

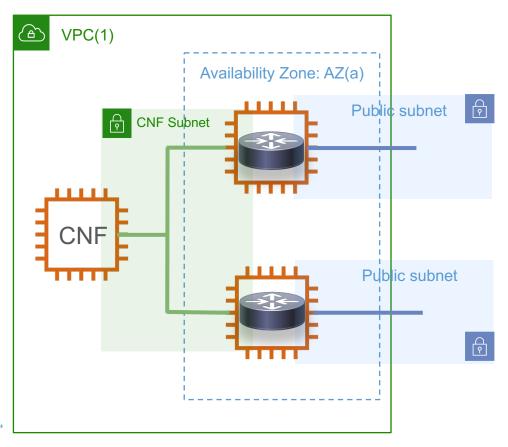
# XRv9k on AWS High Availability (HA) Solution

Akshat Sharma, TME May 2021





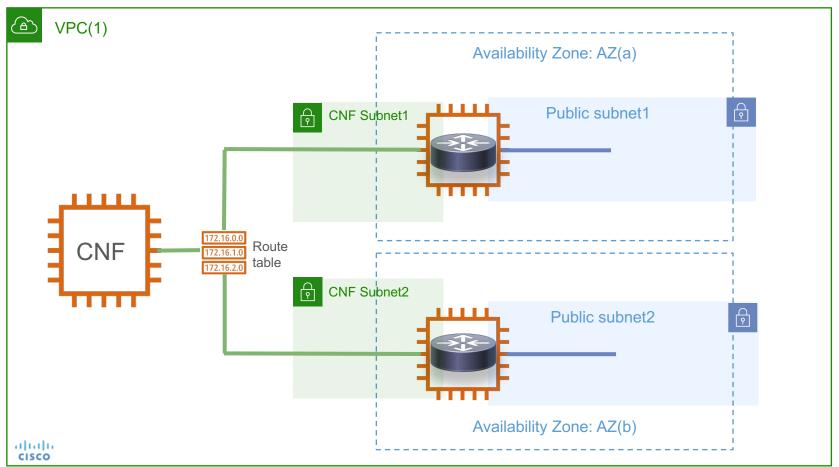
## vRouter HA In Single AZ



CNF = 5G Cloud-Native
Network Functions

Both routers in the same AZ – b2b interfaces in same subnet.

#### vRouter HA Across Two AZ



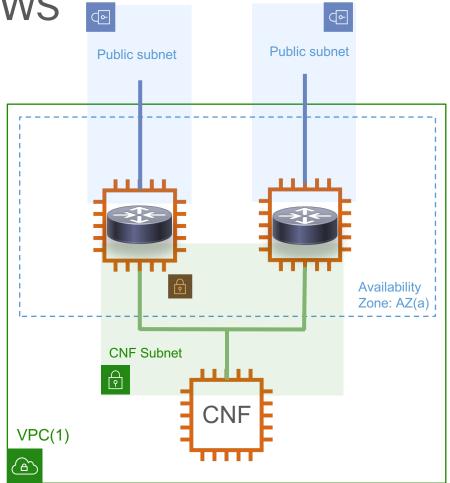
# Why don't we just use HSRP?



## Problems using HSRP on AWS

- AWS as an underlay imposes some limitations on the redundancy design
- AWS underlay blocks multicast/broadcast packets
- HSRP/VRRP hellos are sent to multicast addresses – blocked
- Further HSRP depends on the use of gratuitous ARP packets that are dropped as well.
- Active/Standby Selection and Failover Notifications therefore require some tricks on the vRouter and AWS level to accomplish sub-second failovers

