SONiC User Guide (v0.3)



Table of Contents

[1. Introduction 3](#_Toc512935559)

[2. Document History 3](#_Toc512935560)

[3. Referenced Documents 3](#_Toc512935561)

[4. SONiC System Architecture 4](#_Toc512935562)

[4.1 SONiC Subsystems Description 5](#_Toc512935563)

[4.2 SONiC Subsystems Interactions 10](#_Toc512935564)

[5. System Configuration 19](#_Toc512935565)

[5.1 System Configuration Overview 19](#_Toc512935566)

[5.2 System Configuration Files 19](#_Toc512935567)

[5.3 System Auxiliary Configuration Files 20](#_Toc512935568)

[5.4 System Configuration Workflow -- Provisioning 21](#_Toc512935569)

[5.5 System Configuration Workflow – Verification 23](#_Toc512935570)

[5.6 System Port-Mapping 24](#_Toc512935571)

[5.7 Port-Breakout Configuration 25](#_Toc512935572)

[5.8 Device Metadata Configuration 27](#_Toc512935573)

[5.9 Ports Configuration 28](#_Toc512935574)

[5.10 VLAN Configuration 28](#_Toc512935575)

[5.11 L3 Interfaces Configuration 29](#_Toc512935576)

[5.12 Loopback Interface Configuration 30](#_Toc512935577)

[5.13 Management Interface Configuration 31](#_Toc512935578)

[5.14 ACL Configuration 32](#_Toc512935579)

[5.15 Routing Configuration 37](#_Toc512935580)

[6. System Operation 39](#_Toc512935581)

[6.1 CLI show commands 39](#_Toc512935582)

[6.2 CLI debug / undebug commands 53](#_Toc512935583)

[6.3 CLI clear commands 54](#_Toc512935584)

[7. System Upgrade/Downgrade Process 54](#_Toc512935585)

[8. System Troubleshooting 55](#_Toc512935586)

[8.1 Logging Information (swssloglevel) 55](#_Toc512935587)

[8.2 Databases state (redis-cli) 55](#_Toc512935588)

[9. Annex 55](#_Toc512935589)

[9.1 Example config\_db.json file 55](#_Toc512935590)

# Introduction

The present document aims to serve as a user-guide for Linkedin’s network-operating system (LNOS).

Being LNOS a very thin wrapper over SONiC OS, over the course of this document we will make use of both LNOS and SONiC terms interchangeably, except when there is a need to emphasize areas in which these two diverge.

# Document History

| Revision | Date | Prepared By | Comments |
| --- | --- | --- | --- |
| 0.1 | 2018-02-20 | Rodny Molina | Defining doc skeleton and generating first draft. |
| 0.1.1 | 2018-02-21 | Rodny Molina | Addressing most of the reviewer’s comments. |
| 0.1.2 | 2018-02-28 | Rodny Molina | Adding port-mapping, port-breakout and misc. |
| 0.1.3 | 2018-03-07 | Rodny Molina  Zhenggen Xu | Adding auxiliary config files, VLAN and ACL config items. |
| 0.2 | 2018-03-26 | Rodny Molina | Adding operational commands. |
| 0.3 | 2018-04-11 | Rodny Molina | Adding system-arch description. |

# Referenced Documents

| # | Document Title | Document Identifier & Link |
| --- | --- | --- |
| 1 | SONiC official wiki | <https://github.com/Azure/SONiC/wiki> |
| 2 | SONiC architecture | <https://github.com/Azure/SONiC/wiki/Architecture> |
| 3 | SAI API | <https://github.com/opencomputeproject/SAI> |
| 4 | Redis documentation | <https://redis.io/documentation> |
| 5 | Click module | <http://click.pocoo.org/5/> |
| 6 | JSON introduction | <https://www.json.org/> |
| 7 | SONiC supported platforms | <https://github.com/Azure/SONiC/wiki/Supported-Devices-and-Platforms> |

# SONiC System Architecture

SONiC system’s architecture comprises of various modules that interact among each other through a centralized and scalable infrastructure. This infrastructure relies on the use of a redis-database engine: a key-value database to provide a language independent interface, a method for data persistence, replication and multi-process communication among all SONiC subsystems.

By relying on the publisher/subscriber messaging paradigm offered by the redis-engine infrastructure, applications can subscribe only to the data-views that they require, and avoid implementation details that are irrelevant to their functionality.

SONiC places each module in independent docker containers to keep high cohesion among semantically-affine components, while reducing coupling between disjointed ones. Each of these components are written to be entirely independent of the platform-specific details required to interact with lower-layer abstractions.

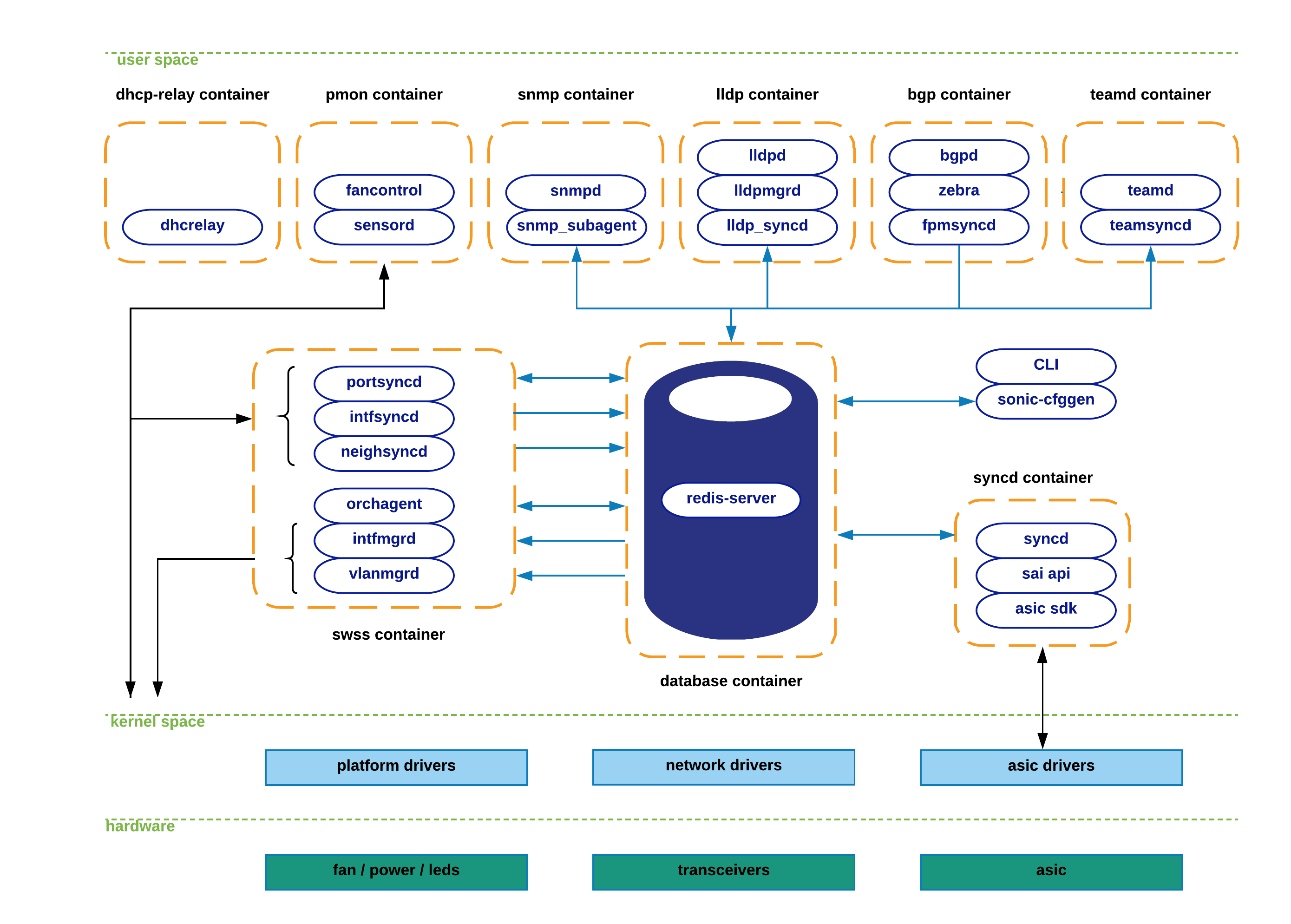
As of today, SONiC breaks its main functional components into the following docker containers:

* Dhcp-relay
* Pmon
* Snmp
* Lldp
* Bgp
* Teamd
* Database
* Swss
* Syncd

The following diagram displays a high-level view of the functionality enclosed within each docker-container, and how these containers interact among themselves. Notice that not all SONiC applications interact with other SONiC components, as some of these collect their state from external entities. We are making use of blue-arrows to represent the interactions with the centralized redis-engine, and black-arrows for all the others (netlink, /sys file-system, etc).

Even though most of SONiC’s main components are held within docker containers, there are some key modules seating within the linux-host system itself. That is the case of SONiC’s configuration module (sonic-cfggen) and SONiC’s CLI.

A more complete picture of all the possible component interactions and the associated state being transferred, will be covered in subsequent sections of this document.



## SONiC Subsystems Description

This section aims to provide a description of the functionality enclosed within each docker container, as well as key SONiC components that operate from the linux-host system. The goal here is to provide the reader with a high-level introduction; a more graphical and (hopefully) intuitive approach will be followed in subsequent sections.

**Teamd container**: Runs Link Aggregation functionality (LAG) in SONiC devices. “teamd” is a linux-based open-source implementation of LAG protocol. “teamsyncd” process allows the interaction between “teamd” and south-bound subsystems.

**Pmon container**: In charge of running “sensord”, a daemon used to periodically log sensor readings from hardware components and to alert when an alarm is signaled. Pmon container also hosts “fancontrol” process to collect fan-related state from the corresponding platform drivers.

**Snmp container**: Hosts snmp features. There are two relevant processes within this container:

* Snmpd: Actual snmp server in charge of handling incoming snmp polls from external network elements.
* Snmp-agent (sonic\_ax\_impl): This is SONiC’s implementation of an AgentX snmp subagent. This subagent feeds the master-agent (snmpd) with information collected from SONiC databases in the centralized redis-engine.

**Dhcp-relay container**: The dhcp-relay agent enables the relay of DHCP requests from a subnet with no DHCP server, to one or more DHCP servers on other subnets.

**Lldp container**: As its name implies, this container hosts lldp functionality. These are the relevant processes running in this container:

* Lldp: Actual lldp daemon featuring lldp functionality. This is the process establishing lldp connections with external peers to advertise/receive system capabilities.
* Lldp\_syncd: Process in charge of uploading lldp’s discovered state to the centralized system’s message infrastructure (redis-engine). By doing so, lldp state will be delivered to applications interested in consuming this information (e.g. snmp).
* Lldpmgr: Process provides incremental-configuration capabilities to lldp daemon; It does so by subscribing to STATE\_DB within the redis-engine. See further below for details in this topic.

**Bgp container**: Runs one of the supported routing-stacks: Quagga or FRR. In LNOS case, we have opted for FRR suite. Even though the container is named after the routing-protocol being used (bgp), in reality, these routing-stacks can run various other protocols (such as ospf, isis, ldp, etc).

BGP container functionalities are broken down as follows:

* bgpd: regular bgp implementation. Routing state from external parties is received through regular tcp/udp sockets, and pushed down to the forwarding-plane through the zebra/fpmsyncd interface.
* zebra: acts as a traditional IP routing-manager; that is, it provides kernel routing-table updates, interface-lookups and route-redistribution services across different protocols. Zebra also takes care of pushing the computed FIB down to both kernel (through netlink interface) and to south-bound components involved in the forwarding process (through Forwarding-Plane-Manager interface –FPM--).

* fpmsyncd: small daemon in charge of collecting the FIB state generated by zebra and dumping its content into the Application-DB table (APPL\_DB) seating within the redis-engine.

**Database container**: Hosts the redis-database engine. Databases held within this engine are accessible to SONiC applications through a UNIX socket exposed for this purpose by the redis-daemon. These are the main databases hosted by the redis engine:

* APPL\_DB: Stores the state generated by all application containers – routes, next-hops, neighbors, etc. This is the south-bound entry point for all applications wishing to interact with other SONiC subsystems.
* CONFIG\_DB: Stores the configuration state created by SONiC applications – port configurations, interfaces, vlans, etc.
* STATE\_DB: Stores “key” operational state for entities configured in the system. This state is used to resolve dependencies between different SONiC subsystems. For example, a LAG portchannel (defined by teamd submodule) can potentially refer to physical ports that may or may-not be present in the system. Another example would be the definition of a VLAN (through vlanmgrd component), which may reference port-members whose presence is undetermined in the system. In essence, this DB stores all the state that is deemed necessary to resolve cross-modular dependencies.
* ASIC\_DB: Stores the necessary state to drive asic’s configuration and operation – state here is kept in an asic-friendly format to ease the interaction between syncd (see details further below) and asic SDKs.
* COUNTERS\_DB: Stores counters/statistics associated to each port in the system. This state can be utilized to satisfy a CLI local request, or to feed a telemetry channel for remote consumption.

**Swss container**: The Switch State Service (SwSS) container comprises of a collection of tools to allow an effective communication among all SONiC modules. If the database container excel at providing storage capabilities, Swss mainly focuses on offering mechanisms to foster communication and arbitration between all the different parties.

Swss also hosts the processes in charge of the north-bound interaction with the SONiC application layer. The exception to this, as previously seen, is fpmsyncd, teamsyncd and lldp\_syncd processes which run within the context of the bgp, teamd and lldp containers respectively. Regardless of the context under which these processes operate (inside or outside the swss container), they all have the same goals: provide the means to allow connectivity between SONiC applications and SONiC’s centralized message infrastructure (redis-engine). These daemons are typically identified by the naming convention being utilized: \*syncd.

* Portsyncd: Listens to port-related netlink events. During boot-up, portsyncd obtains physical-port information by parsing system’s hardware-profile config files. In all these cases, portsyncd ends up pushing all the collected state into APPL\_DB. Attributes such as port-speeds, lanes and mtu are transferred through this channel. Portsyncd also inject state into STATE\_DB. Refer to next section for more details.

* Intfsyncd: Listens to interface-related netlink events and push collected state into APPL\_DB. Attributes such as new/changed ip-addresses associated to an interface are handled by this process.
* Neighsyncd: Listens to neighbor-related netlink events triggered by newly discovered neighbors as a result of ARP processing. Attributes such as the mac-address and neighbor’s address-family are handled by this daemon. This state will be eventually used to build the adjacency-table required in the data-plane for L2-rewrite purposes. Once again, all collected state ends up being transferred to APPL\_DB.
* Teamsyncd: Previously discussed – running within teamd docker container. As in the previous cases, obtained state is pushed into APPL\_DB.
* Fpmsyncd: Previously discussed -- running within bgp docker container. Again, collected state is injected into APPL\_DB.
* Lldp\_syncd: Also previously discussed – running within lldp docker container.

The above processes clearly act as state producers as they inject information into the publisher-subscriber pipeline represented by the redis-engine. But obviously, there must be another set of processes acting as subscribers willing to consume and redistribute all this incoming state. This is precisely the case of the following daemons:

* Orchagent: The most critical component in the Swss subsystem. Orchagent contains logic to extract all the relevant state injected by \*syncd daemons, process and massage this information accordingly, and finally push it towards its south-bound interface. This south-bound interface is yet again another database within the redis-engine (ASIC\_DB), so as we can see, Orchagent operates both as a consumer (for example for state coming from APPL\_DB), and also as a producer (for state being pushed into ASIC\_DB).
* IntfMgrd: Reacts to state arriving from APPL\_DB, CONFIG\_DB and STATE\_DB to configure interfaces in the linux kernel. This step is only accomplished if there is no conflicting or inconsistent state within any of the databases being monitored. Refer to the above database-container section for examples of this undesired behavior.
* VlanMgrd: Reacts to state arriving from APPL\_DB, CONFIG\_DB and STATE\_DB to configure vlan-interfaces in the linux kernel. As in IntfMgrd’s case, this step will be only attempted if there is no dependent state/conditions being unmet.

**Syncd container**: In a nutshell, syncd’s container goal is to provide a mechanism to allow the synchronization of the switch’s network state with the switch’s actual hardware/ASIC. This includes the initialization, the configuration and the collection of the switch’s ASIC current status.

