AppSwitch Documentation

AppSwitch authors

Jun 14, 2018

Getting Started

AppSwitch performs service discovery, access control and traffic management functions on behalf of the applications by transparently taking over the applications' network API calls. In addition, AppSwitch decouples the application from the constructs of underlying network infrastructure by projecting a consistent, virtual view of the network to the application. In abstract terms, it combines the application-level functionality offered by the service mesh approach with the familiarity and compatibility of traditional networking.

Some of the use cases include:

- Automatically curated service registry for seamless service discovery even across hybrid environments.
- Extremely efficient enforcement of label-based access controls without any data path processing.
- Proxy-less load-balancing and traffic management across service instances.
- IP address portability with ability to assign arbitrary IP addresses to applications regardless of underlying network.
- Transparently redirect connection requests to alternate services without using NAT in case of application migration.
- "flat" connectivity with client IP preservation across hybrid network environments.

CHAPTER 1

Functionality Overview

This section walks through a quick demo of AppSwitch functionality.

You can specify any IP address and name you want when bringing up an application. Like this

```
$ ax run --ip 1.1.1.1 nginx -g 'daemon off;'
```

That brings up `nginx` server on 1.1.1.1, no matter the IP address of the host. Note that no interfaces or other network artifacts are created on the host for this to work.

(Please note, AppSwitch currently only supports IPv4. nginx should be configured to listen on IPv4 address only.)

Internally appSwitch runs the application (nginx in this case) in an empty network namespace with no devices and tracks its network API calls through an equivalent of FUSE for network.

You can `curl` the server as follows

```
$ ax run --ip 2.2.2.2 curl -I 1.1.1.1
HTTP/1.1 200 OK
Server: nginx/1.12.2
Date: Tue, 20 Mar 2018 21:21:22 GMT
Content-Type: text/html
Content-Length: 3700
Last-Modified: Wed, 18 Oct 2017 08:08:18 GMT
Connection: keep-alive
ETag: "59e70bf2-e74"
X-Backend-Server: host2
Accept-Ranges: bytes
```

Here curl is given a different IP address. But since it's only acting as a client, you can do without an IP. So this works just as well

```
$ ax run --curl -I 1.1.1.1
```

1.1 Works with names as well

The server can be given a DNS name rather than or in addition to an IP address. Once again, no other changes at the infrastructure level are needed for this to work. The point of is to remove unnecessary interactions between applications and infrastructure.

```
$ ax run --name web nginx -g 'daemon off;'
```

Then you can connect using the name and have `ax` handle the DNS

```
$ ax run -- curl -I web
```

AppSwitch internally makes up an IP address for the purposes of API compatibility but the clients can simply refer to the service by its name without ever knowing about the internal IP address.

1.2 Simple load balancing

You can also bring up another server and give it the same IP address. Don't worry, that won't cause IP or port conflict. Client connections would be automatically load balanced across those servers.

It also works just fine even if the application runs within a container. Just need to pass a couple extra options to "mount" appswitch into the container. In fact, it doesn't matter where the application runs. It could be bare metal, VM, container or somewhere in the cloud as long as there is some type of connectivity underneath.

```
$ docker run -it \
    --net none \
    -v /var/run/appswitch:/var/run/appswitch \
    -v /usr/bin/ax:/usr/bin/ax \
    --entrypoint /usr/bin/ax \
    --cap-add NET_ADMIN \
    --cap-add SYS_ADMIN \
    run --ip 1.1.1.1 nginx -g 'daemon off;'
```

Note that it's an off-the-shelf Docker image with no customizations etc. Also notice `--net none` option. As far as Docker is concerned, the container has no network connectivity. `nginx` gets its network through AppSwitch.

Once the second nginx server is started (as a Docker container, in this case), the client connections would be automatically load balanced across those two instances. You could observe it using `watch`

```
$ watch ax run -- curl -I 1.1.1.1
```

1.3 Security grouping through intuitive labels

You can attach labels to applications while bringing them up under AppSwitch. Among other things, labels are used to enforce isolation. Something like

```
$ ax run --ip 1.1.1.1 --labels role=prod nginx -g 'daemon off;'&
$ ax run --ip 1.1.1.1 --labels role=test nginx -g 'daemon off;'&
```

You can verify that there are multiple servers serving on the same IP address by listing the contents of the *service table* with `ax` command

```
$ ax show srtables servers
NODEID    CLUSTER     APPID     PROTO   SERVICEADDR      IPV4ADDR
-------------------------------------------------------------------------
host0   appswitch     f83e8600    tcp    1.1.1.1:80     10.0.2.15:40010
host0   appswitch     f880063e    tcp    1.1.1.1:80     10.0.2.15:40019
```

At this point, clients only get load balanced to servers that match the labels. For example, the following client with label `role=test` will only connect to the `role=test` server above but never to `role=prod` server even though both carry the same IP address.

```
$ ax --labels role=test curl -I 1.1.1.1
```

A naked client without any labels would not be able access either server.