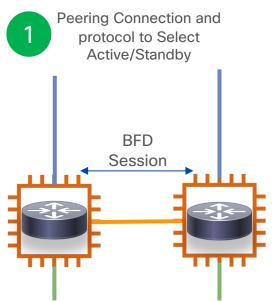




# AWS vRouter HA Redundancy Design Principles



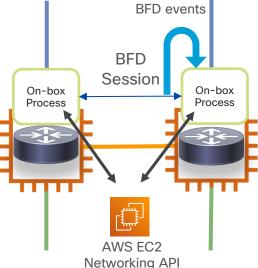
# Components of the Redundancy Design



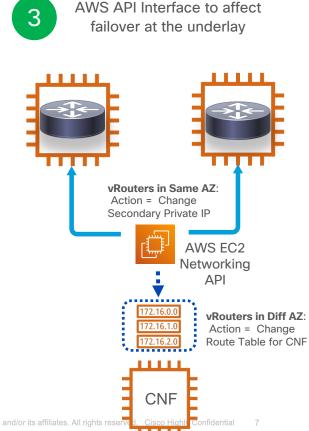
- Regular BFD session (Single-hop BFD for single AZ or Multi-Hop for Diff AZ with local routing in VPC)
- BFDoGRE session (Single-Hop over GRE for both AZ deployments)

adiada CISCO

Process on vRouter to Listen to BFD events, and trigger actions BFD events **BFD** 



- On-box Process listens to BFD events
- Maintains Active/Standby State in relation to the Peer
- Calls AWS API to relevant action



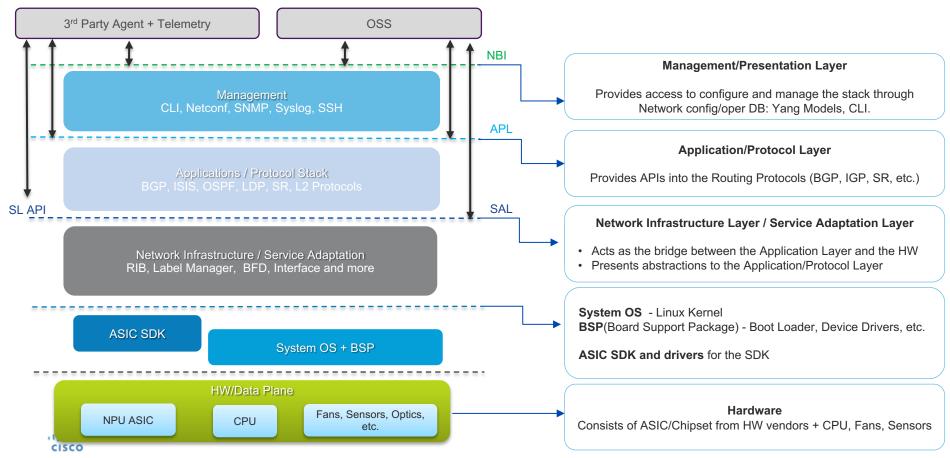
**Peering Connection and** protocol to Select Active/Standby

Process on vRouter to Listen to BFD events, and trigger actions

# IOS-XR SL-API to Setup BFD and Monitor it



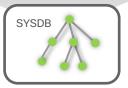
# IOS-XR: API-Driven, Layered SW Architecture



## Service Layer API Architecture

CLI, Yang Models, Streaming Telemetry

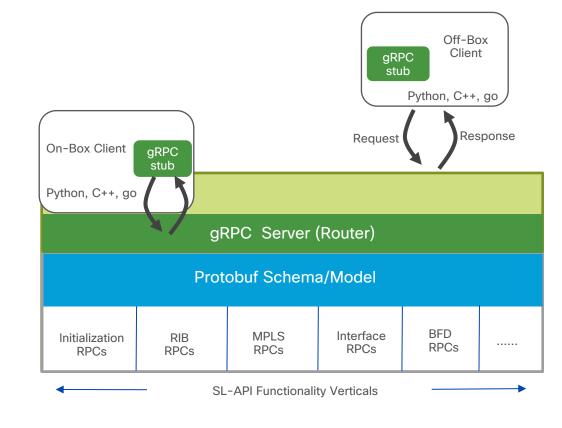
Manageability Layer



Service Layer API

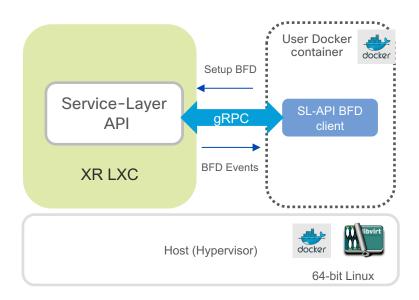
Network Infrastructure Layer (Service Layer)