These are the main logical components present in syncd container:

* Syncd: Process in charge of executing the synchronization logic mentioned above. At compilation time, syncd links with the ASIC SDK library provided by the hardware-vendor, and injects state to the ASICs by invoking the interfaces provided for such effect. Syncd subscribes to ASIC\_DB to receive state from SWSS actors, and at the same time registers as a publisher to push state coming from the hardware.
* SAI API: The Switch Abstraction Interface (SAI) defines the API to provide a vendor-independent way of controlling forwarding elements, such as a switching ASIC, an NPU or a software switch in a uniform manner. Refer to [3] for more details on SAI API.
* ASIC SDK: Hardware vendors are expected to provide a SAI-friendly implementation of the SDK required to drive their ASICs. This implementation is typically provided in the form of a dynamic-linked-library which hooks up to a driving process (syncd in this case) responsible of driving its execution.

**CLI / sonic-cfggen:** SONiC modules in charge of providing CLI functionality and system configuration capabilities.

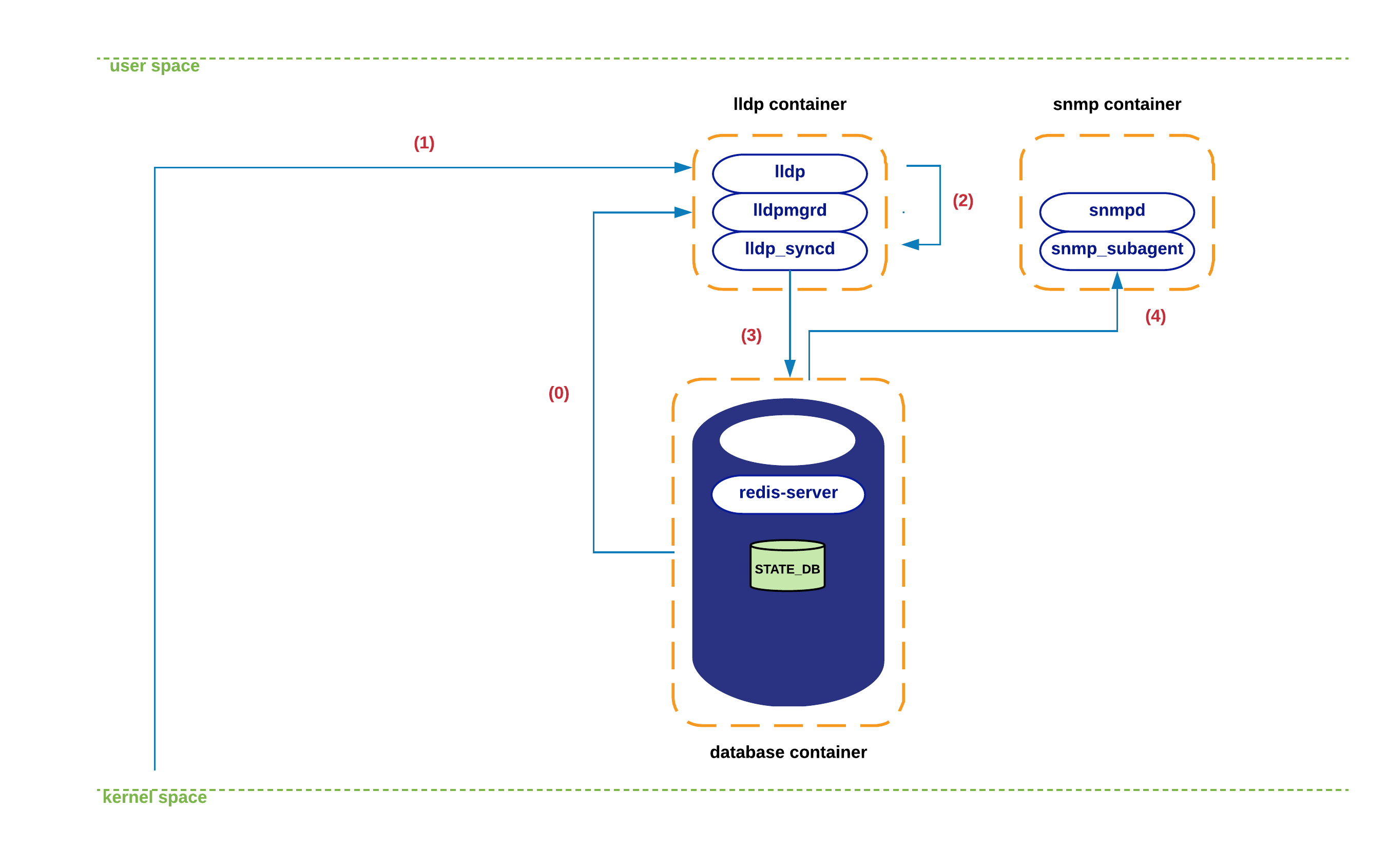
* CLI component heavily relies on Python’s Click library [5] to provide users with a flexible and customizable approach to build command line tools.
* Sonic-cfggen component is invoked by SONiC’s CLI to perform configuration changes or any action requiring config-related interactions with SONiC modules.

## SONiC Subsystems Interactions

This section aims to provide reader with a detailed understanding of the set of interactions that take place among the various SONiC components. To make information more digestible, we have bundled all the system interactions we can envision, attending to the particular state being exchanged by each major functionality.

### LLDP-state interactions.

The following diagram depicts the set of interactions observed during LLDP-state transfer episodes. In this particular example we are iterating through the sequence of steps that take place upon the arrival of an LLDP message carrying state changes.



1. During lldp container initialization, lldpmgrd subscribes to STATE\_DB to get a life-feed of the state of the physical ports in the system – lldpmgrd’s polling cycle runs every 5 seconds. Based on this information, Lldpd (and its network peers), will be kept aware of changes in the system’s port-state and any configuration change affecting its operation.

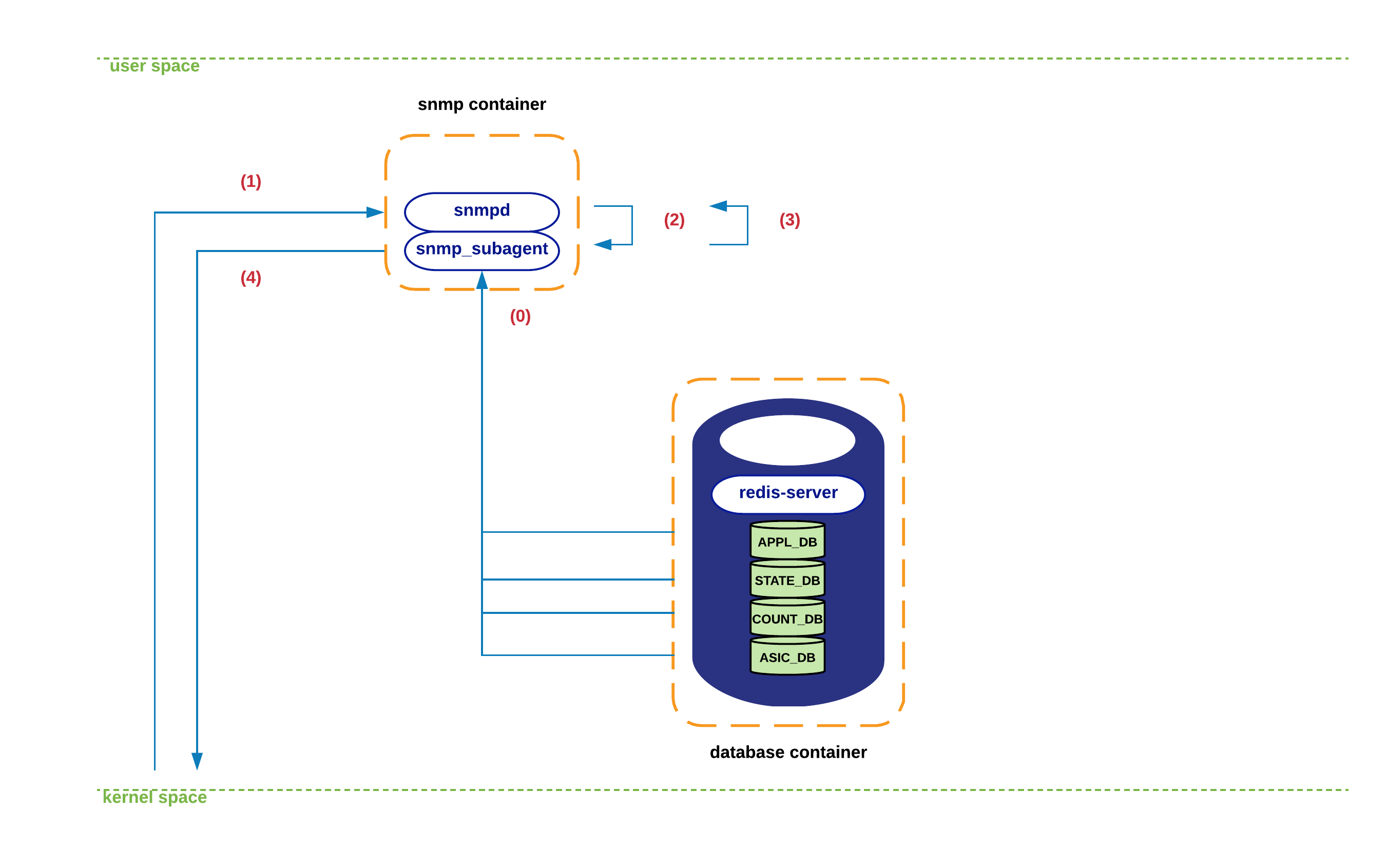
1. At certain point a new LLDP packet arrives at lldp’s socket in kernel space. Kernel’s network-stack eventually delivers the associated payload to lldp process.
2. Lldp parses and digests this new state, which is eventually picked up by lldp\_syncd during its execution of lldpctl cli command -- which typically runs every 10 seconds.
3. Lldp\_syncd pushes this new state into APPL\_DB, concretely to table LLDP\_ENTRY\_TABLE.
4. From this moment on, all entities subscribed to this table should receive a copy of the new state (currently, snmp is the only interested listener).

### SNMP-state interactions.

As previously mentioned, snmp container hosts both a snmp master-agent (snmpd) as well as a SONiC-specific agentX process (snmp\_subagent). This subagent interacts with all those redis databases/tables that provide information from which MIB state can be derived. Concretely, snmp-agent subscribes to the following databases/tables:

* APPL\_DB: PORT\_TABLE, LAG\_TABLE, LAG\_MEMBER\_TABLE, LLDP\_ENTRY\_TABLE
* STATE\_DB: \*
* COUNTERS\_DB: \*
* ASIC\_DB: ASIC\_STATE:SAI\_OBJECT\_TYPE\_FDB\*

The following diagram depicts a typical interaction among various SONiC components during the time an incoming snmp query is processed by the system.

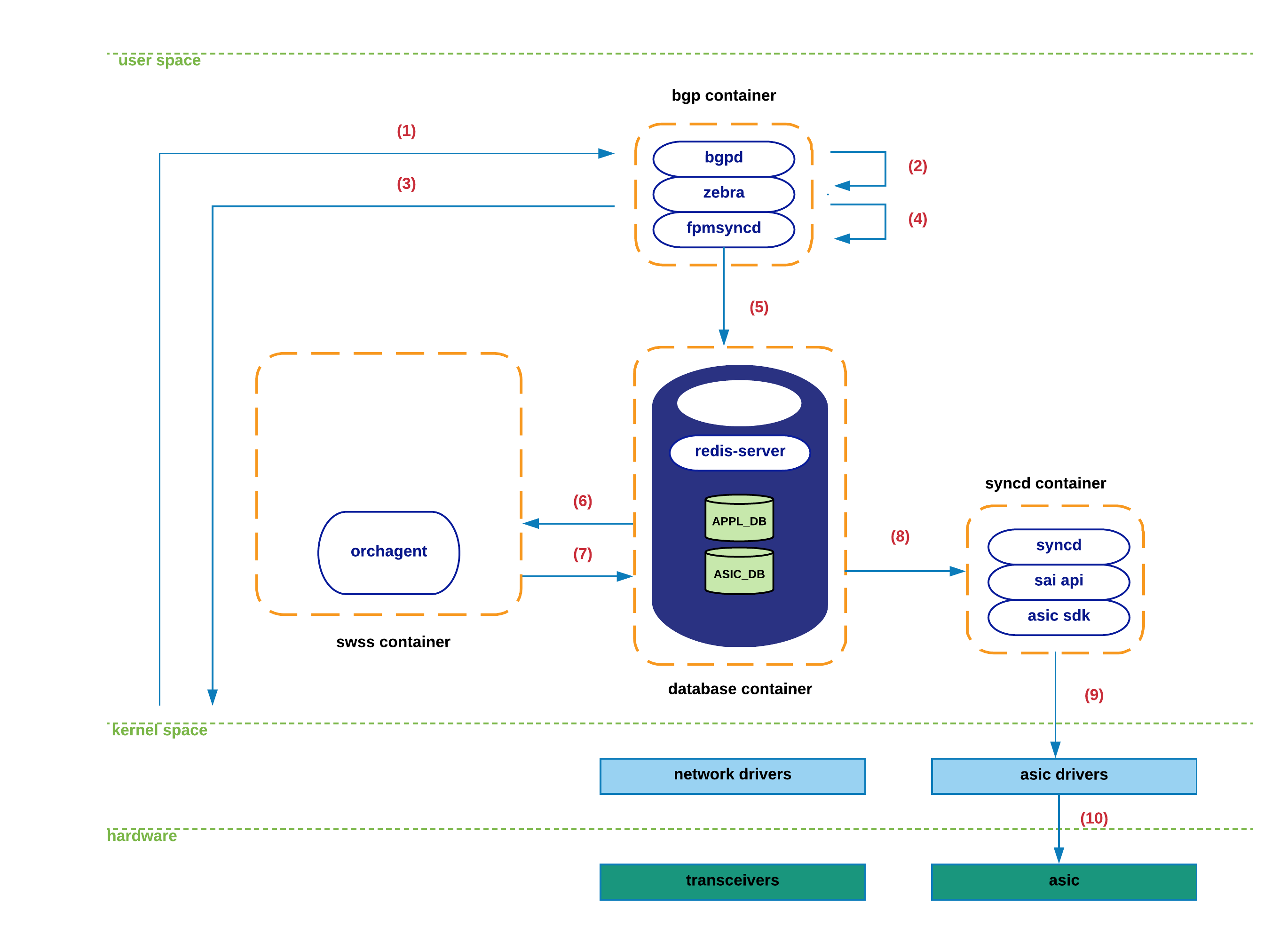


1. During the initialization of the different MIB subcomponents supported in snmp-subagent process, this one establishes connectivity with the various DBs mentioned above. From this moment on, the state obtained from all these DBs is cached locally within snmp-subagent. This information is refreshed every few seconds (< 60) to ensure that DBs and snmp-subagent are fully in-sync.
2. A snmp query arrives at snmp’s socket in kernel space. Kernel’s network-stack delivers the packet to snmpd process.
3. The snmp message is parsed and an associated request is sent towards SONiC’s agentX subagent (i.e. sonic\_ax\_impl).
4. Snmp-subagent serves the query out of the state cached in its local data-structures, and sends the information back to snmpd process.
5. Snmpd eventually sends a reply back to the originator through the usual socket interface.

### Routing-state interactions.

In this section we will iterate through the sequence of steps that take place in SONiC to process a new route received from an eBGP peer. We will assume that this session is already established and that we are learning a new route that makes use of a directly connected peer as its next-hop.

The following figure displays the elements involved in this process. Notice that I’m deliberately obviating details that are not relevant to this SONiC’s architectural description.



