Faculty of Computers and Information

SDN

WITH BLOCKCHAIN

Supervised by:

Dr. Hatem Abdul Qadir

Presented by :

- Ahmed Salem Salem Mohamed
- Mohamed Abdel Gawad Abdel Wahab Salem Omar
- Youssef Yasser Hamed Al-Sheikh
- Ziad Khaled Elawa Abdel Ghaffar Emam

- Rokya Shreef Hassaneen Elgazar

- Nour Ibrahim Al-Ghamry Ibrahim Ghanem

- Riham Mohamed Abu El-Fouth Abd Rabbo

Table of Contents

- Chapter 1: Introduction to Software-Defined Networking (SDN)

  1.1-Overview ……

  1.2-Where we are before SDN.……

  1.3-what a SDN…….

  1.4-looking at the Benefits and Use Cases of SDN……

.

- Chapter 2: SDN Infrastructure

  2.1-layer 1: Infrastructure……

  2.2-layer 2: southbound Interfaces.……

  2.2.1 Types of Southbound interfaces…….

  2.3-layer 3: Northbound Interface.……

  2.3.1 Types of Northbound Interface……

- Chapter 3: SDN Controllers

  3.1-controller…….

  3.1.1 SDN Controller Core Modules……

  3.1.2 Common Types of Controllers…….

❖ Chapter 4: SDN Devices (Switches)

4.1-Types of Switches…….

4.2-Switch components……

4.2.1 Flow Table……...

4.2.2 Group Table4.2.3 Match fields…….

4.2.4 Counters…….

4.2.5 Instructions……

4.2.6 Actions…….

❖ **Chapter 5:** OPEN Daylight Controller

5.1 What is Open-daylight ? ……...

5.1.1Overview…….

5.1.2History and background…….

5.1.3Developed in Java……

5.2open day light architecture……

5. 3 How Open Daylight Works …….

5.4features ……

5.5 installation……

5.5.1 System requirements ……

5. 5. 2 Basic installation steps……

5. 5. 3 How to start the controller…….

❖ **Chapter 6:** Block Chain.

6.1BC with SDNs……

6.2 BC-based security for cloud computing…….

6.3 Proposed "DistB-SDCloud" architecture for cloud computing……

6.4 Data Extraction using SDN techniques in smart IIoT applications

6.4.1. Data plane……

6.4.2. Control plane……

6.4.3. Application layer……

6.5. Distributed Secure BC-SDN methodology…….

6.5.1. Security and access policy…….

6.5.2. BC and SDN convergence…….

6.5.3. Block creation and validation…….

6.6 Attack mitigation in cloud environment BC-based SDN approach……

6.7 Cloud Computing Management and services……

❖ Chapter7: Design and implementation

Chapter 1: Introduction of SDN

1.1 Introduction:

As we know In Traditional computer Networks, Data flow is controlled by switches and routers and contains the following basic elements Data Forwarding plane (Packet streaming), Control plane (Routing algorithms), and Management plane (Configure basic activities) but we found some drawbacks in this way like Inability to Scale, Vendor Dependence, Inconsistent Policies, and Complexity that leads to Static Nature.so we decide to search about new way to overcome this drawbacks. So, we arrive to Software-Defined Networking (SDN).

SDN is an evolutionary approach to network design and functionality based on the ability to programmatically modify the behavior of network devices.

SDN uses user-customizable and configurable software that's independent of hardware to expand data flow control.

In the SDN architecture, the control and data planes are decoupled, network intelligence and state centralized, and the underlying network infrastructure is abstracted from the applications.

It will make networks more flexible, dynamic, and cost-efficient, while greatly simplifying operational complexity and we made in our project by using Linux (Ubuntu) as an environment and setup Mininet as a Controller, and python (Ryu controller) to show the topology that contains Openflow Switches, Hosts, and Controller and setup Wireshark to capture messages between control plane and data plane.

1.2 Where we are before SDN?

To better understand why SDN has become so important, you need to look at what existed before SDN. Traditional networking architectures have significant limitations that must be overcome to meet modern IT requirements. Today's network must scale to accommodate increased workloads with greater agility, while also keeping costs at a minimum. But the traditional approach has substantial limitations:

- Complexity: The abundance of networking protocols and features for specific use cases has greatly increased network complexity. Old technologies were often recycled as quick fixes to address new business requirements.
- Inability to scale: As application workloads change and demand for network bandwidth increases, the IT department either needs to be satisfied with an oversubscribed static network or needs to grow with the demands of the organization. Unfortunately the majority of traditional networks are statically provisioned in such a way that increasing the number of endpoints, services, or bandwidth requires substantial planning and redesign of the network.

1.5 What is SDN?

As originally defined, SDN refers to a network architecture where the forwarding state in the data plane is managed by a remote control plane decoupled from the former.We define an SDN as a network architecture with four Basics:

1- The control and data planes are decoupled. Control functionality is removed from network devices that will become simple (packet) forwarding elements.

2-Forwarding decisions are flow-based, instead of destination-based. A flow is broadly defined by a set of packet field values acting as a match criterion and a set of actions. In the SDN/Openflow, a flow is a sequence of packets between a source and a destination. All packets of a flow receive identical service policies at the forwarding devices.

3- Control logic is moved to an external entity, the so-called SDN controller or Network Operating System (NOS). The NOS is a software platform that runs on commodity server technology and provides the essential resources and abstractions to facilitate the programming of forwarding devices based on a logically centralized, abstract network view. Its purpose is therefore similar to that of a traditional operating system.

4- The network is programmable through software applications running on top of the NOS that interact with the underlying data plane devices. This is a fundamental characteristic of SDN, considered as its main value proposition.

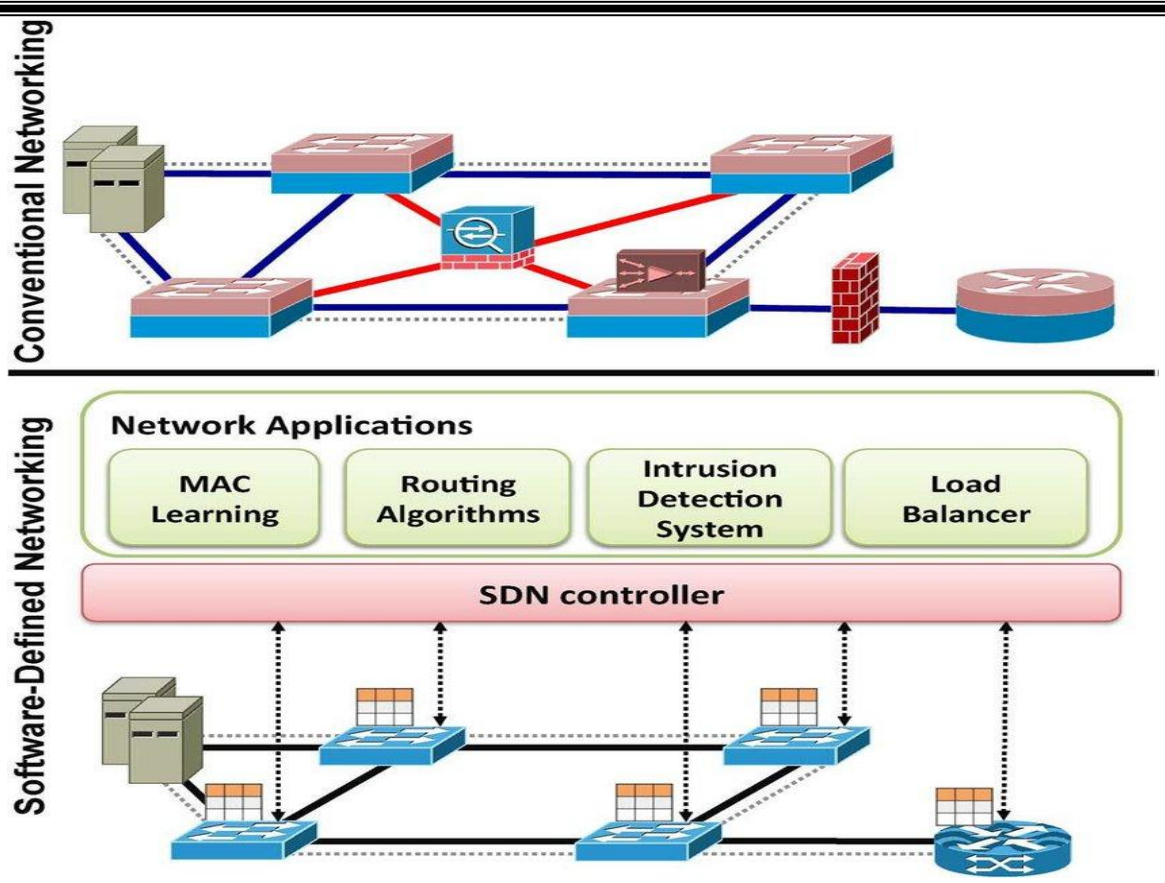


Figure 1. Difference between traditional and SDN Networks

1.3.1 Terminology:

To identify the different elements of an SDN as unequivocally as possible, we now present the essential terminology:

Forwarding Devices (FD): Hardware- or software-based data plane devices that perform a set of elementary operations. The forwarding devices have well-defined instruction sets

- Data Plane (DP): Forwarding devices are interconnected through wireless radio channels or wired cables. The network infrastructure comprises the interconnected forwarding devices, which represent the data plane.

Control Plane (CP): Forwarding devices are programmed by control plane elements through well-defined SI embodiments. The control plane can therefore be seen as the "network brain". All control logic rests in the applications and controllers, which form the control plane.

Southbound Interface (SI): The instruction set of the forwarding devices is defined by the southbound API, which is part of the southbound interface. Furthermore, the SI also defines the communication protocol between forwarding devices and control plane elements. This protocol formalizes the way the control and data plane elements interact.

Northbound Interface (NI): The network operating system can offer an API to application developers. This API represents a northbound interface a common interface for developing applications. Typically, a northbound interface abstracts the low level instruction sets used by southbound interfaces to program forwarding devices.

Management Plane (MP): The management plane is the set of applications that leverage the functions offered by the NI to implement network control and operation logic. This includes applications such as routing, firewalls, load balancers, monitoring, and so forth. Essentially, a management application defines the policies, which are ultimately translated to southbound-specific instructions that program the behavior of the forwarding devices.

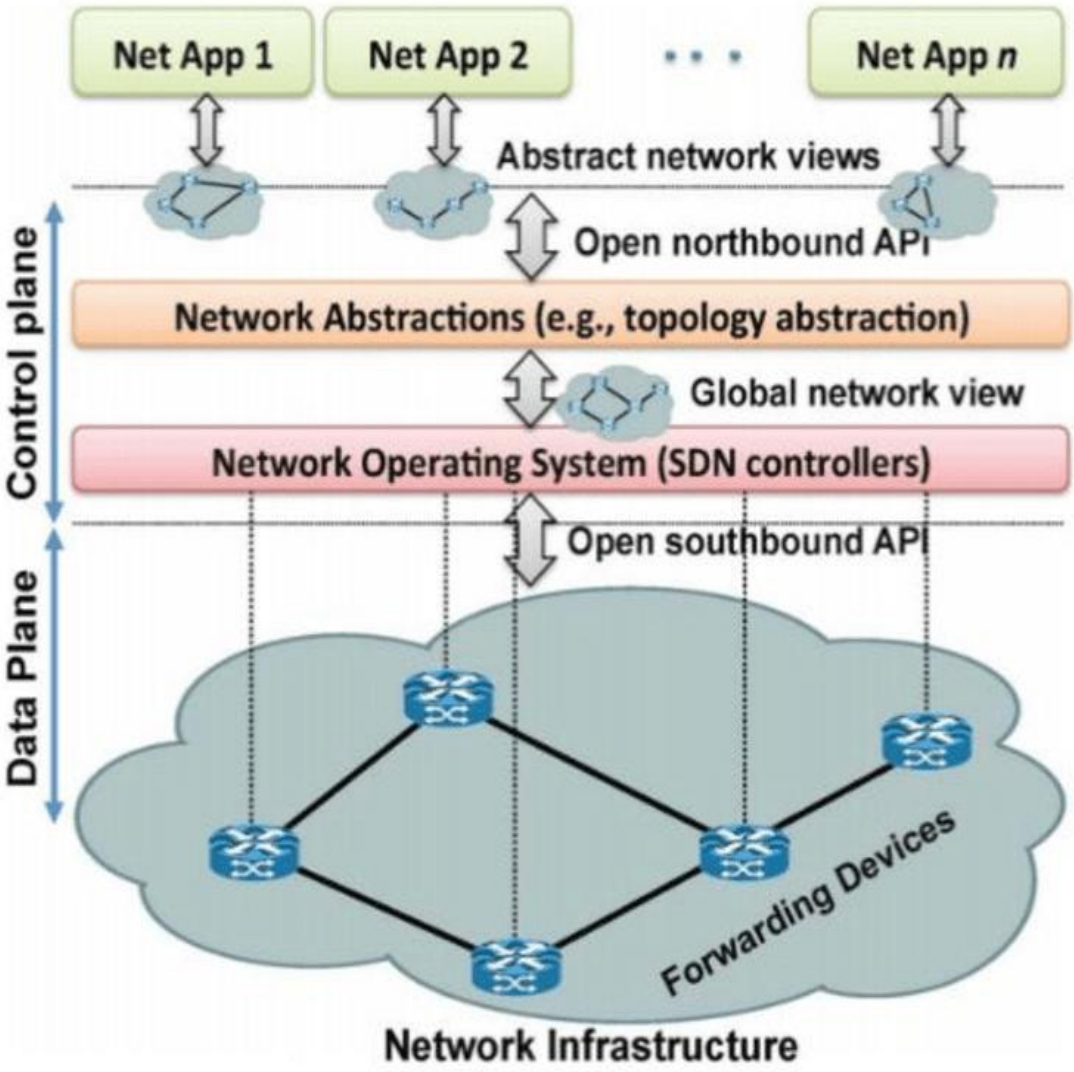


Figure 2. Network Infrastructure

There are a lot of benefits and use cases of SDN:-

- Easier, application-focused way of expressing policy:By creating policies that mirror application semantics framework Provides a simpler, self-documenting mechanism for capturing Policy requirements.
- Improved automation: Grouping policies together allows higher level automation tools to easily manipulate groups Of network endpoints and applications simultaneously.
- Consistency: By grouping endpoints and applying policy to application groups, the framework offers a consistent and concise way to handle policy changes, as well as consistency across both physical and virtual workloads, and physical and virtual application services and security devices.
- Extensible policy model: Because the policy model is open and can be extended to other vendors and other device types, it can easily incorporate switches, routers, security, Layer 4 through 7 services, and so on.

Chapter 2: SDN Infrastructure

2.1 Layer 1: Infrastructure

An SDN infrastructure similarly to a traditional network composed of a set of networking equipment (switches, routers and middlebox appliances). The main difference resides in the fact that those traditional physical devices are now simple forwarding elements without embedded control or software to take autonomous decisions.

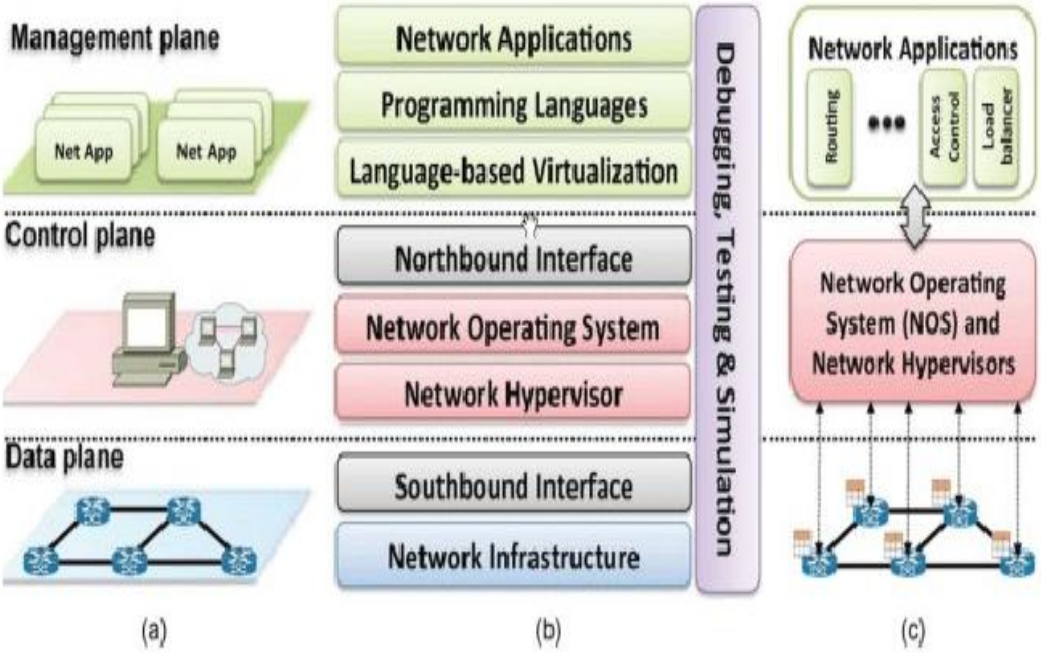


Figure 3 (design architecture)

In SDN/OpenFlow architecture, there are two main elements, the controllers and the forwarding devices. A data plane device is a hardware or software element specialized in packet forwarding, while a controller is a software stack (the "network brain") running on a commodity hardware platform. An OpenFlow-enabled forwarding device is based on a pipeline of flow tables where each entry of a flow table has three parts:

(1) a matching rule.

