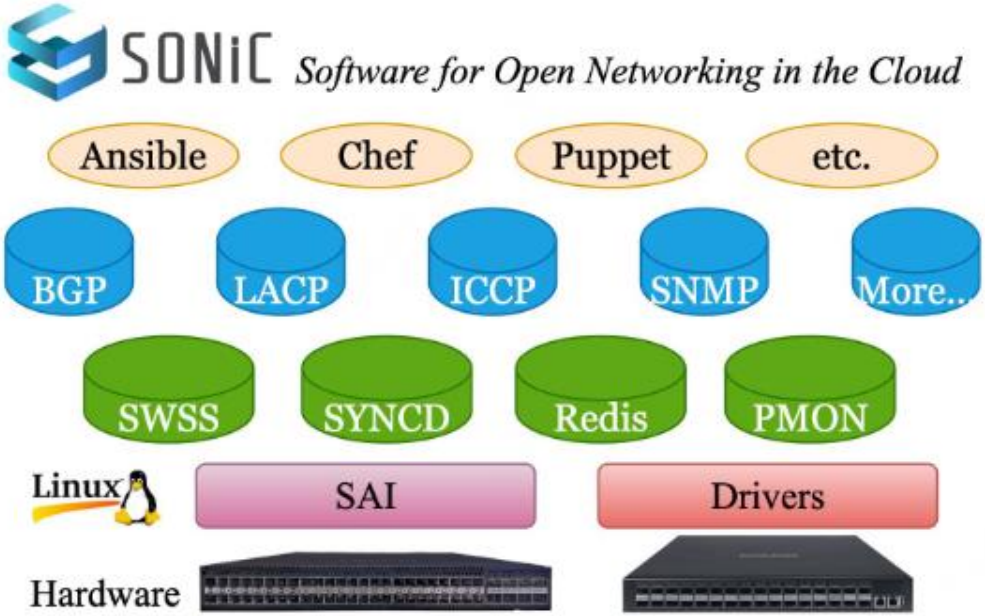
SONiC 架构详解

目录

SONiC 概念

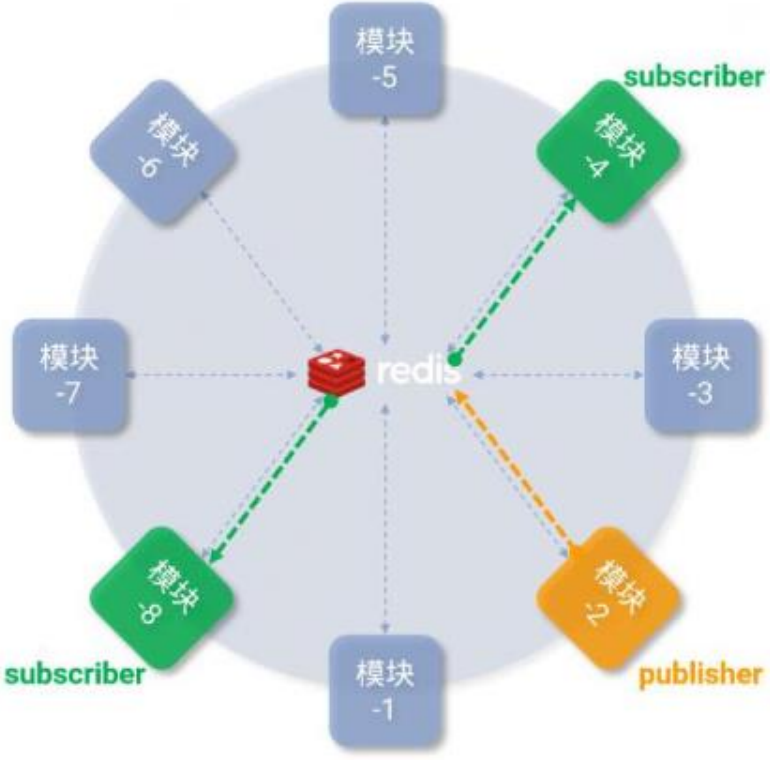


- SONiC是一个基于Linux的开源网络操作系统，包含了构建网络设备所需功能的软件集合，可在多个供应商的交换机和ASIC上运行。
- SONiC大量使用了开源代码和相关技术，如redis，FRRouting，teamd以及自动化运维工具Ansible等。

SONiC 概念

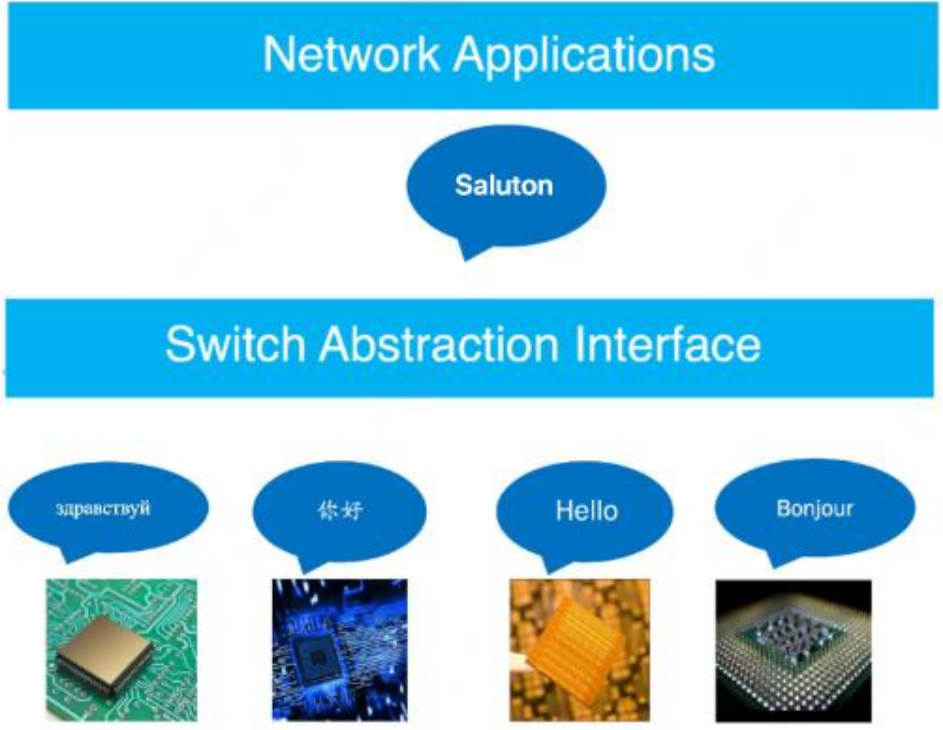
- **Linux-based：**SONiC所使用的基础系统基于Debian GNU/Linux
- **SONiC 基础服务：**Database、Docker 、SWSS、SYNCD等
  - Database：使用Redis数据库，用于子系统间通信、数据持久化
  - Docker：容器化运行环境
- **Network applications：**提供协议交互与网络功能等
  - BGP：提供IP路由协议栈BGP4，BGP4+，MP-BGP
  - SNMP：提供简单网管协议对接功能
- **驱动模块：**
  - 白盒交换驱动：提供对电源、风扇、温度传感器、EEPROM等外设访问与控制
  - ASIC平台驱动：提供对ASIC芯片访问与控制