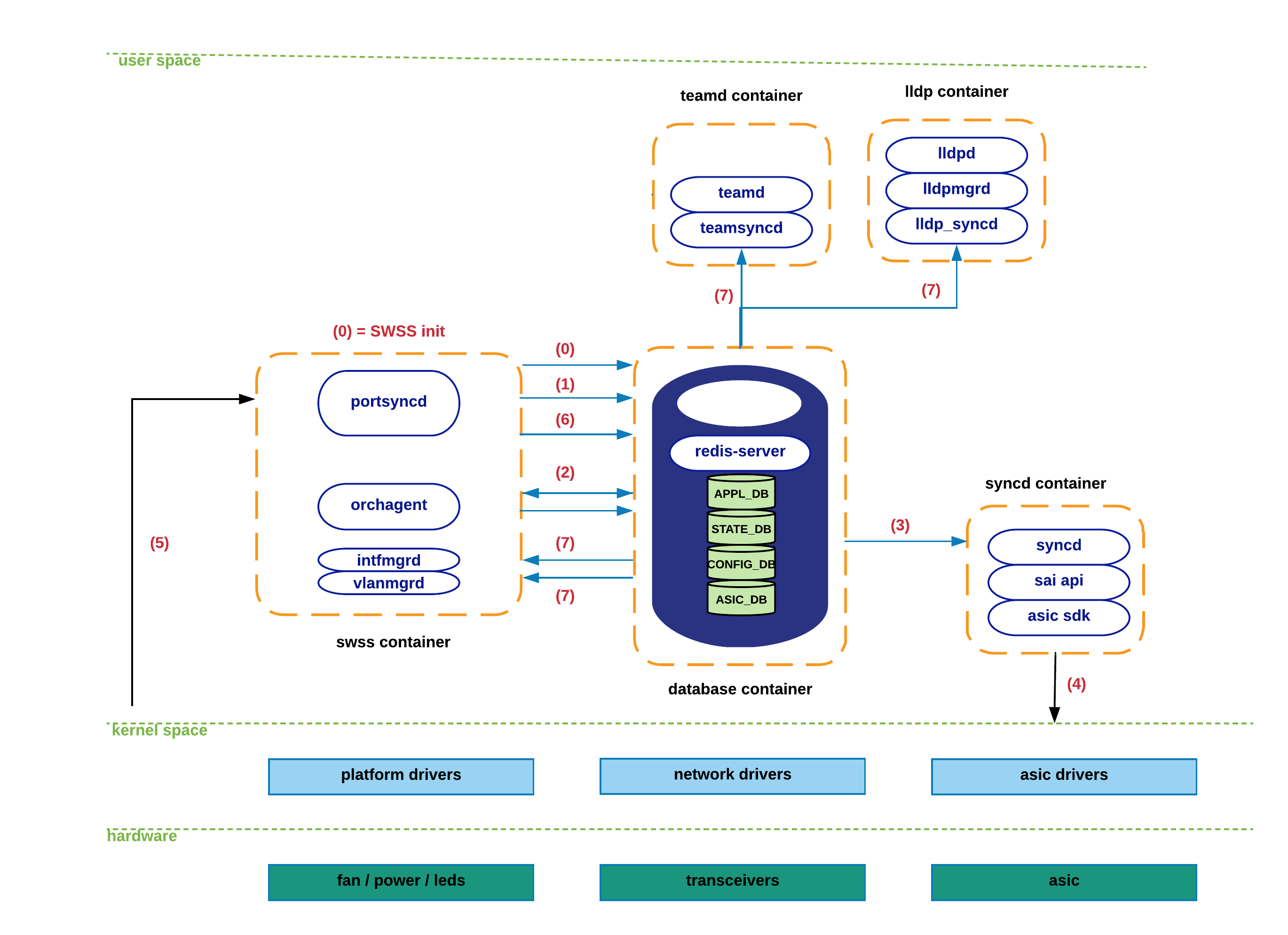
1. During bgp’s container initialization, zebra connects to fpmsyncd through a regular TCP socket. In a stable/non-transient conditions, the routing -tate held within zebra, the linux kernel, APPL\_DB and ASIC\_DB is expected to be fully consistent/equivalent.

1. A new TCP packet arrives at bgp’s socket in kernel space. Kernel’s network-stack eventually delivers the associated payload to bgpd process.
2. Bgpd parses the new packet, process the bgp-update and notifies zebra of the existence of this new prefix and its associated protocol next-hop.
3. Upon determination by zebra of the feasibility/reachability of this prefix (e.g. existing forwarding nh), zebra generates a route-netlink message to inject this new state in kernel.
4. Zebra makes use of the FPM interface to deliver this netlink-route message to fpmsyncd.
5. Fpmsyncd processes the netlink message and pushes this state into APPL\_DB.
6. Being orchagentd an APPL\_DB subscriber, it will receive the content of the information previously pushed to APPL\_DB.
7. After processing the received information, orchagentd will invoke sairedis APIs to inject the route information into ASIC\_DB.
8. Being syncd an ASIC\_DB subscriber, it will receive the new state generated by orchagentd.
9. Syncd will process the information and invoke SAI APIs to inject this state into the corresponding asic-driver.
10. New route is finally pushed to hardware.

### Port-state interactions.

This section describes the system interactions that take place during the transference of port-related information. Taking into account the key-role that portsyncd plays, as well as the dependencies that it imposes in other SONiC subsystems, we start this section by covering its initialization process.

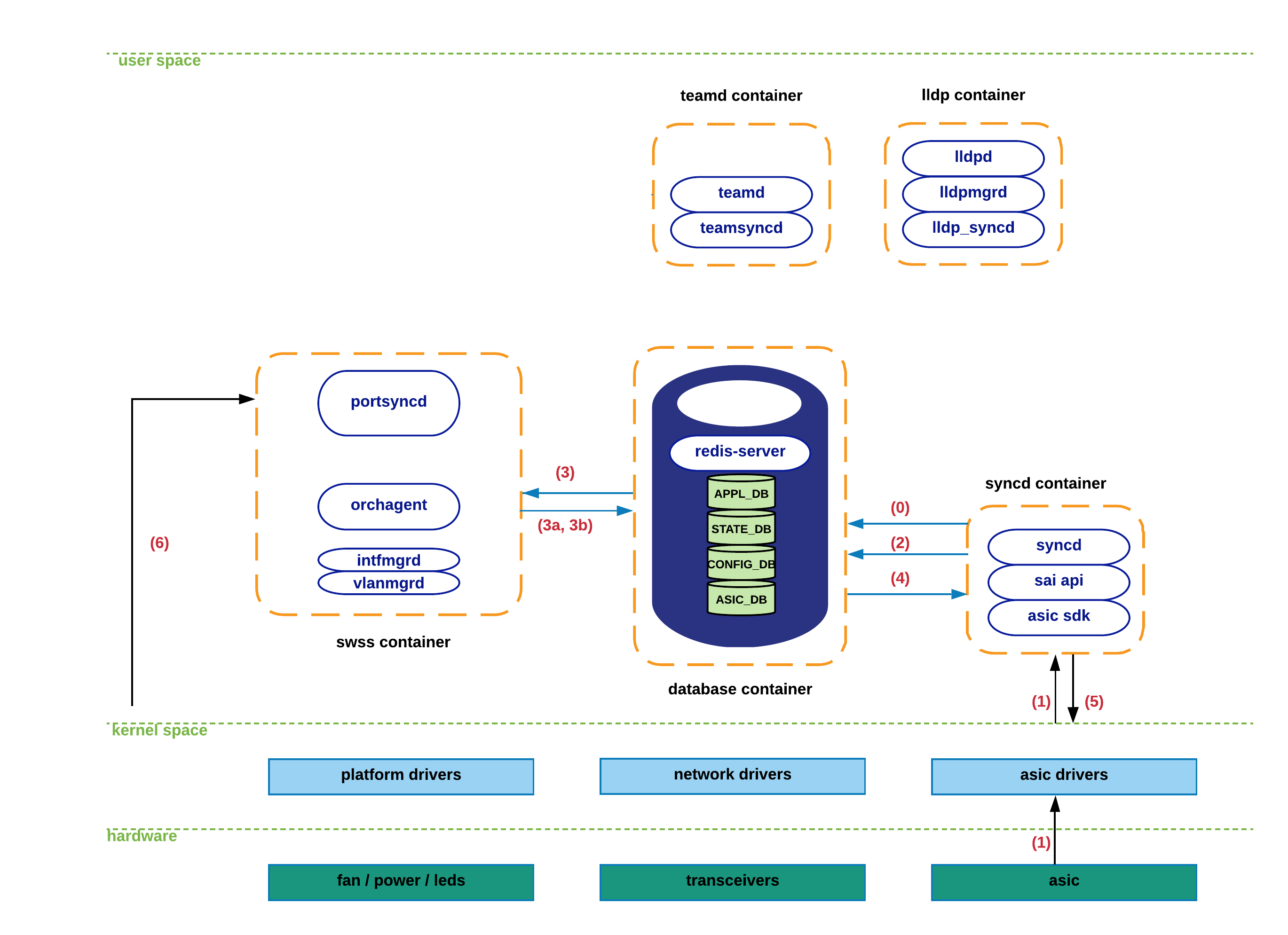
The goal of this exercise is twofold. Firstly, we are exposing the multiple components in the system that are interested in either producing or consuming port-related information. Secondly, we are taking the reader through a graphical example of how the STATE\_DB is used in the system, and how different applications rely on its information for their internal operations.



1. During initialization, portsyncd establishes communication channels with the main databases in the redis-engine. Portsyncd declares its intention to act as a publisher towards APPL\_DB and STATE\_DB, and as a subscriber for CONFIG\_DB. Likewise, portsyncd also subscribes to the system’s netlink channel responsible for carrying port/link-state information.
2. Portsyncd commences by parsing the port-configuration file (port\_config.ini) associated to the hardware-profile/sku being utilized in the system (refer to configuration section for more details). Port-related information such as lanes, interface name, interface alias, speed, etc., is transmitted through this channel on its way to APPL\_DB.
3. Orchagent hears about all this new state but will defer acting on it till portsyncd notifies that it is fully done parsing port\_config.ini information. Once this happens, orchagent will proceed with the initialization of the corresponding port interfaces in hardware/kernel. Orchagent invokes sairedis APIs to deliver this request to syncd through the usual ASIC\_DB interface.
4. Syncd receives this new request through ASIC\_DB and prepares to invoke the SAI APIs required to satisfy Orchagent’s request.
5. Syncd makes use of SAI APIs + ASIC SDK to create kernel host-interfaces associated to the physical ports being initialized.
6. Previous step will generate a netlink message that will be received by portsyncd. Upon arrival to portsyncd of the messages associated to all the ports previously parsed from port\_config.ini (in step 1), portsyncd will proceed to declare the ‘initialization’ process completed.
7. As part of the previous step, portsyncd writes a record-entry into STATE\_DB corresponding to each of the ports that were successfully initialized.
8. From this moment on, applications previously subscribed to STATE\_DB content, will receive a notification to allow these ones to start making use of the ports they are relying on. In other words, if no valid entry is found in STATE\_DB for a particular port, no application will be able to make use of it.

Note: As of today, these are the applications actively listening to the changes in STATE\_DB: teamsyncd, intfmgrd, vlanmgrd and lldpmgr. We will cover all these components in subsequent sections -- lldpmgr has been already tackled above.

Let’s know iterate through the sequence of steps that take place when a physical port goes down:



1. As previously mentioned in the overview section, syncd performs both as a publisher and as a subscriber within the context of ASIC\_DB. The ‘subscriber’ mode is clearly justified by the need for syncd to receive state from the north-bound applications, as has been the case for all the module interactions seen so far. The ‘publisher’ mode is required to allow syncd to notify higher-level components of the arrival of hardware-spawned events.
2. Upon detection of the loss-of-carrier by the corresponding ASIC’s optical module, a notification is sent towards the associated driver, which in turn delivers this information to syncd.
3. Syncd invokes the proper notification-handler and sends the port-down event towards ASIC\_DB.
4. Orchagent makes use of its notification-thread (exclusively dedicated to this task) to collect the new state from ASIC\_DB, and executes the ‘port-state-change’ handler to:
   1. Generate an update to APPL\_DB to alert applications relying on this state for their operation (e.g. CLI – “show interface status”).
   2. Invoke sairedis APIs to alert syncd of the need to update the kernel state associated to the host-interface of the port being brought down. Again, orchagent delivers this request to syncd through the usual ASIC\_DB interface.
5. Syncd receives this new request through ASIC\_DB and prepares to invoke the SAI APIs required to satisfy orchagent’s request.
6. Syncd makes use of SAI APIs + ASIC SDK to update the kernel with the latest operational state (DOWN) of the affected host-interface.
7. A netlink message associated with the previous step is received at portsyncd, which is silently discarded as all SONiC components are by now fully aware of the port-down event.

### Interface-state interactions.

TBD

### Neighbor-state interactions.

TBD

### LAG-Interface-state interactions.

TBD

### Configuration-state interactions.

TBD

More details about the different CLI commands exposed to the user, as well as the system’s configuration files will take place in subsequent sections of this document.

# System Configuration

## System Configuration Overview

As already seen in the previous sections of this document, SONiC’s config-management strategy relies on the use of a unique data-store engine to hold all the configuration state in the system. This data-store is built atop a redis-database engine that offers publisher/subscriber capabilities to the different subsystems that comprise the SONiC ecosystem.

As described earlier, there are various databases being hosted within this redis-engine, but for the purpose of this section we will concentrate on the ones dealing with configuration state, to which we usually refer to as **configDB** and **stateDB**. As its name implies, configDB hosts all the configuration state present in the system, whereas stateDB serves as a centralized location to resolve dependency requirements imposed by various SONiC components (refer to System Architecture section for more details).

Upon system’s reboot, SONiC’s configuration files will be parsed to upload the relevant state to configDB and stateDB. The south-bound applications interested in receiving this information would have previously subscribed to these two databases to explicitly state this goal. The SONiC applications subscribed to configDB will make use of the received information to derive their own application-specific configuration.

Reader should be aware that even though configDB is capable of dealing with incremental configuration changes, this feature has not been implemented by all the SONiC applications. In consequence, a manual restart will be required on SONiC’s backend services whenever configuration files are modified. There are a few exceptions to this rule which we will highlight in subsequent sections.

## System Configuration Files

During boot-up phase, SONiC’s configuration files are typically obtained from ZTP servers, and placed within the usual file-configuration path: **/etc/sonic/**. Within the scope of this document, we will focus on these specific files as they deal with virtually the entire configuration of the system:

/etc/sonic/config\_db.json <-- generic system configuration

/etc/sonic/frr/frr.conf <-- routing-stack configuration

**config\_db.json** is a straight-forward serialization of configDB, and as such, it contains a one-to-one mapping with configDB content. File content is laid out making use of JSON format, which is ideal to store/transmit data objects consisting of attribute-value pairs as those ones held in configDB. This file pretty much contains most of the system’s configuration items, with the exception of ACLs and routing functionalities.

**frr.conf** file deals with all the routing features of the system, and it follows a Cisco’s IOS-alike layout to represent its information. Please refer to the routing-configuration section for more details on this.

## System Auxiliary Configuration Files

As previously mentioned, SONiC’s incremental-configuration capabilities aren’t quite there yet, or at least they are not homogenously supported across the entire functionality spectrum. Even though the current CLI displays a relatively broad set of features, in reality, the set of actions associated to them is quite narrow – see example below for interface-configuration options:

admin@lnos-x1-a-asw02:~$ sudo config ?

Usage: config [OPTIONS] COMMAND [ARGS]...

SONiC command line - 'config' command

Options:

--help Show this message and exit.

Commands:

aaa AAA command line

acl ACL-related configuration tasks

bgp BGP-related configuration tasks

ecn ECN-related configuration tasks

interface Interface-related configuration tasks

load Import a previous saved config DB dump file.

load\_mgmt\_config Reconfigure hostname and mgmt interface based...

load\_minigraph Reconfigure based on minigraph.

reload Clear current configuration and import a...

save Export current config DB to a file on disk.

tacacs TACACS+ server configuration

vlan VLAN-related configuration tasks

admin@lnos-x1-a-asw02:~$ sudo config interface ?

Usage: config interface [OPTIONS] COMMAND [ARGS]...

Interface-related configuration tasks

Options:

--help Show this message and exit.

Commands:

shutdown Shut down interface

speed Set interface speed

startup Start up interface

In consequence, for the time being we will generally rely on the system configuration files described above to inject state into **configDB**.