CHAPTER 2

Installation and Configuration

AppSwitch client and daemon are both built as one static binary, `ax`, with no external dependencies. It is packaged into a docker image for convenience. Installation of the binary (copying to /usr/bin) and bringing up of the daemon can be done by running the following comamnd:

```
curl -L http://appswitch.io/docker-compose.yaml | docker-compose -f - up -d
```

It runs the latest release of AppSwitch docker image through a docker-compose file. The compose file includes most common configuration options as environment variables. Additional options can be passed through AX_OPTS.

```
version: '2.3'

volumes:
  appswitch_logs:

services:
  appswitch:
    image: appswitch/ax
    pid: "host"
    network_mode: "host"
    privileged: true
    volumes:
      - /usr/bin:/usr/bin
      - /var/run/appswitch:/var/run/appswitch
      - appswitch_logs:/var/log
    environment:
      - AX_DRIVER=user # Syscall forwarding driver
      - AX_NODE_INTERFACE= # Node interface to use by daemon.  Accepts IP address or␣
↪interface name, eg eth0
      - AX_NEIGHBORS= # List (csv) of IP addresses of cluster neighbors
      - AX_CLUSTER= # Cluster name.  Required if cluster is part of a federation
      - AX_FEDERATION_GATEWAY_IP= # IP address or `interface` name for federation␣
↪connectivity
      - AX_FEDERATION_GATEWAY_NEIGHBORS= # List (`csv`) of IP addresses of federation␣
↪neighbors (other gateway nodes)
```

(continues on next page)

(continued from previous page)

```
    - AX_OPTS=--clean # Remove any saved state from previous sessions
```

2.1 Ports

AppSwitch uses (by default) the following ports. Please make sure that firewalls don't get in the way of these ports.

6664 *REST Port Number*
7946 Cluster gossip channel (see *Gossip Protocol*)
7947 Federation gossip channel (see *Federation*)
6660 Ingress federation proxy (see *Federation*)

CHAPTER 3

Examples

Here we describe some example usage scenarios for AppSwitch. There is a repository on GitHub, please clone it to follow along with these examples.

```
$ git clone https://github.com/appswitch/examples/
$ cd examples
```

All examples require use of Vagrant to bring up the virtual machines used to form an AppSwitch cluster. Instructions on installing Vagrant can be found on the HashiCorp website

We will be progressively working up to an advanced AppSwitch configuration using all four nodes. The node naming convention is based on the requirements of those use cases. To start with the naming convention is not important but for the interested, the nodes are named `hostXY` where `X` is the cluster number and `Y` is the node number.

Bring up the fist node and ssh into it:

```
$ vagrant up host00
$ vagrant ssh host00
```

The AppSwitch binary (`ax`) is distributed as an image via Docker Hub. The latest image is already pulled as part of provisioning the Vagrant VM.

3.1 Single Node Cluster - Basic AppSwitch Usage

First step is to bring up AppSwitch daemon with the docker-compose file. It includes basic configuration:

```
$ cd appswitch
$ docker-compose -f docker-compose-host00.yaml up -d appswitch
```

Let's start a simple Python based HTTP server using the supplied docker-compose file.

```
version: '2.3'
```

(continues on next page)

(continued from previous page)

```
services:
  http:
    image: python:alpine
    entrypoint: /usr/bin/ax run --name test
    command: python -m http.server
    networks: []
    volumes:
      - /usr/bin:/usr/bin
      - /var/run/appswitch:/var/run/appswitch
```

```
$ docker-compose -f docker-compose-httpserver.yaml up -d httpserver
```

That brings up the server on the specified ip (`1.1.1.1`). Note that the container in which the server runs has no networks. All network connectivity is handled by AppSwitch. Also note that it is an unprivileged container.

