# SOFTWARE DEFINED NETWORKS

## PROJECT REPORT

**TOPIC: RESEARCH AND DEMONSTRATE CONTAINER-BASED IMAGES ON SDN**

**TEAM:**

ANJALI PRAJAPATI(N01579923)

ADITI PANDYA(N01579827)

DHARA BAROT(N01559954)

VENKATA NARASIMHA VEDAVYAS(N0158367)

---

## Abstract:

Software application dependencies are frequently packaged as portable, light-weight components using container-based images. On a variety of computing platforms, such as cloud servers, virtual machines, and physical equipment, containers can be readily deployed. The control plane and data plane are separated in a network design known as software-defined networking (SDN), which also centralizes network management and configuration. SDN offers a scalable and adaptable method for managing network resources, enhancing network security, and improving network performance.

We are deploying a snowflake topology using RYU, an open-source SDN controller, and containerization with Docker and Kubernetes to demonstrate container-based images on SDN.

## Docker and Kubernetes Crew

Docker provides containerization technology that allows us to package the Ryu controller and the Snowflake topology components into a single container. This approach makes the deployment process more streamlined, and it makes it easy to move the container between different environments without needing to install dependencies manually.

Kubernetes offers an orchestration layer that enables us to effectively deploy and manage several containers. The number of containers we want to launch, how many we want to scale, and how we want to distribute traffic among them can all be determined using Kubernetes.

The portability of both allows us to quickly and simply deploy the Snowflake topology and Ryu controller to many environments without having to modify the deployment scripts.

## The Snowflake Topology reasoning

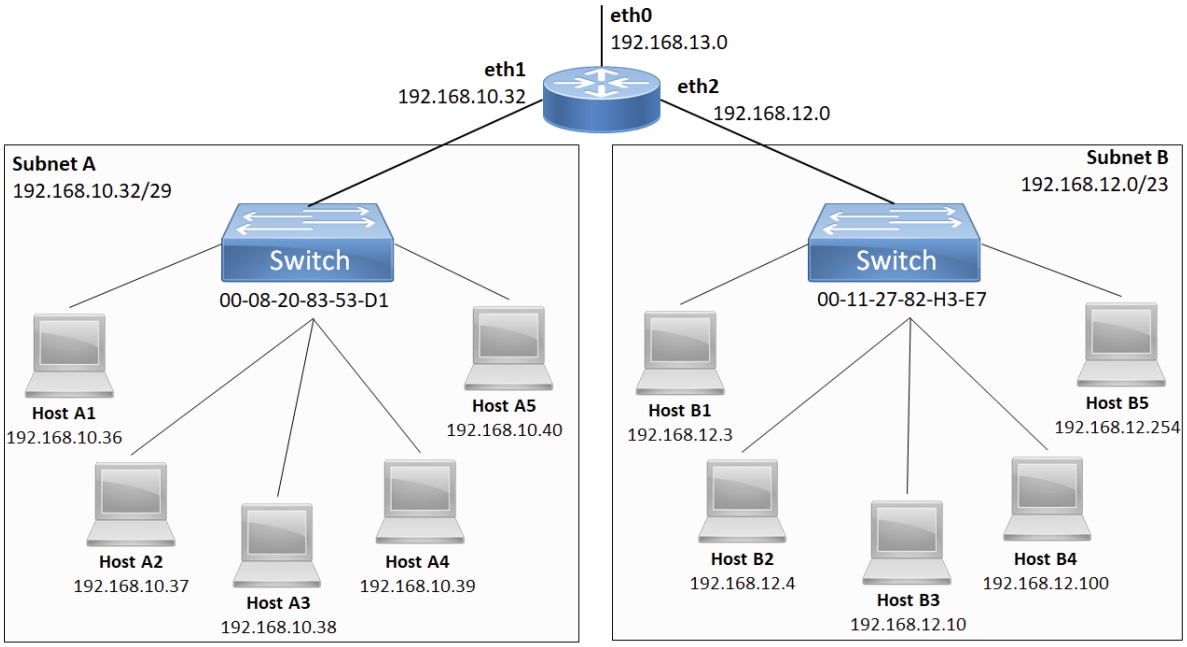
The Snowflake topology reduces the number of switches needed to establish the network while yet offering a high level of fault tolerance and network resilience.

