Hosting Applications on IOS XRv 9000 router for AWS

You can use an On-box docker container on Cisco IOS XRv 9000 router, and host the HA redundancy application within the container to run HA redundancy application on IOS XRv 9000 router for AWS.

Table 1: Feature History Table

Feature Name	Release	Description
Running High Availability (HA) redundancy application on Cisco IOS XRv 9000 router for AWS	Release 7.3.3	This feature lets you run a High Availability (HA) redundancy application that is hosted in an On-box docker container on the Cisco IOS XRv 9000 router for AWS. This feature enables you to have a mechanism for the virtual router to switch over from the active router to the standby router in case of failure. So, if the routing goes down, then you need to detect that failure. You can then initiate a failover to the standby router so that the traffic that was coming into the active router can be diverted to the standby router.

Verify the following requirements before you host an application on a device:

- Suitable build environment to build your application.
- A mechanism to interact with the device and the network outside the device.

Running High Availability redundancy application on Cisco IOS XRv 9000 router for AWS

This section describes having a High Availability (HA) redundancy application that runs on Cisco IOS XRv 9000 router, and you need to have a mechanism for the virtual router to switch over from the active router to the standby router in case of failure. So, if the routing goes down (for example, the router fails, it stops forwarding traffic) then you need to detect that failure, and then initiate a failover to the standby router so that the traffic that was coming into the active router can be diverted to the standby router.

The advantages are:

- Redundancy: For any type of signaling traffic, you must need a redundant gateway solution. The advantage of a redundancy solution is, if one router goes down, the other router continues to send the traffic. So that the drop of traffic is minor.
- Failover: With the failover, only the active router passes traffic while the other router waits in a standby state. If an active router fails, the standby router immediately becomes the active router with little or no delay.

Amazon Web Services (AWS) is a public cloud where you are running a virtual machine on top. Both the active and standby routers running on AWS are available in the same availability zone and they have the same subnet. As a result, both the routers can communicate with each other. You can then set up a system on AWS where this failover can be triggered.

For example, router A is active initially. So all the traffic flows through this active router and you activate or run the application on router A. Spin up the same application on standby router B. The application then uses the service layer API to interact with the BFD process and start the BFD session with the standby router.

Both the routers have the same primary IP address which is unique. Router A also has the secondary IP address initially.

Note Typically, in AWS, the packets are handled by AWS directly. So, the routers do not detect that the same standby IP is configured.

So, when the traffic comes to the router A interface, the traffic destination typically goes through the secondary private IP here. That means only the router that has the secondary private IP configured on the AWS underlay, expects the traffic to come in. That is the router that has the secondary IP configured on the AWS underlay. Both the routers communicate using BFD.

And when this router A goes down or the BFD session goes down, then the session down event is sent to the application running on the router which detects the failover event, use AWS API to connect to AWS for shifting the secondary IP from the active router A to the standby router B. So, when the secondary IP is shifted from active to standby, then the traffic is diverted from the active router A to the standby router because the secondary IP is in the standby router B. So, the standby router B becomes the active router now.

Note If the secondary IP is associated with one router, that is active, and the other one is standby.

To know more about the HA redundancy application and how it works, refer to the HA Solution document.

Configuring the On-box Docker container to enable HA application

The section describes configuring the On-box docker container to enable HA application on Cisco IOS XRv 9000 router using IOS-XR Service-Layer API and AWS API. You must perform these tasks in sequence as listed.

Restrictions

- The HA redundancy application works only on AWS.

  HA application works only when the following functions are supported:

  - Application hosting: The functionality of application hosting allows you to run the application on AWS cloud. You can create your own container on IOS XR, and an application in a separate container. The applications can be developed using any Linux distribution. To know how the app hosting functionality works, see Application Hosting Configuration Guide for Cisco ASR 9000 Series Routers.
  - Service layer API: Service Layer API is a model-driven API over Google-defined remote procedure call (gRPC). It gives you a gRPC server running on the router. You can enable it using a CLI in XR and you can create a client of the service layer API.

    For this feature, we are using the on-box client as an application runs as a docker container on the same router and it communicate s with the gRPC server. The application uses the service layer API in XR to create a BFD session. As a result, the application can trigger the setup of the BFD session and then be able detect events when the BFD session goes down. To know how the service layer API works, see Use Service Layer API to Bring your Controller on Cisco IOS XR Router or Cisco IOS-XR Service Layer.
  - Application Manager: The functionality of application manager allows you to restart all the third-party applications automatically after a router reload or an RP switchover. This process ensures seamless functioning of the hosted applications. The Docker daemon service starts on the router only if you configure a third-party hosting application using the **appmgr** command. Such an on-demand service optimizes operating system resources such as CPU, memory power, and power.

- Hot Standby Router Protocol (HSRP) does not work on AWS.

