SONiC and its Architecture

By Architecture

SONiC

XFLOW RESEARCH

What is SONiC?

- Software for Open Networking in the Cloud
- Developed by microsoft and actively contributes to open source community
- It is compatible with an extensive range of protocols.
- An open-source service mesh designed to simplify microservices management within modern cloud-native architectures.

Why is SONiC important?

1. **Modularity & Decoupling**
   - Sonic's modular architecture enables independent development and scaling of microservices.
   - Decoupling microservices reduces dependencies, enhancing maintainability and resilience.

2. **Agility & Innovation**
   - Allows developers to focus on microservices without infrastructure complexities.

3. **Future-Proofing**
   - Open-source nature and community support ensure long-term sustainability and innovation.

Key Characteristics of SONiC

- **Lightweight**

  Sonic offers a lightweight solution for service mesh, minimizing overhead and complexity.

- **High Performance**

  It ensures minimal impact on application performance while providing essential features.

- **Extensibility**

  Its modular architecture allows for easy customization and integration with existing systems and tools.

Core Functionality

- **Observability**

  Provides comprehensive insights into microservices behavior through monitoring and tracing.

- **Traffic Management**

  Enables dynamic routing, load balancing, and traffic shaping for efficient communication between services.

- **Security**

  Enhances security with features like mutual TLS encryption for secure communication between services.

