

Azure Cloud Switch (ACS) Architecture

Table of Contents

[1 Abstract 4](#_Toc443918881)

[2 High Level Architecture 4](#_Toc443918882)

[3 Definitions 4](#_Toc443918883)

[3.1 Switch Hardware 4](#_Toc443918884)

[3.2 Platform Abstraction Layer (PAL) 4](#_Toc443918885)

[3.3 Switch Abstraction Interface (SAI) 5](#_Toc443918886)

[3.4 Switch State Service 5](#_Toc443918887)

[3.5 Network Application Stack 5](#_Toc443918888)

[4 Switch State Service (SwSS) 5](#_Toc443918889)

[4.1 Overview 5](#_Toc443918890)

[4.2 Redis database 6](#_Toc443918891)

[4.3 Network applications 6](#_Toc443918892)

[4.4 Orchestration Agent 6](#_Toc443918893)

[4.5 sswsyncd 6](#_Toc443918894)

[5 Redis Database Implementation 6](#_Toc443918895)

[5.1 Table Operations 6](#_Toc443918896)

[6 Switch Data Service Database Schema 7](#_Toc443918897)

[6.1 Overview 7](#_Toc443918898)

[6.2 Database 0 – APP\_DB 7](#_Toc443918899)

[6.2.1 ROUTE\_TABLE 7](#_Toc443918900)

[6.2.2 NEIGH\_TABLE 7](#_Toc443918901)

[6.2.3 PORT\_TABLE 8](#_Toc443918902)

[6.2.4 INTF\_TABLE 8](#_Toc443918903)

[6.2.5 VLAN\_TABLE: (later) 8](#_Toc443918904)

[6.2.6 LAG\_INTF\_TABLE: 8](#_Toc443918905)

[6.2.7 ACL\_TABLE (later) 8](#_Toc443918906)

[6.3 Database 7 – ASIC\_DB 8](#_Toc443918907)

[7 Switch state service Layer 3 Implementation 9](#_Toc443918908)

[7.1 Overview 9](#_Toc443918909)

[8 Appendix 9](#_Toc443918910)

[9 References 9](#_Toc443918911)

# Abstract

Azure Cloud Switch (ACS) is a collection of software packages that are installed on Linux running on a network hardware switch which make it a complete, functional router in the Azure physical data center network.

# High Level Architecture

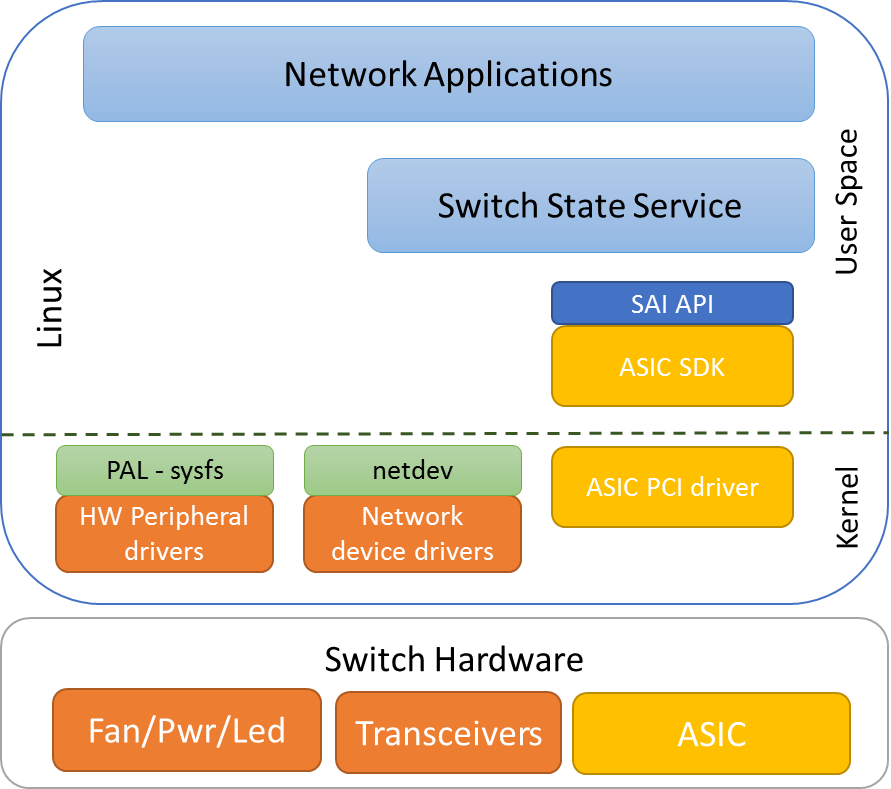


Figure 1 - ACS High Level Software Design

# Definitions

## Switch Hardware

Switch hardware refers to all the physical components inside the network switch enclosure (chassis). This includes fans, power supplies, status LEDs and network transceivers.