Nevertheless, to offer the user certain level of flexibility, we should mention that user can also rely on auxiliary files that can push configuration state into **configDB**. Thus, on top of the system-wide configuration files seen above, we can also have the following feature-specific files:

**acl.json <--** hosts all ACL-related configuration

**vlan.json** <-- hosts all VLAN-related configuration

The syntax of these files is exactly the same as the one expected in **config\_db.json**, only that these ones will cover a much smaller section of the configuration. In fact, the content of these files, once validated and digested by the sonic config parsers, will eventually end up being dumped to the system-wide **config\_db.json** file – we will see how to achieve that in subsequent sections.

So, what’s the added value offered by these files? First of all, by reducing the configuration scope to a unique feature, users are less exposed to potential grammatical/semantical errors being introduced in the configuration.

Secondly, and this is the key point, this approach allows a gradual transition to an incremental-configuration-update model, where certain features support configuration changes that don’t require backend-services being reloaded. Today, Vlan is one of those features offering these capabilities; a monolithic and all-encompassing **config\_db.json** file cannot provide this level of flexibility, that’s the main rationale for the existence of these files.

## System Configuration Workflow -- Provisioning

As previously mentioned, in order to inject configuration state into the system’s configDB, we must either append content to config\_db.json file (default approach), or we must interact with a feature-specific config file (e.g. vlan.json).

We will rely on the following command to manually trigger the processing of newly introduced changes into any of the json configuration files. For example, the following command will regenerate all configDB state by pushing the entries present in config\_db.json file:

admin@lnos-x1-a-asw02:~$ sudo config load /etc/sonic/config\_db.json

Reload all config? [y/N]: y

Running command: sonic-cfggen -j /etc/sonic/config\_db.json --write-to-db

Besides regenerating configDB state, this second instruction will also restart all the backend services to ensure that new configuration state is processed by all SONiC entities. Thereby, we should expect some service disruption during its execution.

admin@lnos-x1-a-asw02:~$ sudo config reload /etc/sonic/config\_db.json

Clear current and reload all config? [y/N]: y

Running command: sonic-cfggen -j /etc/sonic/config\_db.json --write-to-db

Running command: service hostname-config restart

Running command: service interfaces-config restart

Running command: service ntp-config restart

Running command: service rsyslog-config restart

Running command: service swss restart

Running command: service bgp restart

Running command: service teamd restart

Running command: service pmon restart

Running command: service lldp restart

Running command: service snmp restart

Running command: service dhcp\_relay restart

Users will need to explicitly execute one of these two commands every time that config\_db.json file is modified. The **config load** instruction has been conceived assuming that the changes being made will be automatically absorbed by the backend services (configDB subscribers), which requires a system capable of supporting incremental-configuration updates.

Even though there are specific features that support incremental-configuration updates and could benefit from **config load** instruction, there are others that don’t support it, so we will generally rely on **config reload** command to push config-state to the backend. However, in cases where the config changes are contained within a module that supports incremental-configuration updates (e.g. VLAN or ACL), it will be up to the user to decide whether to rely on **config load** or **config reload** command.

Another important aspect to highlight is that, if any change were to be made to configDB without relying on config\_db.json, we would need to make use of the **config save** instruction (see example below) to ensure that configDB state is backed up at config\_db.json file. This could only happen if:

* User relies on a functionality-specific config file (i.e. vlan.json or acl.json), or…
* User makes use of the partially-built CLI tool that is potentially capable of injecting incremental-configuration state into configDB (e.g. **config vlan**), or…
* User directly injects state into configDB making use of redis-cli tool (see next section for more details on this).

Out of these three options, only the first one is supported for the special features that we have talked about earlier. The last two options are highly discouraged at this point, and we are only presenting them here to emphasize that changes made directly into configDB – that is, without config\_db.json involvement -- will not be saved in config\_db.json, so the following instruction will be required in all the above cases:

admin@lnos-x1-a-asw02:~$ sudo config save

Existing file will be overwritten, continue? [y/N]: y

Running command: sonic-cfggen -d --print-data > /etc/sonic/config\_db.json

admin@lnos-x1-a-asw02:~$

## System Configuration Workflow – Verification

Once newly introduced changes are digested by the system (previous step), we will expect the associated state to be held by configDB. Being Redis a key-value-store engine, we should expect entries to be formed by a combination of an object associated to the input key, and a string corresponding to its value.

Redis-engine provides a very useful CLI tool that grants direct read/write access to its users. Our goal here is to make use of this tool to verify that the configuration state is properly recorded in configDB. Users should be mindful of the consequences of using redis write-capable instructions through this tool, as it could easily render the system in an inconsistent state.

The following set of instructions serve as an example of a typical read-only user interaction with redis-cli:

admin@lnos-x1-a-asw02:~$ redis-cli <-- entering redis-engine shell

127.0.0.1:6379>

127.0.0.1:6379> select 4 <-- specify the desired table to query – configDB = 4

OK

127.0.0.1:6379[4]>

127.0.0.1:6379[4]> keys \* <-- displaying all keys in configDB

1) "PORT|Ethernet68"

2) "PORT|Ethernet110"

3) "PORT|Ethernet56"

4) "PORT|Ethernet126"

127.0.0.1:6379[4]> keys "VLAN\_INT\*" <-- displaying keys meeting regexp criteria

1) "VLAN\_INTERFACE|Vlan777|8.1.1.2/24"

2) "VLAN\_INTERFACE|Vlan100|fc00:9:2::2/64"

3) "VLAN\_INTERFACE|Vlan100|9.1.1.2/24"

4) "VLAN\_INTERFACE|Vlan777|fc00:8:2::2/64"

127.0.0.1:6379[4]>

127.0.0.1:6379[4]> hgetall "PORT|Ethernet24" <-- obtaining value associated with a key

1) "alias"

2) "tenGigE1/25"

3) "lanes"

4) "41"

127.0.0.1:6379[4]>

## System Port-Mapping

As is customary in every Linux-based system, users can rely on **ip address** or **ifconfig** (deprecated) native commands to display all the network interfaces in a device. For this purpose, SONiC also provide its own set of commands (e.g. **show interface status**), which will be discussed in **System Operation** section.

Our goal here is to explain how physical ports are mapped to the internal logical-constructs used by SONiC to represent each switch interface. By doing so, user should be able to easily correlate each switch’s physical-port with its associated logical interface, which is a basic requirement to understand the output of operational commands and syslog events.

This port-mapping association will be dictated by the hardware-profile elected to configure a switch. This hardware-profile will define the port-breakout mode in which each switch operates. For example, for the case of Celestica platforms, SONiC supports three different hardware profiles to configure its 32 QSFP28 transceivers:

* 96x10G + 16x50G mode (default)
* 64x50G mode
* 32x100G mode

As each QSFP28 transceiver handles four data-lanes that can be used individually or combined, each port could potentially operate as a single 100GbE interface, or as two 50GbE interfaces, or as four 10GbE interfaces.

As there is no one-to-one mapping between interfaces and physical ports, SONiC provides a per-interface **alias** attribute to serve as an intuitive hint to the user by combining physical-port and lane semantics. The following output provides a visual representation of how all these elements interplay in SONiC’s cli:

admin@lnos-x1-a-asw02:~$ show interfaces status

Command: intfutil status

Interface Admin Oper Alias Lanes Speed MTU

----------- ------- ------ ------- ------- ------- -----

Ethernet0 up down Eth1/1 65 10G 9100

Ethernet1 up down Eth1/2 66 10G 9100 phy-port 1

Ethernet2 up down Eth1/3 67 10G 9100

Ethernet3 up down Eth1/4 68 10G 9100

Ethernet4 up down Eth2/1 69 10G 9100

Ethernet5 up down Eth2/2 70 10G 9100 phy-port 2

Ethernet6 up down Eth2/3 71 10G 9100

Ethernet7 up down Eth2/4 72 10G 9100

physical-port lane-id lane-number

The example above corresponds to the first two physical-ports of an SKU configured in **96x10G + 16x50G** mode, and that is why we observe a list of interfaces being consecutively incremented (Ethernet0..Ethernet7). However, that won’t be the case for scenarios with multi-lane interfaces, as in those cases a single interface will be created to represent each lane aggrupation; hence, a gap will be expected in the ‘Interface’ column displayed below:

admin@lnos-x1-a-asw02:~$ show interfaces status

Command: intfutil status

Interface Admin Oper Alias Lanes Speed MTU

----------- ------- ------ ------- ------- ------- -----

…

Ethernet100 up down Eth26/1 21,22 50G 9100

Ethernet102 up down Eth26/2 23,24 50G 9100

Ethernet104 up down Eth27/1 25,26 50G 9100

Ethernet106 up down Eth27/2 27,28 50G 9100

Ethernet108 up down Eth28/1 29,30 50G 9100

Ethernet110 up down Eth28/2 31,32 50G 9100

admin@lnos-x1-a-asw02:~$ show interfaces status

Command: intfutil status

Interface Admin Oper Alias Lanes Speed MTU

----------- ------- ------ ------- --------------- ------- -----

Ethernet0 up up Eth1 65,66,67,68 100G 9100

Ethernet4 up down Eth2 69,70,71,72 100G 9100

Ethernet8 up down Eth3 73,74,75,76 100G 9100

Ethernet12 up down Eth4 77,78,79,80 100G 9100

Ethernet16 up down Eth5 33,34,35,36 100G 9100

## Port-Breakout Configuration

As discussed in the previous section, the port-breakout layout will be determined by the elected hardware-profile. These hardware-profiles are represented by platform-dependent files associated to every supported SKU. In other words, the speed at which any given port operates is determined in a predefined file. Thereby, as of today, there is no support for dynamic port-breakout functionality.

To allow user to elect a hardware-profile different than default, SONiC provides a useful script to smooth out this transition. User should keep in mind that changing the hardware configuration entails service disruption as all SONiC services will be restarted. See below an example of the use of this script:

admin@lnos-x1-a-asw02:~$ sudo /usr/bin/config-hwsku.sh -h

Usage: /usr/bin/config-hwsku.sh [-h] [-p] [-l] [-s HWSKU]

-h, --help print this usage page

-p, --print print the current HWSKU

-l, --list list the available HWSKUs

-s, --set set the HWSKU

admin@lnos-x1-a-asw02:~$ sudo /usr/bin/config-hwsku.sh -p

Seastone-DX010

admin@lnos-x1-a-asw02:~$ sudo /usr/bin/config-hwsku.sh -l

Seastone-DX010

Seastone-DX010-10-50

Seastone-DX010-50

admin@lnos-x1-a-asw02:~$ sudo /usr/bin/config-hwsku.sh -s Seastone-DX010-10-50

This will reset to the initial configuration with the port mode specified and restart all services. [Y/N] Y

loading the new configuration

Running command: sonic-cfggen -m -j /etc/sonic/init\_cfg.json --write-to-db

Running command: service hostname-config restart

Running command: service interfaces-config restart

Running command: service ntp-config restart

Running command: service rsyslog-config restart

Running command: service swss restart

Running command: service bgp restart

Running command: service teamd restart

Running command: service pmon restart

Running command: service lldp restart

Running command: service snmp restart

Running command: service dhcp\_relay restart

Please note setting loaded from minigraph will be lost after system reboot. To preserve setting, run `config save`.

backup the old config\_db.json to .bak file and save the new one.

Running command: sonic-cfggen -d --print-data > /etc/sonic/config\_db.json

For the case of Celestica SKUs, the default hardware-profile is **Seastone-DX010-10-50**. This configuration allows network elements to operate as LinkedIn’s ASW switches, as they provide both 10G and 50G connectivity. An alternative to this would be the **Seastone-DX010-50** profile, which meets Linkedin’s CSW connectivity requirements. Finally, we have the **Seastone-DX010** mode which isn’t currently utilized today.

For a complete list of the SONiC supported SKUs refer to [[5](https://github.com/Azure/SONiC/wiki/Supported-Devices-and-Platforms)].

## Device Metadata Configuration

This section of the config\_db.json file allows users to configure system-wide attributes such as the **hostname**, **hwsku**, **mac**, etc. Attributes like **bgp\_asn** and **type** are not parsed by any backend service relevant to the LNOS implementation, so they can be skipped by the user.

"DEVICE\_METADATA": {

"localhost": {

"hwsku": "Seastone-DX010-10-50",

"hostname": "sonic",

"mac": "00:e0:ec:3c:0a:16",

"bgp\_asn": "65100",

"deployment\_id": null,

"type": "LeafRouter"

}

}

The **hwsku** attribute deserves further consideration as this one defines the port-breakout scheme to be utilized by the OS. As seen in previous section, there are multiple SKUs supported by SONiC today, and for our purposes we have selected the **Seastone-DX010-10-50** SKU as the default one.

Even through **hwsku** could be potentially modified in this file, we do highly discourage that practice as the rest of the configuration in config\_db.json would be partially inconsistent with the new **hwsku** value. As mentioned above, for hardware-profile/sku changes users should exclusively rely on the script provided for such effect.

Once the configuration is modified and the associated state is processed by backend services, the corresponding entries should be visible in configDB:

127.0.0.1:6379[4]> hgetall "DEVICE\_METADATA|localhost"

1) "hwsku"

2) "Seastone-DX010-10-50"

3) "hostname"

4) "switch1"

5) "mac"

6) "00:e0:ec:3c:0a:16"

7) "bgp\_asn"

8) "12"

9) "deployment\_id"

10) "None"

11) "type"

12) "LeafRouter"

127.0.0.1:6379[4]>

## Ports Configuration

Port attributes can be defined within the **PORT** stanza in config\_db.json. While **lanes** attribute and **speed** are hard-coded --derived from each sku’s hardware-profile and corresponding port\_config.ini --, the **description** field is configurable.

“PORT”: {

"Ethernet112": {

"alias": "Eth29/1",

"lanes": "113,114",

"description": "my port 112",

"speed": "50000"

},

"Ethernet114": {

"alias": "Eth29/2",

"lanes": "115,116",

"description": "my port 114",

"speed": "50000"

},

}

These config entries should be visible in configDB as show below:

127.0.0.1:6379[4]> hgetall "PORT|Ethernet114"

1) "alias"

2) "Eth29/2"

3) "lanes"

4) "115,116"

5) "description"

6) "my port 114"

7) "speed"

8) "50000"

127.0.0.1:6379[4]>

## VLAN Configuration

As previously mentioned in section 5.3, VLAN is one of the features currently supporting incremental-configuration updates, so its associated config-changes can be loaded without requiring backend services being restarted. Regardless of the json file being elected for its configuration, the attributes being supported must be laid out as displayed below:

"VLAN": {

"Vlan100": {

"vlanid": "100",

"admin\_status": "up",

"description": "Data Traffic",

"members": [

"Ethernet16",

"Ethernet17"

],

"mtu": "9100"

},

"Vlan777": {

"vlanid": "777",

"admin\_status": "up",

"description": "IPMI",

"members": [

"Ethernet16",

"Ethernet17"

],

"mtu": "9100"

}

}

Once again, this information should be properly stored at configDB:

127.0.0.1:6379[4]> hgetall "VLAN|Vlan777"

1) "admin\_status"

2) "up"

3) "members@"

4) "Ethernet16,Ethernet17"

5) "description"

6) "IPMI"

7) "vlanid"

8) "777"

9) "mtu"

10) "9100"

127.0.0.1:6379[4]>

## L3 Interfaces Configuration

L3 interfaces will be configured within either the **INTERFACE** or the **VLAN\_INTERFACE** stanza, depending upon the underlying interface on which each L3 entry is being defined.