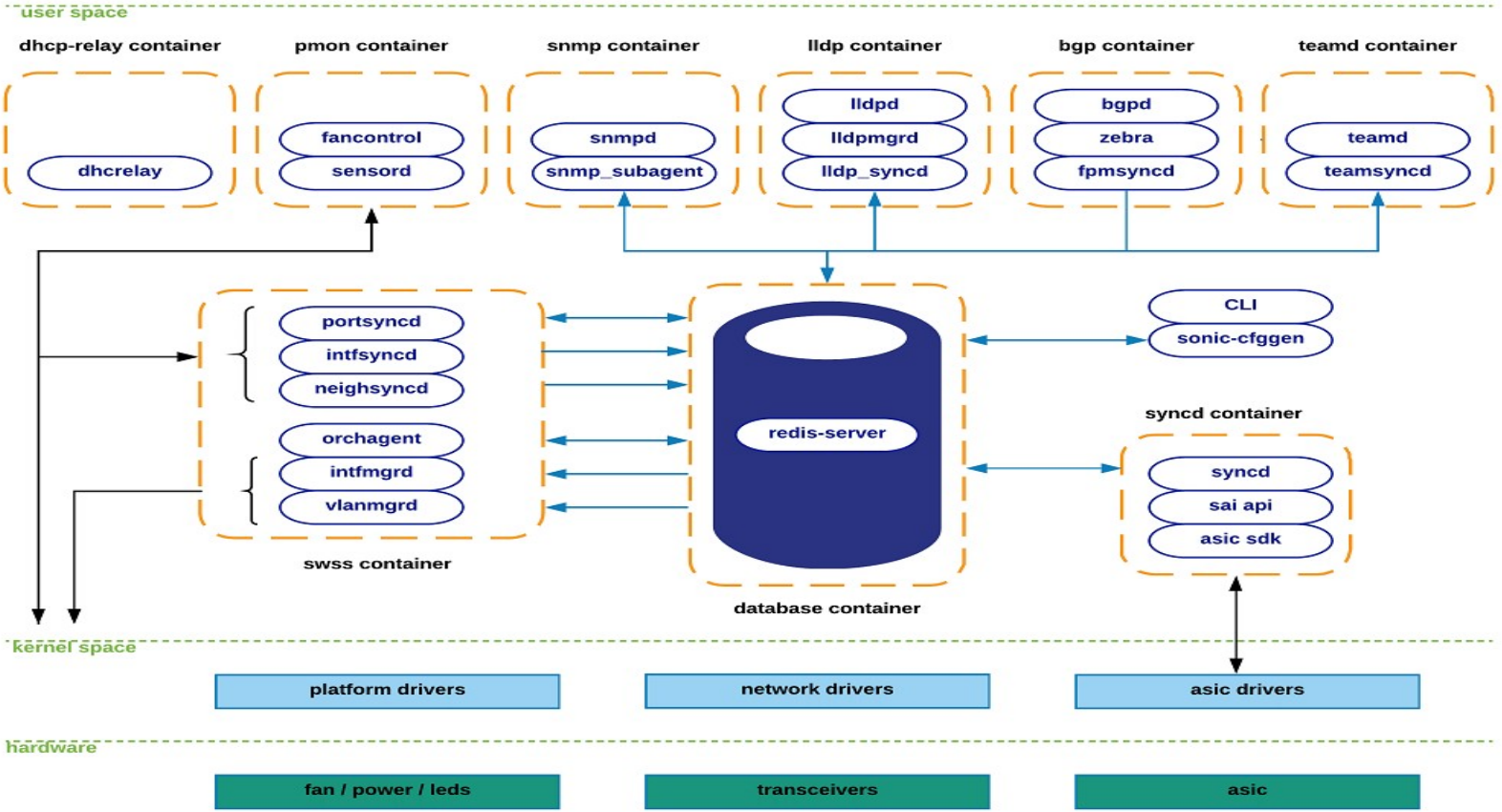
Benefits

- **Simplified Management:** Streamlines the deployment, configuration, and management of microservices-based applications.

- **Enhanced Reliability:** Improves application reliability by providing robust traffic management and fault tolerance mechanisms.

- **Security:** Strengthens application security by enforcing encryption and access control policies

Architecture of SONiC


user space
dhcp-relay container
pmon container
snmp container
lldp container
bgp container
teamd container
dhcrelay
fancontrol
sensord
snmpd
snmp_subagent
lldpd
lldpmgrd
lldp_syncd
bgpd
zebra
fpmsyncd
teamd
teamsyncd
portsyncd
intfsyncd
neighsyncd
orchagent
intfmgrd
vlanmgrd
swss container
redis-server
database container
CLI
sonic-cfggen
syncd container
syncd
sai api
asic sdk
kernel space
platform drivers
network drivers
asic drivers
hardware
fan / power / leds
transceivers
asic

Sonic architecture follows a modular design, facilitating scalability, flexibility, and extensibility in network deployments.

Components:

- **ASIC Abstraction:** Abstracts hardware-specific functions for consistent behavior across different ASICs.
- **Switch Abstraction Interface (SAI):** Provides a standardized API for interacting with ASICs, enabling seamless integration of hardware platforms.
- **Containerized Services:** Modular services such as DHCP Relay, Routing Protocols, and Quality of Service (QoS) run within containers for isolation and scalability.

ASIC Abstraction

- ASIC (Application-Specific Integrated Circuit) abstraction in Sonic refers to the process of decoupling hardware-specific functionality from the network operating system (NOS).

- It enables Sonic to achieve hardware independence and support a wide range of switch platforms.

Benefits

- **Hardware Independence:** Allows Sonic to run on various switch platforms with different ASICs, promoting vendor neutrality and flexibility.
- **Simplified Development:** Developers can focus on building network features without needing to worry about hardware intricacies.
- **Scalability:** Supports a diverse range of ASICs, enabling Sonic to scale from small to large network deployments seamlessly.

Use Cases

- **Multi-Vendor Environments:** Enables deployment of Sonic across switches from different vendors without modification.
- **Upgradability:** Facilitates easy upgrades by abstracting hardware-specific details, reducing the impact of hardware changes on software development.
- **Future-Proofing:** Ensures Sonic remains compatible with new ASICs and switch platforms as technology evolves.