## Platform Abstraction Layer (PAL)

The platform abstraction layer is an interface to network switch hardware peripherals such as transceivers, fans, power supplies and leds. ACS exposes these details via the Linux sysfs interace. More detail will be provided in a future version of this document.

## Switch Abstraction Interface (SAI)

The switch abstraction interface is a standardized C API to the switching ASIC. This API is normally implemented by an SDK specific to the Switch ASIC. More information on SAI is available at the [SAI GitHub](https://github.com/opencomputeproject/SAI) repository.

## Switch State Service

Switch state service (SwSS) is a collection of software components that provide an API and database for storing and synchronizing network switch data with the Switch ASIC. This includes initialization, configuration and current status of the switch ASIC.

## Network Application Stack

Network applications use the switch state service to get and set the state of the Switch State Service.

# Switch State Service (SwSS)

## Overview

The Switch State Service (SwSS) is a collection of software that provides a database interface for communication with and state representation of network applications and network switch hardware.

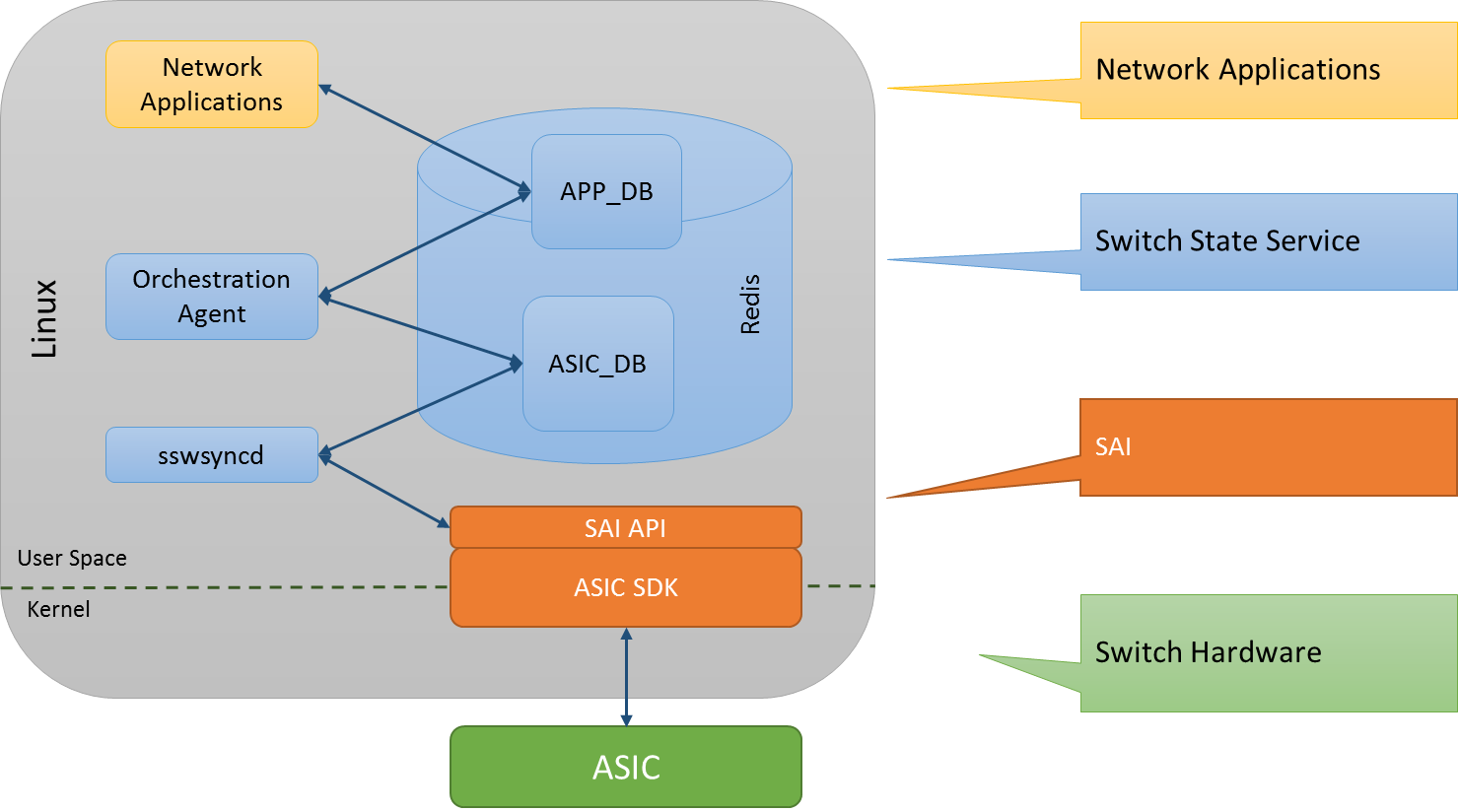


Figure 3 – Switch State Service High Level Design

## Key value database

A key value database was chosen to provide a language independent interface, a method for data persistence, replication and multi-process communication with less complexity for locking access to shared data. Redis was chosen as the underlying database engine. An API wrapper is implemented in swss/common which implements transactions, convienence methods and allows the database storage engine to be changed in the future if necessary.