数据库驱动：以 Redis 为中心的松耦合架构



- 通过 Redis 数据库进行数据交换，实现了模块间调用的解耦
- 各模块可以使用不同的编程语言来实现
- 发布者/订阅者机制，多个模块间不必依赖全连接的复杂通信方式，只订阅所需数据
- 提供数据持久化能力
- 结合相关技术能够在不影响数据转发面的前提下提供应用进程级别故障及恢复能力
- 为新功能扩展提供了便利的基础环境

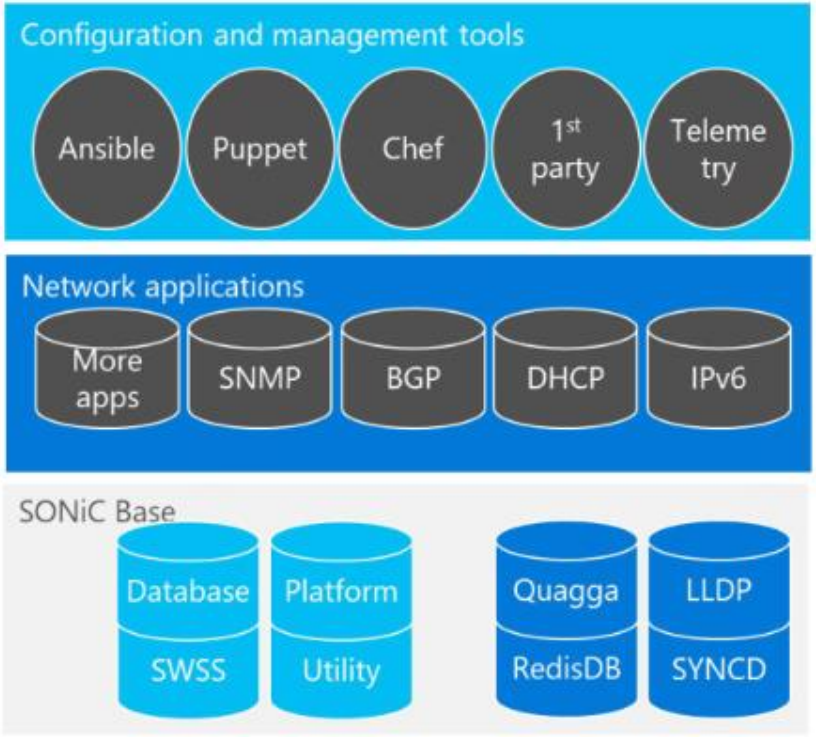
硬件解耦合：抽象的交换机接口



SAI：Switch Abstraction Interface

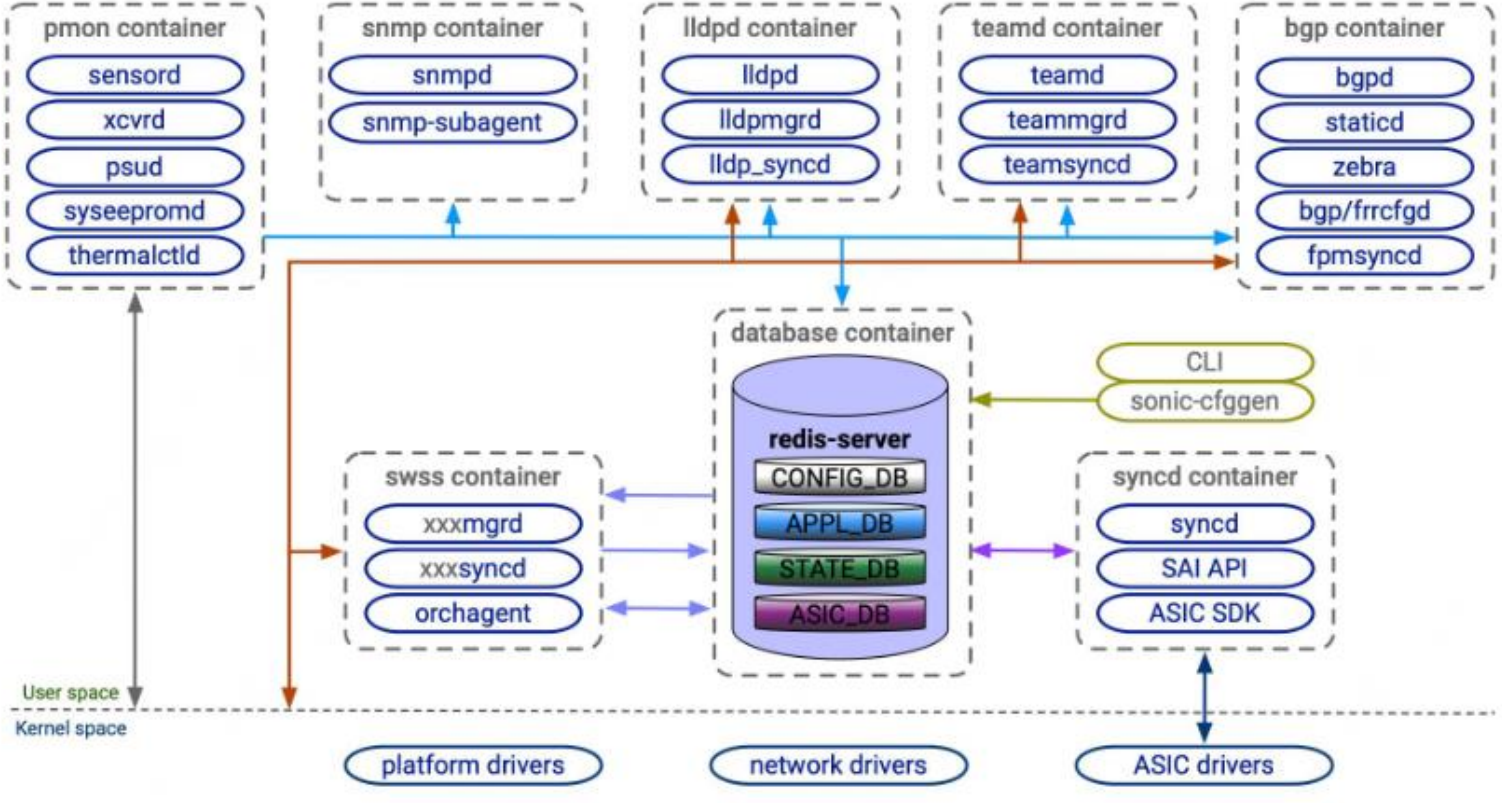
- 交换机抽象接口，定义了一套标准化的API，提供与芯片厂商无关的方式来控制数据转发面
- 采用标准C实现，定义了基于对象的通用CRUD API ，对上层NOS提供统一接口
- 作为南向接口与交换芯片SDK进行适配对接，屏蔽不同芯片厂商的SDK差异