Let's check it with `curl`.

```
$ sudo ax run -- curl -I 1.1.1.1:8000
HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.6.5
Date: Wed, 06 Jun 2018 01:37:55 GMT
Content-type: text/html; charset=utf-8
Content-Length: 882
```

The above command is run as root because ax creates a new network namespace to isolate the application. `sudo` won't be needed with `--new-netns=false` option:

```
$ ax run --new-netns=false curl -I 1.1.1.1:8000
```

3.1.1 Run the server without Docker

Let's start another web server. This time natively without a container. We will assign it an IP address of 1.1.1.1 and give it a DNS resolvable name 'webserver'.

```
$ sudo ax run --ip 1.1.1.1 --name webserver python -m SimpleHTTPServer 8000 &
```

Again we can curl the webserver using either the IP address or the name that we have assigned. Since the previous server and this one are assigned the same IP address, incoming requests would get load balanced across the two.

```
$ sudo ax run -- curl -I webserver:8000
```

or:

```
$ sudo ax run -- curl -I 1.1.1.1:8000
```

3.1.2 Connecting to services without AppSwitch AKA 'Ingress Gateway'

Services run by AppSwitch can be exposed externally with the `--expose` option.

```
$ sudo ax run --ip 1.1.1.1 --name webserver --expose 8000:192.168.0.10:10000 python -
↪m SimpleHTTPServer 8000 &
```

We can now connect to the web server using curl without wrapping the command in `ax`

```
$ curl -I 192.168.0.10:10000
HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/2.7.5
Date: Wed, 06 Jun 2018 03:18:02 GMT
Content-type: text/html; charset=UTF-8
Content-Length: 274
```

When starting the web server you can see we have explicitly specified the IP address of the node we wish to expose the service. We could also have used the wildcard address `0.0.0.0` to expose the service on all nodes in the cluster.

If you start two web server processes both with the same name and IP then AppSwitch will load balance when a connection to that name/IP is made.

Virtual Services

If we start the web server without an IP

```
$ sudo ax run -- python -m SimpleHTTPServer 8000
```

And view the app

```
$ ax get apps
                    NAME                        APPID    NODEID   CLUSTER     APPIP
↪DRIVER     LABELS          ZONES
-------------------------------------------------------------------------------------------
↪-----------------------------------
  <9142a421-00e8-483e-83d0-eea9716c849a>  f000015d  host    appswitch  10.0.2.15
↪user    zone=default  [zone==default]
```

We can associate an IP address with this app by creating a virtual service.

```
$ ax create vservice --ip 1.1.1.1 --backends 10.0.2.15  --expose 8000:10000 myvsvc
Service 'myvsvc' created successfully with IP '1.1.1.1'.
$ ax get vservices
  VSNAME  VSTYPE   VSIP       VSPORTS      VSBACKENDIPS
-----------------------------------------------------------
  myvsvc  Random  1.1.1.1  [{8000 10000}]  [10.0.2.15]
```

Now we can curl to the virtual IP or the virtual name. This feature enables multiple IPs for the same server since the server is still available at the IP assigned it by ax. Furthermore, if we start more than one server we can add them all as backends for the virtual service and AppSwitch will load balance when connecting to the virtual name or IP. Currently round-robin and random load balancing strategies are supported.

```
$ sudo ax run -- curl -I myvsvc:8000
$ sudo ax run -- curl -I 1.1.1.1:8000
```

3.2 Multi-Node AppSwitch Cluster

Bring up a second VM so that we can build a 2 node AppSwitch cluster:

```
$ vagrant up host01
```

ssh in and start the AppSwitch daemon

```
$ vagrant ssh host01
host01 $ docker-compose --file docker-compose-host01.yaml up -d
```

Now we can have a play with these two nodes. We can then try curl'ing the server we brought up earlier on host00 from host01.

```
host01 $ sudo ax run -- curl -I webserver:8000
HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/2.7.5
Date: Wed, 06 Jun 2018 03:18:02 GMT
Content-type: text/html; charset=UTF-8
Content-Length: 274
```

Let's restart the webserver on host00, this time passing it the ingress gateway wildcard address

```
host00 $ sudo ax run --ip 1.1.1.1 --name webserver --expose 8000:0.0.0.0:10000 python␣
↪-m SimpleHTTPServer 8000 &
```

From either host we can now directly curl the web server

```
$ curl -I 192.168.0.11:10000
$ curl -I 192.168.0.10:10000
```

3.3 Multi-Node Multi-Cluster AKA AppSwitch Federation

Let us now configure two AppSwitch clusters each consisting of two nodes. As documented in the Vagrant file, the topology looks as follows:

```
#              cluster0                                          cluster1
#     host01              host00                        host10              host11
#
#                         10.0.0.10  -----------------  10.0.0.11
# 192.168.0.11  -----  192.168.0.10                  192.168.1.10  -----  192.168.1.11
```

Bring up the other two VMs:

```
$ vagrant up host10
$ vagrant up host11
```

3.3.1 Configure and Start the Daemon

To configure the nodes for federation connectivity we must configure two of the nodes as federation gateways, we use `host00` and `host10` as the gateways. Also each node must be configured with a cluster name, `cluster0` or `cluster1`.