(2) Actions to be executed on matching packets.

(3) Counters

that keeps statistics of matching packets. This high-level and simplified model derived from OpenFlow is currently the most widespread design of SDN data plane
devices.

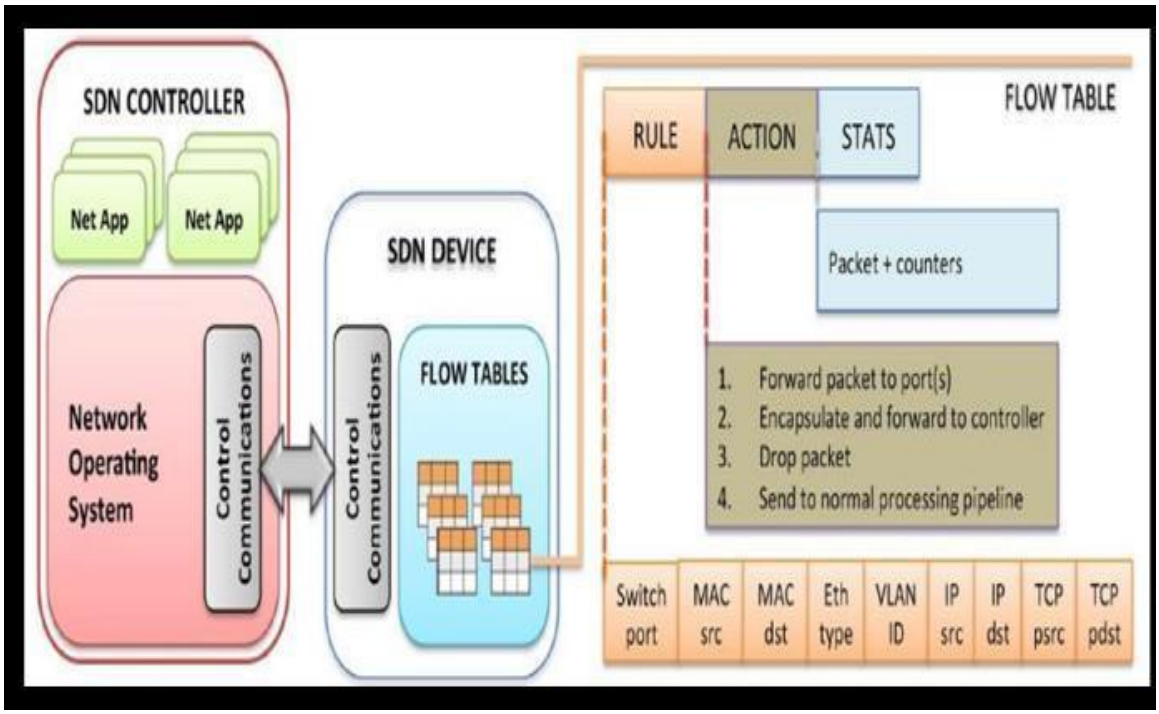


Figure 4. OpenFlow-enabled SDN devices

2.2 Layer 2: Southbound Interfaces

Southbound interfaces (or southbound APIs) are the connecting bridges between control and forwarding elements, thus being the crucial instrument for clearly separating control and data plane functionality.

As a central component of its design the southbound APIs represent one of the major barriers for the introduction and acceptance of any new networking technology. In this light, the emergence of SDN southbound API proposals such as OpenFlow is seen as welcome by many in the industry.

OpenFlow is the most widely accepted and deployed open southbound standard for SDN. It provides a common specification to implement OpenFlow-enabled forwarding devices, and for the communication channel between data and control plane devices (e.g., switches and controllers).

2.2.1 Types of Southbound Interface

OpenFlow is not the only available southbound interface for SDN. There are other API proposals such as:

-OVSDB:is another type of southbound API, designed to provide advanced management capabilities for Open vSwitches, it allows the control elements to create multiple virtual switch instances, set QoS policies on interfaces, attach interfaces to the switches, configure tunnel interfaces on OpenFlow data paths, managequeues, and collect statistics. Therefore, the OVSDB is a complementary protocol to OpenFlow for Open vSwitch.

-POF:One of the main goals of POF is to enhance the current SDN forwarding plane. With Openflow, switches have to understand the protocol headers to extract the required bits to be matched with the flow table's entries. To achieve its goal, POF proposes a generic flow instruction set (FIS) that makes the forwarding plane protocol-oblivious.

-OpFlex:Contrary to Openflow (and similar to ForCES), one of the Ideas behind OpFlex is to distribute part of the complexity of managing the network back to the forwarding devices, with the aim of improving scalability.

-HAL:Is not exactly a southbound API, but is closely related. Differently from the aforementioned approaches, HAL, is rather a translator that enables southbound API such as Openflow to control heterogeneous hardware devices. It thus sits between the southbound API and the hardware device.

2.3 Layer 3: Northbound Interfaces

The North and Southbound interfaces are twa key abetractions Of the SDN ecosystem. A common northbound interface is still an open issue At this moment it may still be a hit too early to define a standard northbound Interface.

It is to be expected a common northbound interface to arise as SDN evolves. That would allow netwerk applications not to depend on specific implementations is important to explore the full potential of SDN.

The northbound interface is mostly a software ecosystem, not a hardware one as is the case of the southbound APIs.

Open and standard northhound interfaces are crucial to promote application portability and interoperability among the different the control platforms

2.3.1 Types of Northbound Interface

There are different types of Northbound API such as

-POSIX:standard in operating systems, representing an abstraction that guarantees programming language and controller independence.

-NOSIX:is one of the first examples of an effort in this direction. It tries to define portable low-level application interfaces, making southbound APIs such as OpenFlow look like "device drivers".

-SFNeo:is another example of a northbound interface. It is a high-level API that translates application requirements into lower level service requests. However, SFNet has a limited scope, targeting queries to request the congestion state of the network and services such as handwidth reservation and multicast.

Chapter 3: SDN Controllers

3.1 Controllers

The controller is the core of an SDN network. By running the control plane as software, the controller facilitates automated network management and makes it easier to integrate and administer applications, SDN controllers uses protocols such as OpenFlow to configure network devices. It manages flow control to enable intelligent networking.

- The controller can delete, add or update flow entries in flow tables existing in the switch, both reactively le. in response to packets or proactively, using the OpenFlow protocol

- Controller make this decision based on policies set by administrator or depending on the conditions of the network and the decision it makes is forwarded to flow table entries of all the switches in the network.

- We have noted that the controller maintains a view of the entire network, implements policy decisions, controls all the SDN devices that comprise the network infrastructure, and provides a Northbound API for applications. When we have said that the controller implements policy decisions regard in routing. forwarding, redirecting, load balancing, and the like, these statements referred to both the controller and the applications that make use of that controller. Controllers often come with their own set of common application modules, such as a learning switch, a router, a basic firewall, and a simple load balancer.

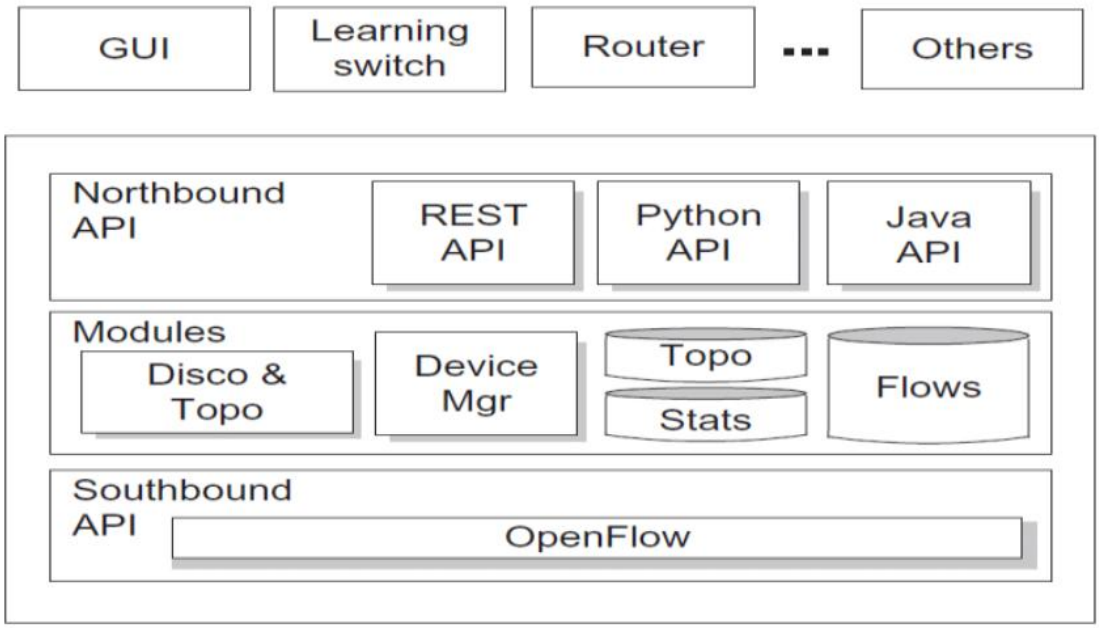


Figure 5.SDN Controller anatomy

The controller abstracts the details of the SDN controller-to-device protocol so that the applications above are able to communicate with those SDN devices without known their nuances, shows the API below the controller, which is OpenFlow in Open SDN, and the interface provided for applications. Every controller provides core functionality between these raw interfaces. Core features in the controller include:

- End-user device discovery. Discovery of end-user devices such as laptops, desktops, printers, mobile devices, and so on.
- Network device discovery. Discovery of network devices that comprise the infrastructure of the network, such as switches, routers, and wireless access points.
- Network device topology management Maintain information about the interconnection details of the network devices to each other and to the end-user devices to which they are directly attached.
- Flow management. Maintain a database of the flows being managed by the controller and perform all necessary coordination with the devices to ensure synchronization of the device flow entries with that database.

3.1.2 Common Types of Controllers

There are different types of Controllers, some types are Open source and the others are vendor dependence. Each type has is different programming language which controls the network such as:

Name	Architecture	Northbound API	Consistency	Faults	License	Prog. Language	Version
Beacon	centralized multi-threaded	ad-hoc API	no	no	GPLv2	Java	v1.0
DISCO	distributed	REST	yes	yes	—	Java	v1.1
Fleet	distributed	ad-hoc	no	no	—	—	—
Floodlight	centralized multi-threaded	RESTful API	no	no	Apache	Java	v1.0
HP VAN SDN	distributed	RESTful API	weak	yes	—	Java	v1.0
HyperFlow	—	—	weak	yes	—	—	v1.0
Kandoo	hierarchically distributed	—	no	no	—	C, C++, Python	v1.0
Onix	—	NVP NBAPI	weak, strong	yes	commercial	Python, C	v1.0
Maestro	centralized multi-threaded	ad-hoc API	no	no	LGPLv2.1	Java	v1.0
Meridian	centralized multi-threaded	extensible API layer	no	no	—	Java	v1.0
MobileFlow	—	SDMN API	no	—	—	—	v1.2
MoL	centralized multi-threaded	multi-level interface	no	no	GPLv2	C	v1.0
NOX	centralized	ad-hoc API	no	no	GPLv3	C++	v1.0
NOX-MT	centralized multi-threaded	ad-hoc API	no	no	GPLv3	C++	v1.0
NVP Controller	distributed	—	—	—	commercial	—	—

Name	Architecture	Northbound API	Consistency	Faults	License	Prog. Language	Version
OpenContrail	—	REST API	no	no	Apache 2.0	Python, C++, Java	v1.0
OpenDaylight	distributed	REST, RESTCONF	weak	no	EPL v1.0	Java	v1.0(3)
ONOS	distributed	RESTful API	weak, strong	yes	—	Java	v1.0
PANE	distributed	PANE API	yes	—	—	—	—
POX	centralized	ad-hoc API	no	no	GPLv3	Python	v1.0
ProgrammableFlow	—	—	—	—	—	—	v1.0
Rosemary	centralized	ad-hoc	—	—	—	C	v1.3
Ryu NOS	centralized multi-threaded	ad-hoc API	no	no	Apache 2.0	Python	v1.0(2.3)
SMaRtLight	distributed	RESTful API	no	no	Apache	Java	v1.0
SNAC	centralized	ad-hoc API	no	no	GPL	C++	—
Trema	centralized multi-threaded	ad-hoc API	no	no	GPLv2	C, Ruby	v1.0
Unified Controller	—	REST API	—	—	commercial	—	v1.0
yanc	distributed	file system	—	—	—	—	—

Chapter 4: SDN Devices (Switches)

4.1 Types of Switches

- Software switch:is implemented as a software application usually running in conjunction with a hypervisor in a data center rack. Like a VM, the virtual switch can be instantiated or moved under software control. It normally serves as a virtual switch and works collectively with a set of other such virtual switches to constitute a virtual network.
- Software SDN device implementations are most often found in software-based network devices, such as the hypervisors of a virtualization system. These hypervisors often incorporate a software switch implementation that connects the various virtual machines to the virtual network. The virtual switch working with a hypervisor is a natural fit for SDN. In fact, the whole virtualization system is often controlled by a centralized management system, which also meshes well with the centralized controller aspect of the SDN paradigm.

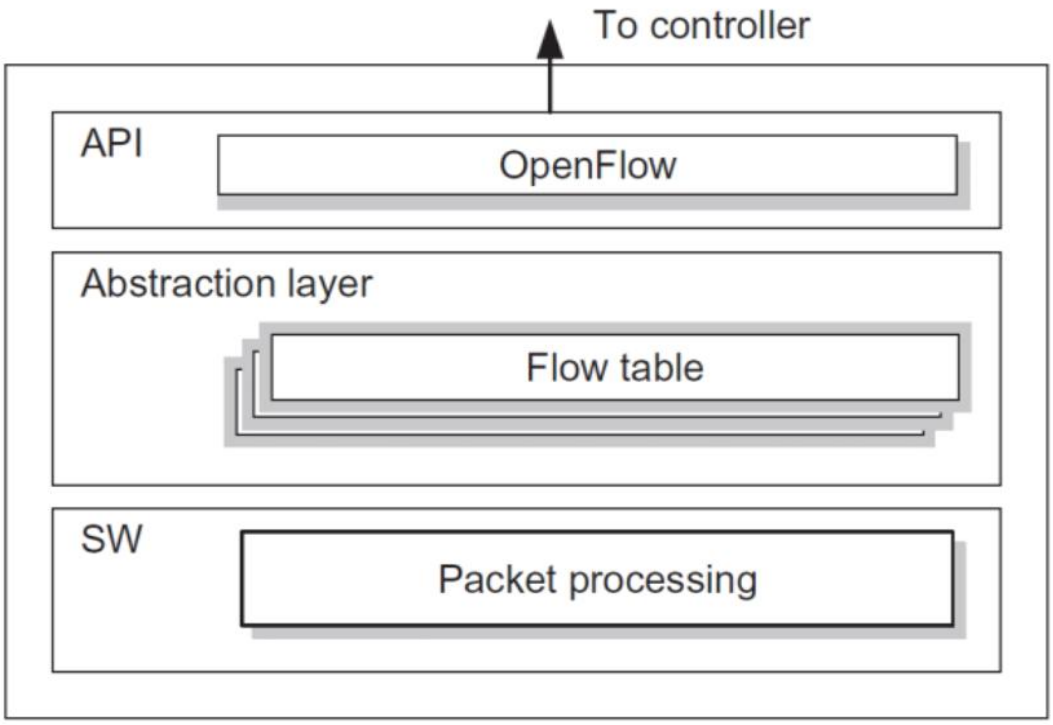


Figure 6 .software switch

-Hardware Switch:Hardware implementations of SDN devices hold the promise of operating much faster than their software counterparts and, are more applicable to performance-sensitive environments, such as in data centers and network cores.

This hardware includes the layer two and layer three forwarding tables, usually implemented using content-addressable memories (CAMs) and ternary content-addressable memories (TCAMs). The layer three forwarding table is used for making IP-level routing decisions.