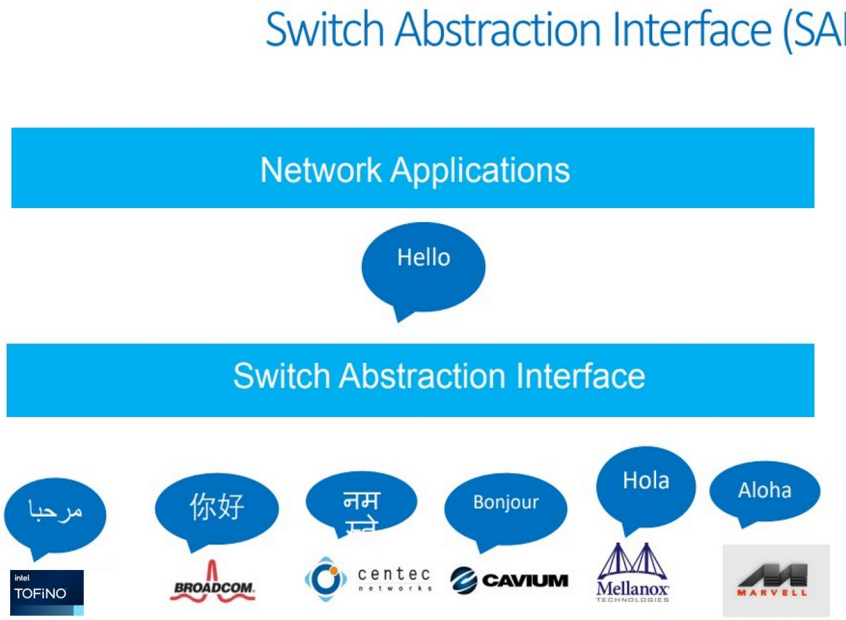
Switch Abstraction Interface(SAI)

SAI acts as a translator between the ASICs(hardware) and the applications

- **Key Components**
  - **Vendor Adaptation:** Vendors translate SAI calls into ASIC-specific commands, ensuring compatibility across platforms.
  - **Plug-and-Play:** Facilitates easy integration with diverse ASICs without modifying the network operating system.

Problem solved by SAI

- Time-consuming
- Non-scalable
- Vendor lock-in
- Costly



Simple, consistent, and stable network application stack

Helps consume the underlying complex, heterogeneous hardware easily and faster

XFLOW RESEARCH

Containerization in SONiC

What is Containerization

- Containerization is a lightweight virtualization method that encapsulates applications and their dependencies into containers.

- Each container runs as an isolated process, providing consistency across different environments.

Benefits of Containerization in SONiC

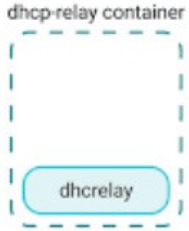
- Improved Deployment Efficiency
- Enhanced Scalability
- Resource Isolation
- Simplified Management
- Portability

Containers at Application Layer

- **DHCP- Relay** (Dynamic Host Configuration Protocol) Container
- **Pmon** (Platform Monitoring) Container
- **SNMP** (Simple Network Management Protocol) Container
- **LLDP** (Link Layer Discovery Protocol) Container
- **BGP** (Border Gateway Protocol) Container
- **Teamd** Container

DHCP-Relay Container

It provides a way for DHCP clients to communicate with DHCP servers when none are available on its local subnet.



Benefits

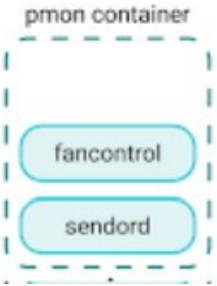
- Simplified Deployment: Easy setup and management within Sonic.
- Integration: Seamlessly integrates with Sonic's management framework for centralized control.

Use Cases

- Multi-VLAN Deployments: Enables dynamic IP assignment across VLANs.
- Data Center Networks: Facilitates efficient IP address assignment in large-scale environments.

Pmon Container

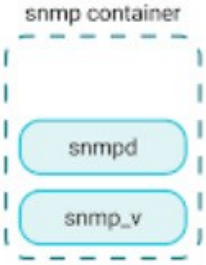
- manages the platform's processes, monitoring their health and ensuring smooth operation, facilitating fault detection and recovery in network environments.
- **Sensord** daemon logs sensor readings at regular intervals from hardware components and triggers alerts when alarms are signaled.
- **Fancontrol:** The process that gathers temperature statistics from platform sensors and adjusts fan speed accordingly, increasing or decreasing it as needed.



SNMP Container

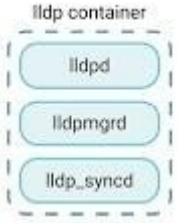
Snmpd: Actual SNMP (master) is responsible for managing incoming SNMP polls from external network elements.

Snmp-agent: This is SONiC's implementation of an AgentX ,SNMP subagent, which gathers information from SONiC databases in the centralized Redis engine and provides it to the master agent (snmpd).