软件解耦合：容器化部署



- SONiC使用Docker容器化技术，将模块放在独立的容器中，屏蔽了与底层平台抽象层交互细节
- 高内聚，低耦合，每个模块尽可能独立完成自己的功能，减少与模块外部的依赖
- 方便部署和升级，结合相关技术提供热升级和业务编排的能力
- 一致的运行环境，便于运维扩展，以及敏捷迭代集成新的应用

SONiC整体架构



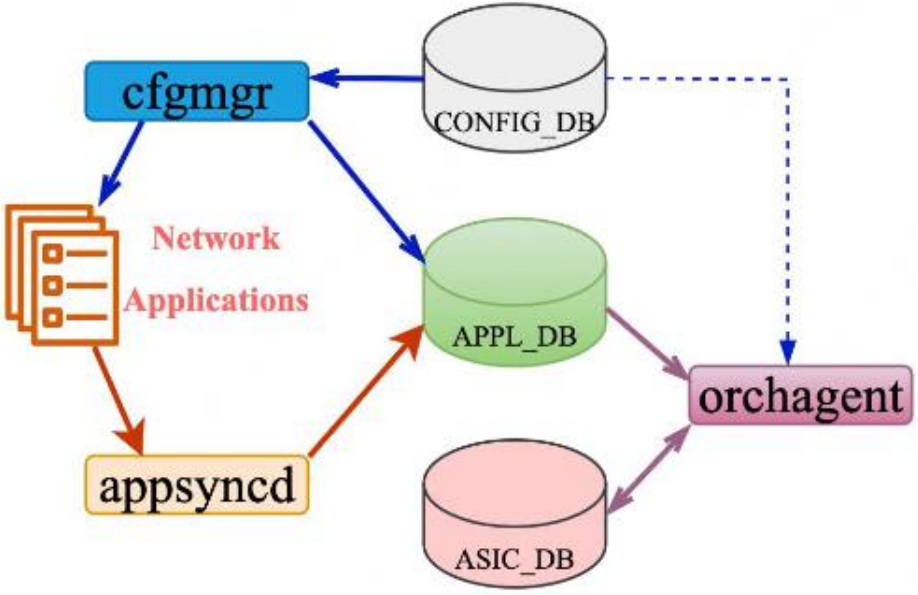
SONiC组件：database container

SONiC 数据库使用 redis-server，采用容器化方式部署在系统中

- 本地应用程序可以通过 Linux 套接字访问数据库中保存的数据，支持不同编程语言实现的模块通过相应的封装接口来访问数据库
- 利用 publish/subscribe 模型和 keyspace 机制，建立各模块间的消息传递机制

索引	数据库名称	描述
0	APPL_DB	存放应用组件控制表项，如应用程序生成的路由信息等
1	ASIC_DB	存放控制底层ASIC的状态和运行的对象信息
2	COUNTERS_DB	存放系统计数器和统计信息
3	LOGLEVEL_DB	存放日志配置信息
4	CONFIG_DB	存放系统和应用程序配置，可用于动态加载与持久化
5	FLEX_COUNTER_DB	存放灵活计数器配置信息
6	STATE_DB	存放运行状态信息，用于处理不同模块间的依赖状态

SONiC组件：SWSS container



SWSS：SWitch State Services，是用于协调组织模块间信息交互的一组服务程序

- cfgmgr：负责监听并处理各类应用配置，控制相关的应用程序配置并写入APPL_DB
- appsyncd：负责监听并处理来自应用程序的消息，写入APPL_DB
- orchagent：提取并处理来自上层应用的信息，转换成SAI原语下发到ASIC_DB

SONiC组件：SWSS相关进程

portmgrd：监听来自CONFIG_DB的物理端口信息，更新Linux接口配置并同步设置APPL_DB和STATE_DB相关表项。

intfmgrd：监听CONFIG_DB中不同类型的接口配置，更新Linux接口配置并同步设置APPL_DB和STATE_DB相关表项。

vlanmgrd：监听CONFIG_DB中的VLAN和VLAN成员配置，在 Linux 中配置 VLAN，并同步APPL_DB和STATE_DB相关表项 。

vrfmgrd：监听CONFIG_DB中的VRF、VNET及带内管理VRF等配置，在 Linux 内核中配置VRF并同步APPL_DB和STATE_DB相关表项 。

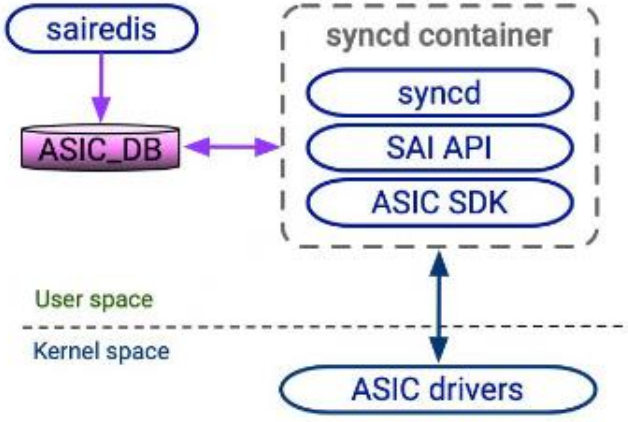
nbrmgrd：监听CONFIG_DB中的邻居配置，向Linux内核下发邻居表；监听来自APPL_DB和STATE_DB表项或状态变更，更新Linux内核邻居表。

portsyncd：监听与端口相关的 netlink 事件及APPL_DB，将端口属性或状态信息写入STATE_DB；在启动过程中，portsyncd 通过解析系统的硬件配置文件获取物理端口信息，并更新STATE_DB中端口相关状态。

fdbsyncd：监听FDB相关的 netlink 事件，更新STATE_DB状态信息，同时提供对EVPN的IMET routes处理。

neighsyncd：监听与邻居相关的 netlink 事件，并根据邻居表状态将包含address-family和mac-address等属性的表项写入 APPL_DB。

SONiC组件：SYNCD container

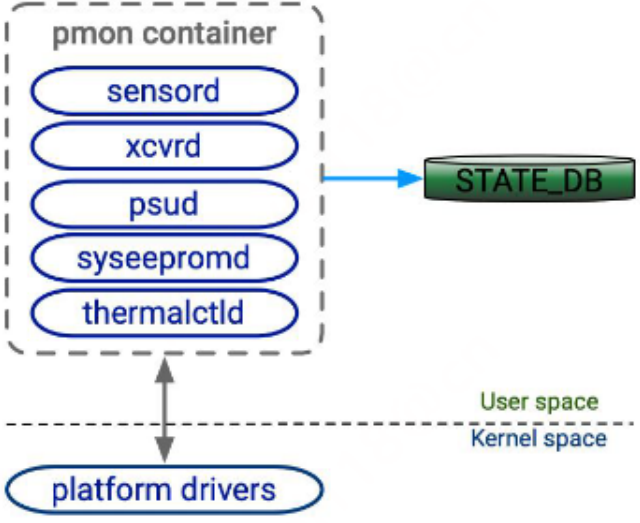


SYNCD 容器提供一种交换机控制面的网络状态与实际硬件转发面表项同步机制。

- **syncd**：负责初始化、配置和收集交换芯片的当前状态，并维护ASIC_DB与底层表项的同步。syncd订阅SWSS写入ASIC_DB的对象信息，调用芯片SDK接口下发到底层硬件；同时syncd也会注册监听来自底层硬件的通知或状态信息，推送给上层处理。
- **ASIC SDK**：以动态链接库的形式提供的驱动ASIC所需的SAI实现，该库文件一般由芯片供应商提供。