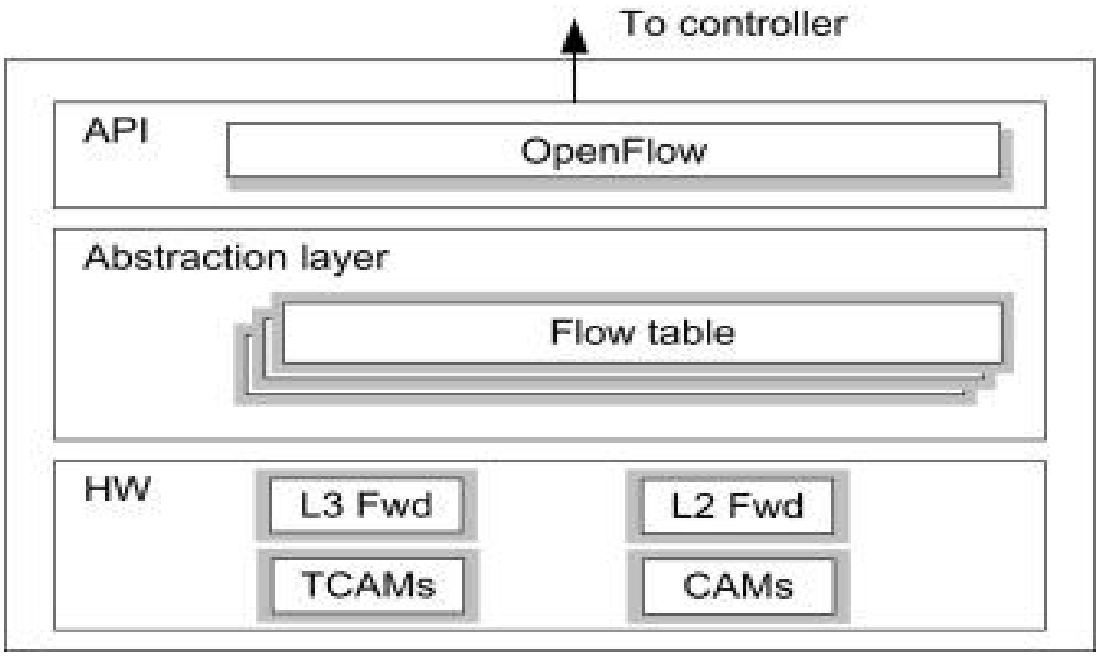
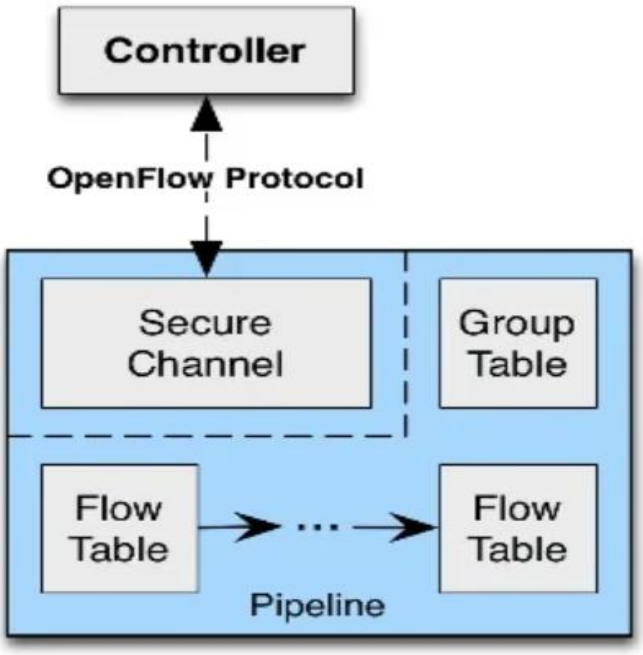


Figure 7.Hasrdware switch

NOTE: Software and Hardware switches may be Openflow-only (support only Openflow operation) and Openflow-hybrid (support Openflow operation and normal Ethernet switching operation).

4.2 Switch Components

An OpenFlow Switch consists of one or more ow tables and a group table, which perform packet lookups and forwarding, and an OpenFlow channel to an external controller. The controller manages the switch via the OpenFlow protocol. Using this protocol, the controller can add, update, and delete ow entries, both reactively and proactively.



OpenFlow Switch

Switch designers are free to implement the internals in any way convenient, provided that correct match and instruction semantics are preserved. For example, while a flow may use an all group to forward to multiple ports, a switch designer may choose to implement this as a single bitmask within the hardware forwarding table. Another example is matching: the pipeline exposed by an OpenFlow switch may be physically implemented with a different number of hardware tables.

4.2.1 Flow Table

A flow table consists of flow entries:

Match Fields	Counters	Instructions

Each ow table entry contains:

- match Fields: to match against packets. These consist of the ingress port and packet headers, and optionally metadata.
- counters: to update for matching packets
- instructions to modify the action set or pipeline processing

4.2.1.1 Pipeline Processing

The OpenFlow pipeline of every OpenFlow switch contains multiple ow tables, each ow table containing multiple ow entries. The OpenFlow pipeline processing defines how packets interact with those flow tables. An OpenFlow switch with only a single ow table is valid, in this case pipeline processing is greatly simplified.

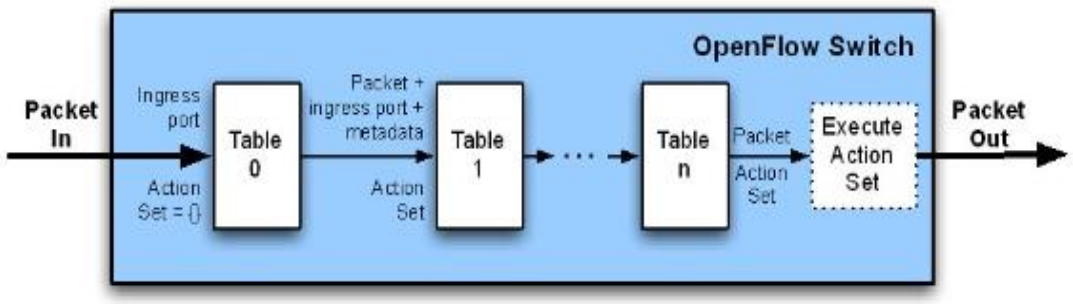


Figure 9. Packets are matched against multiple tables in the pipeline

the flow tables of an OpenFlow switch are sequentially numbered, starting at 0. Pipeline processing always starts at the first flow table: the packet is first matched against entries of ow table 0. Other flow tables may be used depending on the outcome of the match in the first table.

If the packet matches a flow entry in a flow table, the corresponding instruction set is executed. The instructions in the flow entry may explicitly direct the packet to another ow table (using the Gote Instruction, where the same process is repeated again. A ow entry can only direct a packet to a flow table number which is greater than its own flow table number, in other words pipeline processing can only go forward and not backward.

The flow entries of the last table of the pipeline cannot include the Goto instruction. If the matching flow entry does not direct packets to another flow table, pipeline processing stops at this table. When pipeline processing stops, the packet is processed with its associated action set and usually forwarded.

If the packet does not match a flow entry in a flow table, this is a table miss. The behavior on table miss depends on the table configuration:

- The default is to send packets to the controller over the control channel via a packet-in message.
- drop the packet
- A table can also specify that on a table miss the packet processing should continue; in this case the packet is processed by the next sequentially numbered table.

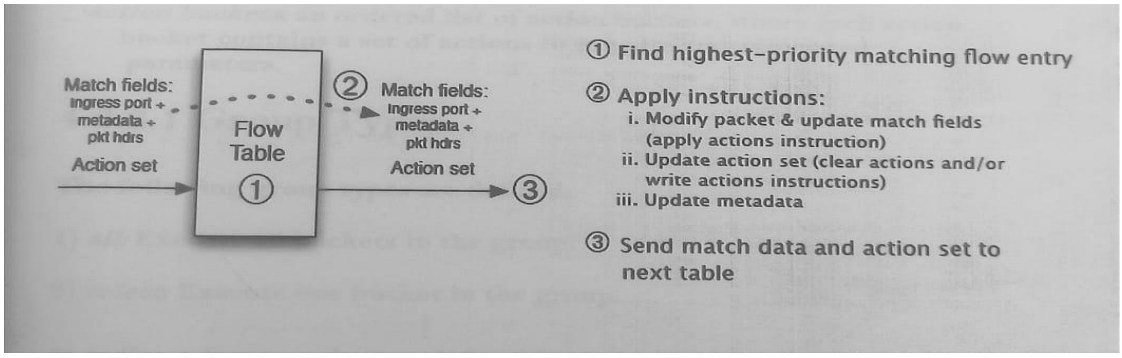


Figure 10. Packet Processing behavior

4.2.2 Group Table

A group table consists of group entries. The ability for a ow to point to a group enables OpenFlow to represent additional methods of forwarding

(e.g. select and all).

Each group entry contains:

Group identifier	Group type	Counters	Action buckets

group identifierza 32 bit unsigned integer uniquely identifying the group

group type: to determine group semantics.

counters: updated when packets are processed by a group.

action buckets an ordered list of action buckets, where each action bucket contains a set of actions to execute and associated parameters.

4.2.2.1 Group Types

The following group types are defined:

1) all Execute all buckets in the group.

2) select Execute one bucket in the group.

3) indirect: Execute the one defined bucket in this group.

4) fast failover: Execute the first live bucket.

4.2.3 Match fields

The match fields an incoming packet is compared against. Each entry contains a specific value, or ANY, which matches any value. If the switch. supports arbitrary bitmasks on the Ethernet source and/or destinations fields, or on the IP source and/or destination fields, these masks can more precisely

specify matches. In addition to packet headers, matches can also be performed against the ingress port and metadata fields. Metadata may be used to pass information between tables in a switch.

Ingress Port
Metadata
Ether src
Ether data
Ethere type
Vlan id
Vlan priority
Mpls label
Mpls traffiic class
Ipv4 src
Ipv4 data
Ipv4 proto

Figure 11. Fields from packets used to match against flow entries.

On receipt of a packet, an OpenFlow Switch performs the functions.The switch starts by performing a table lookup in the first ow table, and, based on pipeline processing, may perform table lookup in other flow tables Match fields used for table lookups depend on the packet type.

A packet matches a flow table entry if the values in the match fields used for the lookup match those defined in the flow table. If a flow table field has a value of ANY, it matches all possible values in the header.

To handle the various Ethernet framing types, matching the Ethernet type is handled based on the packet frame content. In general, the Ethernet type matched by OpenFlow is the one describing what is considered by

OpenFlow as the payload of the packet. If the packet has VLAN tags, the Ethernet type matched is the one found after all the VLAN tags. An exception to that rule is packets with MPLS tags where OpenFlow can not determine the Ethernet type of the MPLS payload of the packet.

If the packet is an Ethernet II frame, the Ethernet type of the Ethernet header (after all VLAN tags) is matched against the flow's Ethernet type. If the packet is an 80%.5 frame with a 809.2 LLC header, a SNAP header and Organizationally Unique Identifier (OUT) of 0x000000, the SNAP protocol id is matched against the flow's Ethernet type. A ow entry that specifies an Ethernet type of 0x05FF, matches all 802.3 frames without a SNAP header and those with SNAP headers that do not have an OUI of 0x000000

4.2.4 Counters

Counters may be maintained for each table, ow, port, queue, group, and bucket. OpenFlow-compliant counters may be implemented in software and maintained by polling hardware counters with more limited ranges.

Duration refers to the amount of time the flow has been installed in the switch. The Receive Errors field is the total of all receive, Counters wrap around with no overflow indicator. If a specific numeric counter is not available in the switch, its value should be set to -1.

4.2.5 Instructions

Each ow entry contains a set of instructions that are executed when a packet matches the entry. These instructions result in changes to the packet, action set and/or pipeline processing. Supported instructions includes -Apply-Actions action(s): Applies the specific action(s) immediately, without any change to the Action Set. This instruction may be used to modify the packet between two tables or to execute multiple actions of the same type. The actions are specified as an action list

- Clear-Actions: Clears all the actions in the action set immediately.
- Write-Actions action(s): Merges the specified action(s) into the current action set If an action of the given type exists in the current set, overwrite it, otherwise add it.
- Write-Metadata metadata/mask: Writes the masked metadata value into the metadata field. The mask specifies which bits of the metadata register should be modified (ie. new metadata = old metadata & mask ¡ value & msk).

- Goto-Table next-table-id: Indicates the next table in the processing pipeline. The table-id must be greater than the current table-id. The flows of last table of the pipeline cannot include this instruction.

The instruction set associated with a flow entry contains a maximum of one instruction of each type. The instructions of the set execute in the order specified by this above list. In practice, the only constraints are that the Clear-Actions instruction is executed before the Write-Actions instruction, and that Goto-Table is executed last.

A switch may reject a flow entry if it is unable to execute the instructions associated with the flow entry. In this case, the switch must return an unsupported ow error. Flow tables may not support every match and every instruction.

4.2.6 Actions

A switch is not required to support all action types When connecting to the controller, a switch indicates which of the "Optional Actions" it supports.

Required Action: Output. The Output action forwards a packet to a specified port. OpenFlow switches must support forwarding to physical ports and switch-defined virtual ports. Standard ports are defined as physical ports, switch-defined virtual ports, and the LOCAL port if supported (excluding other reserved virtual ports). OpenFlow switches must also support forwarding to the following reserved virtual ports:

ALL: Send the packet out all standard ports, but not to the ingress port or ports that are configured OFPPC_NO_FWD.

-CONTROLLER: Encapsulate and send the packet to the controller.

-TABLE: Submit the packet to the first flow table so that the packet can be processed through the regular OpenFlow pipeline. Only valid in the action set of a packet-out message.

-IN PORT: Send the packet out the ingress port.

Optional Action: Output. The switch may optionally support forwarding to the following reserved virtual ports:

-LOCAL: Send the packet to the switch's local networking stack. The local port enables remote entities to interact with the switch via the OpenFlow network, rather than via a separate control network. With a suitable set of default rules it can be used to implement an in-band controller connection.

-NORMAL: Process the packet using the traditional non-OpenFlow pipeline of the switch. If the switch cannot forward packets from the OpenFlow pipeline to the normal pipeline, it must indicate that it does not support this action.

FLOOD: Flood the packet using the normal pipeline of the switch. In general, send the packet out all standard ports, but not to the ingress port, or ports that are in OFPPS_BLOCKED state. The switch may also use the packet VLAN ID to select which ports to flood to.

NOTE:

OpenFlow-only switches do not support output actions to the NORMAL port and FLOOD port, while OpenFlow-hybrid switches may support them. Forwarding packets to the FLOOD port depends on the switch implementation and configuration, while forwarding using a group of type all enables the controller to more flexibly implement flooding.

4.2.6.1 Action Set

An action set is associated with each packet. This set is empty by default. A flow entry can modify the action set using a Write-Action instruction or a Clear-Action instruction associated with a particular match. The action set is carried between ow tables. When an instruction set does not contain a Goto-Table instruction, pipeline processing stops and the actions in the action set are executed.

An action set contains a maximum of one action of each type. When multiple actions of the same type are required, e.g. pushing multiple MPLSlabels or popping multiple MPLS labels, the Apply-Actions instruction may be used.

The actions in an action set are applied in the order specified below, regardless of the order that they were added to the set. If an action set. contains a group action, the actions in the appropriate action bucket of the group are also applied in the order specified below. The switch may support arbitrary action execution order through the action list of the Apply-Actions Instruction.

1. Copy TTL inwards apply copy TTL inward actions to the packet

4. Pop apply all tag pop actions to the packet

3. Push: apply all tag push actions to the packet

4. Copy TTL outwards apply copy TTL outwards action to the packet

5. Decrement TTL: apply decrement TTL action to the packet

6. Set: apply all set-field actions to the packet

7. Qor apply all QoS actions, such as set queue to the packet

8. Group if a group action is specified, apply the actions of the relevant group bucket(s) in the order specified by this list

9. autput: if no group action is specified, forward the packet on the port specified by the output action

The output action in the action set is executed last. If both an output action and a group action are specified in an action set, the output action is ignored and the group action takes precedence. If no output action and no group action were specified in an action set, the packet is dropped. The execution of groups is recursive; a group bucket may specify another group. in which case the execution of actions traverses all the groups specified by the group configuration.

4.2.6.2 Action List

- The Apply-Actions instruction and the Packet-out message include an action list. The actions of an action list are executed in the order specified by the list, and are applied immediately to the packet.
- The execution of action start with the first action in the list and cach action in executed on the packet in sequence. The effect of those actions is cumulative, if the action list contains two Push VLAN actions, two VLAN headers are added to the packet. If the action list contains an output action, a copy of the packet is forwarded in its current state to the desired port. If the list contains a group actions, a copy of the packet in its current state is processed by the relevant group buckets.
- After the execution of the action list in an Apply-Actions instruction, pipeline execution continues on the modified packet. The action set of the packet is unchanged by the execution of the action list.

Chapter 5 : Open DayLigt

1. what is openDayLight

Overview:

OpenDaylight (ODL) is a modular open-source platform designed to act as a central controller in Software-Defined Networking (SDN) architectures. It is developed collaboratively under the Linux Foundation and plays a pivotal role in enabling both SDN and Network Functions Virtualization (NFV) in modern networking environments.