The task also provides you the topics available in the High-Availability GitHub page for HA page. Refer this page to see the detailed configuration examples.

Prerequisites

Before configuring the On-box Docker container to enable HA application, perform the following actions:

- You must have installed Docker on your build server or on your system. For more information, see Install Docker Engine.
- You must have cloned the GitHub repository to your new local directory.

Configuration Example

To configure the On-box docker container to enable High Availability application for Cisco IOS XRv 9000 router on AWS, perform the following actions:

1. Build the Docker image to launch a container.

   Note Router basically accepts RPM files using the Application manager. The purpose of Application manager is to use these RPM files to deploy a docker container. So, after you build the docker image, you can then use the docker image to build an application RPM that you can deploy in the router.

   See the Build the Docker image section in GitHub for more information.

2. Build the Application RPM from the docker image in the router and run the container.

   Note Docker image can be carried as an RPM file to the router and then launched using the Application manager. RPM contains the docker image that you created.

   Build the Application RPM by performing the following actions:

   a. Go to the Application manager folder, and then go to the application-specific folder.

   b. Save your docker image in *tarball* form in the application-specific folder.

   c. Run the *./appmgr_build -b build.yaml* file to build an RPM that the Application manager accepts and deploy the RPM in the router.

   See the Build the Application RPM from docker image for deployment section in GitHub for more information.

3. Create *config.json* for each router based on the router configurations.

   Note The initial configuration is done so that the application can run on Cisco IOS XRv 9000 routers.

   The configuration consists of the following:

   - The BFD session snippet to run the session. The BFD session has the BFD multipliers, VRF name, router IP address with it.

     Note The BFD sessions are in `.json` format. The application supports multiple BFD sessions.

   - The service layer APIs (includes on box gRPC server and port details).
   - Secondary IP shift details for the instances where the secondary IPs can shift, if any of the BFD sessions goes down.

- Endpoint URL to communicate with AWS. To know the functionality of the endpoint URL, see the Create an interface endpoint to an AWS service topic.

Note You can use an interface endpoint to communicate with AWS using a private subnet. That reduces the amount of time that an application would spend on trying to connect to AWS.

- Global retry interval and global retry count. You can modify the value, when required. For example, the interval and count are set to 30 and 5 respectively in the Configuration example section. This example states that when the application is started and you reload the router, then if the gRPC server is down then it tries 5 times in every 30 seconds before failing. Once it fails, the Application manager restarts the session.

See the Create config.json for each router based on the router configuration section in GitHub for more information.

4. Create Host files to reduce the amount of time that the application takes to connect to AWS and to avoid DNS lookup.

   The host files consist of the end point that has a private IP and domain name.

   See the Create hosts file for the router pair section in GitHub for more information.

Deploying the application

After creating the files per router, deploy your application on the router by performing the following actions:

1. Copy the RPM file, config.json file, and the host file to the hard disk of the router.

Note Use SCP command to transfer the rpm file, json file, host file to the hard disk on the router.

2. Run the **appmanager** command to install application RPM packages, and then move both the *config.json* and the host file from the router hard disk to the application-specific folder. It is to start up the application.
3. Apply CLI configuration to activate or launch the application.

   The CLI consists of the following:

   - *BFD echo disable*. It is to run BFD sessions on AWS.
   - *Tpa vra default*. It is to ensure that all your packets that are going out of the router to help the application establish a connection to AWS. AWS must know from where the packets come from.
   - *gRPC details*. It is to connect to the application.
   - *Static IP address*. It is used as a gateway to reach the metadata URL of AWS. It is the standard public cloud operational procedure.

- *Application manager config*. It has the application name to activate it as a docker application, the source image, and the docker run option. This is to launch the docker container and start running the application.

See the Deploy App section in GitHub for more information.

Verifying the application

Based on your requirement, you can now run any of the following **xr-appmgr** commands to verify the application details.

- Run the application info command to get the details on the application that is being run.
- Run the application stats command to view the statistical details of the application run.

  The output displays the following information:

  - CPU percentage
  - Memory use and its percentage
  - Network
  - PIDs
  - Blocks

- Running the application logs command
  - Run the Docker container logs command to view the details related to the standard output logs from the docker container. It displays all the processes that are running in the container.

    The output shows a supervisor process running and it runs two more processes:

    - Application process
    - Redis process which is a database.

    These two processes together constitute your application.

  - Run the Application specific logs command to view the Application manager EXEC and then docker EXEC commands.

    When you run this command, it tails on the log file which is inside the docker container. It shows the behaviour of the application. Running this command displays the application-specific logs and you can troubleshoot the details when failover happens. You can also copy the logs to any location on the router, as required.

- Run the Application CLI commands to view the application-specific json-based CLI running inside a docker container.

  When you run the **show redundancy** command, it displays all show commands that you can run specific to the application.