sairedis：为SWSS写入ASIC_DB提供调用接口，与syncd处理机制紧密结合。

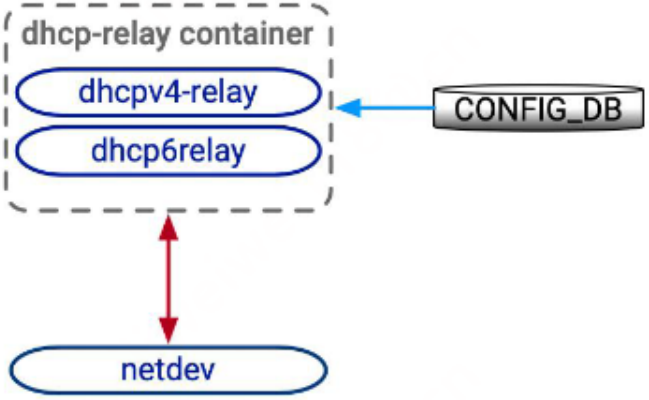
SONiC组件：PMON container



PMON容器为硬件平台外围设备监控管理提供服务，如：电源、风扇、温度传感器、EEPROM等。

- **sensord**：定期获取硬件温度传感器数值并记录到数据库。
- **xcvrd**：定期读取光模块信息并记录到数据库，读取的信息包括模块在位信息、标准EEPROM信息以及DOM信息等，同时监听模块的插入/拔出事件。
- **psud**：定期读取电源的在位和状态信息并记录到数据库。
- **syseepromd**：读取系统EEPROM信息并记录到数据库。
- **thermalctld**：定期读取风扇信息并记录到数据库，并根据热状态运行控制策略。

SONiC组件：DHCP-relay container

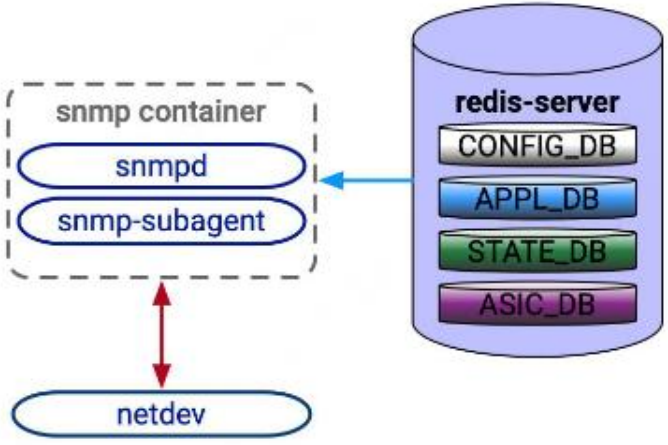


DHCP-relay容器提供DHCP中继代理服务，实现在不同子网和物理网段之间DHCP信息处理和转发。

- **dhcpv4-relay**：提供DHCPv4中继代理服务，使用isc-dhcp开源方案。
- **dhcp6relay**：提供DHCPv6中继代理服务。

ISC DHCP：为实施DHCP服务器、中继代理和客户端提供了一个完整的开源解决方案。ISC DHCP同时支持IPv4和IPv6，并且适合用于高容量和高可靠性的应用。

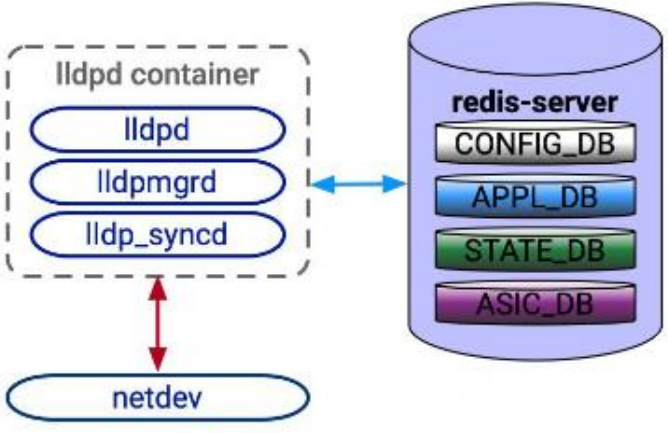
SONiC组件：SNMP container



SNMP容器提供简单网络管理协议功能，处理来自数据库的本地设备状态和业务信息，供外部网管服务查询。

- **snmpd**：作为SNMP服务端，负责处理来自外部网络的snmp请求并返回查询信息。
- **snmp-subagent**：即sonic_ax_impl，用于收集本地数据库中的设备状态和业务信息，并传递给snmpd进程。

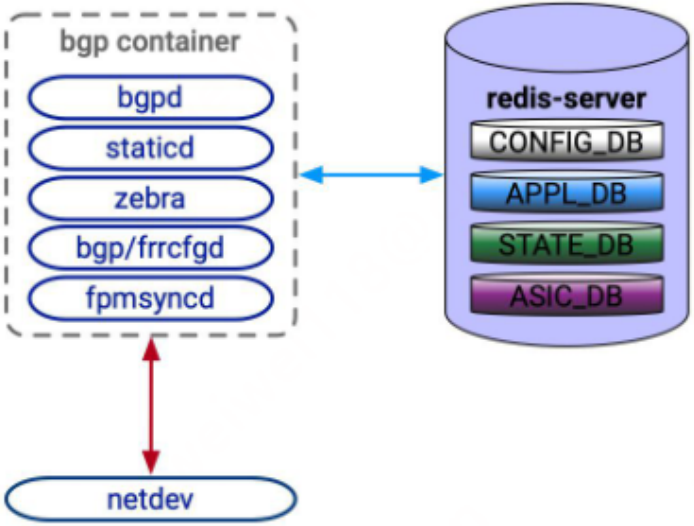
SONiC组件：LLDP container



LLDP容器提供链路层发现协议功能，设备能互相发现并通告状态及交互信息。

- **lldpd**：LLDP协议实现主进程，与外部对端设备建立lldp连接。该应用程序使用开源方案实现。
- **lldpmgr**：为lldp应用程序提供配置逻辑实现。
- **lldp_syncd**：负责将lldp发现的状态信息写入数据库，供其它应用程序访问。