OpenDaylight provides a flexible platform where network applications can control and manage network behavior through programmable interfaces, rather than relying on traditional manual configurations or vendor-specific systems.

Hosted by the Linux Foundation

OpenDaylight is one of several major collaborative networking projects hosted by the Linux Foundation, alongside projects like ONAP (Open Network Automation Platform) and FD.io (Fast Data Project).

Being part of the Linux Foundation ensures:

- Open governance
- A vendor-neutral development environment
- Broad industry collaboration (e.g., Cisco, Ericsson, Huawei, Intel, Red Hat)

Purpose and Vision

The main goal of OpenDaylight is to accelerate the adoption of SDN and NFV by:

- Providing a common platform for network programmability
- Enabling innovation and customization through modular design

- Supporting multiple southbound and northbound interfaces for flexible integration

It addresses the need for open standards and interoperability in a rapidly evolving networking ecosystem, especially as operators shift toward automated**,** dynamic, and virtualized infrastructure.

History and Background

- Announced**:** April 2013
- Initial Release: "Hydrogen" (February 2014)
- Since then, many versions have been released, named alphabetically (Hydrogen, Helium, Lithium, Beryllium, etc.).
- Developed by a consortium of over 50 major networking and IT companies.
- Aimed to reduce fragmentation in SDN solutions by providing a common, extensible platform**.**

Key Milestones:

- 2014–2016**:** Major adoption in research and service provider labs.
- 2016–2020**:** Integration with NFV orchestration platforms and increased use in production networks
- 2021+**:** Focus on 5G, edge computing, and support for cloud-native deployments.

Technology Stack:

- Programming Language: Primarily developed in Java
- Platform: Runs on Java Virtual Machine (JVM)
- Build System**:** Uses Maven
- Provides RESTful APIs for northbound applications.
- Based on a Model-Driven Service Abstraction Layer (MD-SAL) architecture that separates the logic and data models.

2. OpenDaylight Architecture

OpenDaylight (ODL) follows a modular, layered architecture that separates the application logic from the physical infrastructure. This design allows developers and operators to build network applications and services without being tightly coupled to underlying hardware.

The key element of OpenDaylight's architecture is the Model-Driven Service Abstraction Layer (MD-SAL), which facilitates communication between the controller core, applications, and the physical network.

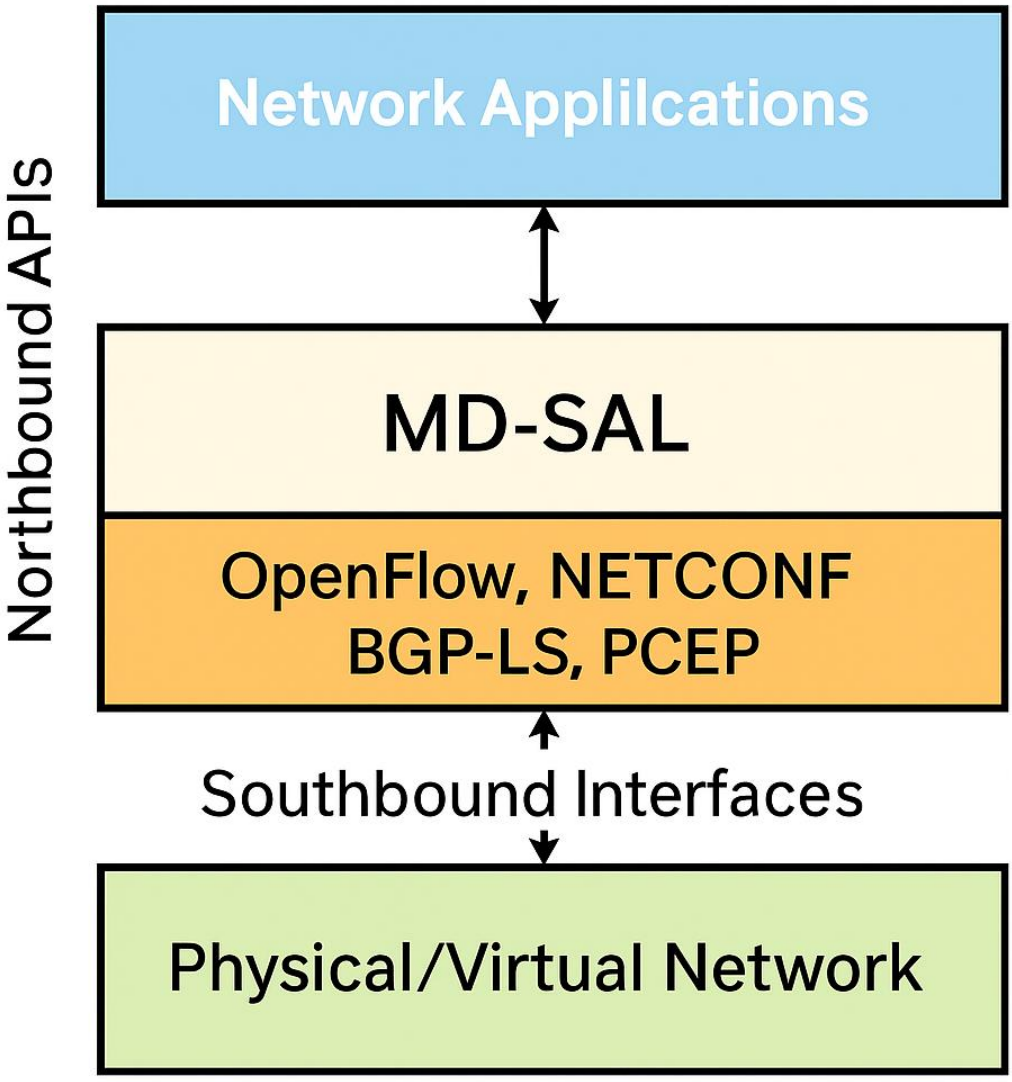


Figure 11 OpenDaylight architecture diagram.

2.2 Model-Driven Service Abstraction Layer (MD-SAL)

MD-SAL is the heart of the OpenDaylight controller.

Responsibilities:

- Acts as a message bus between different modules.
- Stores network data using YANG-based data models.
- Handles both configuration data (desired state) and operational data (actual state).
- Provides RPC (Remote Procedure Call) and **notifications** for inter-module communication.

Key Components:

Data Brokers: Manage read/write access to data stores.

Service Adapters: Bridge between applications and services.

YANG Models: Define data structures and API interfaces.

2.3 Southbound Interfaces

This layer is responsible for interacting with network devices**.**

Protocols Supported:

OpenFlow: Most common SDN protocol.

NETCONF/YANG: For device configuration.

BGP-LS: For collecting topology from routers.

PCEP: For traffic engineering path computation.

OVSDB: For managing Open vSwitches.

SNMP, LISP, CoAP, etc.

2.4 Northbound APIs

Used by **network applications** to interact with the controller.

- REST APIs (HTTP/JSON)
- Java APIs (for in-platform development)
- RPC (Remote Procedure Calls)

These APIs provide:

- Topology information
- Flow statistics
- Access to configuration/state data
- Application-to-controller communication

2.5 Plugins and Features

OpenDaylight is plugin-based, meaning features can be added as modules. Examples include:

- L2Switch: Emulates layer-2 learning switch.
- DLUX**:** Web-based GUI for managing topology and flows.
- AAA (Authentication, Authorization, Accounting)**:** For securing access
- VPP, VTN, BGP, PCEP, and more: Network services and protocol support.

Each plugin interacts with MD-SAL to publish or consume data.

2.6 DLUX – Web GUI

A visual interface that allows users to:

- View the network topology
- Add/delete flows
- Monitor link and node status
- Interact with controller modules visually

Note: DLUX has limited support in newer versions.

Here's how a packet handling workflow might occur:

1. A packet arrives at a switch.
2. The switch sends it to the controller via **OpenFlow**.
3. A plugin receives the packet and passes it to MD-SAL.
4. A network application (e.g., firewall or routing app) inspects it via Northbound API.
5. The app installs a new flow rule via MD-SAL.
6. The flow rule is sent back to the switch via the southbound plugin

2. How OpenDaylight Works?

OpenDaylight (ODL) serves as the centralized brain of a Software-Defined Network (SDN). It orchestrates, monitors, and controls network behavior by translating high-level policies into low-level configurations applied to physical and virtual devices. Its power lies in the use of southbound protocols for communicating with network devices and northbound APIs for interacting with applications and services.

1. Southbound Protocols: Controlling Network Elements

OpenDaylight communicates with the data plane (i.e., routers, switches, firewalls, etc.) using a variety of southbound protocols. These protocols act as the "language" that allows the controller to instruct devices to behave in specific ways.

Supported Protocols:

- **OpenFlow**: Used to add, delete, or modify flow rules on switches.
- **NETCONF/YANG**: Used for configuration management and monitoring.
- **BGP-LS (Link-State)**: Allows ODL to collect topology information from BGP-speaking routers.
- **PCEP (Path Computation Element Protocol)**: Used for computing optimal traffic engineering paths.
- **OVSDB**: Manages Open vSwitch configurations.

Functionality:

- **Topology discovery**: ODL uses LLDP packets through OpenFlow to learn about links and devices in the network.
- **Flow programming**: Installs flow entries in switches, directing how traffic should flow.
- **Device configuration**: Applies settings like IP addressing, interface states, VLANs, etc.
- **Monitoring**: Retrieves statistics and health data from devices.

Example:

If a new switch connects to the network, it sends a message (like `Packet-In` in OpenFlow) to the controller. OpenDaylight recognizes it, updates the network topology, and may push a basic set of rules to start forwarding packets.

2. Northbound APIs: Interfacing with Applications

While southbound protocols deal with devices, **northbound APIs** allow **external applications** to define **what the network should do**—without worrying about how it's done.

Types of Applications:

- **Firewall** (define security rules)
- **Load balancer** (optimize resource use)
- **Traffic shaper** (enforce QoS policies)
- **Routing application** (compute shortest or policy-based paths)
- **Analytics & monitoring dashboards**

Types of APIs:

- **RESTful APIs**: Web-based interface for easy integration.
- **Java APIs**: Used by native OpenDaylight applications.

Applications interact with OpenDaylight by:

- **Requesting information**: e.g., "What is the current network topology?"
- **Issuing instructions**: e.g., "Route video traffic through the fastest path."
- **Subscribing to events**: e.g., "Notify me if a link fails."

OpenDaylight receives these intents, processes them through MD-SAL (Model-Driven Service Abstraction Layer)**,** and pushes necessary flow rules to the infrastructure via southbound plugins.

3. Real-Time Topology Discovery & Flow Programming

One of OpenDaylight's core strengths is its ability to discover, update, and react to network state changes in real time**.**

Topology Discovery:

- ODL sends LLDP (Link Layer Discovery Protocol) packets through OpenFlow-enabled switches.
- It builds a real-time network topology graph**.**
- This topology is stored in the Operational Data Store and made available to applications via APIs.

Flow Programming

When an application requests a specific policy (e.g., allow traffic from IP A to IP B on port 80), OpenDayligh:

- Calculates the best path using its internal services (e.g., topology manager, path calculator).
- Generates flow rules.
- Sends the rules to switches using OpenFlow or another protocol.

Workflow:

1) **A packet from a new source reaches Switch A**.
2) **Switch A doesn't know what to do**, so it sends the packet to the controller (Packet-In).
3) **OpenDaylight processes it**, checks policies, and decides to allow the flow.
4) **The controller installs flow entries** in Switch A and other switches along the path.
5) **Future packets of the same flow** follow the new path without controller involvement.
6) This allows for **dynamic**, **real-time**, and **automated network control**—a key promise of SDN.

4. Role of MD-SAL in the Workflow

At the center of this process is the Model-Driven Service Abstraction Layer (MD-SAL)**:**

- It serves as a message bus between the northbound and southbound components.
- Stores data in structured YANG models.
- Separates logic (applications) from infrastructure (devices), enabling modularity.

5. Features

OpenDaylight (ODL) stands out in the Software Defined Networking ecosystem due to its robust, flexible, and community-driven architecture. Below are its **core features and benefits**, each of which contributes to its strength as a production-ready SDN controller:

1. Modularity: Plugin-Based Architecture

OpenDaylight is designed with a highly modular architecture**,** where different features are implemented as independent plugins or bundles**.** These plugins can be installed, updated, or removed as needed, which offers several advantages:

Customizable deployments: Users can load only the modules required for their use case (e.g., OpenFlow, BGP, DLUX).

Better scalability**:** Modular design allows for lightweight configurations or full-featured systems.

Easier debugging and testing: Faults can be isolated to specific modules.

The modularity is managed via the Apache Karaf OSGi container, which serves as the runtime environment for ODL.

2. Interoperability: Multi-Protocol Support

ODL is protocol-agnostic—it supports a wide range of **southbound protocols**, enabling it to work with diverse types of networking hardware and systems.

Supported Protocols:

- **OpenFlow** – Flow-based forwarding.
- **NETCONF/YANG** – Configuration and state management.
- **BGP-LS / PCEP** – Routing and traffic engineering.
- **OVSDB** – Open vSwitch management.
- **SNMP, LISP, CoAP** – For traditional and IoT devices.

This **broad compatibility** ensures ODL can be deployed in mixed environments, including **legacy**, **virtualized**, and **next-gen SDN-ready** networks.

3. Extensibility: Easily Extendable Framework

OpenDaylight is built with extensibility in mind. Developers can:

- Create custom applications using Java APIs.
- Define new YANG models to describe data structures.
- Build and register new southbound or northbound plugins.

This makes ODL ideal for research, innovation, and rapid prototyping of new networking functions or services such as custom security controls, traffic analytics, or AI-driven policies.

4. Community Support: Strong and Active Ecosystem

ODL benefits from a vibrant open-source community under the Linux Foundation**,** which includes contributors from major network vendors (Cisco, Huawei, Ericsson, etc.) as well as academia and independent developers.

Community Benefits:

- Regular updates and security patches
- Public forums and mailing lists
- Collaborative development process
- Access to training, tutorials, and documentation

This rich ecosystem enhances the reliability, transparency, and trustworthiness of OpenDaylight in both academic and enterprise deployments.

6. Installation and Setup

6.1 System Requirements

To install and run OpenDaylight smoothly, your system should meet the following minimum requirements**:**

Hardware:

- CPU: Dual-core processor (Quad-core recommended)
- RAM: At least **4 GB** (8+ GB recommended for better performance)
- Disk: 2 GB of free space
- Network: Internet access to download dependencies

Software:

Operating System: Linux (Ubuntu/CentOS) or Windows (WSL also works)

Java: Java SE Development Kit (JDK) **version 8**

Verify using: `java -version`

Maven: Apache Maven **version 3.3+**

Verify using: `mvn -v`

Note: Later versions of Java (Java 11 or higher) may not be compatible with older OpenDaylight builds. Stick with Java 8 for compatibility.

Basic Installation Steps:

```
tar -xvzf distribution-karaf-<version>.tar.gz
```

```
cd distribution-karaf-<version>
```

```
./bin/karaf
```

Installing OpenFlow Plugin:

```
feature:install odl-openflowplugin-all
```

Accessing DLUX GUI:

```
feature:install odl-dlux-core
```

Then go to: http://localhost:8181/index.html

Default credentials: admin / admin

Chapter 6 : Block Chain

Through the use of an SDN-intelligent gateway,IoTsensor data can be transformed from the sensor level to the SDN environment. SDN filters the data and BC, and efficiently manages all filtered data. Furthermore, a virtual communication layer is used to establish a connection between the SDN environment and the BC approach. This layer also offers virtualization technology to establish an effective bonding with two methods, as depicted in Fig. 4. In the context of IoT architecture, Sharma et al. presented the DistBlockNet paradigm . This platform offers two significant advantages for various networking technologies such as SDN and BC. The authors proposed a strategy to update the flow rule table by applying the BC technique and formalizing the flow rule table. They also measured the results using different metrics, and the analysis presented a better outcome relative to the surviving piece. Rahman et al. proposed the "DistBlockSDN" architecture with Network Function Virtualization (NFV) implementation for a smart city . The authors used a BC approach to achieve high security and privacy. Furthermore, they presented an algorithm for selecting the cluster head while consuming little energy. The authors evaluated the networks' performances using parameters like throughput and packet arrival rate. In another study, Navid et al. presented a novel model to address IoT challenges by combining SDN and BC technologies for future 5G telecommunication networks.