Service-Layer APIs bypass SysDB giving higher performance





### Max Performance with OnBox SL-API BFD-HA App



- XRv9000 Container Based AppHosting
  - cisco

- Using Xrv9000 Application-hosting capabilities, an SL-API BFD client will be spun up in a Docker container on the vRouter itself.
- BFD sessions are normally associated with a Routing Protocol "Client" like BGP, OSPF, ISIS etc.
- 3) With Service-layer API- "Service-Layer" acts as client for BFD allowing us to add/modify BFD sessions directly WITHOUT ANY IOS-XR CONFIG. BFD session state is transient/ephemeral - tied to the client.
- 4) Real-time BFD events are then received over gRPC, allowing the SLAPI client to react to Peer-router unreachable/down events



Peering Connection and protocol to Select Active/Standby

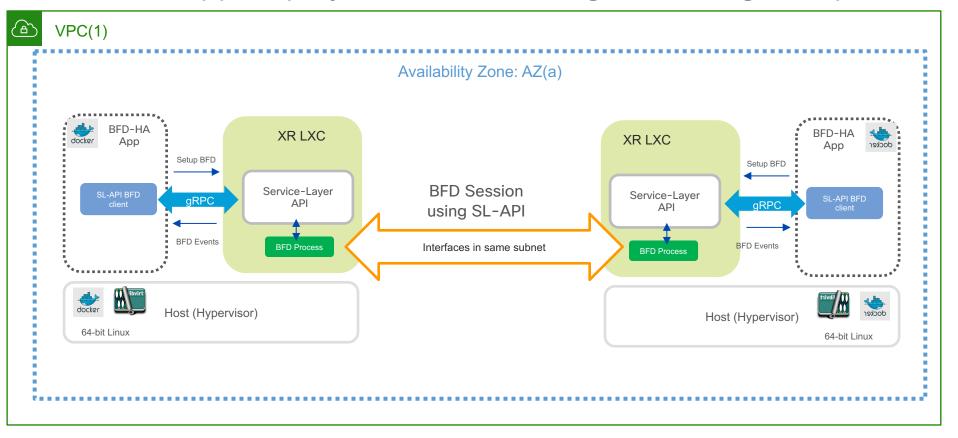


Process on vRouter to Listen to BFD events, and trigger actions

# Deploying the OnBox BFD-HA App

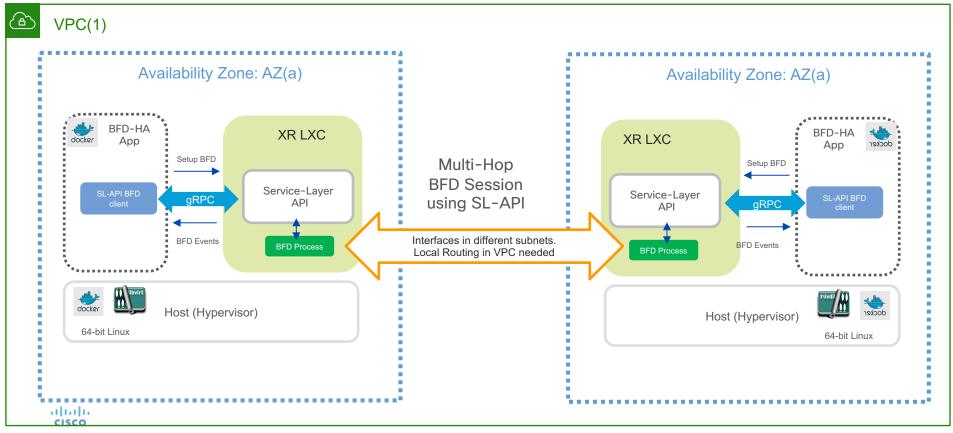


### BFD-HA App Deployment Model: Single AZ, Single-Hop BFD

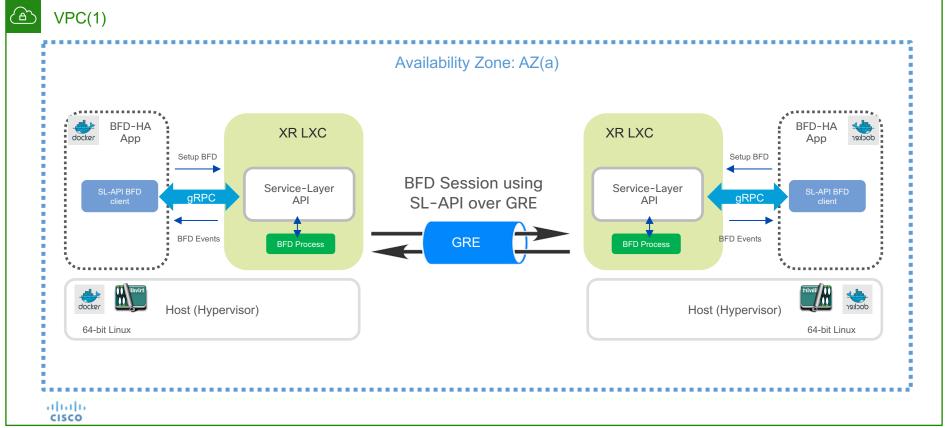




### BFD-HA App Deployment Model: Different AZ



### BFD-HA App Deployment Model: using BFDoGRE (Future)



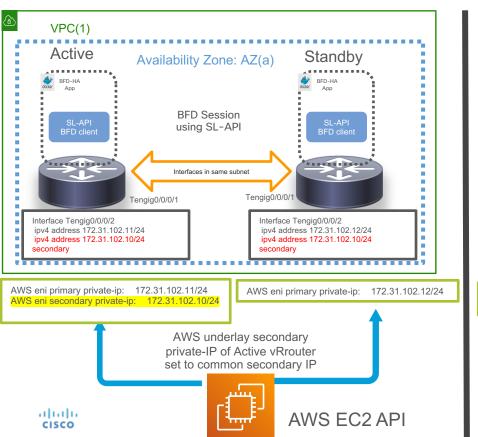
AWS EC2 Networking API to affect failover at the underlay