As can be seen below, L3 interface objects don’t support any attribute. Also, as of today, a single ip-address (ipv4 or ipv6) is supported for each interface-entry; if more than one ip-address is required for a given interface, then a secondary entry needs to be configured. See below an example of this behavior for Ethernet112:

"INTERFACE": {

"Ethernet120|fc00:3:2::2/64": {},

"Ethernet112|fc00:1:2::2/64": {},

"Ethernet8|17:0:50:0:0:0:0:1/96": {},

"Ethernet124|fc00:4:2::2/64": {},

"Ethernet112|10.1.2.2/24": {},

"Ethernet116|10.2.2.2/24": {},

"Ethernet120|10.3.2.2/24": {},

"Ethernet124|10.4.2.2/24": {},

"Ethernet8|12.1.1.2/24": {},

"Ethernet116|fc00:2:2::2/64": {}

},

"VLAN\_INTERFACE": {

"Vlan100|9.1.1.2/24": {

"scope": "global",

"family": "IPv4"

},

"Vlan100|fc00:9:2::2/64": {

"scope": "global",

"family": "IPv6"

},

"Vlan777|8.1.1.2/24": {

"scope": "global",

"family": "IPv4"

},

"Vlan777|fc00:8:2::2/64": {

"scope": "global",

"family": "IPv6"

}

},

Once processed, this information should be visible in configDB:

127.0.0.1:6379[4]> keys "INTERFACE\*"

1) "INTERFACE|Ethernet8|12.1.1.2/24"

2) "INTERFACE|Ethernet120|10.3.2.2/24"

3) "INTERFACE|Ethernet112|fc00:1:2::2/64"

4) "INTERFACE|Ethernet120|fc00:3:2::2/64"

5) "INTERFACE|Ethernet116|10.2.2.2/24"

6) "INTERFACE|Ethernet8|17:0:50:0:0:0:0:1/96"

7) "INTERFACE|Ethernet112|10.1.2.2/24"

8) "INTERFACE|Ethernet124|fc00:4:2::2/64"

9) "INTERFACE|Ethernet124|10.4.2.2/24"

10) "INTERFACE|Ethernet116|fc00:2:2::2/64"

127.0.0.1:6379[4]>

127.0.0.1:6379[4]> keys "VLAN\_INTERFACE\*"

1) "VLAN\_INTERFACE|Vlan777|8.1.1.2/24"

2) "VLAN\_INTERFACE|Vlan100|fc00:9:2::2/64"

3) "VLAN\_INTERFACE|Vlan100|9.1.1.2/24"

4) "VLAN\_INTERFACE|Vlan777|fc00:8:2::2/64"

127.0.0.1:6379[4]>

## Loopback Interface Configuration

In SONiC, the system’s loopback interface has a dedicated location within config\_db.json schema.

"LOOPBACK\_INTERFACE": {

"Loopback0|12.12.12.12/32": {},

"Loopback0|FC00:12::12/128": {}

},

ConfigDB verification:

127.0.0.1:6379[4]> keys "LOOPBACK\*"

1) "LOOPBACK\_INTERFACE|Loopback0|12.12.12.12/32"

2) "LOOPBACK\_INTERFACE|Loopback0|fc00:12::12/128"

127.0.0.1:6379[4]>

## Management Interface Configuration

Management interfaces are defined in **MGMT\_INTERFACE** stanza. Object key is composed of management interface name and IP prefix. The attribute **gwaddr** specify the gateway address of the prefix; whereas **forced\_mgmt\_routes** attribute is used to specify addresses or prefixes to where we want to send traffic to making use of the management network interface.

"MGMT\_INTERFACE": {

"eth0|172.25.11.44/24": {

"forced\_mgmt\_routes": [

"99.99.99.1/30"

],

"gwaddr": "172.25.11.1"

}

},

Again, configDB stores this information accordingly:

127.0.0.1:6379[4]> hgetall "MGMT\_INTERFACE|eth0|172.25.11.44/24"

1) "gwaddr"

2) "172.25.11.1"

3) "forced\_mgmt\_routes@"

4) "99.99.99.1/30"

127.0.0.1:6379[4]>

## ACL Configuration

[ this section needs some adjustments ]

High level configuration flow is that ACL configurations could be part of the config\_db.json that provided by ZTP if no dynamic changes are needed.

In case we want to config/remove ACL dynamically, we push the configurations to configDB through CLI (config load) with json files (or APIs in the future), SONiC daemons will pick up the configurations and push them to HW.

To change the ACL configurations dynamically as mentioned above, we need understand what the ACL design looks like:

There are two tables in configDB to configure ACL features. One is ACL\_TABLE, the other is ACL\_RULE table.

ACL\_TABLE:

The table has a table\_id which is some string specified by the user and should be unique across the configDB. table\_id will be used to refer the table when adding rules and updating or deleting the table. Tables will have the following properties:

* policy\_desc: name of the ACL policy table description
* type: one of the two predefined table types: "L3" or "L3V6” (“mirror" to be described in future)
* stage: ingress or egress, by default is ingress if not specified.
* ports: the list or ports bound to the table

Table type defines a list of supported matches that could be used in rules belonging to this table.

See below exactly what was supported for table type L3 and L3V6:

L3:

|  |  |  |
| --- | --- | --- |
| Keyword for the match criteria | Type | Description |
| ETHER\_TYPE | uint16\_t | Hexadecimal integer [0..FFFF]  (with or w/o leading "0x") |
| IP\_TYPE | string | One of: "IPv4"/"NON\_IPv4"/"ARP" |
| IP\_PROTOCOL | uint8\_t | Hexadecimal unsigned integer [0..FF] |
| SRC\_IP | ip\_address | A valid IPv4 subnet in format IP/Mask |
| DST\_IP | ip\_address | A valid IPv4 subnet in format IP/Mask |
| L4\_SRC\_PORT | uint16\_t | Decimal unsigned integer [0..65535] |
| L4\_DST\_PORT | uint16\_t | Decimal unsigned integer [0..65535] |
| TCP\_FLAGS | uint8\_t | Hexadecimal unsigned integer [0..FF] |
| L4\_SRC\_PORT\_RANGE | uint16\_t, uint16\_t | Two dash separated decimal unsigned integers [0..65535] |
| L4\_DST\_PORT\_RANGE | uint16\_t, uint16\_t | Two dash separated decimal unsigned integers [0..65535] |

L3V6:

|  |  |  |
| --- | --- | --- |
| Keyword for the match criteria | Type | Description |
| ETHER\_TYPE | uint16\_t | Hexadecimal integer [0..FFFF]  (with or w/o leading "0x") |
| IP\_TYPE | string | One of: "IPv4"/"NON\_IPv4"/"ARP" |
| IP\_PROTOCOL | uint8\_t | Hexadecimal unsigned integer [0..FF] |
| SRC\_IPV6 | ip\_address | A valid IPv6 subnet in format IP/Mask |
| DST\_IPV6 | ip\_address | A valid IPv6 subnet in format IP/Mask |
| L4\_SRC\_PORT | uint16\_t | Decimal unsigned integer [0..65535] |
| L4\_DST\_PORT | uint16\_t | Decimal unsigned integer [0..65535] |
| TCP\_FLAGS | uint8\_t | Hexadecimal unsigned integer [0..FF] |
| L4\_SRC\_PORT\_RANGE | uint16\_t, uint16\_t | Two dash separated decimal unsigned integers [0..65535] |
| L4\_DST\_PORT\_RANGE | uint16\_t, uint16\_t | Two dash separated decimal unsigned integers [0..65535] |

Both support ACTION as below:

|  |  |  |
| --- | --- | --- |
| Keyword for the action | Type | Description |
| PACKET\_ACTION | string | Packet action value: "FORWARD" or "DROP" |

ACL RULE:

ACLRules is under the key ACL\_RULE|table\_id|rule\_id. table\_id is the table ID of ACL\_TABLE the rule belongs to and the rule\_id is some string which should be unique across the Table. rule\_id will be used to refer the Rule when it is needed to update or delete the Rule. Rules will have the following properties:

* priority - rule priority in the table, **higher the number means higher the priority, HW will give precedence to the highest priority rule if multiple rules could match. Range: 0-0x7fffffff**
* match:value - packet properties this rule will match
* action:value - action to be applied to the rule if match was successful

The list of allowed matches and actions depends on the ACL\_TABLE the rule will go to.

**By default, there was no implicit deny rule on the box, we need put a rule explicitly with "priority" 0 to deny all traffic.**

Example ACL\_TABLE in acl-tables.json:

{

"ACL\_TABLE": {

"dataacl": {

"type": "L3",

"policy\_desc": "dataacl",

"ports": [

"Ethernet8",

"Ethernet9",

"Ethernet16",

"Ethernet17",

"Ethernet18"

]

},

"data-aclv6": {

"type": "L3V6",

"policy\_desc": "data-aclv6",

"ports": [

"Ethernet8",

"Ethernet9",

"Ethernet16",

"Ethernet17",

"Ethernet18"

]

}

}

}

This typically is part of the config\_db.json as we don’t usually change the table binding etc. But you can load it through CLI to configDB if required: config load acl-tables.json

To change the acl rules dynamically, we load them through the same CLI as above. Example rules and commands as below:

ipv4-rules.json:

{

"ACL\_RULE": {

"dataacl|Rule\_3": {

"PACKET\_ACTION":"DROP",

"SRC\_IP":"11.10.1.0/24",

"priority":3000

}

}

"ACL\_RULE": {

"dataacl|Rule\_4": {

"PACKET\_ACTION":"FORWARD",

"SRC\_IP":"11.10.1.0/24",

"DST\_IP":"12.10.1.8/32",

"priority":4000

}

}

}

ipv6-rules.json:

{

"ACL\_RULE": {

"data-aclv6|Rule\_3": {

"PACKET\_ACTION":"DROP",

"SRC\_IPV6":"2777::0/64",

"priority":3000

}

}

"ACL\_RULE": {

"data-aclv6|Rule\_4": {

"PACKET\_ACTION":"FORWARD",

"SRC\_IPV6":"2777::0/64",

"DST\_IPV6":"3777::8/128",

"priority":4000

}

}

}

#config load ipv4-rules.json

#config load ipv6-rules.json

If you want to delete the rules, the below json files will be used:

ipv4-rules-del.json

{

"ACL\_RULE": {

"dataacl|Rule\_3": null

}

"ACL\_RULE": {

"dataacl|Rule\_4": null

}

}

ipv6-rules-del.json

{

"ACL\_RULE": {

"data-aclv6|Rule\_3": null

}

"ACL\_RULE": {

"data-aclv6|Rule\_4": null

}

}

# config load ipv4-rules-del.json

# config load ipv6-rules-del.json

If no dynamic rule changes are needed, it is recommended to put the ACL\_TABLE and ACL\_RULE together into config\_db.json which was again provided by ZTP and was generated by outside config-gen.

To display the table/rule configured on SONiC and statistics of the rules, command below can be used:

# aclshow -d

ACL Table: dataacl

==================

Property Value

---------------- -----------------------------------------------------------

type............ L3

policy\_desc..... dataacl

ports........... ['Ethernet8', 'Ethernet9', 'Ethernet16', 'Ethernet17', 'Ethernet18']

ACL Table: data-aclv6

=====================

Property Value

---------------- -----------------------------------------------------------

type............ L3V6

policy\_desc..... data-aclv6

ports........... ['Ethernet8', 'Ethernet9', 'Ethernet16', 'Ethernet17', 'Ethernet18']

ACL Rule: Rule\_3

================

Property Value

---------------- -----------------------------------------------------------

PACKET\_ACTION... DROP

SRC\_IP.......... 11.10.1.0/24

priority........ 3000

packets counter. 3256

bytes counter... 345125

ports........... ['Ethernet8', 'Ethernet9', 'Ethernet16', 'Ethernet17', 'Ethernet18']

ACL Rule: Rule\_4

================

Property Value

---------------- -----------------------------------------------------------

PACKET\_ACTION... FORWARD

priority........ 4000

SRC\_IPV6........ 2777::0/64

DST\_IPV6........ 3777::8/128

packets counter. 3209

bytes counter... 404334

ports........... ['Ethernet8', 'Ethernet9', 'Ethernet16', 'Ethernet17', 'Ethernet18']

ACL Rule: Rule\_3

================

Property Value

---------------- -----------------------------------------------------------

PACKET\_ACTION... DROP

priority........ 3000

SRC\_IPV6........ 2777::0/64

packets counter. 3249

bytes counter... 409363

ports........... ['Ethernet8', 'Ethernet9', 'Ethernet16', 'Ethernet17', 'Ethernet18']

ACL Rule: Rule\_4

================

Property Value

---------------- -----------------------------------------------------------

PACKET\_ACTION... FORWARD

SRC\_IP.......... 11.10.1.0/24

priority........ 4000

DST\_IP.......... 12.10.1.8/32

packets counter. 3213

bytes counter... 340578

ports........... ['Ethernet8', 'Ethernet9', 'Ethernet16', 'Ethernet17', 'Ethernet18']

Linux also supports monitoring the CLI output by using “watch” command, it basically refreshes the command output every 2 seconds by default:

#watch aclshow –d

## Routing Configuration

As previously hinted in Configuration Overview section, LNOS’ routing-stack configuration diverge from SONiC’s approach. First and foremost, SONiC makes use of Quagga as the default routing-stack, whereas LNOS rely on FRR. Secondly, FRR configuration does not follow the SONiC traditional configurational model depicted above. LNOS’ FRR configuration is not extracted from config\_db.json, nor this information is placed into configDB.

FRR configuration daemon, which resides within one of the docker containers present in the system, will parse FRR’s configuration file in the host file-system (/etc/sonic/frr/frr.conf) to extract the routing configuration generated by the user. This is made possible by mounting /etc/sonic/frr/ folder in the host file-system, over the path expected by the FRR daemons running within the docker container: /etc/frr/. In other words, the frr.conf file that appears inside the FRR docker container, is just a symbolic representation of the real file that seats in the host file-system.

By deviating from the traditional config\_db.json based approach, we are offering our users a much more flexible interaction with the routing-subsystem. The reason is twofold: first of all this approach exposes the full set of features present in FRR stack through its integrated-shell, and secondly, it allows user to benefit from the incremental-configuration capabilities offered by “vtysh” shell.

Manual changes made over frr.conf file will still require the execution of **config reload** command for the changes to take effect. On the other hand, configuration changes performed through the **vtysh** shell will be immediately activated. Bear in mind that these vtysh-driven changes will not be automatically saved by FRR into frr.conf config file; for that to happen user will need to rely on FRR’s **write** command (see example below).

Finally, user should take into account that even though FRR’s configuration approach is fully decoupled from the rest of SONiC components, the execution of **config reload** instruction will also impact FRR’s applications. On the other hand, the invocation of **config save** command will have no FRR change being saved to disk, as for this purpose we will be relying on FRR’s **write** instruction.

admin@lnos-x1-a-asw02:~$ vtysh

Hello, this is FRRouting (version 3.0-rc2).

Copyright 1996-2005 Kunihiro Ishiguro, et al.

lnos-x1-a-asw02# conf t

lnos-x1-a-asw02(config)# router bgp 10

lnos-x1-a-asw02(config-router)# bgp router-id 12.12.12.12

lnos-x1-a-asw02(config-router)# end

lnos-x1-a-asw02#

lnos-x1-a-asw02# show run

Building configuration...

Current configuration:

!

frr version 3.0-rc2

frr defaults traditional

no log monitor

!

log syslog informational

!

router bgp 10

bgp router-id 12.12.12.12

no bgp default ipv4-unicast

neighbor 10.1.2.1 remote-as 10

neighbor 10.2.2.1 remote-as 10

neighbor fc00:1:2::1 remote-as 10

neighbor fc00:2:2::1 remote-as 10

!

address-family ipv4 unicast

network 9.1.1.0/24

neighbor 10.1.2.1 activate

neighbor 10.2.2.1 activate

exit-address-family

!

address-family ipv6 unicast

network fc00:9:2::/64

neighbor fc00:1:2::1 activate

neighbor fc00:2:2::1 activate

exit-address-family

!

line vty

!

end

lnos-x1-a-asw02# write

Note: this version of vtysh never writes vtysh.conf

Building Configuration...

Integrated configuration saved to /etc/frr/frr.conf

[OK]

lnos-x1-a-asw02#

# System Operation

This section focuses on describing the various operational CLI (show commands) present in SONiC ecosystem. As SONiC CLI is one of the latest subcomponents being developed, it’s natural to expect this section to evolve at a relative-fast pace, so the information displayed here will need to be revisited on a regular basis.

SONiC’s CLI heavily relies on Click [5], which is a popular python-package useful to build and enhance modular command-line-interpreters. In a nutshell, Click…

## CLI show commands

Command: sudo vtysh -c "show bgp summary"

Talk about the “?” attribute

admin@lnos-x1-a-asw02:~$ show logging -?

### show arp <>

Command displays the IP-MAC mappings that result from the execution of system’s address-resolution-process (ARP). See that currently this command only displays IPv4 bindings – IPv6 extension is in the works.

admin@lnos-x1-a-csw01:~$ show arp

Command: /usr/sbin/arp -n

Address HWtype HWaddress Flags Mask Iface

10.1.2.2 ether 00:e0:ec:3c:0a:16 C Ethernet4

172.25.11.1 ether 28:6f:7f:ba:1c:ff C eth0

10.1.1.2 ether 00:e0:ec:3c:0a:0a C Ethernet0

172.25.11.11 ether bc:16:65:b3:ce:8e C eth0

### show bgp <>

As previously discussed, routing-stack features operate slightly different in LNOS, and that also applies to routing operational commands. In this case, the instruction will not be entirely parsed by Click submodule as is the case for all the other commands.