LLDP Container

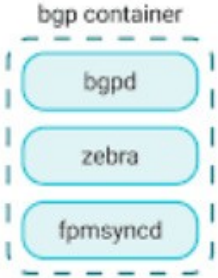
- **Lldpd:** An active LLDP daemon that initiates LLDP connections with external peers to exchange and share system capabilities.
- **Lldp_syncd:** It transfers the discovered state of LLDP to the centralized system's message infrastructure (Redis engine). Through this action, the LLDP state will be forwarded to applications keen on utilizing this data (e.g., SNMP)
- **Lldpmgr:** enhances the lldpd's functionality by enabling incremental configuration.



BGP Container

- **Bgpd:** A standard BGP implementation where routing updates from external parties are received via regular TCP/UDP sockets and then forwarded to the forwarding plane via zebra or fpmsyncd

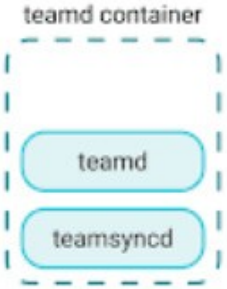
- **Zebra:** Zebra manages IP routing updates and redistributions, distributing the Forwarding Information Base (FIB) to the kernel and south bound components via FPM interfaces.



- **Fpmsyncd:** A compact daemon tasked with gathering the FIB state generated by Zebra and storing it in the Application-DB table (APPL_DB) housed within the Redis engine.

Teamd Container

- **teamd:** Implements LAG(Link Aggregation) Protocol
- **teamsyncd:** Transfers the current state into APPL_DB

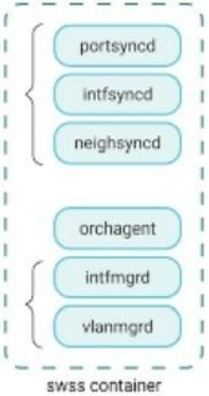


SouthBound Containers

- **SWSS**(Switch State Services) Container
- **Central Database** Container (Redis Engine)
- **Syncd** Container

SWSS(Switch State Services) Container

- Interacts with Redis- Database
- Components:
- **Portsyncd:** Portsyncd monitors port-related netlink events, transferring data to APPL_DB including attributes like port speeds and lanes. Additionally, it injects state information into STATE_DB.
- **Intfsyncd:** Monitors interface-related netlink events and pushes collected state to APPL_DB, handling attributes like new or changed IP addresses associated with an interface.
- **neighsyncd:** Listens to neighbor-related netlink events from ARP processing to gather data for the adjacency-table, essential for L2-rewrite in the data-plane.



- **Orchagent:** Processes and messages relevant state from syncd daemons and forwards it through its south-bound interface.

- **intfMgrd:** Utilizes data from APPL_DB, CONFIG_DB, and STATE_DB to configure Linux kernel interfaces.

- **vlanMgrd:** Configures Linux kernel VLAN interfaces based on state from APPL_DB, CONFIG_DB, and STATE_DB.

Database Container (Redis Engine)

- Manages **Redis-Engine** databases for housing SONiC network device state data.
- Utilizes a **Key-Value** format for storing information.
- Offers fast and effective access for all SONiC components to retrieve, modify, and exchange data.

Databases:

APPL_DB

ASIC_DB

STATE_DB

CONFIG_DB

COUNTERS_DB

- APPL_DB: Stores application-specific data in a database.
- ASIC_DB: Manages data related to Application-Specific Integrated Circuits (ASICs).
- STATE_DB: Tracks the current state of various system components or entities.
- CONFIG_DB: Stores configuration settings and parameters for the system.
- COUNTERS_DB: Keeps track of various counters or metrics within the system.

Syncd Container

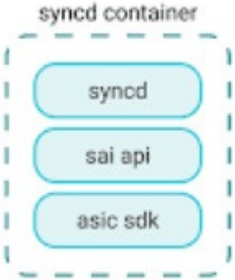
- **Goal:** Synchronize switch's network state with hardware/ASIC.

  Main Logical Components:

- **Syncd**

  Executes synchronization logic.

  Links with vendor's ASIC SDK library.



- **SAI API**

  Provides vendor-independent control of forwarding elements.

- **ASIC SDK**

  Implementation by hardware vendors for ASIC control.

Thankyou!
Any suggestions and feedback is most welcome