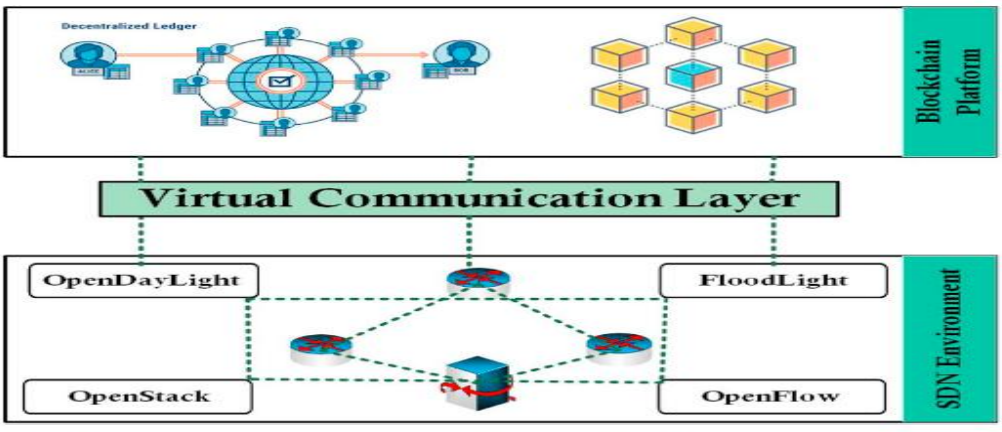


Fig. 4. An architecture combining SDN with Blockchain to provide security.

6.1. BC-based security for cloud computing Cloud computing

is a model that allows for the remote delivery of hardware, software, storage, and other resources as services via the internet. Different implementation models have been developed according to the application scenario and business purpose, such as restricting access to cloud resources only to the employees of an enterprise. There can be personal, public, hybrid, and community deployment models, as reported in Fig. 5. Gaetani et al. first presented some research questions based on BC for cloud computing. They presented accurate, high-level solutions to these questions for the European project SUNFISH. In related research, Park et al. established the notion of BC technology, and some possible technological cloud computing directions. They also presented a high-level security approach to cloud computing on various parameters based on the BC concept. In , BC technology was used to provide various security services to IoT forwarding devices. Cloud computing and Edge transparent computing technologies were described in detail in the study. Furthermore, the authors explicitly discussed BC measures to safeguard IoT networks from unauthorized threats. Similarly in, Sharma et al. introduced a new BC-based decentralized cloud platform with Software-Defined Networking(SDN)enabled controller fog nodes at the network's edge. They proposed an excellent combination of fog computing, SDN, and BC. Furthermore, the authors presented an architecture designed to support high availability, real-time data collection, enhanced scalability, security, and resiliency while maintaining low latency. They also evaluated parameters like throughput, response time, and accuracy in detecting real-time attacks. In addition to the traditional BC, artificial intelligence-based BC technology was used in the mechanism to provide cyber-physical security.

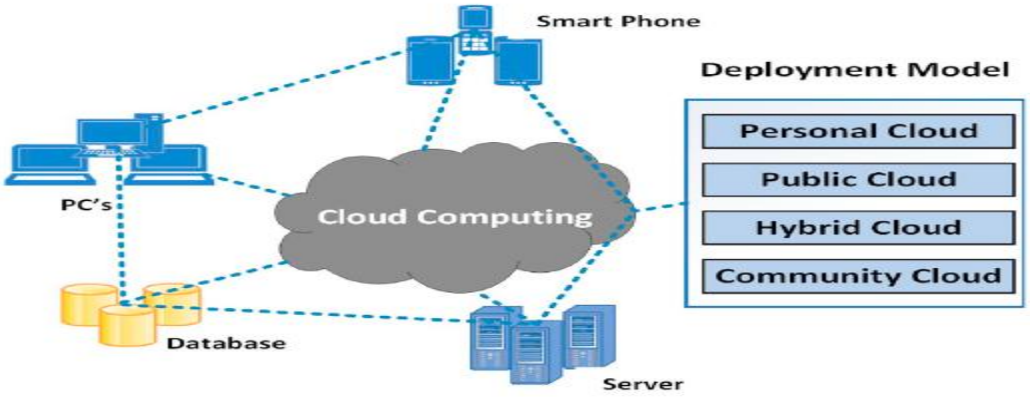


Fig. 5. A notional architecture of Cloud computing.

6.2. Proposed "DistB-SDCloud" architecture for cloud computing

To provide individual security and reliability to cloud applications, we propose a distributed, reliable, BC-based architecture framework that runs these tasks efficiently, as shown in Fig. 6. The presented architecture is divided into four distinct layers. These layers are–Data Extraction, SDN Environment, Distributed Secure BC Methodology, Cloud Computing Management, and Services for the smart IIoT applications.

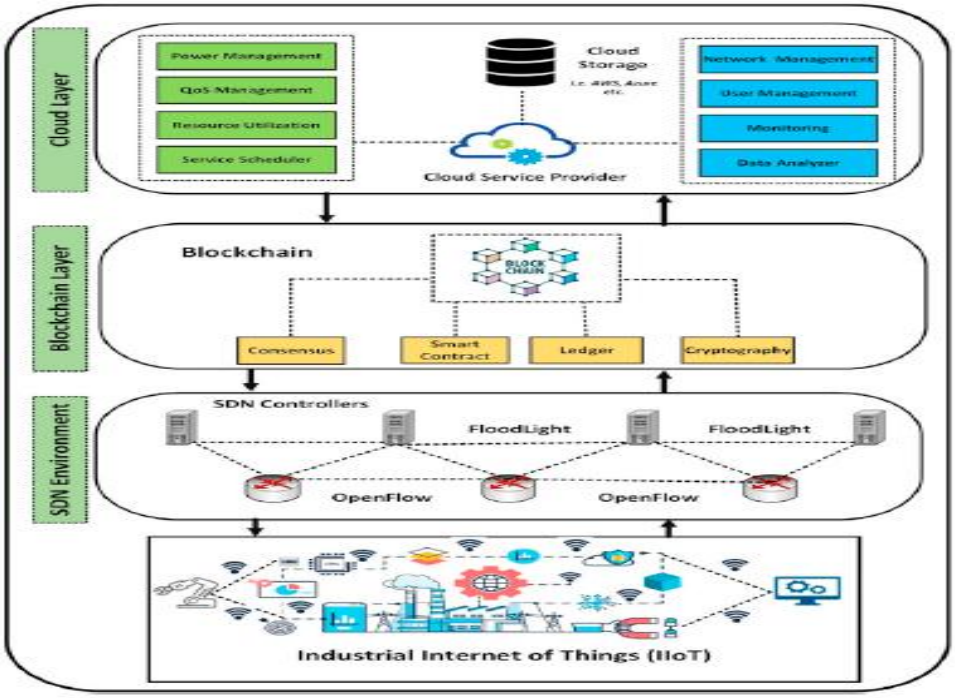


Fig. 6. Proposed "DistB-SDCloud" architecture.

6.2. Data Extraction using SDN techniques in smart IIoT applications

Several smart sensor-based devices (for example, an IoT forwarding host) can send sensor data through SDN-enabled gateways controllers, such as firewalls, switches, routers, and various data storage devices. These sensor data can be securely used in an SDN-based BC network for various operations. Furthermore, sensor data contributes to efficient performance in the distributed architecture for the smart IIoT environment. The SDN can then be separated into several planes using sensor data, including data, control, and implementation planes.

6.2.1. Data plane

As depicted in Fig. 7, the data layer is the lowest in the SDN architecture. This plane enables the SDN gateway to communicate properly with sensor-based devices (base station, switches, firewalls, data transfer, and so on). It offers two types of switches: virtual and software-based switches, which are commonly used with the Linux operating system. Another type is physical switches, which are related to hardware-based switches and take advantage of the higher flow of physical forwarding devices in the SDN infrastructure plane. These switches are responsible for forwarding and exchanging packets in network-based applications. In our architecture, the data plane collects the sensor data as it moves to and from the cloud computing environment in a smart IIoT design.

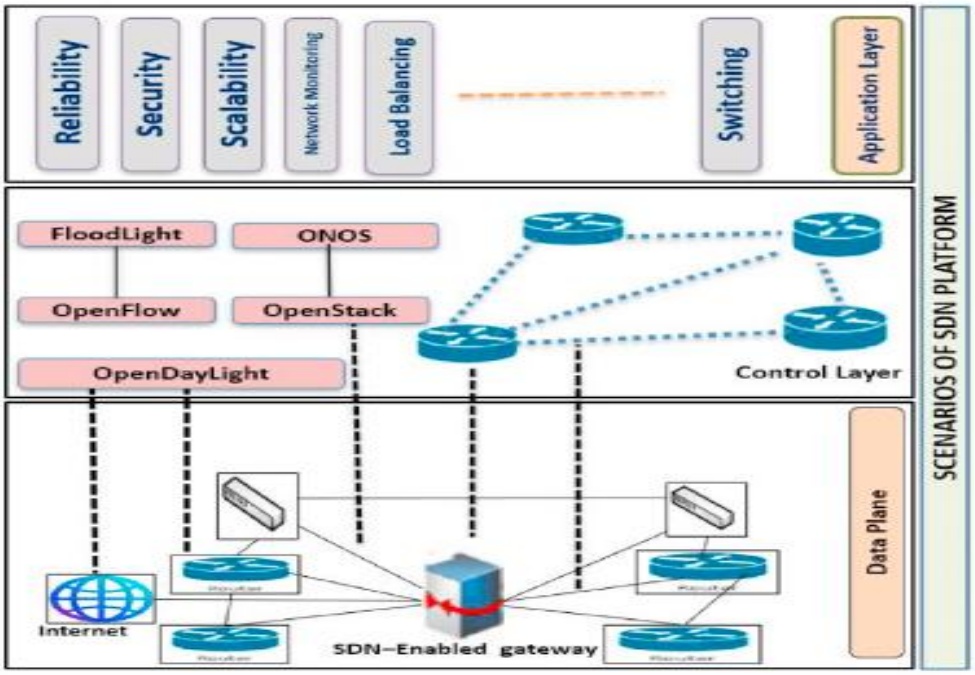


Fig. 7. Layered-based performance of SDNs.

6.2.2. Control plane

The control plane is the core foundation of the SDN architecture and the foundation of network layer communication. Although there are multiple distributed controllers at the physical level, this plane provides a high-level abstraction, giving the applications a logically centralized view. Furthermore, this plane is responsible for the interaction between infrastructure and applications planes in the SDN architecture with smart IIoT

applications. Consequently, it enables numerous networking ser vices for the desired platform. The plane's functionalities are provided through several controllers such as POX, Floodlight, OpenDayLight, Openstack, and Beacon, as well as leveraging protocols like Open Flow. Different interfaces are used to interact with devices: different interfaces–southbound, northbound, and east/west-bound. The south bound interface communicates with forwarding devices, while the northbound interface communicates with application fields; the east and westbound interfaces communicate with distributed controllers. Furthermore, this controller provides a highly flexible and stable architecture for data to flow into the cloud computing platform.

6.2.3. Application layer

The service layer is the topmost layer of the SDN. While the lower layers of the SDN-based scheme enable an effective dynamic updating of forwarding flow rules, the application layer for the smart IIot framework increases network infrastructure between the control and implementation platforms via virtual or physical forwarding. By leveraging the functionalities of the lower layers, the applications can realize complex network configuration and management, network data analytics, or specialized functions targeting specific scenarios such as large data centers.This layer enables services such as smart optimization ,mobility management, loadbalancing ,routing, switching, reliability, and network monitoring in the cloud networks ,as depicted in Fig.7.

6.3.Distributed Secure BC-SDN methodology

ABC is a form of ledger or data system that enables the addition of many functionalities to be added to a distributed or decentralized, temperature-resistant facility, as depicted in Fig. 8. It is based on the identity node of the miner and the general members' requesting node. Simultaneously, BC can provide effective access control and security for system design. In essence, BC serves as a secure ledger for recording transactions. However, it does not save all user activity in central storage or database. Furthermore, at the user end, each user uses the same storage. They also keep all transaction activities and updated copies in the same place to ensure a consensus system. In the BC environment, each block can accurately deal with multiple transactions in the smart IIoT environment. Furthermore, every block is linked by a hash chain and contains detailed information such as a timestamp, records, existing hashes, previous data, and non-conflicting transactions. Based on this concept, we believe that BC is an appropriate technology for ensuring access control in the presented cloud environment. This approach is organized as follows: security and access policy.

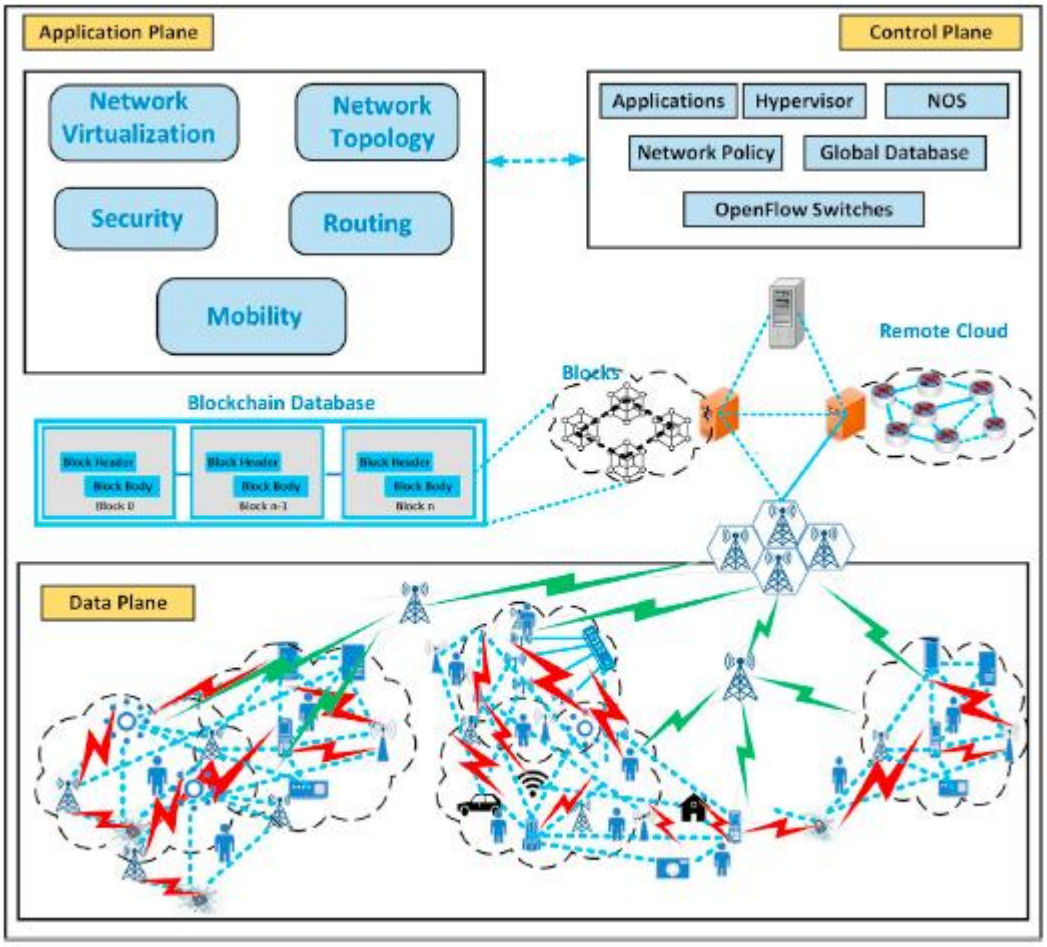


Fig. 8. BC and SDN convergence.

6.3.1. Security and access policy

By combining a Blockchain BC with a smart IIoT approach we provide adequate security for our desired system model This model prevents access by undesired users or unauthorized third parties attempting to perform access control as well as addressing data tampering and data loss Moreover the user sends a request to the BC based server which then takes the necessary action based on the request This request is simultaneously checked in the server database If the request is valid the server sends a positive response to the user allowing them to access the various services from the desired server
If the request is not valid it is discarded.

6.3.2. BC and SDN convergence

This segment discusses the relationship between BC and SDN which is depicted in Fig 8
Initially the data layer is responsible for adequately forwarding intelligent device information to another layer Moreover the data passing gateway stores all data to provide a secure platform The authors established a secure platform BC
This block helps a proffer secure the temporary database for processing the node data before entering the cloud applications Furthermore these cloud services can be managed by the desired user remotely The entire process is executed by the SDN platform efficiently SDN uses OpenFlow switch to conduct all activities in the desired application
After the completion of the convergence between BC and SDN the user will be able to integrate various services such as security remote routing mobility privacy and virtualization into the cloud data management applicatio

6.3.3. Block creation and validation

To provide security in the presented system the authors have considered the distributed platform BC The block creation and validation process of BC is depicted in Fig 9
First the authors send an encrypted block to the peer to peer networking environment
Then these blocks are validated by network nodes through the mining process
After validating the process the block is created as a novel data block
Furthermore this block is appended to the BC network as a new block
Finally the distributed ledger is efficiently updated

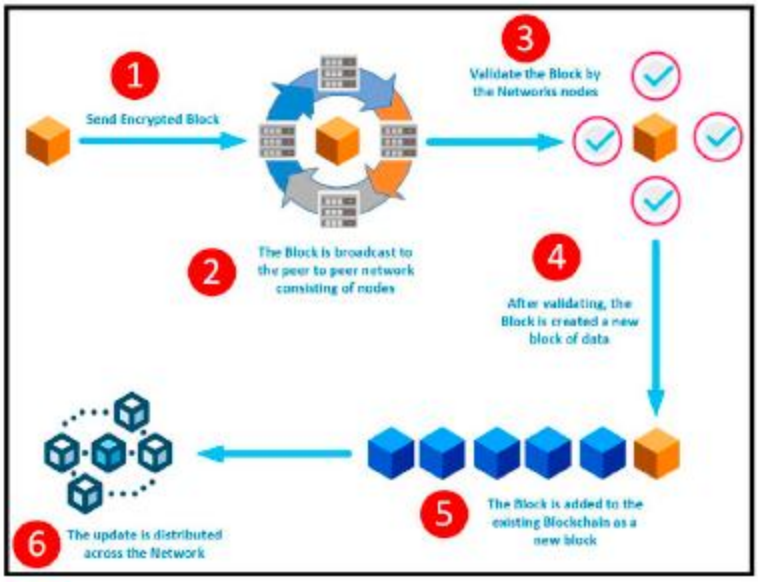


Fig. 9. Block creation and validation.

