager VMs implements this function.

  The databases that are included in Session Manager are:

  - Admin
  - Audit
  - Custom Reference Data
  - Policy Reporting
  - Sessions
  - Balance
  - SPR

For more information on Persistence Layer, refer to Persistence Layer, on page 3.

Persistence Layer

The Persistence Layer is responsible for storing session information, as well as subscriber profile and quota information if applicable. This is done using the Session Manager application. It is the persistence layer that maintains state within CPS. Geographic redundancy is achieved through data synchronization of the persistence layer between sites.

The Session Manager is built using MongoDB, a high-performance and high-availability document-oriented database.

The MongoDB obtains high performance by using a 'file-backed in-memory database'. To achieve high performance, the MongoDB stores as much of the data as possible in memory (and thus is very fast), but the data is mirrored and written out to disk to preserve the database information across restarts.

Access to the database is typically performed using the Unified API (SOAP/XML) interface. GUI access is typically limited to lab environments for testing and troubleshooting, and can be used to perform the following tasks:

- Manage subscriber data (if SPR used), that is, find, create or edit subscriber information
- Stop database or check the availability of subscriber sessions
- Review and manage subscriber sessions
- Populate custom reference data tables: Custom reference data tables allow service providers to create their own data structures that can be used during policy evaluation. Examples of information that can be included in custom reference data tables include:
  - Device parameters
  - Location data mapping (for example, map network sites and cell sites into the subscriber's home network)
  - Roaming network or preferred roaming network
  - IMEI data tagging for smart phone, Apple, android device, and so on

Unified Subscriber Manager (USuM)/Subscriber Profile Repository (SPR)

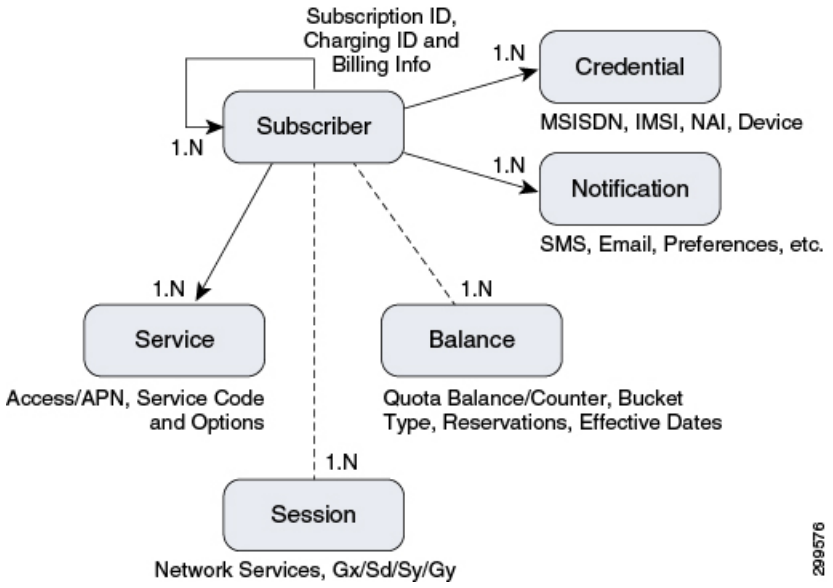
USuM manages subscriber data in a Subscriber Profile Repository (SPR). This includes the credentials with which a subscriber is able to log in and services allocated to the subscriber. The details of what a service means are stored in Policy Builder.

Each subscriber record that is stored in the USuM is a collection of the data that represents the real world end-subscriber to the system. Examples include which of the service provider's systems that the subscriber can access (mobile, broadband, and so on.) or to identify specific plans and service offerings that the subscriber can utilize.

Additionally, USuM can correlate balance and session data to a subscriber. Balance data is owned by Multi-Service Balance Manager (MsBM) and is correlated by the Charging Id. Session data is correlated by the credential on the session which should match an USuM credential. Session data is managed by CPS core and can be extended by components.

In 3GPP terminology, the USuM is a Subscriber Profile Repository (SPR). The following is a symbolic representation of how data portions of the Cisco SPR relate and depend on each other.

Figure 2: Subscriber Profile Repository Architecture



SPR primarily maintains the subscriber profile information such as Username, Password, Domain, devices configured, services (plans), and so on. SPR database is updated by provisioning process and queried at start of session.

Session Manager (SM)

The session manager database contains all the state information for a call flow. Components such as Diameter or custom components can add additional data to the session database without core code changes.

Multi-Service Balance Manager (MsBM)

MsBM is used to support any use cases that require balance, for example, volume monitoring over Gx also uses the Balance database (without need for Gy). It also handles the CPS implementation of an online charging server (OCS). It handles quota and manages the subscriber accounting balances for CPS. Quota information is stored separately from a subscriber so that it can be shared or combined among multiple subscribers.

MsBM defines the times, rates and balances (quota) which are used as part of CPS. In addition, it performs the following functions:

- Maintains the multiple balances that can be assigned for each subscriber. Balance types can include:
  - Recurring balances (for example, reset daily, monthly, or per billing cycle)
  - One-time balances such as an introductory offer might be assigned per subscriber
  - Both recurring and one time balances can be topped-up (credited) or debited as appropriate
- Balances can be shared among subscribers, as in family or corporate plans.
- Operators can set thresholds to ensure some pre-defined action is taken after a certain amount of quota is utilized. This can be used to prevent bill shock, set roaming caps, and to implement restrictions around family or corporate plans.
- Operators can configure policies to take action when a given threshold is breached. Examples include:

- Sending a notification to the subscriber
- Redirecting the subscriber to a portal or a URL
- Downgrading the subscriber's balance

Note The decision to utilize quota is made on a per service basis. Users who do not have quota-based services would not incur the overhead of querying/updating the MsBM database.