A central switch (the Snowflake switch) in a Snowflake architecture links to numerous edge switches. The end hosts or other switches are connected to the edge switches in turn. The Snowflake topology is so named because, when graphically represented, the network resembles a snowflake, with the central switch representing the centre and the edge switches representing the branches.

In SDN systems, the Snowflake topology is advantageous because it enables effective network administration and lowers the number of switches required to set up the network. Since all switches are connected to the central switch, even if one fails, the others can still interact with one another, making the Snowflake topology extremely fault tolerant.

The Snowflake topology is a popular option for many businesses because it offers an effective and durable network design that is suitable for usage in SDN environments.

# Snowflake Topology Architecture



**Different Container-Based Images That Used In SDN**

Open vSwitch (OVS): OVS is a popular open-source virtual switch that provides high-performance, multi-layer switching with support for SDN protocols such as OpenFlow. It can be run as a containerized image in SDN environments to enable flexible network virtualization and programmable network control.

Docker-SDN: Docker-SDN is a container networking solution provided by Docker, which is a widely used containerization platform. It offers networking features such as multi-host networking, network isolation, and overlay networking, making it suitable for SDN deployments.

Calico: Calico is an open-source networking and network security solution for containers and virtual machines. It provides scalable, secure, and policy-driven networking for container-based applications. Calico can be deployed as a containerized image in SDN environments to enable advanced networking capabilities.

Weave: Weave is a popular open-source container networking solution that provides a simple and secure way to connect containers across different hosts. It supports network segmentation, encryption, and DNS-based service discovery, making it suitable for SDN deployments.

Cilium: Cilium is a modern, API-aware networking and security project that provides enhanced networking features for containers and micro services. It offers advanced networking capabilities such as load balancing, API-aware network security, and service discovery, making it suitable for SDN environments.

Flannel: Flannel is an open-source networking solution that provides a simple and lightweight way to create overlay networks for containers. It uses various backbends such as VXLAN, GRE, and host-gw to enable communication between containers running on different hosts, making it suitable for SDN deployments.

## Topo.py file

```
from ryu.base import app_manager
from ryu.topology import event
from ryu.topology import switches

from ryu.base import app_manager
from ryu.topology import event
from ryu.topology import switches
import logging

class MyTopology(app_manager.RyuApp):
    def _init_(self, *args, **kwargs):
        super(MyTopology, self)._init_(*args, **kwargs)
        self.topology_api_app = self

    def _register_switches(self):
        # Register switches
        self.topology_api_app.switch_map = switches.SwitchMap()

        for i in range(1, 4):
            switch = switches.Switch(i)
            self.topology_api_app.switch_map.register_switch(switch)
        # Example of print statement
        print("Registered switches in the topology")

    def _register_links(self):
        # Register links
        self.topology_api_app.link_list = []

        # Switches connected to controller
        for i in range(1, 4):
            link = event.EventLinkAdd(i, 0, 1, i)
            self.topology_api_app.link_list.append(link)
        # Example of logging statement
        logging.info("Registered links in the topology")
```

```
    def _register_hosts(self):
        # Register hosts
        self.topology_api_app.host_map = {}

        for i in range(1, 4):
            host = event.EventHostAdd(str(i), 0, i)
            self.topology_api_app.host_map[str(i)] = host
        # Example of logging statement
        logging.info("Registered hosts in the topology")

    def switch_features_handler(self, ev):
        self._register_switches()
        self._register_links()
        self._register_hosts()

if _name_ == '_main_':
    # Example of logging configuration
    logging.basicConfig(filename='my_topology.log', level=logging.DEBUG)
    MyTopology().run()
```