6.4. Attack mitigation in cloud environment BC-based SDN approach

SDN is extremely vulnerable to cyber-attack. Some of the most common attacks are DDoS attacks , DoS attacks, flooding attacks, and many more, now that the cloud environment is fundamental to various innovations. Attacks on this cloud storage scenario have become a significant attraction for many intruders. The overall system could be down for some time because of these attacks. We also proposed a method to recognize and block these attacks to address this problem so that the device can operate efficiently. The mechanism of the BC has been used to support the SDN controllers. The controller will detect potential attacks according to this instruction. The second issue is the avoidance or defense after the intruder has been identified. In the BC approach, the data packets are transmitted block by block. In this case, only the authorized blocks can take their place, and unauthorized blocks are discarded from the chain

6.5. Cloud Computing Management and services

Our proposed model enhances various services in the cloud computing environment by using a distributed BC approach. This BC based SDN architecture provides benefits like flexibility, accessibility, security, and privacy when retrieving, and storing numerous resources in the cloud computing platform. Without the involvement of the SDN approach, BC alone [54] cannot provide improved reliability, high stability, a logically centralized controller, and an increased load balancing capability in the proposed architecture. While the accessibility and availability of services and resources in the cloud eventually depend on internet speed, cloud computing resources perform better with Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS) when the presented architecture correctly, as depicted in Fig. 10

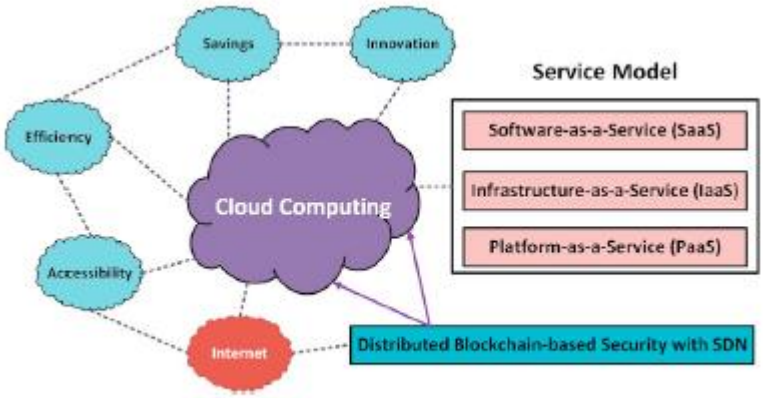
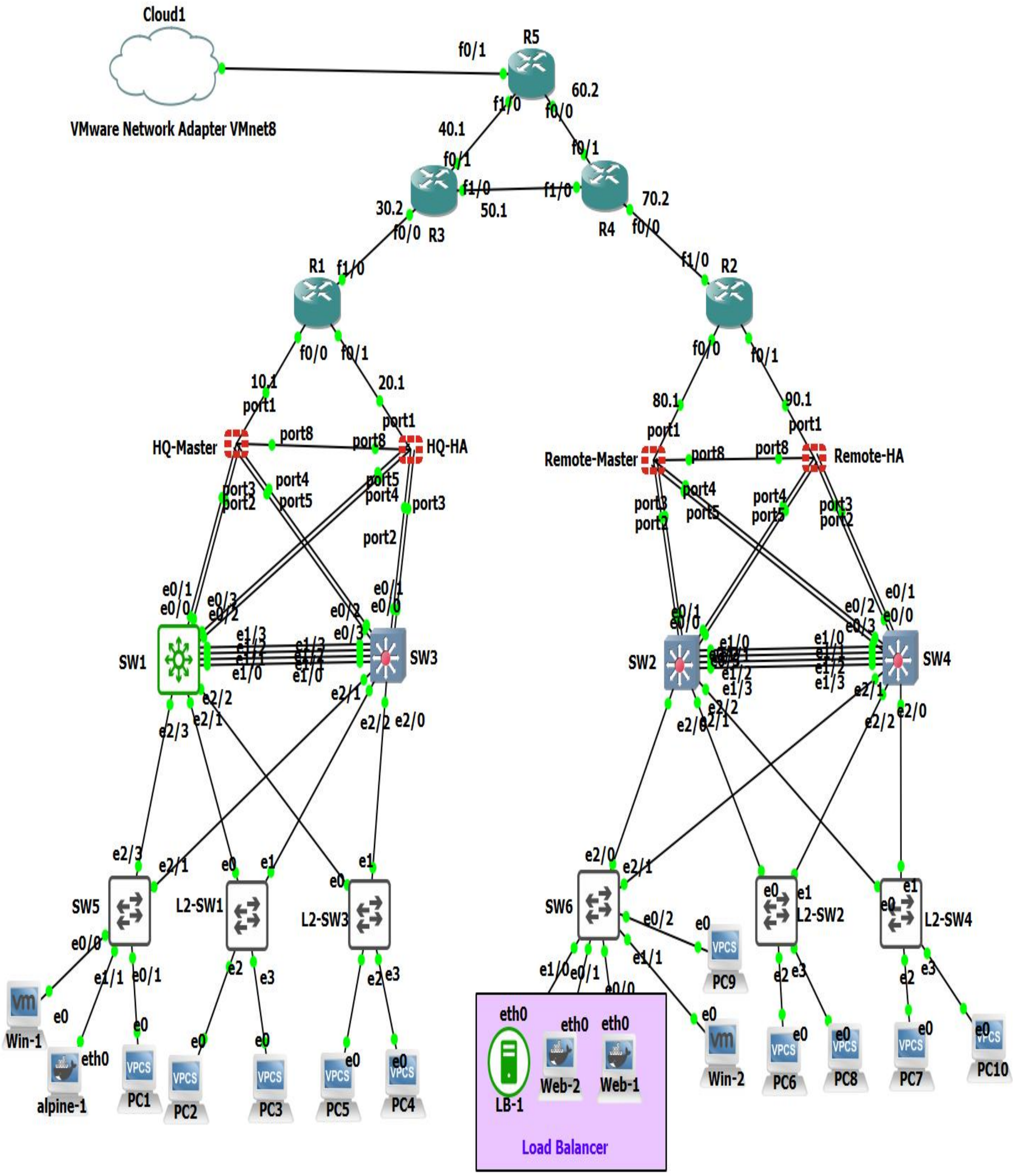


Fig. 10. Distributed BC-SDN based Cloud computing services.

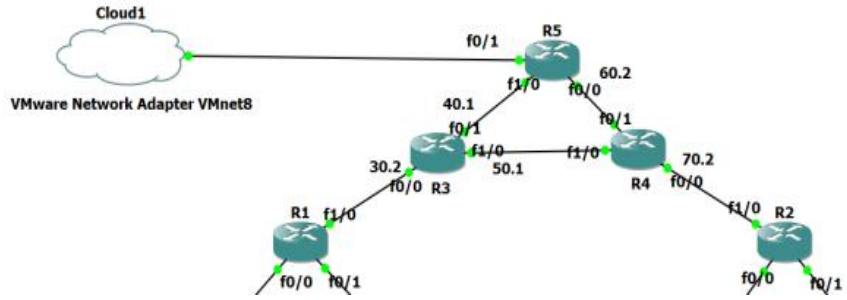
Chapter 7 : Design And Implementation

Now, let's take a closer look at the network topology used in this project. The diagram provides a visual representation of the network's layout and connections:



.Evaluate the IP Addressing Strategy for the Given Network Design:

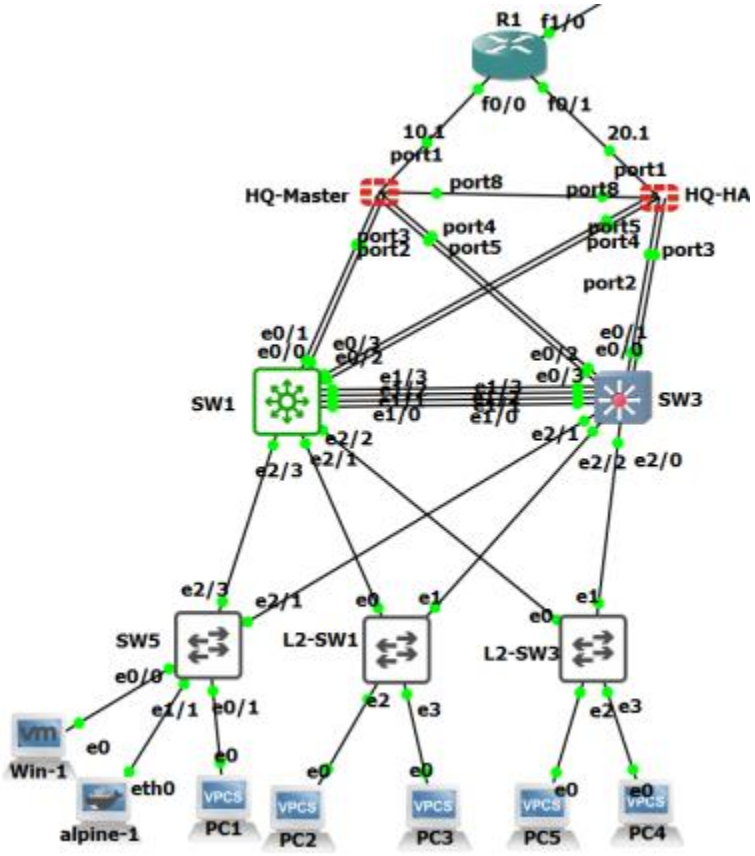
. This segment represents an ISP's infrastructure, where BGP protocol routing is in use :



Network Addressing Scheme:

Source Device	Source Port	Source IP Address	Destination Device	Destination Port	Destination IP Address
HQtest	Port-1	50.50.10.1/30	R1	Fo/0	50.50.10.2/30
HQ-HA	Port	50.50.20.1/30	R1	Fo/1	50.50.20.2/30
R1	Fo/0	50.50.30.1/30	R3	Fo/0	50.50.30.2/30
R3	Fo/1	50.50.40.1/30	R5	Fo/0	50.50.40.2/30
R3	F1/0	50.50.50.1/30	R4	F1/0	50.50.50.2/30
R4	Fo/1	50.50.60.1/30	R5	Fo/0	50.50.60.2/30
R4	Fo/0	50.50.70.2/30	R2	F1/0	50.50.70.1/30
Remote-HQ	Port-1	50.50.80.1/30	R2	Fo/0	50.50.80.2/30
Remote-HA	Port-	50.50.90.1/30	R2	Fo/1	50.50.90.2/30

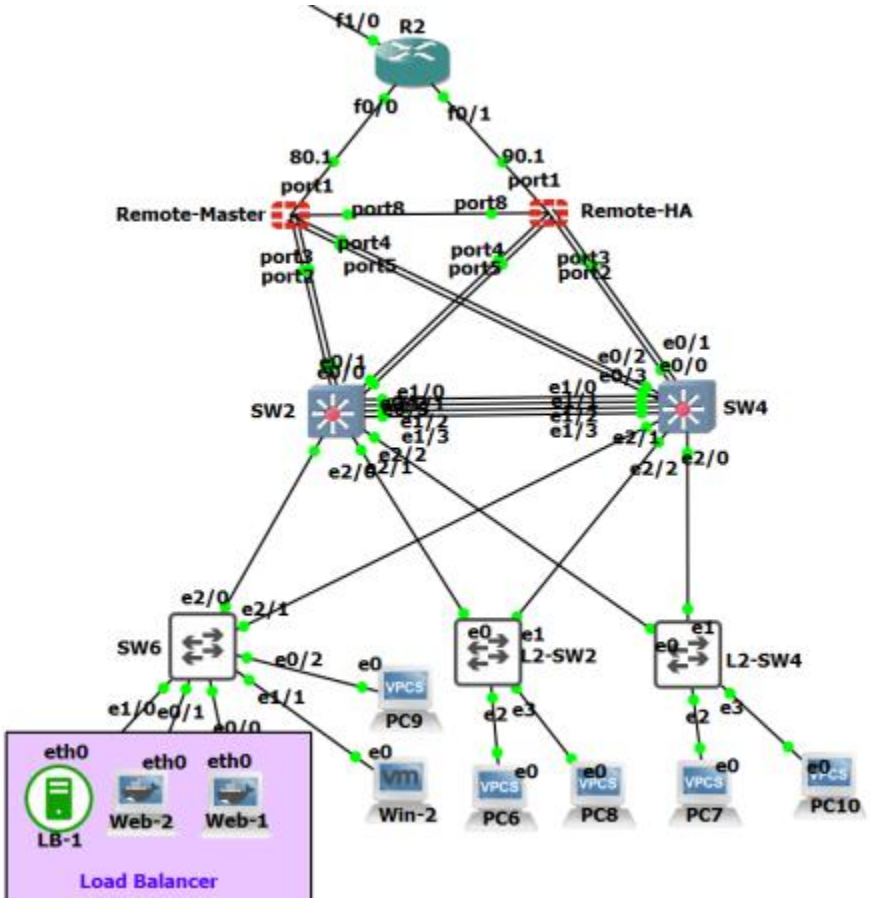
.Internal Network Segmentation: VLANs and IP Addressing:



Network Addressing Scheme:

VLAN ID	VLAN Name	IP Subnet	Associated Switches
10	IT	10.10.10.0/24	SW1, SW3
20	CS	10.10.20.0/24	SW3
30	IS	10.10.30.0/24	SW1, SW3
40	Management	10.10.40.0/24	SW1, SW3
50	Servers	10.10.50.0/24	SW1, SW3
60	Employee	10.10.60.0/24	SW1, SW3

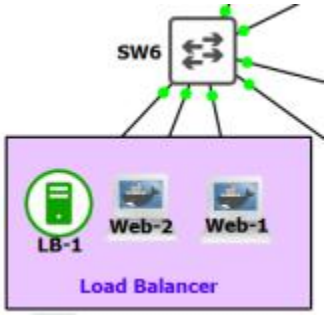
Remote Site Network Segmentation: VLANs and IP Addressing



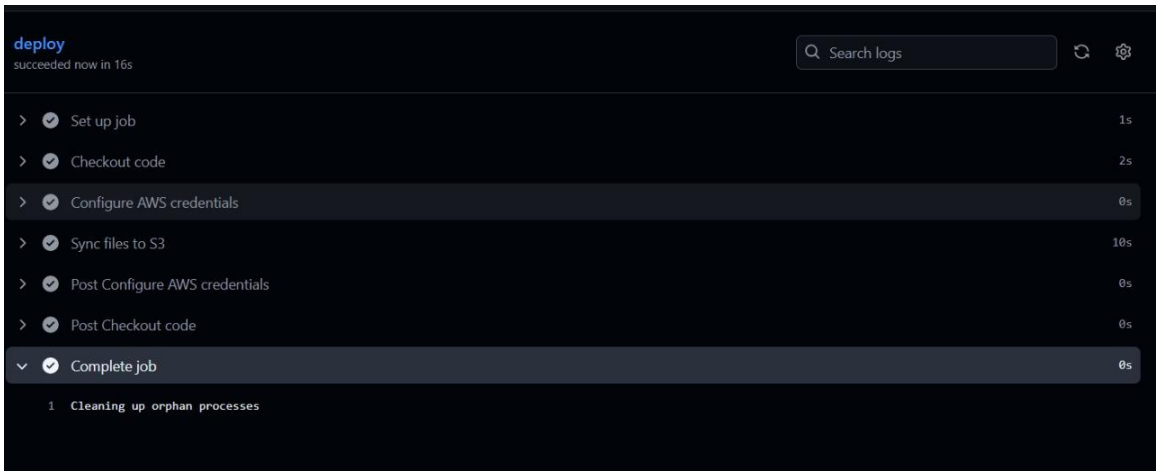
Network Addressing Scheme:

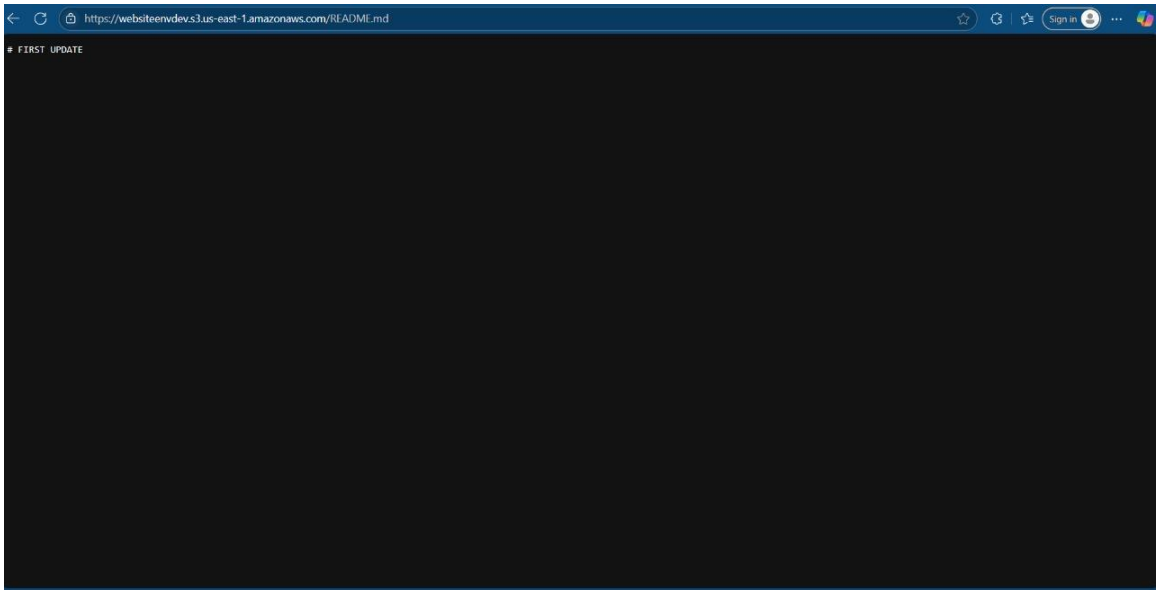
VLAN ID	VLAN Name	IP Subnet	Associated Switches
10	IT	20.20.10.0/24	SW2, SW4
20	CS	20.20.20.0/24	SW2, SW4
30	IS	20.20.30.0/24	SW2, SW4
40	Management	20.20.40.0/24	SW2, SW4
50	Servers	20.20.50.0/24	SW2, SW4
60	Employee	20.20.60.0/24	SW2, SW4

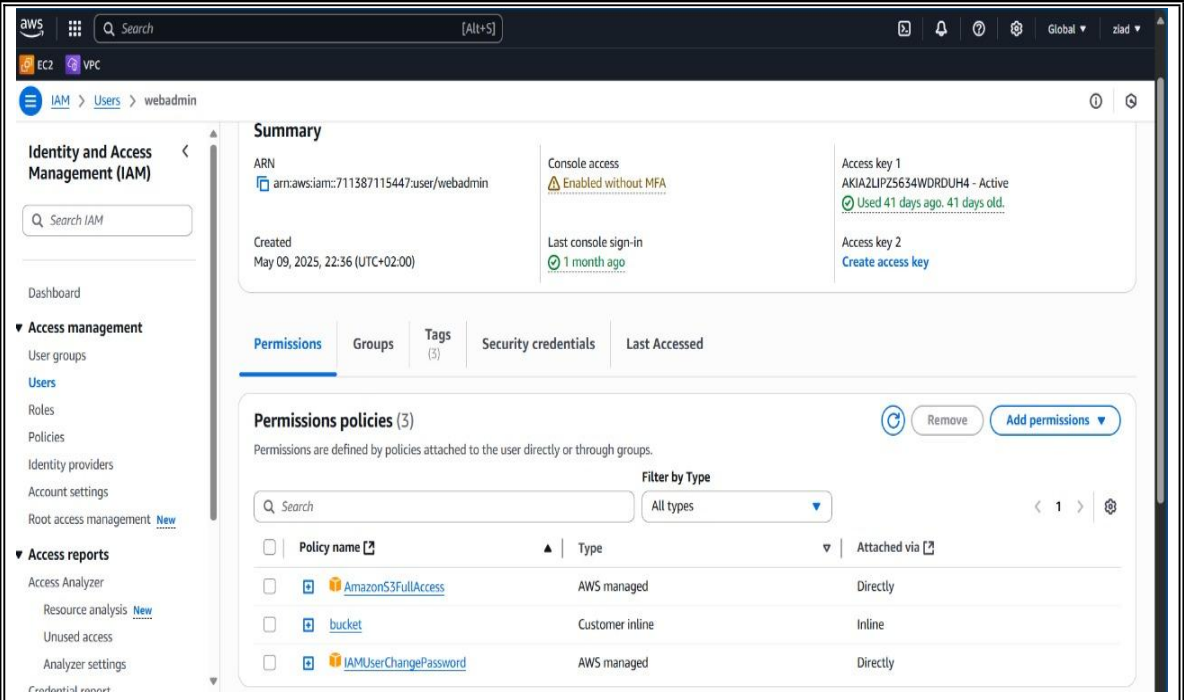
.Additionally, the remote network utilizes Docker for its Load Balancer (LB-1) and web servers (Web-1, Web-2) to manage and scale applications:

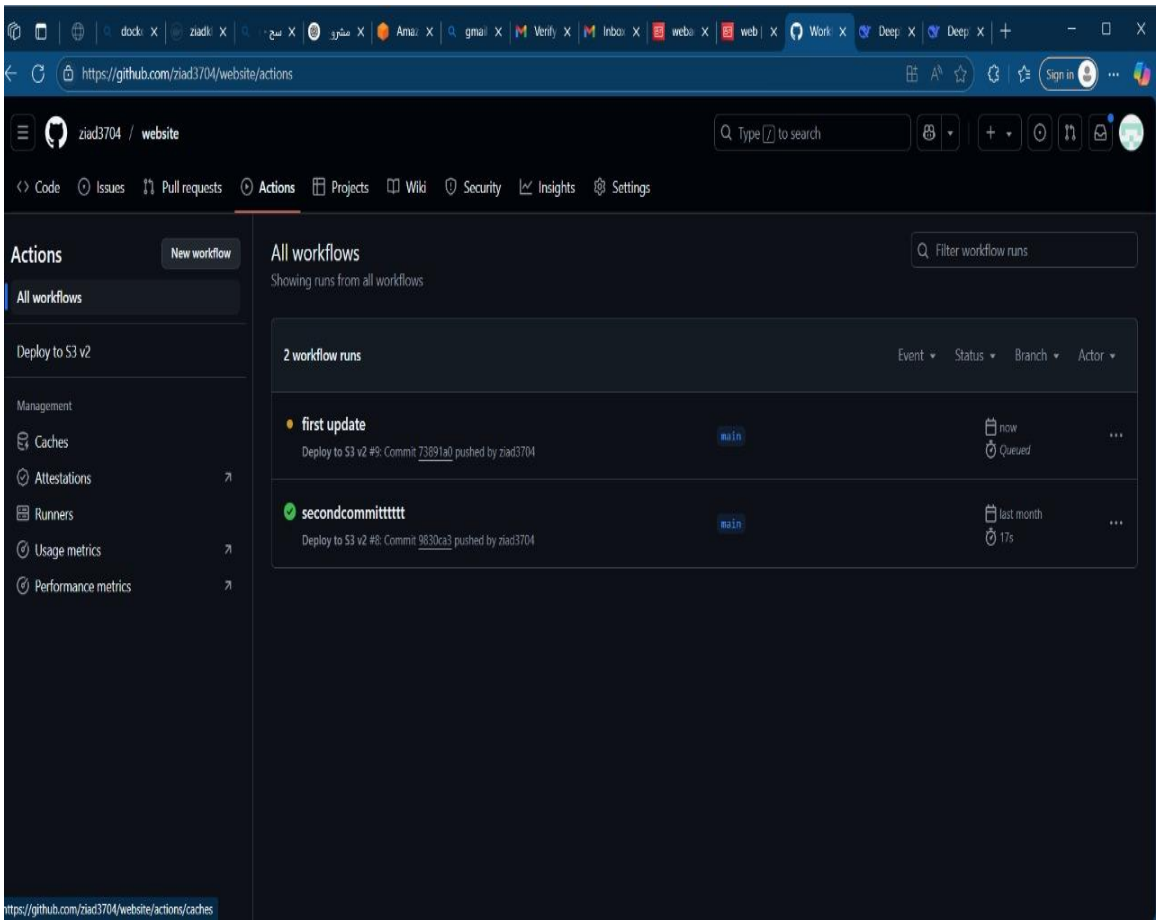


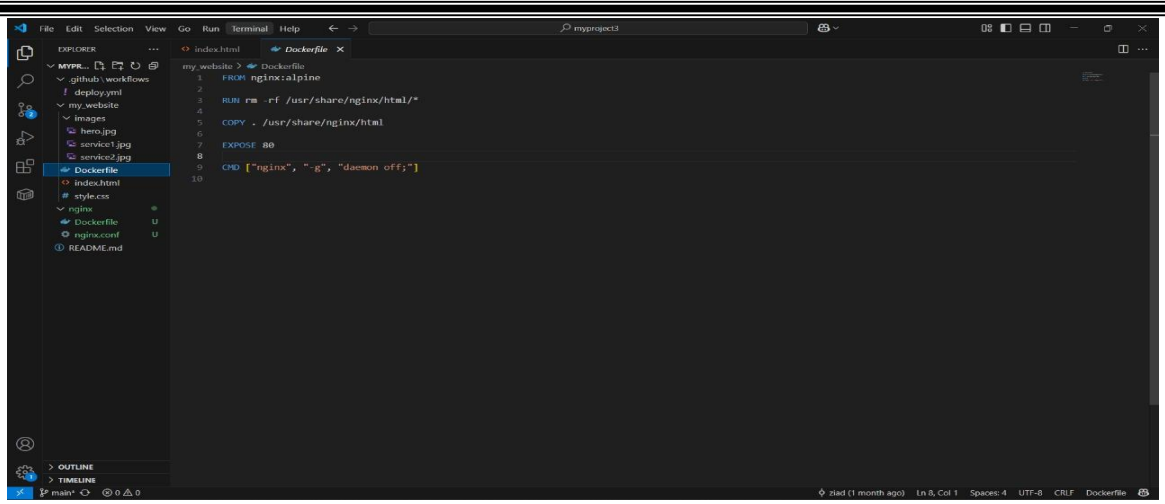
Here are the Docker configuration steps:

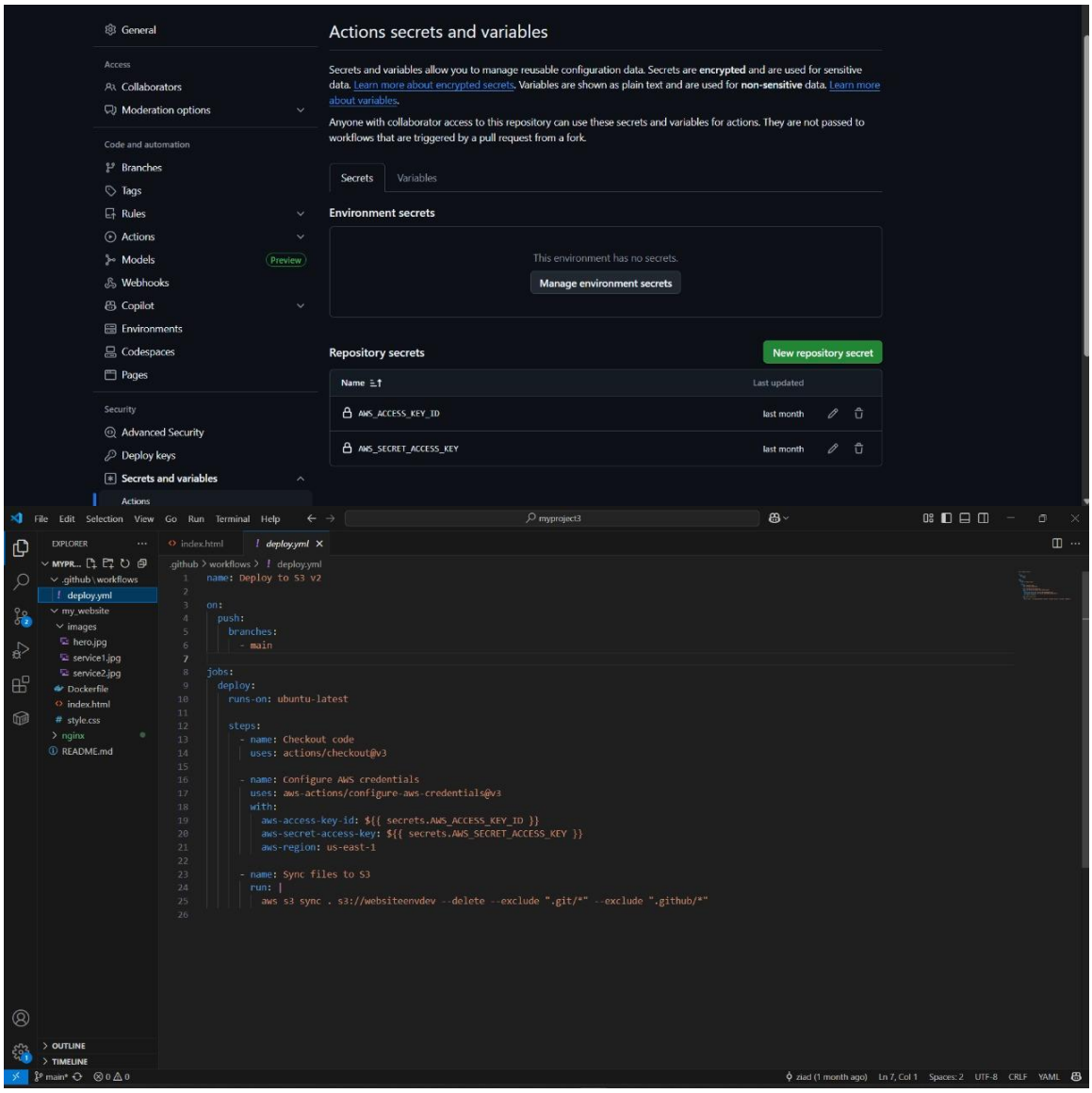


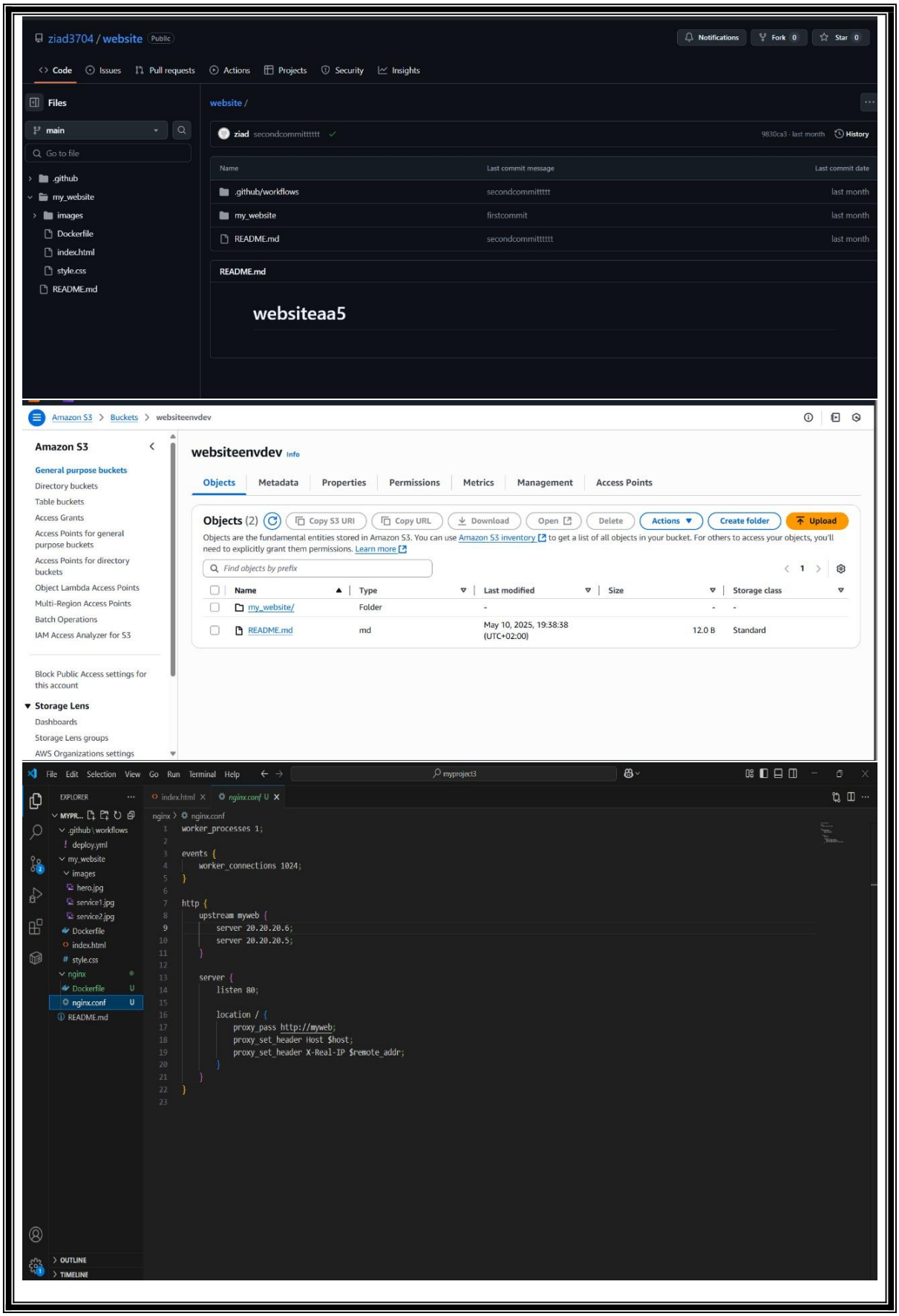



aws
Search
[Alt+S]
Global
ziad
EC2
VPC
IAM > Users > webadmin
Identity and Access Management (IAM)
Search IAM
Dashboard
Access management
User groups
Users
Roles
Policies
Identity providers
Account settings
Root access management New
Access reports
Access Analyzer
Resource analysis New
Unused access
Analyzer settings
Summary
ARN
arn:aws:iam::711387115447:user/webadmin
Console access
Enabled without MFA
Access key 1
AKIA2LIPZ5634WDRDUH4 - Active
Used 41 days ago. 41 days old.
Created
May 09, 2025, 22:36 (UTC+02:00)
Last console sign-in
1 month ago
Access key 2
Create access key
Permissions
Groups
Tags (3)
Security credentials
Last Accessed
Permissions policies (3)
Remove
Add permissions
Permissions are defined by policies attached to the user directly or through groups.
Filter by Type
Search
All types
Policy name
Type
Attached via
AmazonS3FullAccess
AWS managed
Directly
bucket
Customer inline
Inline
IAMUserChangePassword
AWS managed
Directly