## Network applications

Network applications may be written which are independent of lower layer communication details to the hardware. Applications may subscribe only to the data views they want and avoid implementation details that are not specific to their scenario. Examples of applications that may want to interface with SwSS include: Layer 3 routing, Layer 2 bridging, Access control lists (packet filtering), Quality of service, Telemetry streaming, tunneling, link aggregation, load balancing and policy based routing to name a few.

## Orchestration Agent

This process contains logic for transforming data from the APP tables to the ASIC table. There must only be one producer for each ASIC table. Currently there is just one orchestration agent.

## sswsyncd

The switch sync daemon sswsyncd copies data between the ASIC\_DB tables and a SAI compliant ASIC SDK.

Network applications read and write to APP\_DB.

The Sswsyncd process reads and writes switch ASIC data to SAI and ASIC\_DB.

Only a single Orchestration Agent may write to an ASIC\_DB table. Today, SwSS only implements one Orchestration Agent.

# Database Implementation

SwSS implements the concept of a table in redis by naming keys with prefixes. A producer / consumer design is implemented to ensure integrity of data.

APP\_ tables are designed for each use case. For example, ROUTE\_TABLE and NEIGH\_TABLE.

ASIC\_ tables are created from the SAI header files. For example ASIC\_sai\_unicast\_route\_entry\_t and ASIC\_sai\_neighbor\_entry\_t.

## Table Operations

[TODO: link to API’s for sswcommon table operations]

Producer

SET – insert or update a key -> fields and values.

DEL – deletes a key

Consumer

POP – get a table change notification, the key name and the key->fields and values and operation [SET, DEL].

SELECT – check if a table notification exists.

[TODO: add examples]

## Transactions

For each ‘TABLE’, there are keys used for internal implementation of notifications. See the code for more details.

Tablename+”\_KEY\_QUEUE”

Tablename+”\_VALUE\_QUEUE”

Tablename+”\_OP\_QUEUE”

# Switch Data Service Database Schema

## Overview

Two databases are defined, APP and ASIC. Applications outside of SwSS are expected to store data by adding keys with well-defined names into the APP database. The ASIC database stores data used by hardware sync agents. Keys in the ASIC database are expected named strictly following SAI attributes.

Keys must be prefixed with a string that looks like a table name. The allowed keys are “[a-z][A-Z][0-9]\_” and end with “\_TABLE:”

In redis, databases are only defined my numbers:

Database 0 = APP\_DB

Database 7 = ASIC\_DB

## Database 0 – APP\_DB

Description: This is a database containing application data. Data is organized into tables. Table schema is defined in ABNF form according to [RFC 5234](http://www.ietf.org/rfc/rfc5234.txt).

### ROUTE\_TABLE

;Stores a list of routes

;Status: Mandatory

Key = ROUTE\_TABLE:IP\_PREFIX

Nexthop = (String list) ;IP addresses separated “,” (empty strings indicate no gateway)

Ifindex = (String list) ;ifindex for route separated with “,” (0 indicated no ifindex)

Blackhole = Boolean ;Set to 1 if this route is a blackhole (or null0)

### NEIGH\_TABLE

; Stores the neighbors or next hop IP address and output port or

; interface for routes

; Note: neighbor\_sync process will resolve mac addr for neighbors

; using libnl to get neighbor table

;Status: Mandatory

ip\_address PORT\_TABLE.name / VLAN\_INTF\_TABLE.name / LAG\_INTF\_TABLE.name = macaddress ; (may be empty)

### PORT\_TABLE

;Data loaded from configuration file by ????

;Status: Mandatory

Key = name

admin\_status = Boolean

oper\_status = B**oolean**

lanes = list of lanes (need spec???)