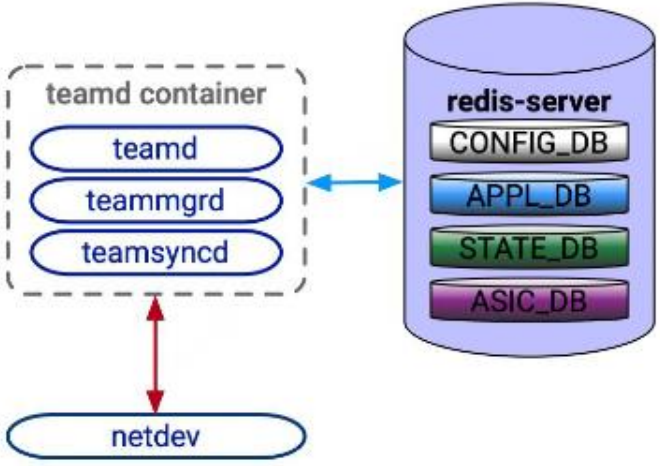
SONiC组件：BGP container



BGP容器运行路由协议栈模块，提供对BGP、OSPF、ISIS、BFD等协议支持。该容器可运行的路由协议栈支持多种开源方案，默认使用FRR。

- **bgpd**：提供BGP路由协议栈实现。
- **staticd**：提供对静态路由机制实现。
- **zebra**：IP路由管理模块，提供内核路由表更新、接口查询和不同路由协议之间的路由分发。
- **bgp/frrcfgd**：为容器中运行的路由协议模块提供配置逻辑实现。
- **fpmsyncd**：负责监听由zebra生成的FIB信息，并将信息处理后同步下发到APPL_DB。

SONiC组件：teamd container



Teamd容器运行链路聚合控制协议模块，为系统提供链路聚合功能。

- **teamd**：提供LACP实现，是基于 Linux 的链路聚合协议的一种开源实现。
- **teammgrd**：为运行链路聚合协议模块提供配置逻辑实现。
- **teamsyncd**：监听链路聚合组与状态变化，维护聚合组、聚合组成员及更新状态信息。

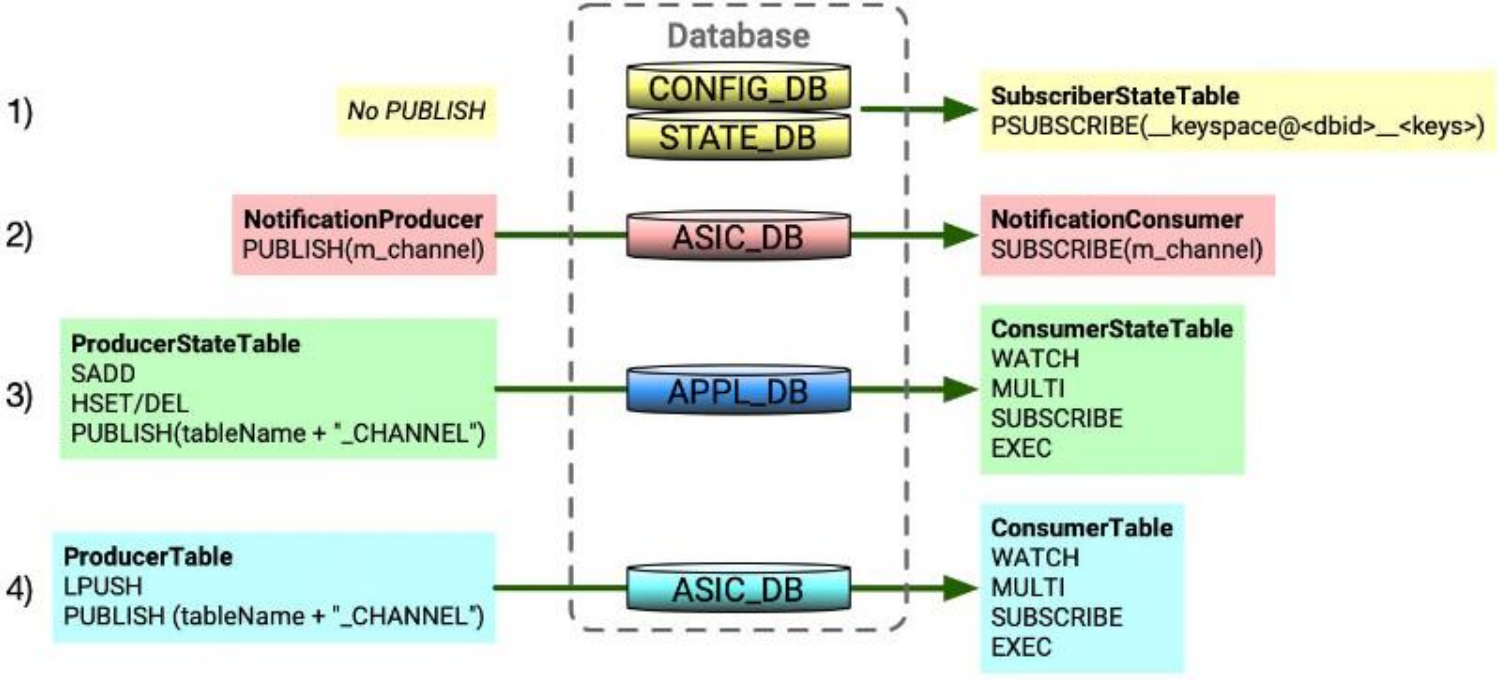
SONiC 配置方式


```
admin@sonic:~$ sudo config ?
Usage: config [OPTIONS] COMMAND [ARGS]...

  SONiC command line - 'config' command

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

Commands:
  aaa                       AAA command line
  acl                       ACL-related configuration tasks
  apply-patch               Apply given patch of updates to Config.
  auto-techsupport          AUTO_TECHSUPPORT part of config_db.json
  auto-techsupport-feature  AUTO_TECHSUPPORT_FEATURE part of config_db.json
  bgp                       BGP-related configuration tasks
  buffer                    Configure buffer_profile
  cbf                       CBF-related configuration tasks
  chassis                   Configure chassis commands group
```

系统提供多种配置方式：

- **SONiC CLI**：基于Python Click库实现的命令行界面，提供了一种灵活定制扩展的方法来构建命令行的工具。
- **CONFIG_DB**：通过导入json到数据库，实现对配置的添加或修改。
- **Management Framework**：提供Cisco-Like CLI，RESTAPI，gNMI等配置方式。

以数据库为中心的通信模型



数据库通信模型（一）

1）SubscriberStateTable

- 使用Redis的keyspace机制，向订阅者传递key-value消息
- 不需要执行PUBLISH，订阅指定KEY
- 用于CONFIG_DB和STATE_DB的监听处理

