转载自：soul orbit: <https://r12f.com/posts/sonic-2-key-components/>

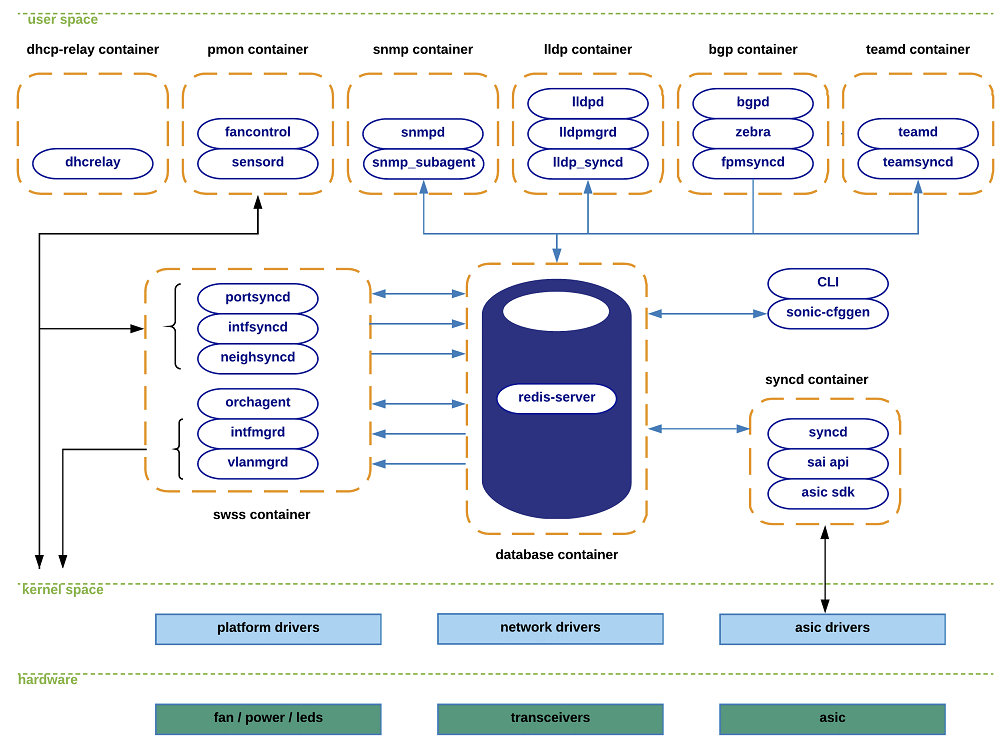
# SONiC学习笔记（二）：核心组件

[在上一篇文章中，我们简单的介绍了SONiC的基本架构，以及它的一些特点](https://r12f.com/posts/sonic-1-intro/)。在这篇文章中，我们来继续自上向下的来看看SONiC，更加深入一点的来了解SONiC的核心组件的构成。

## **1. 前置知识**

这篇文章和SONiC的官方文档中会经常出现这几个词：ASIC（Application-Specific Integrated Circuit）和ASIC状态（State）。它们指的是交换机中用来进行包处理的Pipeline的状态，比如，ACL，转发方式等等，这个和其他交换机的硬件状态，比如，端口状态（端口速度，接口类型），IP信息等等硬件状态是非常不同的。如果大家有兴趣了解更深入的细节，可以先移步阅读两个相关资料：[SAI (Switch Abstraction Interface) API](https://github.com/opencomputeproject/SAI/wiki/SAI-APIs" \t "https://r12f.com/posts/sonic-2-key-components/_blank)和一篇RMT（Reprogrammable Match Table）的相关论文：[Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN](http://yuba.stanford.edu/~grg/docs/sdn-chip-sigcomm-2013.pdf" \t "https://r12f.com/posts/sonic-2-key-components/_blank)。这些都会对我们阅读SONiC的文档有很大的帮助。

为了方便我们的理解和阅读，我们把上一篇文章中已经放过的SONiC架构图在这里再放一次 [[1]](https://github.com/sonic-net/SONiC/wiki/Architecture" \t "https://r12f.com/posts/sonic-2-key-components/_blank)：



## **2. 数据库**

首先，在SONiC里面最核心的服务，自然是当之无愧的中心数据库Redis了！它的主要目的有两个：存储所有服务的配置和状态，并且为各个服务提供通信的媒介。具体的细节，我们这里不会展开，后面会用专门的文章来介绍。