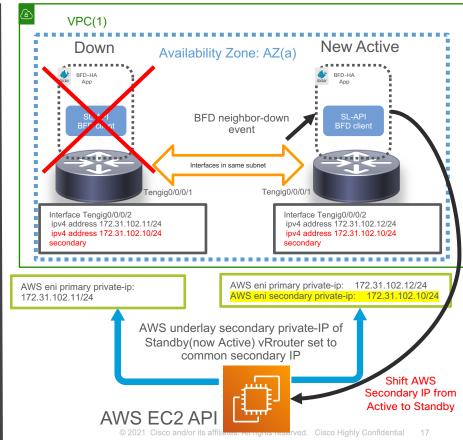


# Triggering Failover

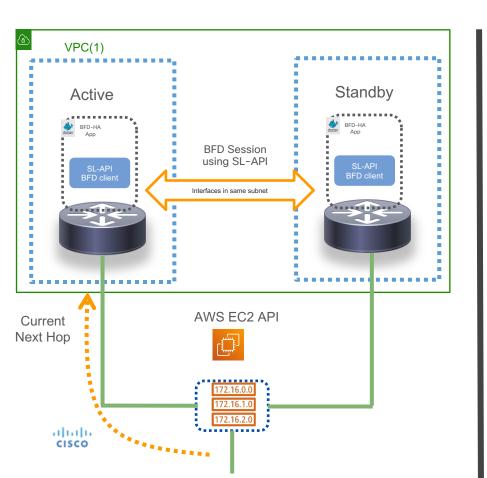


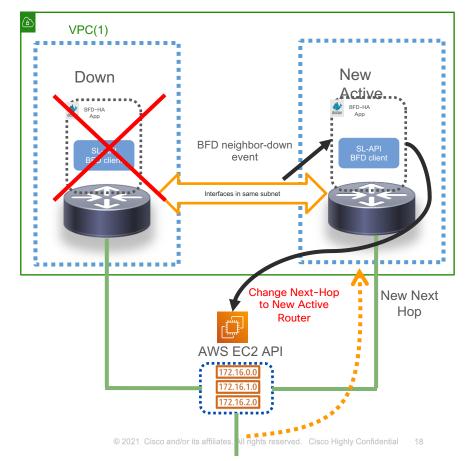
# Single AZ: BFD HA App detects BFD event and triggers Secondary IP reassignment





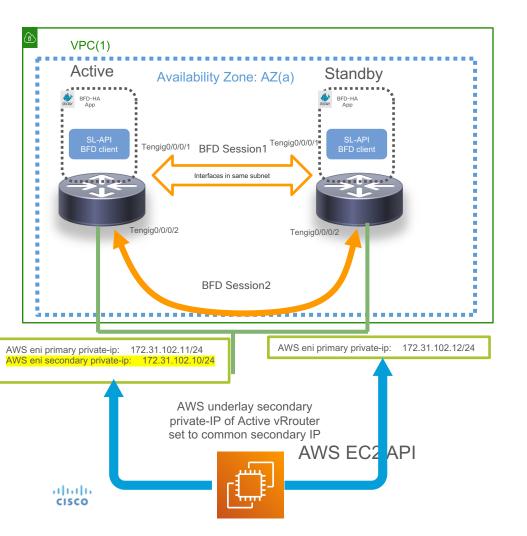
#### Different AZ: BFD HA App detects BFD event and triggers Route-Table change





# BFD Sessions – where do they run?





#### BFD session 1:

- Running over the Private-HA subnet (Not carrying traffic)
- Useful to detect complete failure of the active router
- Cannot be used to detect flap/failure of the Ingress interface on Active (without router failure)
- Any failover measurements with this session (using intf shut/no-shut) =

Time taken for AWS underlay to switch traffic when secondary IP is updated.

#### BFD session 2:

- Running over the Ingress Subnet (where secondary-IPs are assigned)
- Useful to detect complete failure of the active router
- Also useful to detect ingress interface failure.
- Any failover measurements with this session (using intf shut/no-shut) =

Time taken for BFD Event to be received by App

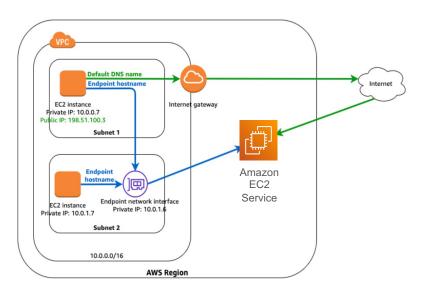
Time taken for App to invoke AWS API via endpoint service

Time taken for AWS underlay to switch traffic when secondary IP is updated.

# Accessing AWS API from onBox HA App



# Using AWS private-link (Interface Endpoint Service)

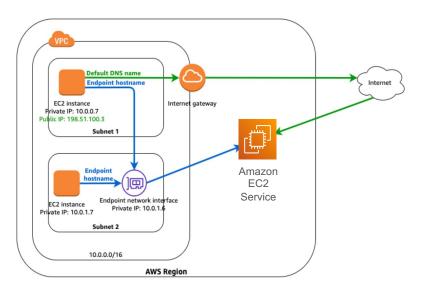


- Private Interface Endpoint Service is created so that routers can access EC2 API without internet access
- The endpoint URL utilized by the router HA App is obtained from the interface endpoint Service configuration
- Sample boto3 resource creation for interface endpoint service:

```
resource = boto3.resource(
    service name='ec2',
   endpoint url='https://vpce-0786bfdf8aad8840c-
vacmlrog-us-west-2a.ec2.us-west-2.vpce.amazonaws.com',
    aws_access_key_id=ACCESS_KEY,
    aws_secret_access_key=SECRET_KEY,
    aws session token=SESSION TOKEN,
    region name=REGION NAME
```