Start the AppSwitch daemon on each node

```
$ docker-compose --file docker-compose-$(hostname).yaml up -d
```

Now we can have a play with these four nodes. First let's start a web server on `host11`

```
$ sudo ax run --ip 2.2.2.2 python -m SimpleHTTPServer 8000 &
```

We can then try curl'ing this server from host01:

```
$ sudo ax run -- curl -I 2.2.2.2:8000
HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/2.7.5
Date: Wed, 06 Jun 2018 03:18:02 GMT
Content-type: text/html; charset=UTF-8
Content-Length: 274
```

In this case, the client is able to reach the server on the specified IP address (`2.2.2.2`) even though it is in a completely different network, which could have been somewhere in the cloud. AppSwitch is able to flatten the network even across hybrid environments without complex tunneling etc.

CHAPTER 4

AppSwitch Command Line Options

AppSwitch features are provided through the `ax` command.

This guide goes over various options supported by `ax` command thereby describing the related features. The top level commands include:

```
daemon              Start the AppSwitch daemon
run                 Run an application under AppSwitch
get                 Get active resources
create              Create a resource
delete, del         Delete a resource
```

4.1 Daemon

The `daemon` sub command is used to start the AppSwitch daemon.

4.1.1 System Call Forwarding Driver

```
--driver name         System call forwarding driver name ('user' or 'kernel')
```

AppSwitch can be used with either a user-space driver or a kernel-space driver. If the *–driver* flag is not set then the default behavior is to use the kernel driver if the AppSwitch kernel module is loaded. If the module is not loaded then user-space driver is used.

4.1.2 Node Interface

```
--node-interface interface           Node interface to use by daemon. Accepts IP␣
↪address or interface name, eg eth0
```

The host interface that AppSwitch daemon should use to communicate with its peers on other nodes. You can specify the interface by name (eg `eth0`) or by IP address. If unspecified, it defaults to the first interface listed on the node.

4.1.3 Neighbors

```
--neighbors csv           List (csv) of IP addresses of cluster neighbors
```

In order to configure more than one node into an AppSwitch cluster you may provide to each node a comma separated list of IP addresses of neighbor nodes.

```
$ ax daemon --node-interface 192.168.0.2 --neighbors 192.168.0.2,192.168.0.3
```

4.1.4 Clean Start

```
--clean                       Remove any saved state from previous sessions
```

When AppSwitch daemon is shutdown, the state of sockets maintained on behalf of the applications is saved under /var/run/appswitch. If this flag is set, then any saved state from earlier runs of the daemon is removed before starting the daemon. The default behavior is to restore the saved state.

4.1.5 Node Name

```
--node-name name              Name of the node (defaults to host name)
```

Name used to identify the node. Defaults to the host name of the node that the daemon is running on.

4.1.6 Profiling AppSwitch

```
--cpu-profile file   Write CPU profile output to file
--mem-profile file   Write memory profile output to file
```

Golang profiling can be enabled using the `--cpu-profile` and `--mem-profile` options. Each option sets the file name (relative or absolute path) for profiling output. Output file is a binary file that can be viewed using standard `go` tools.

```
$ ax daemon --cpu-profile /tmp/cpu.prof
... Ctl-C

$ go tool pprof /tmp/cpu.prof
```

4.1.7 Domain Name System

```
--dns                 Start the built-in DNS server (default: true)
```

If this flag is set then AppSwitch starts a DNS server on port 53. The built-in DNS server resolves the names associated with the applications run under AppSwitch. This option is required to use the `--name <name>` option to the `run` command.

DNS Configuration Options

```
--dns-domain name     DNS domain name suffix (default: "appswitch.local")
--dns-servers csv     List (csv) of 'IP[:port]' strings of forwarding servers
```

Configuration options for the built-in DNS server. You can optionally specify a DNS suffix and a list of forwarding DNS servers. For example

```
$ ax daemon --dns-servers 127.0.0.1:5533,8.8.8.8
```

4.1.8 Transport Layer Security

AppSwitch can be configured to use Transport Layer Security for internal communication among the daemons.

```
--tls                         Enable Transport Layer Security
```

TLS Configuration Options

```
--tls-ca-cert file            Path to TLS CA certificate file
--tls-cert file               Path to TLS certificate file
--tls-key file                Path to TLS key file
```

Currently AppSwitch needs the absolute path to all files when configuring TLS. If you create a self signed certificate you can start the AppSwitch daemon like this

```
$ ax daemon --tls \
    --tls-ca-cert /etc/ssl/certs/cacert.pem \
    --tls-cert /etc/ssl/certs/ax.crt \
    --tls-key /etc/ssl/private/ax.key
```

4.1.9 Ports

```
--ports csv           List (csv) of ports, or port ranges, reserved for applications␣
↪(default: "40000-60000")
```

AppSwitch binds application sockets to ports on the host from this port space.

```
$ ax daemon --ports '4000,6000-8000'
```

4.1.10 REST Port Number

```
--rest-port number            REST API port number (default: 6664)
```

AppSwitch exposes most of its functionality through the REST API. Most of the the CLI commands are simply a front end to the REST API. This option specifies the port number used for the REST endpoint.