To take advantage of FRR’s feature-rich CLI we will be sending the entire string typed by the user towards FRR’s CLI interpreter. As can be seen in the “Command” field below, this **show bgp <>** instruction is simply a wrapper that invokes **vtysh** module to carry user-provided parameters into FRR module.

admin@lnos-x1-a-asw02:~$ show bgp ?

Command: sudo vtysh -c "show bgp ?"

<cr>

A.B.C.D Network in the BGP routing table to display

A.B.C.D/M IPv4 prefix

X:X::X:X Network in the BGP routing table to display

X:X::X:X/M IPv6 prefix

attribute-info List all bgp attribute information

cidr-only Display only routes with non-natural netmasks

community Display routes matching the communities

community-info List all bgp community information

community-list Display routes matching the community-list

dampening Display detailed information about dampening

filter-list Display routes conforming to the filter-list

ipv4 Address Family

ipv6 Address Family

json JavaScript Object Notation

l2vpn Layer 2 Virtual Private Network

large-community Display routes matching the large-communities

large-community-list Display routes matching the large-community-list

memory Global BGP memory statistics

multicast Address Family modifier

neighbors Detailed information on TCP and BGP neighbor connections

nexthop BGP nexthop table

paths Path information

peer-group Detailed information on BGP peer groups

prefix-list Display routes conforming to the prefix-list

regexp Display routes matching the AS path regular expression

route-map Display routes matching the route-map

statistics BGP RIB advertisement statistics

summary Summary of BGP neighbor status

unicast Address Family modifier

update-groups Detailed info about dynamic update groups

view BGP view

views Show the defined BGP views

vpn Address Family modifier

vpnv4 Address Family

vrf BGP VRF

vrfs Show BGP VRFs

There is a cost we are incurring in by addressing FRR’s CLI through vtysh pipeline, and that is the lack of command auto-completion features. To beneficiate from this perk user would need to access FRR’s shell and type the commands there:

admin@lnos-x1-a-asw02:~$ vtysh

Hello, this is FRRouting (version 3.0-rc2).

Copyright 1996-2005 Kunihiro Ishiguro, et al.

lnos-x1-a-asw02#

### show clock

Surprise! This one displays the current time/date.

admin@lnos-x1-a-asw02:~$ show clock

Command: date

Tue Mar 20 19:05:50 UTC 2018

### show environment

This command displays platform-level information such as system’s temperature, fans speed, and voltage/current/wattage consumption.

admin@lnos-x1-a-asw02:~$ show environment

Command: sudo sensors

coretemp-isa-0000

Adapter: ISA adapter

Core 0: +20.0 C (high = +98.0 C, crit = +98.0 C)

Core 1: +20.0 C (high = +98.0 C, crit = +98.0 C)

Core 2: +20.0 C (high = +98.0 C, crit = +98.0 C)

Core 3: +20.0 C (high = +98.0 C, crit = +98.0 C)

dx010\_lm75b-i2c-5-48

Adapter: i2c-0-mux (chan\_id 3)

Rear-panel temp sensor 1: +31.6 C (high = +43.0 C, hyst = +28.0 C)

dx010\_lm75b-i2c-6-49

Adapter: i2c-0-mux (chan\_id 4)

Front-panel temp sensor 2: +34.6 C (high = +43.0 C, hyst = +28.0 C)

…

dx010\_lm75b-i2c-15-4e

Adapter: i2c-0-mux (chan\_id 5)

Rear-panel temp sensor 2: +28.5 C (high = +43.0 C, hyst = +28.0 C)

emc2305-i2c-13-2e

Adapter: i2c-0-mux (chan\_id 3)

fan1: 8124 RPM (div = 4)

fan2: 8141 RPM (div = 4)

fan3: 8124 RPM (div = 4)

fan4: 8041 RPM (div = 4)

fan5: 8124 RPM (div = 4)

emc2305-i2c-13-4d

Adapter: i2c-0-mux (chan\_id 3)

fan1: 7943 RPM (div = 4)

fan2: 8057 RPM (div = 4)

fan3: 8024 RPM (div = 4)

fan4: 7992 RPM (div = 4)

fan5: 8090 RPM (div = 4)

dps460-i2c-10-5a

Adapter: i2c-0-mux (chan\_id 0)

vin: +211.50 V (crit min = -0.50 V, min = -0.50 V)

(max = -0.50 V, crit max = -0.50 V)

vcap: -0.50 V

vout1: +12.06 V (crit min = +10.40 V, min = +128.00 V)

(max = +128.00 V, crit max = +14.20 V)

fan1: 5840 RPM

Power Supply 1 temp sensor 1: +38.0 C (low = -0.5 C, high = +119.0 C)

(crit low = -0.5 C, crit = -0.5 C)

Power Supply 1 temp sensor 2: +49.0 C (low = -0.5 C, high = +119.0 C)

(crit low = -0.5 C, crit = -0.5 C)

Power Supply 1 temp sensor 3: +43.0 C (low = -0.5 C, high = +119.0 C)

(crit low = -0.5 C, crit = -0.5 C)

pin: 140.25 W (max = 1.15 kW)

pout1: 126.62 W (max = 900.00 W, crit = 1.10 kW)

(cap = -500.00 mW)

iin: +0.69 A (max = +6.00 A, crit max = +7.00 A)

iout1: +10.62 A (crit min = -0.50 A, max = +76.00 A)

(crit max = +80.00 A)

…

### show interface <>

“show interface” stanza offers a set of commands that are quite useful for system’s daily operation. Find below an example of these commands:

admin@lnos-x1-a-asw02:~$ show interfaces ?

Usage: show interfaces [OPTIONS] COMMAND [ARGS]...

Show details of the network interfaces

Options:

-?, -h, --help Show this message and exit.

Commands:

alias Show Interface Name/Alias Mapping

counters Show interface counters

description Show interface status, protocol and...

portchannel Show PortChannel information

status Show Interface status information

summary Show interface status and information

transceiver Show SFP Transceiver information

admin@lnos-x1-a-asw02:~$ show interfaces status

Command: intfutil status

Interface Admin Oper Alias Lanes Speed MTU

----------- ------- ------ ------- ------- ------- -----

…

Ethernet16 up up Eth5/1 33 10G 9100

Ethernet17 up down Eth5/2 34 10G 9100

Ethernet112 up up Eth29/1 113,114 50G 9100

Ethernet116 up up Eth30/1 117,118 50G 9100

Ethernet120 up up Eth31/1 121,122 50G 9100

Ethernet124 up up Eth32/1 125,126 50G 9100

…

admin@lnos-x1-a-asw02:~$ show interfaces description

Command: intfutil description

Interface Admin Oper Alias Description

----------- ------- ------ ------- -------------

Ethernet0 down down Eth1/1 Description 1

Ethernet1 down down Eth1/2 Description 2

Ethernet2 down down Eth1/3 Description 3

Ethernet3 down down Eth1/4 Description 4

…

admin@lnos-x1-a-asw02:~$ show interfaces transceiver

Usage: show interfaces transceiver [OPTIONS] COMMAND [ARGS]...

Show SFP Transceiver information

Options:

-?, -h, --help Show this message and exit.

Commands:

eeprom Show interface transceiver EEPROM information

lpmode Show interface transceiver low-power mode...

presence Show interface transceiver presence

admin@lnos-x1-a-asw02:~$ show interfaces transceiver eeprom

Command: sudo sfputil show eeprom

Ethernet0: SFP EEPROM detected

Connector: Unknown

Encoding: NRZ

Extended Identifier: Unknown

Extended RateSelect Compliance: QSFP+ Rate Select Version 1

Identifier: Unknown

Length(km): 2

Nominal Bit Rate(100Mbs): 255

Specification compliance:

Vendor Date Code(YYYY-MM-DD Lot): 2016-05-20 19

Vendor Name: INNOLIGHT

Vendor OUI: 44-7c-7f

Vendor PN: TR-VC13T-N00

Vendor Rev: 01

Vendor SN: INGAU6650590

### show ip <>

“show ip” commands display IPv4 related information. A similar stanza is also offered for IPv6 state – refer to “show ipv6” section below. The following subcommands are available for IPv4 case:

admin@lnos-x1-a-asw02:~$ show ip ?

Usage: show ip [OPTIONS] COMMAND [ARGS]...

Show IP (IPv4) commands

Options:

-?, -h, --help Show this message and exit.

Commands:

interfaces Show interfaces IPv4 address

protocol Show IPv4 protocol information

route Show IP (IPv4) routing table

admin@lnos-x1-a-asw02:~$ show ip interfaces

Interface IPv4 address/mask Admin/Oper

----------- ------------------- ------------

Ethernet112 10.1.2.2/24 up/up

Ethernet116 10.2.2.2/24 up/up

Ethernet120 10.3.2.2/24 up/up

Ethernet124 10.4.2.2/24 up/up

Vlan100 172.18.1.1/24 up/up

Vlan777 12.10.1.1/24 up/up

…

admin@lnos-x1-a-asw02:~$ show ip route

Command: sudo vtysh -c "show ip route"

Codes: K - kernel route, C - connected, S - static, R - RIP,

O - OSPF, I - IS-IS, B - BGP, P - PIM, N - NHRP, T - Table,

v - VNC, V - VNC-Direct,

> - selected route, \* - FIB route

K>\* 0.0.0.0/0 via 172.25.11.1, eth0

B>\* 10.1.1.0/24 [20/0] via 10.1.2.1, Ethernet112, 14:32:48

C>\* 10.1.2.0/24 is directly connected, Ethernet112

B>\* 10.2.1.0/24 [20/0] via 10.2.2.1, Ethernet116, 14:32:44

C>\* 10.2.2.0/24 is directly connected, Ethernet116

B>\* 10.3.1.0/24 [20/0] via 10.3.2.1, Ethernet120, 14:32:47

C>\* 10.3.2.0/24 is directly connected, Ethernet120

B>\* 10.4.1.0/24 [20/0] via 10.4.2.1, Ethernet124, 14:32:47

C>\* 10.4.2.0/24 is directly connected, Ethernet124

B>\* 10.10.1.1/32 [20/0] via 10.1.2.1, Ethernet112, 14:32:44

\* via 10.2.2.1, Ethernet116, 14:32:44

\* via 10.3.2.1, Ethernet120, 14:32:44

\* via 10.4.2.1, Ethernet124, 14:32:44

C>\* 10.10.1.2/32 is directly connected, lo

### show ipv6 <>

Same as above, but this time we are displaying IPv6 state.

admin@lnos-x1-a-asw02:~$ show ipv6

Usage: show ipv6 [OPTIONS] COMMAND [ARGS]...

Show IPv6 commands

Options:

-?, -h, --help Show this message and exit.

Commands:

interfaces Show interfaces IPv6 address

protocol Show IPv6 protocol information

route Show IPv6 routing table

admin@lnos-x1-a-asw02:~$ show ipv6 interfaces

Interface IPv6 address/mask Admin/Oper

----------- -------------------------------------- ------------

Bridge fe80::2452:c5ff:fe1b:7df5%Bridge/64 up/up

Ethernet16 fe80::2e0:ecff:fe3c:a16%Ethernet16/64 up/up

Ethernet112 fc00:1:2::2/64 up/up

fe80::2e0:ecff:fe3c:a16%Ethernet112/64

Ethernet116 fc00:2:2::2/64 up/up

fe80::2e0:ecff:fe3c:a16%Ethernet116/64

Ethernet120 fc00:3:2::2/64 up/up

fe80::2e0:ecff:fe3c:a16%Ethernet120/64

Ethernet124 fc00:4:2::2/64 up/up

fe80::2e0:ecff:fe3c:a16%Ethernet124/64

Vlan100 3100::1/64 up/up

fe80::2e0:ecff:fe3c:a16%Vlan100/64

Vlan777 3777::1/64 up/up

admin@lnos-x1-a-asw02:~$ show ipv6 route

Command: sudo vtysh -c "show ipv6 route"

Codes: K - kernel route, C - connected, S - static, R - RIPng,

O - OSPFv3, I - IS-IS, B - BGP, N - NHRP, T - Table,

v - VNC, V - VNC-Direct,

> - selected route, \* - FIB route

B>\* 2100::/64 [20/0] via fe80::2e0:ecff:fe3b:d835, Ethernet112, 14:38:50

\* via fe80::2e0:ecff:fe3b:da41, Ethernet116, 14:38:50

\* via fe80::2e0:ecff:fe3b:d9be, Ethernet120, 14:38:50

\* via fe80::2e0:ecff:fe3b:d8b8, Ethernet124, 14:38:50

B>\* 2777::/64 [20/0] via fe80::2e0:ecff:fe3b:d835, Ethernet112, 14:38:50

\* via fe80::2e0:ecff:fe3b:da41, Ethernet116, 14:38:50

\* via fe80::2e0:ecff:fe3b:d9be, Ethernet120, 14:38:50

\* via fe80::2e0:ecff:fe3b:d8b8, Ethernet124, 14:38:50

C>\* 3100::/64 is directly connected, Vlan100

C>\* 3777::/64 is directly connected, Vlan777

B>\* fc00:1:1::/64 [20/0] via fe80::2e0:ecff:fe3b:d835, Ethernet112, 14:40:50

C>\* fc00:1:2::/64 is directly connected, Ethernet112

B>\* fc00:2:1::/64 [20/0] via fe80::2e0:ecff:fe3b:da41, Ethernet116, 14:38:50

C>\* fc00:2:2::/64 is directly connected, Ethernet116

### show lldp <>

“show lldp” stanza provides user with information about lldp’s current operation. For a summarized output we can rely on “show lldp table”; for more details we can make use of “show lldp neighbor” command.

admin@lnos-x1-a-asw02:~$ show lldp

Usage: show lldp [OPTIONS] COMMAND [ARGS]...

LLDP (Link Layer Discovery Protocol) information

Options:

-?, -h, --help Show this message and exit.

Commands:

neighbors Show LLDP neighbors

table Show LLDP neighbors in tabular format

admin@lnos-x1-a-asw02:~$ show lldp table

Command: sudo lldpshow

Capability codes: (R) Router, (B) Bridge, (O) Other

LocalPort RemoteDevice RemotePortID Capability RemotePortDescr

----------- --------------- ----------------- ------------ -----------------

Ethernet16 04:62:73:f7:e5:ce

Ethernet112 lnos-x1-a-csw01 00:e0:ec:3b:d8:35 BR Ethernet4

Ethernet116 lnos-x1-a-csw02 00:e0:ec:3b:da:41 BR Ethernet4

Ethernet120 lnos-x1-a-csw03 00:e0:ec:3b:d9:be BR Ethernet4

Ethernet124 lnos-x1-a-csw04 00:e0:ec:3b:d8:b8 BR Ethernet4

--------------------------------------------------

Total entries displayed: 5

admin@lnos-x1-a-asw02:~$ show lldp neighbors

Command: sudo lldpctl

-------------------------------------------------------------------------------

LLDP neighbors:

-------------------------------------------------------------------------------

Interface: Ethernet112, via: LLDP, RID: 2, Time: 0 day, 14:54:57

Chassis:

ChassisID: mac 00:e0:ec:3b:d8:35

SysName: lnos-x1-a-csw01

SysDescr: Debian GNU/Linux 8 (jessie) Linux 3.16.0-4-amd64 #1 SMP Debian 3.16.43-2+deb8u5 (2015-12-19) x86\_64

TTL: 120

MgmtIP: 10.10.2.1

MgmtIP: fc00:1:1::1

Capability: Bridge, on

Capability: Router, on

Capability: Wlan, off

Capability: Station, off

Port:

PortID: mac 00:e0:ec:3b:d8:35

PortDescr: Ethernet4

### show logging <>

As expected this CLI stanza takes care of displaying logging information in chronological order. There are a few interest options to be aware of: “-l <num>” option displays the last few “num” lines dumped by syslog agents, where as “-f” behaves as a regular “tail -f” command.