# Using AWS private-link (Interface Endpoint Service)



#### Authentication:

- IAM role attached to router instances.
- App downloads the temporary credentials using metadata URLs

```
headers = {
    'X-aws-ec2-metadata-token': response.text,
response = requests.get('http://169.254.169.254/latest/meta-
data/iam/security-credentials/ec2access', headers=headers)
ACCESS KEY=response.json()["AccessKeyId"]
SECRET KEY=response.ison()["SecretAccessKev"]
SESSION TOKEN=response.json()["Token"]
```



### Additional Considerations

- The BFD-HA app will accept input parameters associated with the API call in a JSON file format
- The HA state of each router is determined independently by each router without peerpeer messaging by checking the required end state of the AWS underlay (Whether secondary-ip of own interface = intended secondary IP specified in json config).
- Because of independent HA state determination split-brain scenarios do not arise.



#### Initial Test Results using Python HA App

est Results:								
	Iteration	HA trigger Interface (shutdown)	Packet Size (bytes)	Python SLAPI-AWS client (Draft)			Ping Interval	Failover Time
				BFD event (BFD configured with interval=50ms and multiplier=3) (ms)	AWS EC2 call completion (ms)	Packet Loss (Packets)	(ms)	(ms)
		Private HA Interface	64	152	597	75	10	750
	1	TenGigE0/0/0/1	800	99	549	8	10	80
		Private HA Interface	64	147	519	14	10	140
	2	TenGigE0/0/0/1	800	130	700	22	10	220
			000	150	700		10	220
	3	Private HA Interface	64	152	545	58	10	580
	3	TenGigE0/0/0/1	800	177	587	40	10	400
	4	Private HA Interface	64	128	571	45	10	450
		TenGigE0/0/0/1	800	137	620	34	10	340
		Private HA Interface	64	146	667	14	10	140
	5	TenGigE0/0/0/1	800	174	788	50	10	500
		0-1,1,1	800	1/4	700	30	10	300
		Private HA Interface	64	119	550	16	10	160
	6	TenGigE0/0/0/1	800	117	598	60	10	600
			Average>	139.8333333	607.5833333	36.33333333		363.3333333
			Average>	139.6333333	007.3633333	30.3333333		303.333333
		Downstream		171	627	98	40	980
	7	(standby IP associated)	64	1/1	627	98	10	980
		TenGigE0/0/0/2	800	105	653	98	10	980
	8	Downstream	64	143	633	117	10	1170
		(standby IP associated)						
		TenGigE0/0/0/2	800	109	620	132	10	1320
	9	Downstream (standby IP associated)	64	120	544	117	10	1170
		TenGigE0/0/0/2	800	156	674	89	10	890
			800	130	0/4	83	10	890
		Downstream						
	10	(standby IP associated)	64	168	634	73	10	730
		TenGigE0/0/0/2	800	143	632	72	10	720
	11	Downstream	64	161	572	113	10	1130
		(standby IP associated)						
		TenGig0/0/0/2	800	125	491	60	10	600
	12	Downstream (standby IP associated)	64	118	594	64	10	640
	12	(standby IP associated) TenGig0/0/0/2	***	***			40	0.50
			800	118	605	96	10	960

BFD Events are detected by App in about 139ms (TimeA)

Call to AWS API via private endpoint service takes about 600ms (TimeB)

Time taken by AWS underlay to failover traffic (post API call): Approx 350ms (TimeC) – very volatile.

Average HA failover time (Measured via PING traffic drop: average 940ms

HA failover time comes out to be approximately TimeA+TimeB+TimeC

# cisco