4.1.11 Gossip Protocol

AppSwitch uses Serf as the gossip channel. Serf can be configured with the following options

```
--gossip-port number          Gossip protocol port number (default: 7946)
--gossip-auto-discover        Auto discover neighbors
```

4.1.12 Egress Gateway

```
--egress-gateway              Configure node as egress gateway
```

If this flag is set, connections to external services would be proxied through this daemon. However, the presence of the intermediate egress gateway would be transparent to the client running under AppSwitch. That is, client would directly connect to the external service and not the egress gateway.

4.1.13 Cluster Name

```
--cluster name                Cluster name.  Required if cluster is part of a␣
↪federation
```

Name used to identify the cluster. All cluster names within a federation must be unique. Cluster name is only needed if this node is part of a cluster that will be part of a federation of clusters. Otherwise the default 'appswitch' can be used. All nodes within the cluster should be configured with the same name.

4.1.14 Federation

Multiple AppSwitch clusters may be connected together to form a federation (see *AppSwitch Hierarchical Model*). To achieve this one or more nodes in each cluster must be configured as a federation gateway node. Connections to services from one cluster to another will be made through the federation gateway nodes.

A federation gateway node has two listening services. One, referred to as the egress federation gateway service, accepts connections from other cluster nodes. Data flows out of a cluster via the egress federation gateway service. The second, referred to as the ingress federation gateway service accepts connections on the wide area network from other federation gateway nodes. Data flows into a cluster via the ingress federation gateway service.

Federation Gateway Node Configuration Options

```
--federation-gateway-ip interface             IP address or interface name for␣
↪federation connectivity
--federation-gateway-advertise-ip value       Required iff proxy node is behind NAT
--federation-gateway-port number              TCP port number for federation gateway␣
↪sessions (default: 6660)
--federation-gateway-gossip-port number       Federation gossip protocol port number␣
↪(default: 7947)
--federation-gateway-neighbors csv            List (csv) of IP addresses of federation␣
↪neighbors (other gateway nodes)
```

Please note also; when configuring a federation each and every node must be configured with it's cluster name, and furthermore cluster names must be unique within a federation (See *cluster-label* for details).

4.2 Run

The `run` sub command is used to run an application under AppSwitch.

4.2.1 IP Address

```
--ip address          IPv4 address at which services of this application would be␣
↪reachable
```

The specified IP address is associated with the application. When an AppSwitch-managed client connects to the IP address, it would be automatically directed to the services of this application. To achieve that, a *vservice* is implicitly created. The same IP address could be used for other applications, in which case, all those applications become backends for the vservice.

4.2.2 DNS Resolvable Name

```
--name value          DNS resolvable name of the application
```

The specified name is associated with the application. When an AppSwitch-managed client looks up this name, it is resolved to the IP address associated with the application by AppSwith daemon's built-in DNS server.

4.2.3 Labels

```
--labels csv          Labels of this application (default: "zone=default")
```

Allows arbitrary labels of the form `label=value` to be associated with the application. This option accepts a comma separated list of labels all of which will be associated with the application. Accepts arbitrary string values for both 'label' and 'value'. A client would be able to reach a service only if they share at least one matching label. For example, a client with a label `role=test` cannot connect to a service with a label `role=prod` or one without any labels.

4.2.4 Exposed Ports

`--expose` option is used to expose an internal application port on the cluster node(s) such that the service can be accessed by external non-AppSwitch clients. There are three variations of it:

```
--expose internal-port:host-port
```

The specified application port would be exposed on the specified external port only on the node where the application is running.

For example, a python web server (port 8000) can be exposed on external port 9999 as follows:

```
$ ax run --expose '8000:9999' python -m http.server
$ curl -I 192.168.0.2:9999
HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.5.2
Date: Mon, 30 Apr 2018 05:23:33 GMT
Content-type: text/html; charset=utf-8
Content-Length: 2377
```

```
--expose internal-port:<node-IP>:node-port
```

The specified application port would be exposed on the specified external port only on the specified node. The specified node-IP must belong to one of the nodes in the AppSwitch cluster.

```
--expose internal-port:0.0.0.0:node-port
```

The specified application port would be exposed on the specified external port on every node in the AppSwitch cluster. This is equivalent to the nodePort feature of Kubernetes. A similar result can also be produced by creating an external *vservice*.