### INTF\_TABLE

;defines logical network interfaces, an attachement point and ip address

;Status: mandatory

Key = name

attach\_to = PORT\_TABLE.name / VLAN\_INTF\_TABLE.name / LAG\_INTF\_TABLE.name

v4prefix = IPv4prefix

IPv4prefix = 1\*3DIGIT "." 1\*3DIGIT "." 1\*3DIGIT "." 1\*3DIGIT “/” %d1-32

### VLAN\_TABLE

;Defines VLANs and the interfaces which are members of the vlan

;Status: work in progress

Key = name

vland\_id = 0-4095

admin\_status = Boolean

oper\_status = boolean

attach\_to = nPORT\_TABLE.portname

PORT\_TABLE.portname = %s”PORT\_TABLE.”64\*ALPHA

### LAG\_INTF\_TABLE:

;a logical, link aggregation interface (802.3ad) made of one or more ports

;Status: work in progress

Key = name

ports = nPORT\_TABLE.portname

admin\_status = Boolean

oper\_status = boolean

minimum\_links = 1\*2DIGIT

### ACL\_TABLE (later)

## Database 7 – ASIC\_DB

The ASIC database stores data used by hardware sync agents. Keys in the ASIC database are named strictly following SAI attributes.

# Switch state service Layer 3 Implementation

## Overview

This section describes a high level overview of how a BGP route is learned by Quagga and propgated to the ASIC.

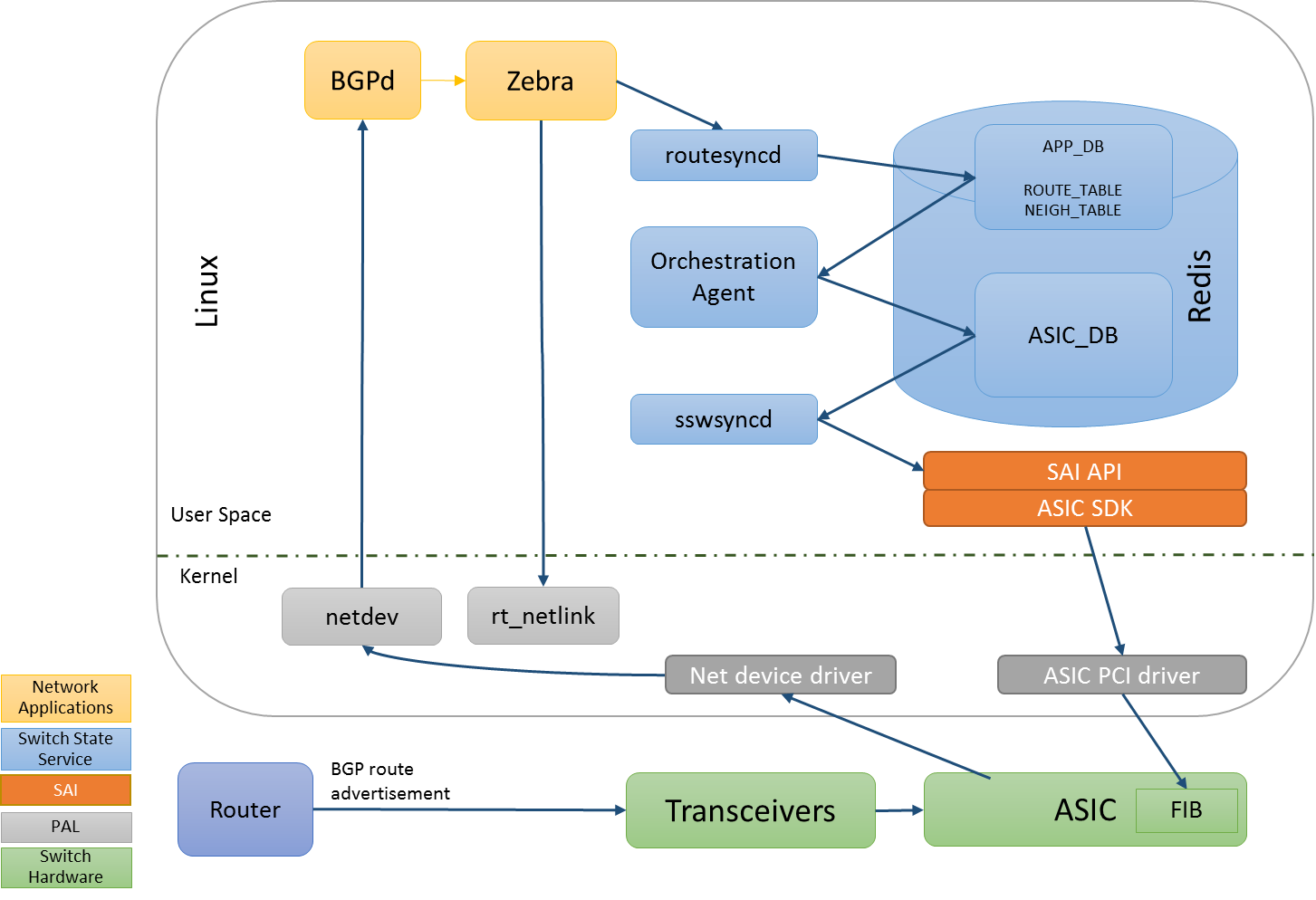


Figure 4 – Learn bgp route

# Appendix

# References