2）NotificationProducer & NotificationConsumer

- 使用Redis的publish/subscribe模型
- 通过消息队列来传递信息
- 用于模块间信息通知处理，如端口状态通知、FDB事件通知

数据库通信模型（二）

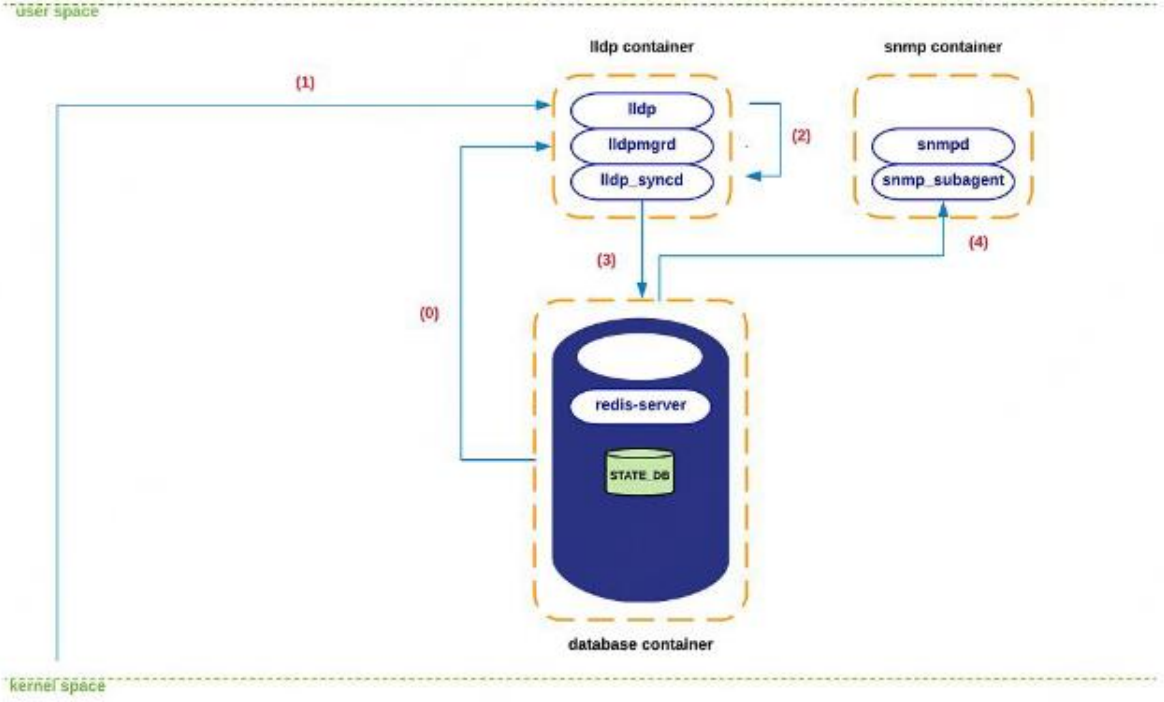
3）ProducerStateTable & ConsumerStateTable

- 使用Redis的publish/subscribe模型封装
- 利用KeySet机制来传递信息
- 用于围绕APPL_DB的消息传递

4）ProducerTable & ConsumerTable

- 使用Redis的publish/subscribe模型封装
- 通过有序链表（list）来传递key-value-op三元消息
- 用于围绕ASIC_DB和FLEX_COUNTER_DB的消息传递

LLDP状态交互



LLDP状态交互过程

(0) 在 LLDP 服务初始化过程中，lldpmgrd 订阅 CONFIG_DB 获取基本配置，并通过 STATE_DB 获取系统物理端口的实时状态。lldpd 负责与对端连接网络设备进行协议信息交互，通知系统端口状态变化以及相关配置信息变更。

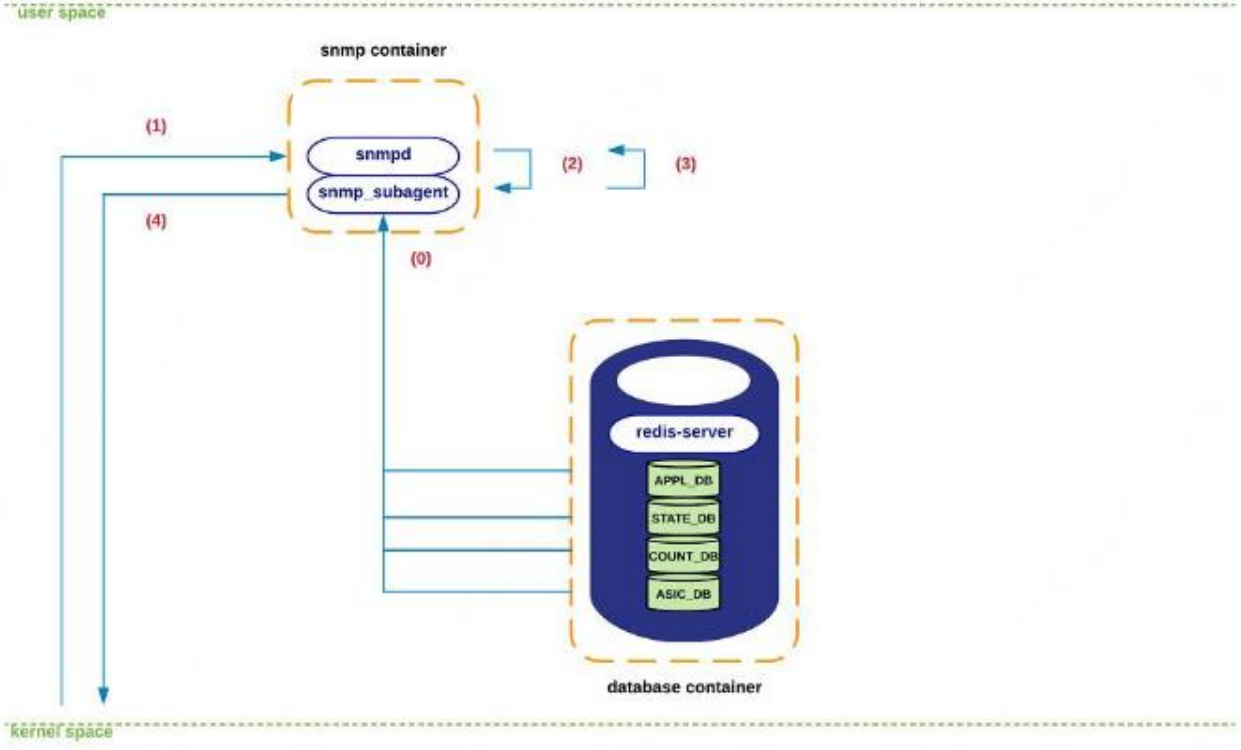
(1) LLDP 容器通过内核socket收到 LLDP 数据包，交由 lldpd 进程进一步处理。

(2) lldpd 负责解析并处理收到的协议数据内容，并通过 lldpctl 和 lldpcli 获取更新后的相关信息。

(3) lldp_syncd 负责将获取的信息格式化为指定的json，写入APPL_DB中对应的表项中。

(4) SNMP容器的应用程序作为该表项的订阅者，监听相关信息的变化通过简单网络管理协议同步给管理平台。

SNMP状态交互



SNMP状态交互过程

(0) 在SNMP服务初始化阶段，snmp-subagent 根据不同 MIB 需要与对应的 DB 建立连接。从 DB 获得的信息都缓存在 snmp-subagent 中，每间隔一段时间刷新一次，保持 DB 内容与 snmp-subagent 完全同步。