4.2.5 User

```
--user name          UID or user name to run the child process
```

When the client runs an application it is run by default as the same user that invoked AppSwitch. If AppSwitch is run as root (which is required to create a new network namespace) then the application being run will be run as root. This is often *not* the desired behavior. Using the `--user` option the name or UID of a valid user can be given to the client and the application being run will be run as that user.

```
$ ax run -- whoami
root

$ ax run --user alice whoami
alice
```

4.2.6 Interface Name

```
--interface name     Name of the dummy interface created within the application's␣
↪network namespace
```

Some applications require the presence of a non-loopback network interface in order to function. AppSwitch places the application in a new network namespace by default. With this option, a dummy interface with the specified name can be created in the new network namespace before the application is executed. A new network namespace has, by default, only the loopback interface. This option requires that `--no-new-netns` flag is not used.

```
$ ax run -- ip addr show
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN group default qlen 1000
 link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

```
$ ax run --interface eth0 ip addr show
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN group default qlen 1
 link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN group␣
↪default qlen 1000
 link/ether d2:a0:cb:e5:b0:33 brd ff:ff:ff:ff:ff:ff
 inet 192.168.178.2/32 scope global eth0
    valid_lft forever preferred_lft forever
 inet6 fe80::d0a0:cbff:fee5:b033/64 scope link
    valid_lft forever preferred_lft forever
```

4.2.7 Network Namespace

```
--new-netns          Create a new network namespace (default: true)
```

Each application run by AppSwitch is run in a separate namespace. Creation of a new namespace is handled by AppSwitch by default. Sometimes this behavior is not required, for example when running within a Docker container. Note that creating a new network namespace also requires privilege. To prevent from new network namespace from being created `--new-netns=false` can be used.

4.2.8 DNS Override

```
--dns-override       Take over application's DNS requests (default: true)
```

This option overrides existing resolv.conf file for the application with one that points to the built-in DNS server by mounting over it. Host is not affected by this. To prevent dns override use `--dns-override=false`.

4.3 Get

The `get` command is used to display current AppSwitch resources.

Examples:

- The IP address of the host machine is 192.168.178.2
- The daemon was started with: `ax daemon --node-name node1`
- Two AppSwitch client instances were started - `ax run -- nc -l 6000` - `ax run -- iperf3 -s`

4.3.1 `ax get apps`

Displays information about applications currently running under AppSwitch

```
                  NAME                      APPID    NODEID   CLUSTER        APPIP
↪DRIVER     LABELS          ZONES
-------------------------------------------------------------------------------------
↪----------------------
<ab856b81-7db0-4d88-8a1e-1bfbf0c5fe9f>  f00001bb  node1   appswitch   192.168.178.2
↪user    zone=default  [zone==default]
<04a275bc-b9b4-4496-9c9d-a838daecdffb>  f000028e  node1   appswitch   192.168.178.2
↪user    zone=default  [zone==default]
```

4.3.2 `ax get servers`

Shows information about currently running services.

```
NODEID   CLUSTER     APPID    PROTO     SERVICEADDR           IPV4ADDR
-------------------------------------------------------------------------
node1   appswitch   f00001bb  tcp    192.168.178.2:6000  10.0.23.11:40000
node1   appswitch   f000028e  tcp    192.168.178.2:5201  10.0.23.11:40001
```

SERVICEADDR above represents the virtual IP address where the service is available to AppSwitch-managed clients and the IPV4ADDR represents the host IP and port where the service is actually bound.

4.3.3 ax get sockets

Displays socket information for currently running applications

```
                  ID                    NODEID    APPID    INODE  PROTO  FLAGS     ␣
↪BINDIP      BACKLOG
-------------------------------------------------------------------------------------
↪--------------
4daa64c4-2091-46e0-8a67-5428fae9775d  node1   f00001bb  829    tcp    0      0.0.0.
↪0:6000   1
f6dd4f27-bb4e-4c4a-a076-dc07c29af7be  node1   f000028e  833    tcp    0      0.0.0.
↪0:5201   5
```

4.3.4 ax get proxies

Displays current proxies. Example listing is off a node configured to be a federation gateway node (see *Federation* for details).

```
  ID  PROTO       LISTENER          DIALERS
-------------------------------------------
  1   tcp    10.0.0.10:6660      [0.0.0.0:0]
  2   tcp    192.168.0.10:36869  [0.0.0.0:0]
```

4.4 Create

Create command is used to create a resource. Resource is a general construct that represents a particular AppSwitch feature.

Currently supported resources are:

```
vservice                Create a virtual service
```

4.4.1 vservice

A virtual service (vservice) is a virtual-IP:virtual-port combination that acts as a load balancing front end to a set of backend services. Backend services consist of services listed in the service table or external services specified as IP:port pairs. The vIP, vPort and the IP:Ports of the backend services are specified by the user.

The following options are provided by vservice command.