Geographic Redundancy

Overview

CPS can be deployed in a geographic redundant manner in order to provide service across a catastrophic failure, such as data center failure of a site hosting a Policy Suite cluster. In a GR deployment, two Policy Suite clusters are linked together for redundancy purposes with the clusters located either locally or remotely from each other in separate geographic sites.

Geo-redundancy is achieved through data synchronization between the two sites in a geographic redundant pair through a shared persistence layer. The specific subscriber profile, balance, session data replicated across sites is determined based on the deployment architecture, service requirements, and the network environment.

CPS supports active/standby redundancy in which data is replicated from the active to standby cluster. The active site provides services in normal operation. If the active site fails, the standby site becomes the primary and takes over operation of the cluster. In order to achieve a geographically distributed system, two active/standby pairs can be setup where each site is actively processing traffic and acting as backup for the remote site.

CPS also supports active/active deployment model in which data can be replicated across sites in both directions in order to achieve a geographically distributed system. If one system fails, entire traffic would failover to one site that can handle traffic of both sites simultaneously.

Concepts

The following HA/GR concepts and terms are useful in understanding a GR implementation of CPS:

Active/Standby

The Active site is one which is currently processing sessions. The Standby site is idle, waiting to begin processing sessions upon failure of one or more systems at the Active site.

Note Active/Standby and Primary/Secondary are used interchangeably in the context of Active/Standby GR solutions.

Failure

Failure refers to the failure of a given part in functioning. The part may be hardware, software, networking, or other infrastructure (power).

Failover

Failover refers to termination of the application/system at one site and the initiation of the same application/system at another site at the same level. Failovers can be manually triggered, where the system is brought down at the direction of an administrator and restored at a different site, or automatically triggered, in scenarios like, if the master database is not available at primary site without the direction of an administrator.

In both cases, failovers proceed through a set of predefined steps. Manual failover differs from manual intervention wherein depending upon the situation, faults, and so on, there are some additional steps that are executed to make a system Up or Down. Such steps might include patch installations, cleanup, and so on.

Failover Time

Failover Time refers to the duration needed to bring down the system, and start the system at the other site, until it starts performing its tasks. This usually refers to automatic failover.

Heartbeat

Heartbeat is a mechanism in which redundant systems monitor health of each other. If one system detects that the other system is not healthy/available, it can start failover process to start performing the tasks.

Split Brain

Split Brain situation arrives when the link between Primary and Secondary sites goes down, and due to unavailability of Heartbeat response, each site tries to become Primary. Depending upon technologies/solutions used for high availability, the behavior of each site might differ (both becoming Primary or both becoming Secondary and so on.) In general, this is an undesirable situation, and is typically avoided using solutions like Arbiter.

Cross-site Referencing

In a GR deployment, traffic is coming at one or both sites depending upon the nature of deployment. It is required that all the queries to the database be restricted to local sites. However, in case of certain failures, the servers on one site might query databases instances on another site. Since there is a time latency between the two sites, these queries are slow in nature and hence responses get delayed. This is cross-site referencing.

Arbiter

Arbiter is a lightweight 'observer' process that monitors the health of Primary and Secondary systems. Arbiter takes part in the election process of the Primary (active) system. It breaks any ties between systems during the voting process making sure that no split-brain occurs if there is a network partition (network partition is an example). To make sure that this process works smoothly, you can have odd number of participants in the system (for example, Primary, Secondary, and Arbiter).

Data Redundancy

There are two ways to achieve Data Redundancy.

- Data Replication

- Shared Data

Data Replication

In this mechanism, data is replicated between the Primary and Secondary sites so that it is always available to the systems. There are various factors that affect efficiency of replication.

- Bandwidth: Bandwidth is important when the amount of data that is replicated is large. With higher bandwidth, more data can be sent simultaneously. Also, if the data is compressed, it helps further better utilization of bandwidth.
- Latency: Latency is the time required to send a chunk of data from one system to another. The round-trip latency is an important factor that determines speed of replication. Lower latency equates to higher round-trip speed. Latency typically increases with the distance and number of hops.
- Encryption: During replication, the data might travel on public network, it is important to have encryption of data for protection. Encryption involves time and slows the replication. Data needs to be encrypted before it is transmitted for replication which takes additional time.
- Synchronous/Asynchronous: In an asynchronous write and asynchronous replication model, a write to local system is immediately replicated without first waiting for confirmation of the local write. With this form of replication there are chances of data loss if the replication could not take place due to some issue.

  This risk can be mitigated by the replication system through maintenance of operations log (oplog) which can be used to reconfirm replication. In the combination of asynchronous write and synchronous replication, oplog plays a vital role. The application is made efficient by responding fast to writes and data synchronization can also be ensured.

Shared Data

This is mostly applicable in case of local high availability where the data can be stored on an external shared disk which is not part of the system. This disk is connected to both the systems. In case a system goes down, the data is still available to redundant host. Such a system is difficult to achieve in case of Geographic Redundancy as write time to disk would be significant due to latency.

Operations Log

In the context of MongoDB, the operations log (oplog) is a special capped collection that keeps a rolling record of all the operations that modify the data stored in a database. Operations are applied to the primary database instance which then records the operation in the primary's oplog. The secondary members then copy and apply these operations in an asynchronous process, which allows them to maintain the current state of the database. Whether applied once or multiple times to the target data set, each operation in the oplog produces the same results.