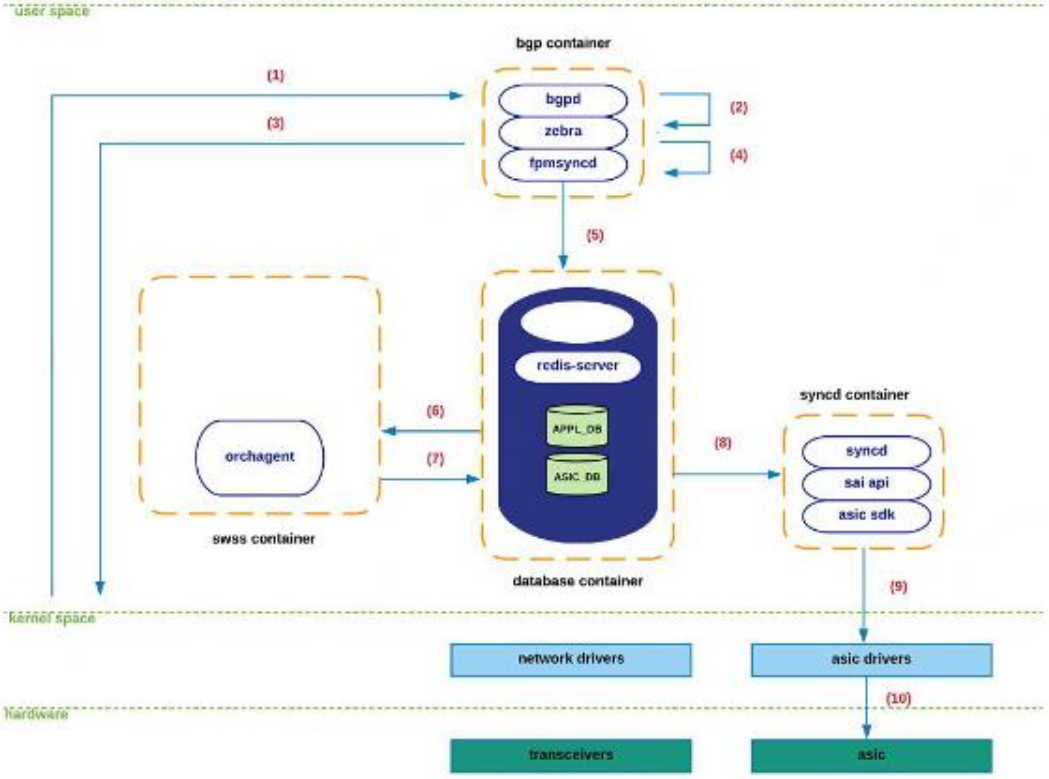
(1) SNMP 容器通过内核 socket 收到 SNMP 查询报文，交由 snmpd 进程进一步处理。

(2) snmpd 进程解析 SNMP 消息并向 snmp-subagent (sonic_ax_impl) 发送相关请求。

(3) snmp-subagent 根据相关请求获取本地数据结构中的状态信息，并将对应信息发送回 snmpd 进程。

(4) snmpd 通过通用 socket 接口向发起者发送回复信息。

路由状态交互



路由状态交互过程

(0) 在 BGP 容器初始化时，zebra 通过 TCP socket 与 fpmsyncd 连接。在稳定或非瞬态条件下，zebra、linux 内核、APPL_DB 以及 ASIC_DB 中保存的通用路由表可以认为是一致的（非过滤情况下）。

(1) 通过内核 socket 收到 BGP 协议报文，交由 bgpd 进程进一步处理。

(2) bgpd 解析数据包，完成协议处理并通知 zebra 关于路由前缀及关联下一跳的更新。

(3) zebra 检查路由前缀和下一跳的可达性，生成路由 netlink 消息并发送给 Linux 内核。

(4) 同时 zebra 通过 FPM 接口将此 netlink 消息传递给 fpmsyncd。

(5) fpmsyncd 处理 netlink 消息并将路由表项信息写入 APPL_DB。

(6) orchagent 作为一个APPL_DB消费者，订阅处理相关路由表项。

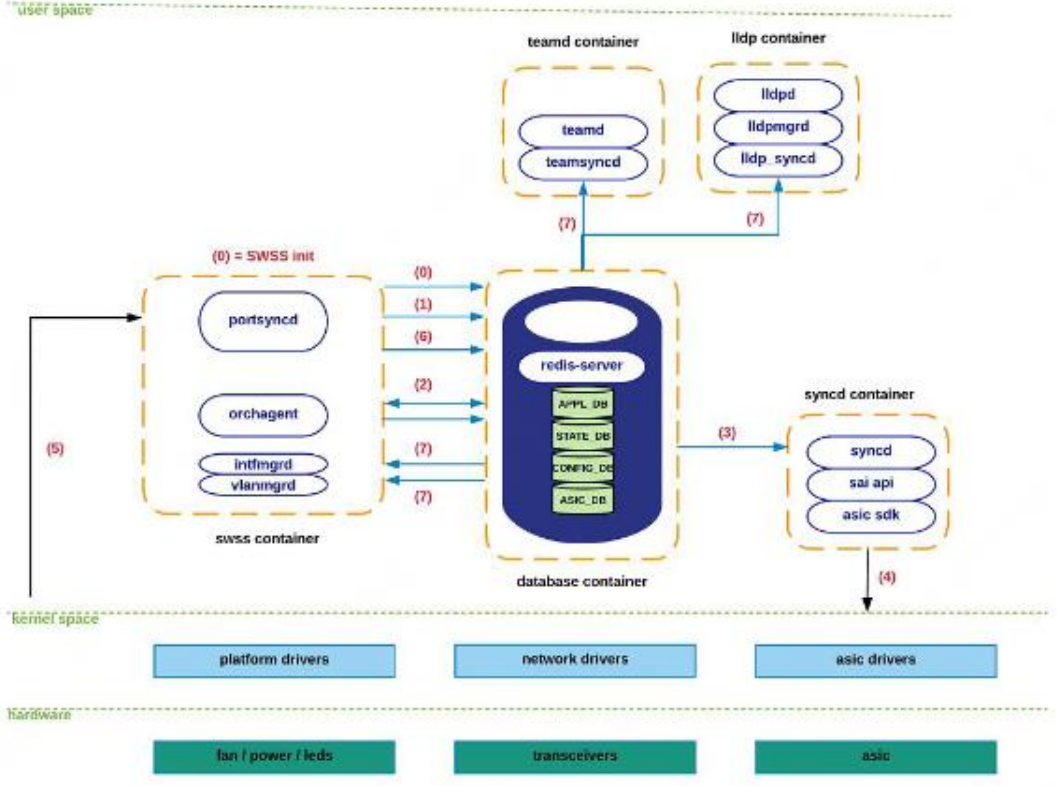
(7) orchagent 对应的路由线程完成信息处理后，调用 sairedis 接口将路由信息写入 ASIC_DB 。

(8) syncd作为消费者订阅处理 ASIC_DB 中的路由表项信息。

(9) syncd 完成信息处理并调用 SAI API ，利用芯片厂商提供的 SAI 实现通过 ASIC driver 向芯片下发配置。

(10) 路由最终下发到转发面 ASIC。

端口状态交互



端口状态交互过程

(0) 在 SWSS 服务初始化时，portsyncd 与 redis-server 的关联数据库建立通信连接，portsyncd 既作为 CONFIG_DB 的订阅者，也作为 APPL_DB 和 STATE_DB 的发布者，同时负责监听接口的 netlink 消息。