If no option is specified, this command will display the content of the last two syslog files being collected: the current file (/var/log/syslog) and the previous one (/var/log/syslog.1).

admin@lnos-x1-a-asw02:~$ show logging -?

Usage: show logging [OPTIONS] [PROCESS]

Show system log

Options:

-l, --lines TEXT

-f, --follow

-?, -h, --help Show this message and exit.

admin@lnos-x1-a-asw02:~$ show logging -l 5

Command: sudo cat /var/log/syslog.1 /var/log/syslog | tail -5

Mar 20 20:35:37.388719 lnos-x1-a-asw02 INFO supervisord 2018-03-20 20:35:36,589 INFO exited: arp\_update (exit status 0; expected)

Mar 20 20:35:37.594289 lnos-x1-a-asw02 INFO swss.sh[18129]: 2018-03-20 20:35:37,593 INFO spawned: 'arp\_update' with pid 3030

Mar 20 20:35:38.596299 lnos-x1-a-asw02 INFO swss.sh[18129]: 2018-03-20 20:35:38,595 INFO success: arp\_update entered RUNNING state, process has stayed up for > than 1 seconds (startsecs)

Mar 20 20:35:40.944634 lnos-x1-a-asw02 WARNING snmp-subagent [sonic\_ax\_impl] WARNING: 0 - b'LLDP\_ENTRY\_TABLE' is empty. No LLDP information could be retrieved.

Mar 20 20:35:43.202741 lnos-x1-a-asw02 NOTICE ZTP: Retrying to get the Bootstrap script URL...

### show mac <>

“show mac” displays L2 information related to the system’s forwarding state, specifically the MAC entries learned through each port interface.

admin@lnos-x1-a-asw02:~$ show mac -?

Usage: show mac [OPTIONS]

Show MAC (FDB) entries

Options:

-v, --vlan TEXT

-p, --port TEXT

-?, -h, --help Show this message and exit.

admin@lnos-x1-a-asw02:~$ show mac

Command: fdbshow

No. Vlan MacAddress Port

----- ------ ----------------- ----------

1 100 04:62:73:F7:E5:CE Ethernet16

2 100 04:62:73:F7:E5:D3 Ethernet16

Total number of entries 2

### show ntp

Just a thin wrapper over “ntpq” linux command. This command provides the list of ntp peers with whom the local ntp server interacts with.

admin@lnos-x1-a-asw02:~$ show ntp

Command: ntpq -p

remote refid st t when poll reach delay offset jitter

==============================================================================

+atl0.fairy.matt 128.59.0.245 2 u 590 1024 377 81.606 0.575 2.007

+85.254.217.3 131.188.3.220 2 u 1034 1024 377 182.748 0.779 1.538

\*resolver1.skyfi 216.218.192.202 2 u 711 1024 337 10.328 1.349 0.370

-awesome.bytesta 243.50.127.182 2 u 504 1024 377 40.857 -1.466 1.244

### show platform <>

The information displayed in this set of commands partially overlaps with the one generated by “show envinronment” instruction. In this case though, the information is presented in a more succinct fashion. In the future these two CLI stanzas may end up getting combined.

admin@lnos-x1-a-asw02:~$ show platform

Usage: show platform [OPTIONS] COMMAND [ARGS]...

Show platform-specific hardware info

Options:

-?, -h, --help Show this message and exit.

Commands:

psustatus Show PSU status information

summary Show hardware platform information

syseeprom Show system EEPROM information

admin@lnos-x1-a-asw02:~$ show platform summary

Platform: x86\_64-cel\_seastone-r0

HwSKU: Seastone-DX010-10-50

ASIC: broadcom

admin@lnos-x1-a-asw02:~$ show platform syseeprom

Command: sudo decode-syseeprom

TlvInfo Header:

Id String: TlvInfo

Version: 1

Total Length: 191

TLV Name Code Len Value

-------------------- ---- --- -----

Platform Name 0x28 8 RANGELEY

Manufacturer 0x2B 9 CELESTICA

Product Name 0x21 14 Seastone-DX010

Part Number 0x22 14 R0872-F0010-01

Serial Number 0x23 22 DX010B2F016304LI000023

Base MAC Address 0x24 6 00:E0:EC:3C:0A:16

Manufacture Date 0x25 19 03/25/2016 09:10:25

Device Version 0x26 1 1

Label Revision 0x27 9 DX010 B2F

ONIE Version 0x29 22 Seastone 2014.08.0.0.4

MAC Addresses 0x2A 2 4

Manufacture Country 0x2C 3 CHN

Vendor Name 0x2D 14 Seastone-DX010

Diag Version 0x2E 5 1.0.5

Service Tag 0x2F 2 LB

Vendor Extension 0xFD 3

CRC-32 0xFE 4 0x88FA0DF9

(checksum valid)

### show processes <>

“show processes” commands provide a wrapper over linux’s “top” command. “show process cpu” sorts the processes being displayed by cpu-utilization, whereas “show process memory” does it attending to processes’ memory-utilization.

admin@lnos-x1-a-asw02:~$ show processes ?

Usage: show processes [OPTIONS] COMMAND [ARGS]...

Display process information

Options:

-?, -h, --help Show this message and exit.

Commands:

cpu Show processes CPU info

memory Show processes memory info

summary Show processes info

admin@lnos-x1-a-asw02:~$ show processes cpu | head -10

Command: top -bn 1 -o %CPU

top - 22:30:09 up 1 day, 4:34, 6 users, load average: 0.74, 0.58, 0.45

Tasks: 187 total, 2 running, 185 sleeping, 0 stopped, 0 zombie

%Cpu(s): 5.6 us, 2.7 sy, 0.0 ni, 91.6 id, 0.1 wa, 0.0 hi, 0.1 si, 0.0 st

KiB Mem: 4041448 total, 2080756 used, 1960692 free, 328564 buffers

KiB Swap: 0 total, 0 used, 0 free. 809928 cached Mem

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND

18299 root 20 0 944636 173000 41408 S 68.6 4.3 233:25.28 syncd

1159 root 20 0 55648 10164 2924 R 56.2 0.3 157:42.77 redis-server

admin@lnos-x1-a-asw02:~$ show processes memory | head -10

Command: top -bn 1 -o %MEM

top - 22:30:33 up 1 day, 4:35, 6 users, load average: 0.67, 0.57, 0.45

Tasks: 187 total, 1 running, 186 sleeping, 0 stopped, 0 zombie

%Cpu(s): 5.6 us, 2.7 sy, 0.0 ni, 91.6 id, 0.1 wa, 0.0 hi, 0.1 si, 0.0 st

KiB Mem: 4041448 total, 2080076 used, 1961372 free, 328568 buffers

KiB Swap: 0 total, 0 used, 0 free. 809932 cached Mem

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND

18378 root 20 0 718692 191996 5940 S 0.0 4.8 0:01.87 syncd

18299 root 20 0 944636 172984 41408 S 49.9 4.3 233:30.77 syncd

### show services

This command displays the state of all the SONiC processes running inside a docker container. This helps to identify the status of SONiC’s critical processes.

admin@lnos-x1-a-asw02:~$ show services

dhcp\_relay docker

---------------------------

UID PID PPID C STIME TTY TIME CMD

root 1 0 0 05:26 ? 00:00:12 /usr/bin/python /usr/bin/supervi

root 24 1 0 05:26 ? 00:00:00 /usr/sbin/rsyslogd -n

snmp docker

---------------------------

UID PID PPID C STIME TTY TIME CMD

root 1 0 0 05:26 ? 00:00:16 /usr/bin/python /usr/bin/supervi

root 24 1 0 05:26 ? 00:00:02 /usr/sbin/rsyslogd -n

Debian-+ 29 1 0 05:26 ? 00:00:04 /usr/sbin/snmpd -f -LS4d -u Debi

root 31 1 1 05:26 ? 00:15:10 python3.6 -m sonic\_ax\_impl

syncd docker

---------------------------

UID PID PPID C STIME TTY TIME CMD

root 1 0 0 05:26 ? 00:00:13 /usr/bin/python /usr/bin/supervi

root 12 1 0 05:26 ? 00:00:00 /usr/sbin/rsyslogd -n

root 17 1 0 05:26 ? 00:00:00 /usr/bin/dsserve /usr/bin/syncd

root 27 17 22 05:26 ? 04:09:30 /usr/bin/syncd --diag -p /usr/sh

root 51 27 0 05:26 ? 00:00:01 /usr/bin/syncd --diag -p /usr/sh

swss docker

---------------------------

UID PID PPID C STIME TTY TIME CMD

root 1 0 0 05:26 ? 00:00:29 /usr/bin/python /usr/bin/supervi

root 25 1 0 05:26 ? 00:00:00 /usr/sbin/rsyslogd -n

root 30 1 0 05:26 ? 00:00:13 /usr/bin/orchagent -d /var/log/s

root 42 1 1 05:26 ? 00:12:40 /usr/bin/portsyncd -p /usr/share

root 45 1 0 05:26 ? 00:00:00 /usr/bin/intfsyncd

root 48 1 0 05:26 ? 00:00:03 /usr/bin/neighsyncd

root 59 1 0 05:26 ? 00:00:01 /usr/bin/vlanmgrd

root 92 1 0 05:26 ? 00:00:01 /usr/bin/intfmgrd

root 3606 1 0 23:36 ? 00:00:00 bash -c /usr/bin/arp\_update; sle

root 3621 3606 0 23:36 ? 00:00:00 sleep 300

…

### show system-memory

Displays system-wide memory utilization information – just a wrapper over linux native “free” command.

admin@lnos-x1-a-asw02:~$ show system-memory

Command: free -m -h

total used free shared buffers cached

Mem: 3.9G 2.0G 1.8G 33M 324M 791M

-/+ buffers/cache: 951M 2.9G

Swap: 0B 0B 0B

### show techsupport

For troubleshooting and debugging purposes, this command collects a considerable amount of information across the entire system. Information as diverse as syslog entries, database state, routing-stack state, etc.

The collected information is compressed in a single file and placed in this path: /var/dump/tmp/sonic\_dump\_<hostname>.tar.gz

admin@lnos-x1-a-asw02:~$ show techsupport

Command: sudo generate\_dump -v

main

whoami

vtysh -c "show ip bgp neighbors" | grep "BGP neighbor is" | awk -F '[, ]' '{print $4}'

/usr/local/bin/sonic-cfggen -v platform

find -L /var/log -type f

…

admin@lnos-x1-a-asw02:~$ ls -lrt /var/dump/sonic\_dump\_lnos-x1-a-asw02\_20180320\_235347.tar.gz

-rw-r--r-- 1 root root 2756808 Mar 20 23:54 /var/dump/sonic\_dump\_lnos-x1-a-asw02\_20180320\_235347.tar.gz

### show version

This command displays relevant information as the SONiC and Linux kernel version being utilized, as well as the commit-id used to build the SONiC image. The second section of the output displays the various docker images and their associated id’s.

admin@lnos-x1-a-csw01:~$ show version

SONiC Software Version: SONiC.master.0-dirty-20180301.084517

Distribution: Debian 8.10

Kernel: 3.16.0-4-amd64

Build commit: 1d24e78

Build date: Thu Mar 1 09:02:59 UTC 2018

Built by: rmolina@server04

Docker images:

REPOSITORY TAG IMAGE ID SIZE

docker-syncd-brcm latest 444b31cbd5e3 346 MB

docker-syncd-brcm master.0-dirty-20180301.084517 444b31cbd5e3 346 MB

docker-orchagent-brcm latest 930841ed1898 287.1 MB

docker-orchagent-brcm master.0-dirty-20180301.084517 930841ed1898 287.1 MB

docker-dhcp-relay latest dfe0bbd254d4 280.6 MB

docker-dhcp-relay master.0-dirty-20180301.084517 dfe0bbd254d4 280.6 MB

docker-database latest eb4faaa6993f 279.3 MB

docker-database master.0-dirty-20180301.084517 eb4faaa6993f 279.3 MB

docker-teamd latest 0569238531d4 284.2 MB

docker-teamd master.0-dirty-20180301.084517 0569238531d4 284.2 MB

docker-snmp-sv2 latest 36724ae269ab 319.1 MB

docker-snmp-sv2 master.0-dirty-20180301.084517 36724ae269ab 319.1 MB

docker-router-advertiser latest 6e2a3fcfe5f2 276.9 MB

docker-router-advertiser master.0-dirty-20180301.084517 6e2a3fcfe5f2 276.9 MB

docker-platform-monitor latest 6268105b364b 298.9 MB

docker-platform-monitor master.0-dirty-20180301.084517 6268105b364b 298.9 MB

docker-lldp-sv2 latest c5b2d17f8742 284.1 MB

docker-lldp-sv2 master.0-dirty-20180301.084517 c5b2d17f8742 284.1 MB

docker-fpm-frr latest 882126b8f45c 289.3 MB

docker-fpm-frr master.0-dirty-20180301.084517 882126b8f45c 289.3 MB

### show uptime

As expected, here we are just displaying the system uptime.

admin@lnos-x1-a-asw02:~$ show uptime

Command: uptime -p

up 1 day, 8 hours, 15 minutes

### show users

A simple wrapper over “who” linux native command – displays current users logged in the system.

admin@lnos-x1-a-asw02:~$ show users

Command: who

admin ttyS0 Mar 19 17:56

admin pts/10 Mar 19 18:07 (172.29.120.222:S.0)

admin pts/11 Mar 19 18:07 (172.29.120.222:S.1)

admin pts/12 Mar 19 18:18 (172.29.120.222:S.2)

### show vlan <>

“show vlan” commands display VLAN specific information such as the port members to which each VLAN associates with, the mac-address learned through each vlan interface, etc.

admin@lnos-x1-a-asw02:~$ show vlan

Usage: show vlan [OPTIONS] COMMAND [ARGS]...

Show VLAN information

Options:

-?, -h, --help Show this message and exit.

Commands:

brief Show all bridge information

config

id Show list of learned MAC addresses for...

admin@lnos-x1-a-asw02:~$ show vlan brief

Command: sudo brctl show

bridge name bridge id STP enabled interfaces

Bridge 8000.00e0ec3c0a16 no Ethernet16

Ethernet17

docker0 8000.0242c6c11673 no

admin@lnos-x1-a-asw02:~$ show vlan id Bridge

Command: sudo brctl showmacs Bridge

port no mac addr is local? ageing timer

1 00:e0:ec:3c:0a:16 yes 0.00

1 04:62:73:f7:e5:ce no 1.22

1 04:62:73:f7:e5:d3 no 2.29

## CLI debug / undebug commands

Aside from the multiple debugging/troubleshooting tools offered by Linux ecosystem, which are easily installable in SONiC through the typical apt-get instructions cycle, SONiC also offers debugging tools to aid user in the daily troubleshooting tasks.

In particular, we are referring to the debugging features offered by FRR’s routing-stack for control-plane troubleshooting. User can activate this feature by running the commands displayed below, and then refer to the global syslog file (/var/log/syslog) to visualize the generated information:

admin@lnos-x1-a-asw02:~$ debug ?

Usage: debug [OPTIONS] COMMAND [ARGS]...

SONiC command line - 'debug' command

Options:

-?, -h, --help Show this message and exit.

Commands:

bgp debug bgp events

disable disable debugging for routing events

enable enable debugging for routing events

zebra debug bgp events

To allow debugging information to get dumped to /var/log/syslog:

admin@lnos-x1-a-asw02:~$ debug enable

Command: sudo vtysh -c "configure terminal" -c "log syslog debugging"

To disable debugging state being written to /var/log/syslog:

admin@lnos-x1-a-asw02:~$ debug disable

Command: sudo vtysh -c "configure terminal" -c "no log syslog debugging"

To deactivate specific debugging items we can also make use of the “undebug” commands:

admin@lnos-x1-a-asw02:~$ undebug bgp ?

Usage: undebug bgp [OPTIONS] COMMAND [ARGS]...

undebug bgp events

Options:

-?, -h, --help Show this message and exit.

Commands:

as4 undebug bgp AS4 actions

bestpath undebug bgp bestpath

keepalives undebug bgp keepalives

neighborevents undebug bgp neighbor events

nht undebug bgp nexthop tracking events

updategroups undebug bgp update-group events

updates undebug bgp updates

zebra undebug bgp zebra messages

admin@lnos-x1-a-asw02:~$ undebug bgp keepalives

Command: sudo vtysh -c "no debug bgp keepalives"

BGP keepalives debugging is off

## CLI clear commands

The “clear” CLI stanza is utilized for various scenarios such as clearing interface counters, eliminating ARP entries, bringing down bgp sessions, etc.

admin@lnos-x1-a-asw02:~$ clear ?