SONiC会在Redis中创建一个名为sonic-db的数据库，其配置和分库信息我们可以在/var/run/redis/sonic-db/database\_config.json中找到：

|  |
| --- |
| admin@sonic:~$ cat /var/run/redis/sonic-db/database\_config.json {  "INSTANCES": {  "redis": {  "hostname": "127.0.0.1",  "port": 6379,  "unix\_socket\_path": "/var/run/redis/redis.sock",  "persistence\_for\_warm\_boot": "yes"  }  },  "DATABASES": {  "APPL\_DB": { "id": 0, "separator": ":", "instance": "redis" },  "ASIC\_DB": { "id": 1, "separator": ":", "instance": "redis" },  "COUNTERS\_DB": { "id": 2, "separator": ":", "instance": "redis" },  "LOGLEVEL\_DB": { "id": 3, "separator": ":", "instance": "redis" },  "CONFIG\_DB": { "id": 4, "separator": "|", "instance": "redis" },  "PFC\_WD\_DB": { "id": 5, "separator": ":", "instance": "redis" },  "FLEX\_COUNTER\_DB": { "id": 5, "separator": ":", "instance": "redis" },  "STATE\_DB": { "id": 6, "separator": "|", "instance": "redis" },  "SNMP\_OVERLAY\_DB": { "id": 7, "separator": "|", "instance": "redis" },  "RESTAPI\_DB": { "id": 8, "separator": "|", "instance": "redis" },  "GB\_ASIC\_DB": { "id": 9, "separator": ":", "instance": "redis" },  "GB\_COUNTERS\_DB": { "id": 10, "separator": ":", "instance": "redis" },  "GB\_FLEX\_COUNTER\_DB": { "id": 11, "separator": ":", "instance": "redis" },  "APPL\_STATE\_DB": { "id": 14, "separator": ":", "instance": "redis" }  },  "VERSION": "1.0" } |

虽然我们可以看到SONiC中的数据库有十来个，但是我们大部分时候只需要关注以下几个最重要的数据库就可以了：

* ****CONFIG\_DB****：ID为4，存储所有服务的****配置信息****，比如端口配置，VLAN配置等等。它代表着****用户想要交换机达到的状态****的数据模型，这也是所有CLI和外部应用程序修改配置时的主要操作对象。
* ****APPL\_DB（Application DB）****：ID为0，存储****所有服务的内部状态信息****。这些信息有两种：一种是各个服务在读取了CONFIG\_DB的配置信息后，自己计算出来的。我们可以理解为****各个服务想要交换机达到的状态****（Goal State），还有一种是当最终硬件状态发生变化被写回时，有些服务会直接写回到APPL\_DB，而不是我们下面马上要介绍的STATE\_DB。这些信息我们可以理解为****各个服务认为交换机当前的状态****（Current State）。
* ****STATE\_DB****：ID为6，存储着交换机****各个部件当前的状态****（Current State）。当SONiC中的服务收到了STATE\_DB的状态变化，但是发现和Goal State不一致的时候，SONiC就会重新下发配置，直到两者一致。（当然，对于那些回写到APPL\_DB状态，服务就会监听APPL\_DB的变化，而不是STATE\_DB了。）
* ****ASIC\_DB****：ID为1，存储着****SONiC想要交换机ASIC达到状态信息****，比如，ACL，转发方式等等。和APPL\_DB不同，这个数据库里面的数据模型是面向ASIC设计的，而不是面向服务抽象的。这样做的目的是为了方便各个厂商进行SAI和ASIC驱动的开发。

目前我的感觉是APPL\_DB和STATE\_DB这两个DB的设计并不是很干净，它们的职责经常重叠，不过也有可能是我理解不够深入。欢迎大家一起讨论。

这里，我们会发现一个很直观的问题：交换机里面这么多服务，难道所有的配置和状态都放在一个数据库里面没有隔离的么？如果两个服务用了同一个Redis Key怎么办呢？这个问题非常的好，SONiC的解决也很直接，那就是在每个数据库里面继续分表！

我们知道Redis在每个数据库里面并没有表的概念，而是使用key-value的方式来存储数据。所以，为了进一步分表，SONiC的解决方法是将表的名字放入key中，并且使用分隔符将表和key隔开。上面的配置文件中separator字段就是做这个了。比如：APPL\_DB中的PORT\_TABLE表中的Ethernet4端口的状态，我们可以通过PORT\_TABLE:Ethernet4来获取，如下：

|  |
| --- |
| 127.0.0.1:6379> select 0 OK 127.0.0.1:6379> hgetall PORT\_TABLE:Ethernet4  1) "admin\_status"  2) "up"  3) "alias"  4) "Ethernet6/1"  5) "index"  6) "6"  7) "lanes"  8) "13,14,15,16"  9) "mtu" 10) "9100" 11) "speed" 12) "40000" 13) "description" 14) "" 15) "oper\_status" 16) "up" |