(1) portsyncd 通过解析与硬件平台关联的端口配置文件，将与端口相关的信息，如接口名称、接口速度、lane 等信息写入 APPL_DB。

(2) orchagent 负责订阅并处理这些表项信息，判断端口配置解析完成的状态，当满足条件时，orchagent 调用 sairedis API 将相关信息写入 ASIC_DB ；否则挂起相关操作，直到 portsyncd 完成端口初始化解析。

(3) syncd作为消费者订阅处理 ASIC_DB 中的端口表项信息。

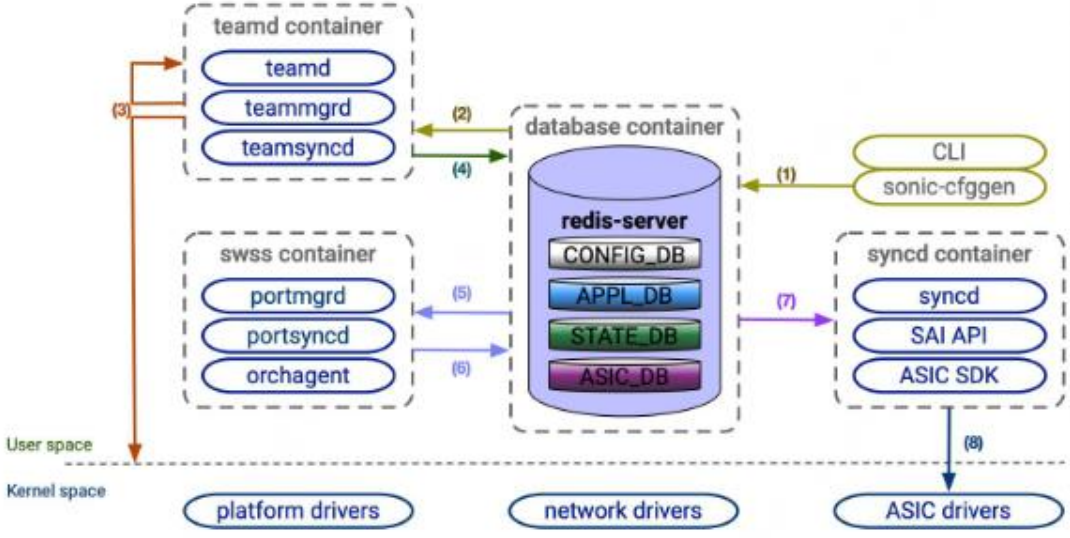
(4) syncd 创建与正在初始化的物理端口相关联的内核主机接口并调用 SAI API ，利用芯片厂商提供的 SAI 实现通过 ASIC driver 向芯片下发配置。

(5) portsyncd 收到对内核主机接口操作生成的相应 netlink 消息。

(6) portsyncd 将每个端口初始化完成状态写入 STATE_DB 的对应表项中。

(7) 其它依赖 STATE_DB 关于端口初始化状态的应用程序将收到通知，允许这些应用程序开始使用它们所依赖的端口。

端口聚合组配置流程示例

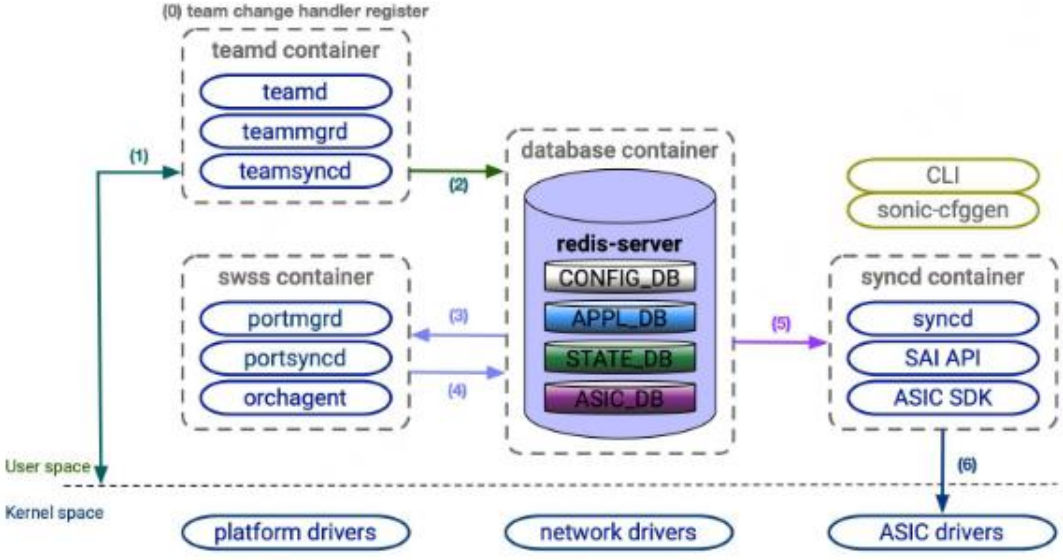



```
admin@sonic:~$ redis-cli -n 4 KEYS PORTCHANNEL*
1) "PORTCHANNEL|PortChannel0001"
2) "PORTCHANNEL_MEMBER|PortChannel0001|Ethernet4"
3) "PORTCHANNEL_MEMBER|PortChannel0001|Ethernet0"
```

```
admin@sonic:~$ redis-cli -n 0 KEYS LAG*
1) "LAG_MEMBER_TABLE:PortChannel0001:Ethernet0"
2) "LAG_MEMBER_TABLE:PortChannel0001:Ethernet4"
3) "LAG_TABLE:PortChannel0001"
```

```
admin@sonic:~$ redis-cli -n 1 KEYS *LAG*
1) "ASIC_STATE:SAI_OBJECT_TYPE_LAG_MEMBER:oid:0x1b000000000569"
2) "ASIC_STATE:SAI_OBJECT_TYPE_LAG:oid:0x2000000000568"
3) "ASIC_STATE:SAI_OBJECT_TYPE_LAG_MEMBER:oid:0x1b000000000056a"
```

端口聚合流程示例




```
admin@sonic:~$ redis-cli -n 6 HGETALL "LAG_TABLE|PortChannel0001"
1) "state"
2) "ok"
redis-cli -n 6 HGETALL "PORT_TABLE|Ethernet0"
1) "state"
2) "ok"
admin@sonic:~$ redis-cli -n 6 HGETALL "PORT_TABLE|Ethernet4"
1) "state"
2) "ok"
```

```
admin@sonic:~$ redis-cli -n 0 HGETALL "LAG_TABLE:PortChannel0001"
1) "admin_status"
2) "up"
3) "oper_status"
4) "up"
5) "mtu"
6) "9100"
admin@sonic:~$ redis-cli -n 0 HGETALL "LAG_MEMBER_TABLE:PortChannel0001:Ethernet0"
1) "status"
2) "enabled"
admin@sonic:~$ redis-cli -n 0 HGETALL "LAG_MEMBER_TABLE:PortChannel0001:Ethernet4"
1) "status"
2) "enabled"
```

Thanks

感 谢 您 的 聆 听