Usage: sonic-clear [OPTIONS] COMMAND [ARGS]...

SONiC command line - 'Clear' command

Options:

-?, -h, --help Show this message and exit.

Commands:

arp Clear IPv4 ARP table

bgp BGP information

counters Clear counters

ip Clear IPv4 information

ipv6 Clear IPv6 information

# System Upgrade/Downgrade Process

TBD

# System Troubleshooting

## Logging Information (swssloglevel)

## Databases state (redis-cli)

# Annex

## Example config\_db.json file

admin@lnos-x1-a-asw02:~$ more /etc/sonic/config\_db.json

{

"onie\_arch": "x86\_64",

"VLAN": {

"Vlan100": {

"vlanid": "100",

"admin\_status": "up",

"description": "Data Traffic",

"members": [

"Ethernet16",

"Ethernet17"

],

"mtu": "9100"

},

"Vlan777": {

"vlanid": "777",

"admin\_status": "up",

"description": "IPMI",

"members": [

"Ethernet16",

"Ethernet17"

],

"mtu": "9100"

}

},

"onie\_platform": "x86\_64-cel\_seastone-r0",

"onie\_machine\_rev": "0",

"VLAN\_INTERFACE": {

"Vlan100|172.18.1.1/24": {

"scope": "global",

"family": "IPv4"

},

"Vlan100|3100::1/64": {

"scope": "global",

"family": "IPv6"

},

"Vlan777|12.10.1.1/24": {

"scope": "global",

"family": "IPv4"

},

"Vlan777|3777::1/64": {

"scope": "global",

"family": "IPv6"

}

},

"onie\_version": "2014.08.0.0.4",

"DEVICE\_METADATA": {

"localhost": {

"hwsku": "Seastone-DX010-10-50",

"hostname": "sonic",

"mac": "00:e0:ec:3c:0a:16",

"bgp\_asn": "None",

"deployment\_id": "None",

"type": "LeafRouter"

}

},

"onie\_machine": "cel\_seastone",

"onie\_config\_version": "1",

"DEVICE\_NEIGHBOR": {

"Ethernet4": {

"name": "ixia-card2-port8",

"port": "Eth1/2/8"

},

"Ethernet112": {

"name": "lnos-x1-a-csw01",

"port": "Eth29/1"

},

"Ethernet116": {

"name": "lnos-x1-a-csw02",

"port": "Eth30/1"

},

"Ethernet120": {

"name": "lnos-x1-a-csw03",

"port": "Eth31/1"

},

"Ethernet124": {

"name": "lnos-x1-a-csw04",

"port": "Eth32/1"

}

},

"PORT": {

"Ethernet0": {

"alias": "Eth1/1",

"lanes": "65",

"speed": "10000"

},

"Ethernet1": {

"alias": "Eth1/2",

"lanes": "66",

"speed": "10000"

},

"Ethernet2": {

"alias": "Eth1/3",

"lanes": "67",

"speed": "10000"

},

"Ethernet3": {

"alias": "Eth1/4",

"lanes": "68",

"speed": "10000"

},

"Ethernet4": {

"alias": "Eth2/1",

"lanes": "69",

"speed": "10000"

},

"Ethernet5": {

"alias": "Eth2/2",

"lanes": "70",

"speed": "10000"

},

"Ethernet6": {

"alias": "Eth2/3",

"lanes": "71",

"speed": "10000"

},

"Ethernet7": {

"alias": "Eth2/4",

"lanes": "72",

"speed": "10000"

},

"Ethernet8": {

"alias": "Eth3/1",

"lanes": "73",

"speed": "10000"

},

"Ethernet9": {

"alias": "Eth3/2",

"lanes": "74",

"speed": "10000"

},

"Ethernet10": {

"alias": "Eth3/3",

"lanes": "75",

"speed": "10000"

},

"Ethernet11": {

"alias": "Eth3/4",

"lanes": "76",

"speed": "10000"

},

"Ethernet12": {

"alias": "Eth4/1",

"lanes": "77",

"speed": "10000"

},

"Ethernet13": {

"alias": "Eth4/2",

"lanes": "78",

"speed": "10000"

},

"Ethernet14": {

"alias": "Eth4/3",

"lanes": "79",

"speed": "10000"

},

"Ethernet15": {

"alias": "Eth4/4",

"lanes": "80",

"speed": "10000"

},

"Ethernet16": {

"alias": "Eth5/1",

"lanes": "33",

"speed": "10000"

},

"Ethernet17": {

"alias": "Eth5/2",

"lanes": "34",

"speed": "10000"

},

"Ethernet18": {

"alias": "Eth5/3",

"lanes": "35",

"speed": "10000"

},

"Ethernet19": {

"alias": "Eth5/4",

"lanes": "36",

"speed": "10000"

},

"Ethernet20": {

"alias": "Eth6/1",

"lanes": "37",

"speed": "10000"

},

"Ethernet21": {

"alias": "Eth6/2",

"lanes": "38",

"speed": "10000"

},

"Ethernet22": {

"alias": "Eth6/3",

"lanes": "39",

"speed": "10000"

},

"Ethernet23": {

"alias": "Eth6/4",

"lanes": "40",

"speed": "10000"

},

"Ethernet24": {

"alias": "Eth7/1",

"lanes": "41",

"speed": "10000"

},

"Ethernet25": {

"alias": "Eth7/2",

"lanes": "42",

"speed": "10000"

},

"Ethernet26": {

"alias": "Eth7/3",

"lanes": "43",

"speed": "10000"

},

"Ethernet27": {

"alias": "Eth7/4",

"lanes": "44",

"speed": "10000"

},

"Ethernet28": {

"alias": "Eth8/1",

"lanes": "45",

"speed": "10000"

},

"Ethernet29": {

"alias": "Eth8/2",

"lanes": "46",

"speed": "10000"

},

"Ethernet30": {

"alias": "Eth8/3",

"lanes": "47",

"speed": "10000"

},

"Ethernet31": {

"alias": "Eth8/4",

"lanes": "48",

"speed": "10000"

},

"Ethernet32": {

"alias": "Eth9/1",

"lanes": "49",

"speed": "10000"

},

"Ethernet33": {

"alias": "Eth9/2",

"lanes": "50",

"speed": "10000"

},

"Ethernet34": {

"alias": "Eth9/3",

"lanes": "51",

"speed": "10000"

},

"Ethernet35": {

"alias": "Eth9/4",

"lanes": "52",

"speed": "10000"

},

"Ethernet36": {

"alias": "Eth10/1",

"lanes": "53",

"speed": "10000"

},

"Ethernet37": {

"alias": "Eth10/2",

"lanes": "54",

"speed": "10000"

},

"Ethernet38": {

"alias": "Eth10/3",

"lanes": "55",

"speed": "10000"

},

"Ethernet39": {

"alias": "Eth10/4",

"lanes": "56",

"speed": "10000"

},

"Ethernet40": {

"alias": "Eth11/1",

"lanes": "57",

"speed": "10000"

},

"Ethernet41": {

"alias": "Eth11/2",

"lanes": "58",

"speed": "10000"

},

"Ethernet42": {

"alias": "Eth11/3",

"lanes": "59",

"speed": "10000"

},

"Ethernet43": {

"alias": "Eth11/4",

"lanes": "60",

"speed": "10000"

},

"Ethernet44": {

"alias": "Eth12/1",

"lanes": "61",

"speed": "10000"

},

"Ethernet45": {

"alias": "Eth12/2",

"lanes": "62",

"speed": "10000"

},

"Ethernet46": {

"alias": "Eth12/3",

"lanes": "63",

"speed": "10000"

},

"Ethernet47": {

"alias": "Eth12/4",

"lanes": "64",

"speed": "10000"

},

"Ethernet48": {

"alias": "Eth13/1",

"lanes": "81",

"speed": "10000"

},

"Ethernet49": {

"alias": "Eth13/2",

"lanes": "82",

"speed": "10000"

},

"Ethernet50": {

"alias": "Eth13/3",

"lanes": "83",

"speed": "10000"

},

"Ethernet51": {

"alias": "Eth13/4",

"lanes": "84",

"speed": "10000"

},

"Ethernet52": {

"alias": "Eth14/1",

"lanes": "85",

"speed": "10000"

},

"Ethernet53": {

"alias": "Eth14/2",

"lanes": "86",

"speed": "10000"

},

"Ethernet54": {

"alias": "Eth14/3",

"lanes": "87",

"speed": "10000"

},

"Ethernet55": {

"alias": "Eth14/4",

"lanes": "88",

"speed": "10000"

},

"Ethernet56": {

"alias": "Eth15/1",

"lanes": "89",

"speed": "10000"

},

"Ethernet57": {

"alias": "Eth15/2",

"lanes": "90",

"speed": "10000"

},

"Ethernet58": {

"alias": "Eth15/3",

"lanes": "91",

"speed": "10000"

},

"Ethernet59": {

"alias": "Eth15/4",

"lanes": "92",

"speed": "10000"

},

"Ethernet60": {

"alias": "Eth16/1",

"lanes": "93",

"speed": "10000"

},

"Ethernet61": {

"alias": "Eth16/2",

"lanes": "94",

"speed": "10000"

},

"Ethernet62": {

"alias": "Eth16/3",

"lanes": "95",

"speed": "10000"

},

"Ethernet63": {

"alias": "Eth16/4",

"lanes": "96",

"speed": "10000"

},

"Ethernet64": {

"alias": "Eth17/1",

"lanes": "97",

"speed": "10000"

},

"Ethernet65": {

"alias": "Eth17/2",

"lanes": "98",

"speed": "10000"

},

"Ethernet66": {

"alias": "Eth17/3",

"lanes": "99",

"speed": "10000"

},

"Ethernet67": {

"alias": "Eth17/4",

"lanes": "100",

"speed": "10000"

},

"Ethernet68": {

"alias": "Eth18/1",

"lanes": "101",

"speed": "10000"

},

"Ethernet69": {

"alias": "Eth18/2",

"lanes": "102",

"speed": "10000"

},

"Ethernet70": {

"alias": "Eth18/3",

"lanes": "103",

"speed": "10000"

},

"Ethernet71": {

"alias": "Eth18/4",

"lanes": "104",

"speed": "10000"

},

"Ethernet72": {

"alias": "Eth19/1",

"lanes": "105",

"speed": "10000"

},

"Ethernet73": {

"alias": "Eth19/2",

"lanes": "106",

"speed": "10000"

},

"Ethernet74": {

"alias": "Eth19/3",

"lanes": "107",

"speed": "10000"

},

"Ethernet75": {

"alias": "Eth19/4",

"lanes": "108",

"speed": "10000"

},

"Ethernet76": {

"alias": "Eth20/1",

"lanes": "109",

"speed": "10000"

},

"Ethernet77": {

"alias": "Eth20/2",

"lanes": "110",

"speed": "10000"

},

"Ethernet78": {

"alias": "Eth20/3",

"lanes": "111",

"speed": "10000"

},

"Ethernet79": {

"alias": "Eth20/4",

"lanes": "112",

"speed": "10000"

},

"Ethernet80": {

"alias": "Eth21/1",

"lanes": "1",

"speed": "10000"

},

"Ethernet81": {

"alias": "Eth21/2",

"lanes": "2",

"speed": "10000"

},

"Ethernet82": {

"alias": "Eth21/3",

"lanes": "3",

"speed": "10000"

},

"Ethernet83": {

"alias": "Eth21/4",

"lanes": "4",

"speed": "10000"

},

"Ethernet84": {

"alias": "Eth22/1",

"lanes": "5",

"speed": "10000"

},

"Ethernet85": {

"alias": "Eth22/2",

"lanes": "6",

"speed": "10000"

},

"Ethernet86": {

"alias": "Eth22/3",

"lanes": "7",

"speed": "10000"

},

"Ethernet87": {

"alias": "Eth22/4",

"lanes": "8",

"speed": "10000"

},

"Ethernet88": {

"alias": "Eth23/1",

"lanes": "9",

"speed": "10000"

},

"Ethernet89": {

"alias": "Eth23/2",

"lanes": "10",

"speed": "10000"

},

"Ethernet90": {

"alias": "Eth23/3",

"lanes": "11",

"speed": "10000"

},

"Ethernet91": {

"alias": "Eth23/4",

"lanes": "12",

"speed": "10000"

},

"Ethernet92": {

"alias": "Eth24/1",

"lanes": "13",

"speed": "10000"

},

"Ethernet93": {

"alias": "Eth24/2",

"lanes": "14",

"speed": "10000"

},

"Ethernet94": {

"alias": "Eth24/3",

"lanes": "15",

"speed": "10000"

},

"Ethernet95": {

"alias": "Eth24/4",

"lanes": "16",

"speed": "10000"

},

"Ethernet96": {

"alias": "Eth25/1",

"lanes": "17,18",

"speed": "50000"

},

"Ethernet98": {

"alias": "Eth25/2",

"lanes": "19,20",

"speed": "50000"

},

"Ethernet100": {

"alias": "Eth26/1",

"lanes": "21,22",

"speed": "50000"

},

"Ethernet102": {

"alias": "Eth26/2",

"lanes": "23,24",

"speed": "50000"

},

"Ethernet104": {

"alias": "Eth27/1",

"lanes": "25,26",

"speed": "50000"

},

"Ethernet106": {

"alias": "Eth27/2",

"lanes": "27,28",

"speed": "50000"

},

"Ethernet108": {

"alias": "Eth28/1",

"lanes": "29,30",

"speed": "50000"

},

"Ethernet110": {

"alias": "Eth28/2",

"lanes": "31,32",

"speed": "50000"

},

"Ethernet112": {

"alias": "Eth29/1",

"lanes": "113,114",

"speed": "50000"

},

"Ethernet114": {

"alias": "Eth29/2",

"lanes": "115,116",

"speed": "50000"

},

"Ethernet116": {

"alias": "Eth30/1",

"lanes": "117,118",

"speed": "50000"

},

"Ethernet118": {

"alias": "Eth30/2",

"lanes": "119,120",

"speed": "50000"

},

"Ethernet120": {

"alias": "Eth31/1",

"lanes": "121,122",

"speed": "50000"

},

"Ethernet122": {

"alias": "Eth31/2",

"lanes": "123,124",

"speed": "50000"

},

"Ethernet124": {

"alias": "Eth32/1",

"lanes": "125,126",

"speed": "50000"

},

"Ethernet126": {

"alias": "Eth32/2",

"lanes": "127,128",

"speed": "50000"

}

},

"platform": "x86\_64-cel\_seastone-r0",

"onie\_partition\_type": "gpt",

"onie\_build\_date": "\"2015-02-03T15:59-0500\"",

"INTERFACE": {

"Ethernet112|10.1.2.2/24": {},

"Ethernet112|fc00:1:2::2/64": {},

"Ethernet116|10.2.2.2/24": {},

"Ethernet116|fc00:2:2::2/64": {},

"Ethernet120|10.3.2.2/24": {},

"Ethernet120|fc00:3:2::2/64": {},

"Ethernet124|10.4.2.2/24": {},

"Ethernet124|fc00:4:2::2/64": {}

},

"NTP\_SERVER": {

"0.debian.pool.ntp.org": {},

"1.debian.pool.ntp.org": {},

"2.debian.pool.ntp.org": {},

"3.debian.pool.ntp.org": {}

},

"VLAN\_MEMBER": {

"Vlan100|Ethernet16": {

"tagging\_mode": "untagged"

},

"Vlan100|Ethernet17": {

"tagging\_mode": "untagged"

},

"Vlan777|Ethernet16": {

"tagging\_mode": "tagged"

},

"Vlan777|Ethernet17": {

"tagging\_mode": "tagged"

}

},

"onie\_vendor\_id": "12244",

"LOOPBACK\_INTERFACE": {

"Loopback0|10.10.1.2/32": {}

},

"MIRROR\_SESSION": {

"everflow0": {

"src\_ip": "10.1.0.32",

"dst\_ip": "2.2.2.2"

}

},

"onie\_kernel\_version": "3.2.35"

}