当然在SONiC中，不仅仅是数据模型，包括通信机制，都是使用类似的方法来实现“表”级别的隔离的。

## **3. 服务**

### **3.1. 服务分类**

接下来就是服务了。SONiC里面的服务（常驻进程）非常的多，有二三十种，它们会在随着交换机启动而启动，并一直保持运行，直到交换机关机。如果我们想快速掌握SONiC，一个一个服务的去了解，会很容易陷入细节的泥潭，所以，我们最好把这些服务进行一个大的分类。

我们这里不会深入到某一个具体的服务中去，而是先从整体上来看看SONiC中的服务的结构，帮助我们建立一个整体的认识。关于各个具体的服务，如果有机会，我们会在后面的文章中再进行介绍。

总体而言，SONiC中的服务可以分为以下几类：\*syncd, \*mgrd，feature实现，orchagent和syncd。

#### **3.1.1. \*syncd服务**

这类服务名字中都以syncd结尾。它们做的事情都很类似：它们负责将硬件状态同步到Redis中，一般目标都以APPL\_DB或者STATE\_DB为主。

比如，portsyncd就是通过监听netlink的事件，将交换机中所有Port的状态同步到STATE\_DB中，而natsyncd则是监听netlink的事件，将交换机中所有的NAT状态同步到APPL\_DB中。

#### **3.1.2. \*mgrd服务**

这类服务名字中都以mgrd结尾。顾名思义，这些服务是所谓的“Manager”服务，也就是说它们负责各个硬件的配置，和\*syncd完全相反。它们的逻辑主要有两个部分：

* ****配置下发****：负责读取配置文件和监听Redis中的配置和状态改变（主要是CONFIG\_DB，APPL\_DB和STATE\_DB），然后将这些修改推送到交换机硬件中去。推送的方法有多种，取决于更新的目标是什么，可以通过更新APPL\_DB并发布更新消息，或者是直接调用linux下的命令行，对系统进行修改。比如：nbrmgr就是监听CONFIG\_DB，APPL\_DB和STATE\_DB中neighbor的变化，并调用netlink和command line来对neighbor和route进行修改，而intfmgr除了调用command line还会将一些状态更新到APPL\_DB中去。
* ****状态同步****：对于需要Reconcile的服务，\*mgrd还会监听STATE\_DB中的状态变化，如果发现硬件状态和当前期望状态不一致，就会重新发起配置流程，将硬件状态设置为期望状态。这些STATE\_DB中的状态变化一般都是\*syncd服务推送的。比如：intfmgr就会监听STATE\_DB中，由portsyncd推送的，端口的Up/Down状态和MTU变化，一旦发现和其内存中保存的期望状态不一致，就会重新下发配置。

#### **3.1.3. 功能实现服务**

有一些功能并不是依靠OS本身来完成的，而是由一些特定的进程来实现的，比如BGP，或者一些外部接口。这些服务名字中经常以d结尾，表示deamon，比如：bgpd，lldpd，snmpd，teamd等，或者干脆就是这个功能的名字，比如：fancontrol。

#### **3.1.4. orchagent服务**

这个是SONiC中最重要的一个服务，不像其他的服务只负责一两个特定的功能，orchagent作为交换机ASIC状态的编排者（orchestrator），会检查数据库中所有来自\*syncd服务的状态，整合起来并下发给用于保存交换机ASIC配置的数据库：ASIC\_DB。这些状态最后会被syncd接收，并调用SAI API经过各个厂商提供的SAI实现和ASIC SDK和ASIC进行交互，最终将配置下发到交换机硬件中。

#### **3.1.5. syncd服务**

syncd服务是orchagent的下游，它虽然名字叫syncd，但是它却同时肩负着ASIC的\*mgrd和\*syncd的工作。

* 首先，作为\*mgrd，它会监听ASIC\_DB的状态变化，一旦发现，就会获取其新的状态并调用SAI API，将配置下发到交换机硬件中。
* 然后，作为\*syncd，如果ASIC发送了任何的通知给SONiC，它也会将这些通知通过消息的方式发送到Redis中，以便orchagent和\*mgrd服务获取到这些变化，并进行处理。这些通知的类型我们可以在[SwitchNotifications.h](https://github.com/sonic-net/sonic-sairedis/blob/master/syncd/SwitchNotifications.h" \t "https://r12f.com/posts/sonic-2-key-components/_blank)中找到。