```
--ip value            IPv4 address for the virtual service
--external            Make this vservice external
```

A virtual service can be marked external. In that case, in addition to creating the vservice with the specified vIP and vPort, the virtual service represented by the load balanced backend services is exposed on the vPort on all nodes of the cluster.

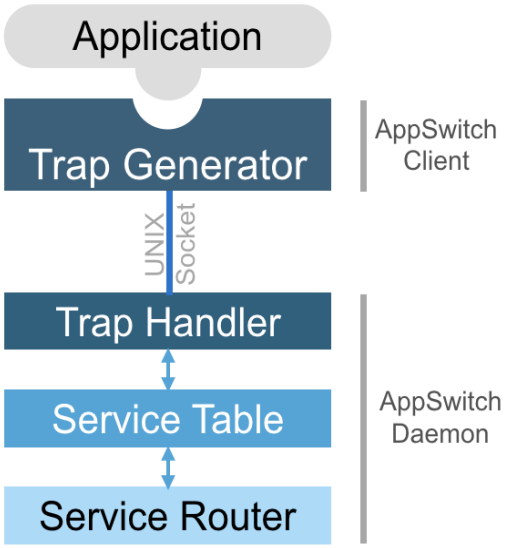
```
--lbtype value        Load Balancer Type <possible values: Random, RoundRobin>␣
↪(default: "Random")
--backends value      Comma separated list of IPv4 IPs
--ports value         Comma separated list of <virtual port:application port>. The␣
↪service will be made available on all cluster nodes on virtual port
--source-ip           Host IP address to use when making the outbound connection
```

4.5 Delete

CHAPTER 5

Architecture

AppSwitch consists of two components, a client and a daemon. One instance of daemon runs per host (physical / virtual machine) and exposes an interface that the client uses. The client transparently plugs into the application and tracks the network system calls it makes such as socket(), connect(), bind(), accept() and forwards them to the daemon over a UNIX socket. Client is not a separate process or a thread. Rather it directly runs in the context of the application itself. The daemon handles the system calls forwarded to it from applications based on the policy.



5.1 Service Table

AppSwitch provides a logically centralized data structure called *service table* that maintains a record of all currently running services across the cluster. The table is automatically updated as services come and go.

When an application calls listen() system call, a new service is automatically added to the service table along with a set of system and user-defined attributes. The service attributes include app-id, name (if specified), protocol, app-ip:app-port, host-ip:host-port and labels.

Clients running in an AppSwitch cluster would be able to access the services listed in the server table by simply connect()ing to the app-ip:app-port listed in the service table entry. AppSwitch transparently performs service discovery, access control and traffic management based on specified policy and the contents of the service table and appropriately directs the connection.

5.2 Service Router

Service router propagates the information about services as they come and go to other instances of AppSwitch daemons across hosts and clusters. It provides flexible ingress, egress, and federation capabilities such that applications running on an AppSwitch cluster can communicate with non-AppSwitch external entities or to communicate across clusters.

5.3 AppSwitch Hierarchical Model

AppSwitch model has the following hierarchy.

Federation:

- Contains a set of clusters.

Cluster:

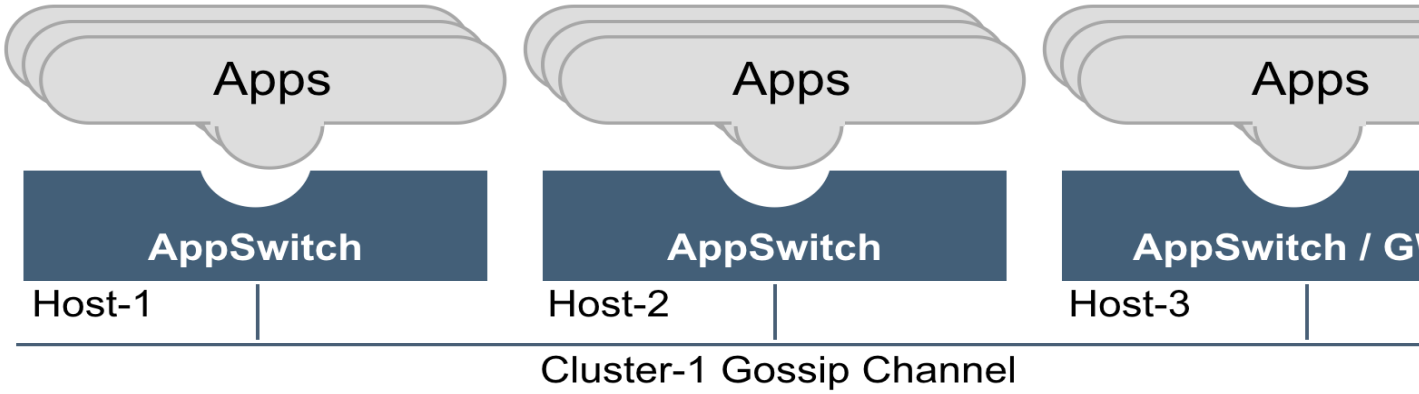
- Contains a set of nodes with mutual IP connectivity.
- A cluster may have one or more federation gateways.
- Federation gateways across clusters peer with each other over a federation gossip channel.

Node:

- Contains applications managed by AppSwitch.
- Each node runs an AppSwitch daemon instance. Daemons across nodes peer with each other over the cluster gossip channel.

Application:

- Contains zero or more client / server sockets, maintained by the AppSwitch daemon on behalf of the application.


Apps
Apps
Apps
AppSwitch
AppSwitch
AppSwitch / G
Host-1
Host-2
Host-3
Cluster-1 Gossip Channel

CHAPTER 6

Integrations

This guide describes AppSwitch's integration with Kubernetes and Istio.

6.1 Kubernetes

AppSwitch can serve as the network backend for Kubernetes. In a Kubernetes cluster, AppSwitch can be deployed with the DaemonSet spec like below. The only required customization before creating the DaemonSet is `args`: a comma-separated list of one or more of existing nodes in the AppSwitch cluster (Eg., `args: ["--neighbors", "<ip1>,<ip2>"]`). For simplicity and availability, all nodes in the Kubernetes cluster shown by `kubectl get nodes` can be added.