### **3.2. 服务间控制流分类**

有了这些分类，我们就可以更加清晰的来理解SONiC中的服务了，而其中非常重要的就是理解服务之间的控制流。有了上面的分类，我们这里也可以把主要的控制流有分为两类：配置下发和状态同步。

#### **3.2.1. 配置下发**

配置下发的流程一般是这样的：

* ****修改配置****：用户可以通过CLI或者REST API修改配置，这些配置会被写入到CONFIG\_DB中并通过Redis发送更新通知。或者外部程序可以通过特定的接口，比如BGP的API，来修改配置，这种配置会通过内部的TCP Socket发送给\*mgrd服务。
* **\*mgrd**下发配置****：服务监听到CONFIG\_DB中的配置变化，然后将这些配置推送到交换机硬件中。这里由两种主要情况（并且可以同时存在）：
  + ****直接下发****：
* \*mgrd服务直接调用linux下的命令行，或者是通过netlink来修改系统配置
* \*syncd服务会通过netlink或者其他方式监听到系统配置的变化，并将这些变化推送到STATE\_DB或者APPL\_DB中。
* \*mgrd服务监听到STATE\_DB或者APPL\_DB中的配置变化，然后将这些配置和其内存中存储的配置进行比较，如果发现不一致，就会重新调用命令行或者netlink来修改系统配置，直到它们一致为止。
* ****间接下发****：
* \*mgrd将状态推送到APPL\_DB并通过Redis发送更新通知。
* orchagent服务监听到配置变化，然后根据所有相关的状态，计算出此时ASIC应该达到的状态，并下发到ASIC\_DB中。
* syncd服务监听到ASIC\_DB的变化，然后将这些新的配置通过统一的SAI API接口，调用ASIC Driver更新交换机ASIC中的配置。

配置初始化和配置下发类似，不过是在服务启动的时候读取配置文件，这里就不展开了。

#### **3.2.2. 状态同步**

如果这个时候，出现了一些情况，比如网口坏了，ASIC中的状态变了等等，这个时候我们就需要进行状态更新和同步了。这个流程一般是这样的：

* ****检测状态变化****：这个状态变化主要来源于\*syncd服务（netlink等等）和syncd服务（[SAI Switch Notification](https://github.com/sonic-net/sonic-sairedis/blob/master/syncd/SwitchNotifications.h" \t "https://r12f.com/posts/sonic-2-key-components/_blank)），这些服务在检测到变化后，会将它们发送给STATE\_DB或者APPL\_DB。
* ****处理状态变化****：orchagent和\*mgrd服务会监听到这些变化，然后开始处理，将新的配置重新通过命令行和netlink下发给系统，或者下发到ASIC\_DB中，让syncd服务再次对ASIC进行更新。

#### **3.2.3. 具体例子**

SONiC的官方文档中给出了几个典型的控制流流转的例子，这里就不过多的展开了，有兴趣的朋友可以去这里看看：[SONiC Subsystem Interactions](https://github.com/sonic-net/SONiC/wiki/Architecture" \l "sonic-subsystems-interactions" \t "https://r12f.com/posts/sonic-2-key-components/_blank)。

## **4. 容器**

然后，就是SONiC的设计中最具特色的地方：容器化。

从SONiC的上面的设计图中，我们可以看出来，SONiC中，所有的服务都是以容器的形式存在的。在登录进交换机之后，我们可以通过docker ps命令来查看当前运行的容器：

|  |
| --- |
| admin@sonic:~$ docker ps CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES ddf09928ec58 docker-snmp:latest "/usr/local/bin/supe…" 2 days ago Up 32 hours snmp c480f3cf9dd7 docker-sonic-mgmt-framework:latest "/usr/local/bin/supe…" 2 days ago Up 32 hours mgmt-framework 3655aff31161 docker-lldp:latest "/usr/bin/docker-lld…" 2 days ago Up 32 hours lldp 78f0b12ed10e docker-platform-monitor:latest "/usr/bin/docker\_ini…" 2 days ago Up 32 hours pmon f9d9bcf6c9a6 docker-router-advertiser:latest "/usr/bin/docker-ini…" 2 days ago Up 32 hours radv 2e5dbee95844 docker-fpm-frr:latest "/usr/bin/docker\_ini…" 2 days ago Up 32 hours bgp bdfa58009226 docker-syncd-brcm:latest "/usr/local/bin/supe…" 2 days ago Up 32 hours syncd 655e550b7a1b docker-teamd:latest "/usr/local/bin/supe…" 2 days ago Up 32 hours teamd 1bd55acc181c docker-orchagent:latest "/usr/bin/docker-ini…" 2 days ago Up 32 hours swss bd20649228c8 docker-eventd:latest "/usr/local/bin/supe…" 2 days ago Up 32 hours eventd b2f58447febb docker-database:latest "/usr/local/bin/dock…" 2 days ago Up 32 hours database |

这里我们来简单介绍一下这些容器。

### **4.1. 数据库容器：database**

这个容器中运行的就是我们多次提到的SONiC中的中心数据库Redis了，它里面存放着所有交换机的配置和状态信息，SONiC也是主要通过它来向各个服务提供底层的通信机制。

我们通过docker进入这个容器，就可以看到里面正在运行的redis进程了：

|  |
| --- |
| admin@sonic:~$ sudo docker exec -it database bash root@sonic:/*# ps aux* USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND ... root 82 13.7 1.7 130808 71692 pts/0 Sl Apr26 393:27 /usr/bin/redis-server 127.0.0.1:6379 ... root@sonic:/*# cat /var/run/redis/redis.pid* 82 |

那么别的容器是如何来访问这个Redis数据库的呢？答案是通过Unix Socket。我们可以在database容器中看到这个Unix Socket，它将交换机上的/var/run/redis目录map进database容器，让database容器可以创建这个socket：

|  |
| --- |
| *# In database container* root@sonic:/*# ls /var/run/redis* redis.pid redis.sock sonic-db *# On host* admin@sonic:~$ ls /var/run/redis redis.pid redis.sock sonic-db |

然后再将这个socket给map到其他的容器中，这样所有容器就都可以来访问这个中心数据库啦，比如，swss容器：

|  |
| --- |
| admin@sonic:~$ docker inspect swss ...  "HostConfig": {  "Binds": [  ...  "/var/run/redis:/var/run/redis:rw",  ...  ],  ... |

### **4.2. 交换机状态管理容器：swss（Switch State Service）**

这个容器可以说是SONiC中最关键的容器了，****它是SONiC的大脑****，里面运行着大量的\*syncd和\*mgrd服务，用来管理交换机方方面面的配置，比如Port，neighbor，ARP，VLAN，Tunnel等等等等。另外里面还运行着上面提到的orchagent，用来统一处理和ASIC相关的配置和状态变化。

这些服务大概的功能和流程我们上面已经提过了，所以就不再赘述了。这里我们可以通过ps命令来看一下这个容器中运行的服务：

|  |
| --- |
| admin@sonic:~$ docker exec -it swss bash root@sonic:/*# ps aux* USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND ... root 43 0.0 0.2 91016 9688 pts/0 Sl Apr26 0:18 /usr/bin/portsyncd root 49 0.1 0.6 558420 27592 pts/0 Sl Apr26 4:31 /usr/bin/orchagent -d /var/log/swss -b 8192 -s -m 00:1c:73:f2:bc:b4 root 74 0.0 0.2 91240 9776 pts/0 Sl Apr26 0:19 /usr/bin/coppmgrd root 93 0.0 0.0 4400 3432 pts/0 S Apr26 0:09 /bin/bash /usr/bin/arp\_update root 94 0.0 0.2 91008 8568 pts/0 Sl Apr26 0:09 /usr/bin/neighsyncd root 96 0.0 0.2 91168 9800 pts/0 Sl Apr26 0:19 /usr/bin/vlanmgrd root 99 0.0 0.2 91320 9848 pts/0 Sl Apr26 0:20 /usr/bin/intfmgrd root 103 0.0 0.2 91136 9708 pts/0 Sl Apr26 0:19 /usr/bin/portmgrd root 104 0.0 0.2 91380 9844 pts/0 Sl Apr26 0:20 /usr/bin/buffermgrd -l /usr/share/sonic/hwsku/pg\_profile\_lookup.ini root 107 0.0 0.2 91284 9836 pts/0 Sl Apr26 0:20 /usr/bin/vrfmgrd root 109 0.0 0.2 91040 8600 pts/0 Sl Apr26 0:19 /usr/bin/nbrmgrd root 110 0.0 0.2 91184 9724 pts/0 Sl Apr26 0:19 /usr/bin/vxlanmgrd root 112 0.0 0.2 90940 8804 pts/0 Sl Apr26 0:09 /usr/bin/fdbsyncd root 113 0.0 0.2 91140 9656 pts/0 Sl Apr26 0:20 /usr/bin/tunnelmgrd root 208 0.0 0.0 5772 1636 pts/0 S Apr26 0:07 /usr/sbin/ndppd ... |