```
---
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: kube-appswitch
  namespace: kube-system
  labels:
    k8s-app: appswitch
spec:
  selector:
    matchLabels:
      name: kube-appswitch
  template:
    metadata:
      labels:
        name: kube-appswitch
    spec:
      hostNetwork: true
      hostPID: true
      containers:
      - name: appswitch-daemon
        image: appswitch/ax
```

(continues on next page)

(continued from previous page)

```
        args: ["--neighbors","<ip1>,<ip2>"]
        resources:
          requests:
            cpu: "200m"
            memory: "200Mi"
          limits:
            cpu: "200m"
            memory: "200Mi"
        securityContext:
          privileged: true
        volumeMounts:
        - name: bin
          mountPath: /usr/bin
        - name: sock
          mountPath: /var/run/appswitch
      volumes:
        - name: bin
          hostPath:
            path: /usr/bin
        - name: sock
          hostPath:
            path: /var/run/appswitch
```

Then the DaemonSet itself can be created as follows.

```
$ kubectl create -f kube-daemonset.yaml
```

At this point, the Kubernetes cluster is primed to run applications based on AppSwitch. The pod spec needs a few small changes to use AppSwitch. This sample nginx pod spec illustrates those changes:

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx
    command: ['/usr/bin/ax','run','--ip','1.1.1.1',nginx, '-g', 'daemon off;']
    ports:
    - containerPort: 80
    volumeMounts:
      - name: bin
        mountPath: /usr/bin/ax
      - name: run
        mountPath: /var/run/appswitch
  volumes:
    - name: bin
      hostPath:
        path: /usr/bin/ax
    - name: run
      hostPath:
        path: /var/run/appswitch
```

This creates an nginx pod enabled to run with AppSwitch. The changes from a typical nginx pod spec are:

- `command` is prefixed with `ax` and its parameters

- `/usr/bin/ax` and `/var/run/appswitch` host paths are mounted into the container

The application can be deployed to the cluster the normal way.

```
$ kubectl create -f nginx.yaml
```

Once deployed, you can connect to this *nginx* pod from any node in the cluster as follows.

```
$ ax run --ip 2.2.2.2 curl -I 1.1.1.1
HTTP/1.1 200 OK
Server: nginx/1.12.2
Date: Tue, 20 Mar 2018 21:21:22 GMT
Content-Type: text/html
Content-Length: 3700
Last-Modified: Wed, 18 Oct 2017 08:08:18 GMT
Connection: keep-alive
ETag: "59e70bf2-e74"
X-Backend-Server: host2
Accept-Ranges: bytes
```

6.2 Istio

AppSwitch is integrated with Istio to serve as a highly efficient dataplane through the AppSwitch Istio agent that consumes Pilot (XDS) API and conveys traffic management policies to AppSwitch.

CHAPTER 7

References and Blogs

- AppSwitch: Resolving the Application Identity Crisis
- Test drive of AppSwitch, the “network stack from the future”
- So, What’s this AppSwitch Thing Going Around

CHAPTER 8

Indices and tables

- genindex
- modindex
- search