### **4.3. ASIC管理容器：syncd**

这个容器中主要是用于管理交换机上的ASIC的，里面运行着syncd服务。我们之前提到的各个厂商提供的SAI（Switch Abstraction Interface）和ASIC Driver都是放在这个容器中的。正是因为这个容器的存在，才使得SONiC可以支持多种不同的ASIC，而不需要修改上层的服务。换句话说，如果没有这个容器，那SONiC就是一个缸中大脑，除了一些基本的配置，其他只能靠想的，什么都干不了。

在syncd容器中运行的服务并不多，就是syncd，我们可以通过ps命令来查看，而在/usr/lib目录下，我们也可以找到这个为了支持ASIC而编译出来的巨大无比的SAI文件：

|  |
| --- |
| admin@sonic:~$ docker exec -it syncd bash root@sonic:/*# ps aux* USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND ... root 20 0.0 0.0 87708 1544 pts/0 Sl Apr26 0:00 /usr/bin/dsserve /usr/bin/syncd --diag -u -s -p /etc/sai.d/sai.profile -b /tmp/break\_before\_make\_objects root 32 10.7 14.9 2724404 599408 pts/0 Sl Apr26 386:49 /usr/bin/syncd --diag -u -s -p /etc/sai.d/sai.profile -b /tmp/break\_before\_make\_objects ... root@sonic:/*# ls -lh /usr/lib* total 343M ... lrwxrwxrwx 1 root root 13 Apr 25 04:38 libsai.so.1 -> libsai.so.1.0 -rw-r--r-- 1 root root 343M Feb 1 06:10 libsai.so.1.0 ... |

### **4.4. 各种实现特定功能的容器**

SONiC中还有很多的容器是为了实现一些特定功能而存在的。这些容器一般都有着特殊的外部接口（非SONiC CLI和REST API）和实现（非OS或ASIC），比如：

* bgp：用来实现BGP协议（Border Gateway Protocol，边界网关协议）的容器
* lldp：用来实现LLDP协议（Link Layer Discovery Protocol，链路层发现协议）的容器
* teamd：用来实现Link Aggregation（链路聚合）的容器
* snmp：用来实现SNMP协议（Simple Network Management Protocol，简单网络管理协议）的容器

和SWSS类似，为了适应SONiC的架构，它们中间也都会运行着上面我们提到的那几种服务：

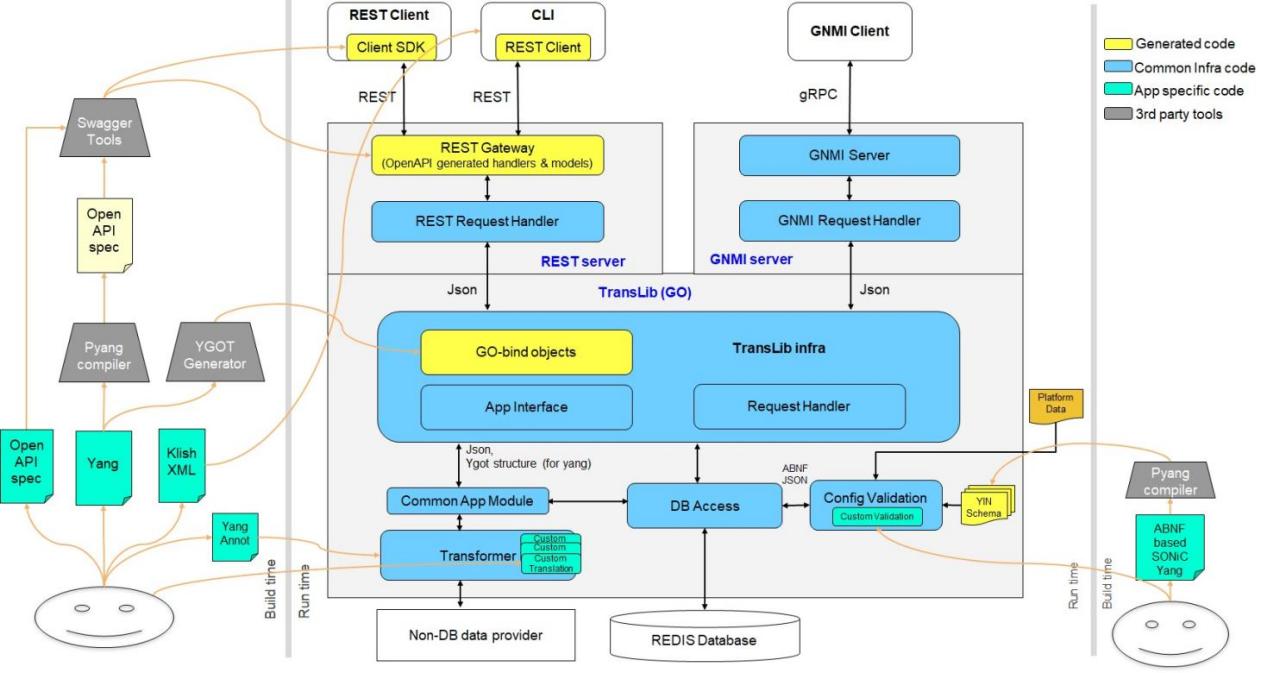
* 配置管理和下发（类似\*mgrd）：lldpmgrd，zebra（bgp）
* 状态同步（类似\*syncd）：lldpsyncd，fpmsyncd（bgp），teamsyncd
* 服务实现或者外部接口（\*d）：lldpd，bgpd，teamd，snmpd

### **4.5. 管理服务容器：mgmt-framework**

我们[在上一篇中](http://r12f.com/posts/sonic-1-intro/)的安装部分已经看过如何使用SONiC的CLI来进行一些交换机的配置，但是在实际生产环境中，手动登录交换机使用CLI来配置所有的交换机是不现实的，所以SONiC提供了一个REST API来解决这个问题。这个REST API的实现就是在mgmt-framework容器中。我们可以通过ps命令来查看：

|  |
| --- |
| admin@sonic:~$ docker exec -it mgmt-framework bash root@sonic:/*# ps aux* USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND ... root 16 0.3 1.2 1472804 52036 pts/0 Sl 16:20 0:02 /usr/sbin/rest\_server -ui /rest\_ui -logtostderr -cert /tmp/cert.pem -key /tmp/key.pem ... |

其实除了REST API，SONiC还可以通过其他方式来进行管理，如gNMI，这些也都是运行在这个容器中的。其整体架构如下图所示 [[7]](https://github.com/sonic-net/SONiC/blob/master/doc/mgmt/Management Framework.md" \t "http://r12f.com/posts/sonic-2-key-components/_blank)：



这里我们也可以发现，其实我们使用的CLI，底层也是通过调用这个REST API来实现的～

### **4.6. 平台监控容器：pmon（Platform Monitor）**

这个容器里面的服务基本都是用来监控交换机一些基础硬件的运行状态的，比如温度，电源，风扇，SFP事件等等。同样，我们可以用ps命令来查看这个容器中运行的服务：

|  |
| --- |
| admin@sonic:~$ docker exec -it pmon bash root@sonic:/*# ps aux* USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND ... root 28 0.0 0.8 49972 33192 pts/0 S Apr26 0:23 python3 /usr/local/bin/ledd root 29 0.9 1.0 278492 43816 pts/0 Sl Apr26 34:41 python3 /usr/local/bin/xcvrd root 30 0.4 1.0 57660 40412 pts/0 S Apr26 18:41 python3 /usr/local/bin/psud root 32 0.0 1.0 57172 40088 pts/0 S Apr26 0:02 python3 /usr/local/bin/syseepromd root 33 0.0 1.0 58648 41400 pts/0 S Apr26 0:27 python3 /usr/local/bin/thermalctld root 34 0.0 1.3 70044 53496 pts/0 S Apr26 0:46 /usr/bin/python3 /usr/local/bin/pcied root 42 0.0 0.0 55320 1136 ? Ss Apr26 0:15 /usr/sbin/sensord -f daemon root 45 0.0 0.8 58648 32220 pts/0 S Apr26 2:45 python3 /usr/local/bin/thermalctld ... |

其中大部分的服务从名字我们就能猜出来是做什么的了，中间只有xcvrd不是那么明显，这里xcvr是transceiver的缩写，它是用来监控交换机的光模块的，比如SFP，QSFP等等。

## **5. SAI**

最后就是SAI（Switch Abstraction Interface，交换机抽象接口）了，它是SONiC的基石，正因为有了它，SONiC才能支持多种硬件平台。我们在[这个SAI API的文档](https://github.com/opencomputeproject/SAI/wiki/SAI-APIs" \t "https://r12f.com/posts/sonic-2-key-components/_blank)中，可以看到它定义的所有接口。

上面我们提到，SAI运行在syncd容器中。不过和其他组件不同，它并不是一个服务，而是一个组动态链接库（.so）。在syncd容器中，SONiC会在启动时启动统一的syncd服务，而syncd服务会加载依赖的SAI组件。这个组件由各个厂商提供，它们会根据自己的硬件平台来实现SAI的接口，从而让SONiC可以支持多种硬件平台。

我们可以通过ps, ls和nm命令来简单的对这个进行验证：

|  |
| --- |
| *# Enter into syncd container* admin@sonic:~$ docker exec -it syncd bash *# List all processes. We will only see syncd process here.* root@sonic:/*# ps aux* USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND ... root 21 0.0 0.0 87708 1532 pts/0 Sl 16:20 0:00 /usr/bin/dsserve /usr/bin/syncd --diag -u -s -p /etc/sai.d/sai.profile -b /tmp/break\_before\_make\_objects root 33 11.1 15.0 2724396 602532 pts/0 Sl 16:20 36:30 /usr/bin/syncd --diag -u -s -p /etc/sai.d/sai.profile -b /tmp/break\_before\_make\_objects ... *# Find all libsai\*.so.\* files.* root@sonic:/*# find / -name libsai\*.so.\** /usr/lib/x86\_64-linux-gnu/libsaimeta.so.0 /usr/lib/x86\_64-linux-gnu/libsaimeta.so.0.0.0 /usr/lib/x86\_64-linux-gnu/libsaimetadata.so.0.0.0 /usr/lib/x86\_64-linux-gnu/libsairedis.so.0.0.0 /usr/lib/x86\_64-linux-gnu/libsairedis.so.0 /usr/lib/x86\_64-linux-gnu/libsaimetadata.so.0 /usr/lib/libsai.so.1 /usr/lib/libsai.so.1.0 *# Copy the file out of switch and check libsai.so on your own dev machine.* *# We will see the most important SAI export functions here.* $ nm -C -D ./libsai.so.1.0 > ./sai-exports.txt $ vim sai-exports.txt ... 0000000006581ae0 T sai\_api\_initialize 0000000006582700 T sai\_api\_query 0000000006581da0 T sai\_api\_uninitialize ... |

## **6. 小结**

好了，到此我们就把所有SONiC的关键组件都简单的过完了！

我们能感觉到，交换机上的组件其实非常的多，而且由于SONiC中Redis的解耦，我们很难简单的对代码进行跟踪来理解服务之间的关系，这就需要我们先建立一个比较抽线的整体模型，然后再去深入的学习每个组件的细节。这也是为什么我们会这里先对每个组件都做一个点到为止的介绍。

后面如果有时间，我们会继续自上向下的来看各个组件的细节。由于篇幅原因，这篇文章就先到这里吧～

## **7. 参考资料**

* [1] [SONiC Architecture](https://github.com/sonic-net/SONiC/wiki/Architecture" \t "https://r12f.com/posts/sonic-2-key-components/_blank)
* [2] [SONiC Roadmap Planning](https://github.com/sonic-net/SONiC/wiki/Sonic-Roadmap-Planning" \t "https://r12f.com/posts/sonic-2-key-components/_blank)
* [3] [SONiC Landing Page](https://sonic-net.github.io/SONiC/index.html" \t "https://r12f.com/posts/sonic-2-key-components/_blank)
* [4] [SONiC Workgroups](https://sonic-net.github.io/SONiC/workgroups.html" \t "https://r12f.com/posts/sonic-2-key-components/_blank)
* [5] [SONiC Supported Devices and Platforms](https://sonic-net.github.io/SONiC/Supported-Devices-and-Platforms.html" \t "https://r12f.com/posts/sonic-2-key-components/_blank)
* [6] [SONiC User Manual](https://github.com/sonic-net/SONiC/blob/master/doc/SONiC-User-Manual.md" \t "https://r12f.com/posts/sonic-2-key-components/_blank)
* [7] [SONiC Management Framework](https://github.com/sonic-net/SONiC/blob/master/doc/mgmt/Management Framework.md" \t "https://r12f.com/posts/sonic-2-key-components/_blank)
* [8] [SAI API](https://github.com/opencomputeproject/SAI/wiki/SAI-APIs" \t "https://r12f.com/posts/sonic-2-key-components/_blank)
* [9] [Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN](http://yuba.stanford.edu/~grg/docs/sdn-chip-sigcomm-2013.pdf" \t "https://r12f.com/posts/sonic-2-key-components/_blank)
* [10] [Github: sonic-net/sonic-sairedis](https://github.com/sonic-net/sonic-sairedis/" \t "https://r12f.com/posts/sonic-2-key-components/_blank)