https://github.com/ziad3704/website/actions
Sign in
ziad3704 / website
Type / to search
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
Actions
New workflow
All workflows
Deploy to S3 v2
Management
Caches
Attestations
Runners
Usage metrics
Performance metrics
All workflows
Showing runs from all workflows
Filter workflow runs
2 workflow runs
Event
Status
Branch
Actor
first update
Deploy to S3 v2 #9: Commit 73891a0 pushed by ziad3704
main
now
Queued
secondcommittttttt
Deploy to S3 v2 #8: Commit 9830ca3 pushed by ziad3704
main
last month
17s
https://github.com/ziad3704/website/actions/caches


File Edit Selection View Go Run Terminal Help
myproject3
EXPLORER
index.html
Dockerfile
my_website > Dockerfile
1 FROM nginx:alpine
2
3 RUN rm -rf /usr/share/nginx/html/*
4
5 COPY . /usr/share/nginx/html
6
7 EXPOSE 80
8
9 CMD ["nginx", "-g", "daemon off;"]
10
MYPR...
.github\workflows
deploy.yml
my_website
images
hero.jpg
service1.jpg
service2.jpg
Dockerfile
index.html
style.css
nginx
Dockerfile
nginx.conf
README.md
OUTLINE
TIMELINE
main*
ziad (1 month ago) Ln 8, Col 1 Spaces: 4 UTF-8 CRLF Dockerfile


General
Access
Collaborators
Moderation options
Code and automation
Branches
Tags
Rules
Actions
Models
Preview
Webhooks
Copilot
Environments
Codespaces
Pages
Security
Advanced Security
Deploy keys
Secrets and variables
Actions
Actions secrets and variables
Secrets and variables allow you to manage reusable configuration data. Secrets are encrypted and are used for sensitive data. Learn more about encrypted secrets. Variables are shown as plain text and are used for non-sensitive data. Learn more about variables.
Anyone with collaborator access to this repository can use these secrets and variables for actions. They are not passed to workflows that are triggered by a pull request from a fork.
Secrets
Variables
Environment secrets
This environment has no secrets.
Manage environment secrets
Repository secrets
New repository secret
Name
Last updated
AWS_ACCESS_KEY_ID
last month
AWS_SECRET_ACCESS_KEY
last month
File Edit Selection View Go Run Terminal Help
myproject3
EXPLORER
index.html
deploy.yml
.github > workflows > deploy.yml
1 name: Deploy to S3 v2
2
3 on:
4 push:
5 branches:
6 - main
7
8 jobs:
9 deploy:
10 runs-on: ubuntu-latest
11
12 steps:
13 - name: Checkout code
14 uses: actions/checkout@v3
15
16 - name: Configure AWS credentials
17 uses: aws-actions/configure-aws-credentials@v3
18 with:
19 aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
20 aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
21 aws-region: us-east-1
22
23 - name: Sync files to S3
24 run: |
25 aws s3 sync . s3://websiteenvdev --delete --exclude ".git/*" --exclude ".github/*"
26
MYPR...
.github\workflows
deploy.yml
my_website
images
hero.jpg
service1.jpg
service2.jpg
Dockerfile
index.html
style.css
nginx
README.md
OUTLINE
TIMELINE
main*
ziad (1 month ago) Ln 7, Col 1 Spaces: 2 UTF-8 CRLF YAML


ziad3704 / website
Public
Notifications
Fork 0
Star 0
Code
Issues
Pull requests
Actions
Projects
Security
Insights
Files
main
Go to file
.github
my_website
images
Dockerfile
index.html
style.css
README.md
website /
ziad secondcommittttttt
9830ca3 · last month
History
Name
Last commit message
Last commit date
.github/workflows
secondcommittttttt
last month
my_website
firstcommit
last month
README.md
secondcommittttttt
last month
README.md
websiteaa5
Amazon S3 > Buckets > websiteenvdev
Amazon S3
General purpose buckets
Directory buckets
Table buckets
Access Grants
Access Points for general purpose buckets
Access Points for directory buckets
Object Lambda Access Points
Multi-Region Access Points
Batch Operations
IAM Access Analyzer for S3
Block Public Access settings for this account
Storage Lens
Dashboards
Storage Lens groups
AWS Organizations settings
websiteenvdev Info
Objects
Metadata
Properties
Permissions
Metrics
Management
Access Points
Objects (2)
Copy S3 URI
Copy URL
Download
Open
Delete
Actions
Create folder
Upload
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more
Find objects by prefix
Name
Type
Last modified
Size
Storage class
my_website/
Folder
-
-
-
README.md
md
May 10, 2025, 19:38:38 (UTC+02:00)
12.0 B
Standard
File Edit Selection View Go Run Terminal Help
myproject3
EXPLORER
index.html
nginx.conf U
nginx > nginx.conf
.github \ workflows
deploy.yml
my_website
images
hero.jpg
service1.jpg
service2.jpg
Dockerfile
index.html
style.css
nginx
Dockerfile U
nginx.conf U
README.md
OUTLINE
TIMELINE
worker_processes 1;
events {
worker_connections 1024;
}
http {
upstream myweb {
server 20.20.20.6;
server 20.20.20.5;
}
server {
listen 80;
location / {
proxy_pass http://myweb;
proxy_set_header Host $host;
proxy_set_header X-Real-IP $remote_addr;
}
}
}

Web Hosting Infrastructure and CI/CD Pipeline Integration

Introduction

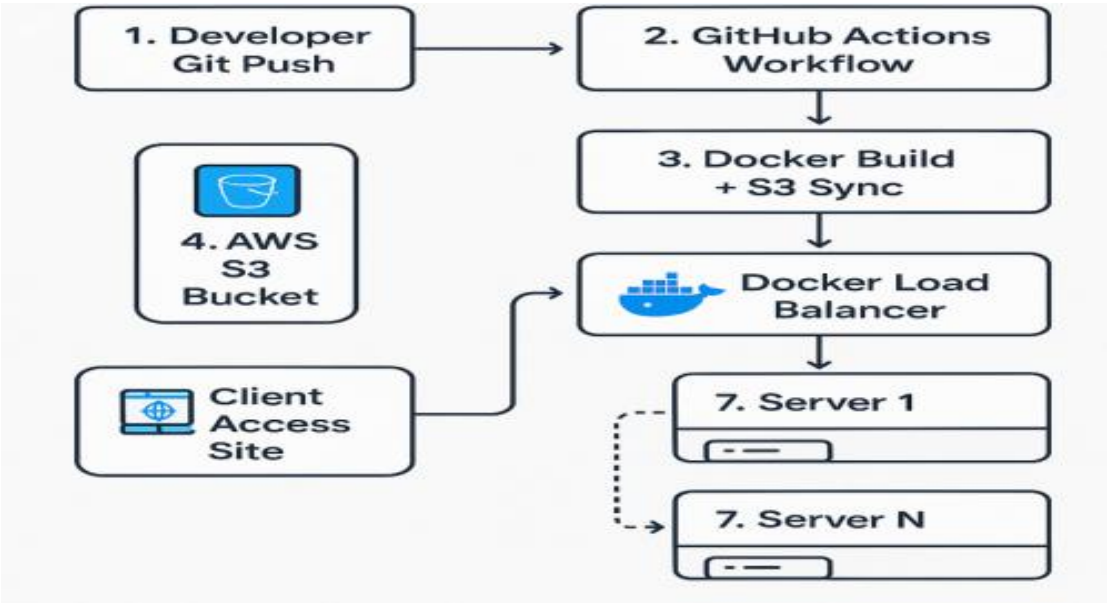
This integration utilizes Docker, NGINX, AWS S3, and GitHub Actions, thereby simulating a realistic enterprise-grade software delivery pipeline. The goal is to

streamline deployment, enable collaboration, enforce automation, and enhance scalability—key attributes in modern IT infrastructures.

This chapter presents a comprehensive walkthrough of the architectural components, implementation steps, automation pipelines, and security considerations, supported with code snippets and real deployment workflows.

Architecture Design

The architecture combines version control, CI/CD automation, containerized services, and cloud storage as illustrated conceptually below:



This structure provides a CI/CD-powered development loop:

- Developer pushes changes to GitHub.
- GitHub Actions automatically builds the site and syncs to AWS S3.
- Docker-based containers and load balancer reflect changes live.

Docker Web Servers

Each web server instance is encapsulated in a lightweight Docker container using `nginx:alpine`. The setup ensures fast deployment, reliable replication, and minimal overhead.

Dockerfile Sample:

```
Dockerfile

CopyEdit

FROM nginx:alpine

COPY ./site /usr/share/nginx/html

EXPOSE 80
```

Deployment Command:

```
bash

CopyEdit

docker build -t mysite .

docker run -d -p 8080:80 mysite
```

Advantages:

- Consistent across environments
- Containerization reduces “it works on my machine” problems.
- Rollback is easy by switching image tags.

Load Balancer with NGINX

The load balancer is another Docker container running **NGINX** with a reverse proxy configuration. It uses a round-robin strategy to distribute traffic evenly among backend containers.

Sample NGINX Config:

```
nginx

CopyEdit

upstream backend {

    server web1:80;

    server web2:80;

}

server {

    listen 80;



    location / {

        proxy_pass http://backend;

  }

}
```

Docker Compose Snippet:

```
yaml

CopyEdit

services:

  lb:
```

```
    image: nginx:alpine

    volumes:

      - ./nginx.conf:/etc/nginx/nginx.conf

    ports: - "80:80"
```

Benefits:

- High availability and fault tolerance.
- Scales horizontally by adding more containers.
- Instant reconfiguration without downtime.

To automate deployments, we utilized GitHub Actions. It responds to code commits on the `main` branch and initiates a workflow defined in `.github/workflows/deploy.yml`.

Workflow YAML Example:

```
yaml

CopyEdit

name: Deploy to AWS S3

on:

  push:

    branches: [ "main" ]

jobs:

  deploy:

    runs-on: ubuntu-latest

    steps:

      - name: Checkout code

        uses: actions/checkout@v3
```

```
      - name: Configure AWS credentials
        uses: aws-actions/configure-aws-credentials@v2
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region: us-east-1

      - name: Sync files to S3
        run: aws s3 sync ./site s3://my-sdn-project-bucket --delete
```

Automation Highlights:

- Eliminates manual deployment steps.
- Ensures synchronization with S3 for frontend hosting.
- Secure through GitHub Secrets.

Cloud Integration and IAM

For secure access, we created an IAM user with restricted permissions specifically for S3 operations.

IAM Policy Snippet:

```
json
CopyEdit
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
```

```
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:ListBucket",
        "s3:DeleteObject"
      ],
      "Resource": [
        "arn:aws:s3:::my-sdn-project-bucket/*",
        "arn:aws:s3:::my-sdn-project-bucket" ]
    }
  ]}
```

Security Practices:

- Least privilege principle applied.
- Access keys never hardcoded; stored in GitHub Secrets.
- Optionally rotate credentials periodically.

Architecture Benefits Summary

Component	Role	Benefits
Docker Web Servers	Host site inside containers	Portable, isolated, and fast deployment
NGINX Load Balancer	Route traffic to containers	Scalability and reliability
GitHub Actions	Automate build and deployment	CI/CD pipeline with no manual intervention
AWS S3	Static hosting	High availability and global reach
IAM Security	Controlled access	Enforces least privilege and audit-ability

Future Improvements

We identified potential enhancements to further modernize the platform:

1-Enable HTTPS with Let's Encrypt:

Secure content delivery and ensure compliance.

Automate certificate renewal via Certbot inside Docker.

2-Container Reloads via Webhooks:

Configure GitHub to trigger a reload script or endpoint after each deployment.

3-Add Slack/Discord Integration:

Notify team members of successful or failed deployments.

4-CloudFront + S3 Integration:

Serve content with CDN-level performance and caching.

Advanced Pipelines with Jenkins or Matrix GitHub Jobs:

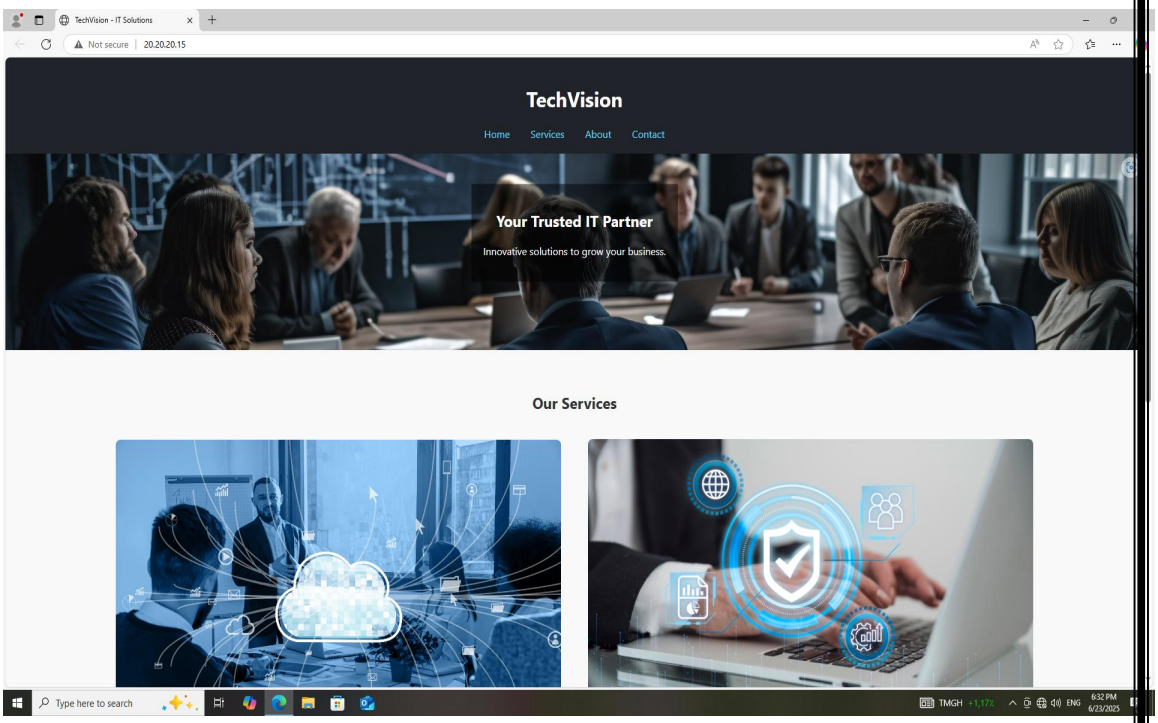
Add testing stages (unit, integration).

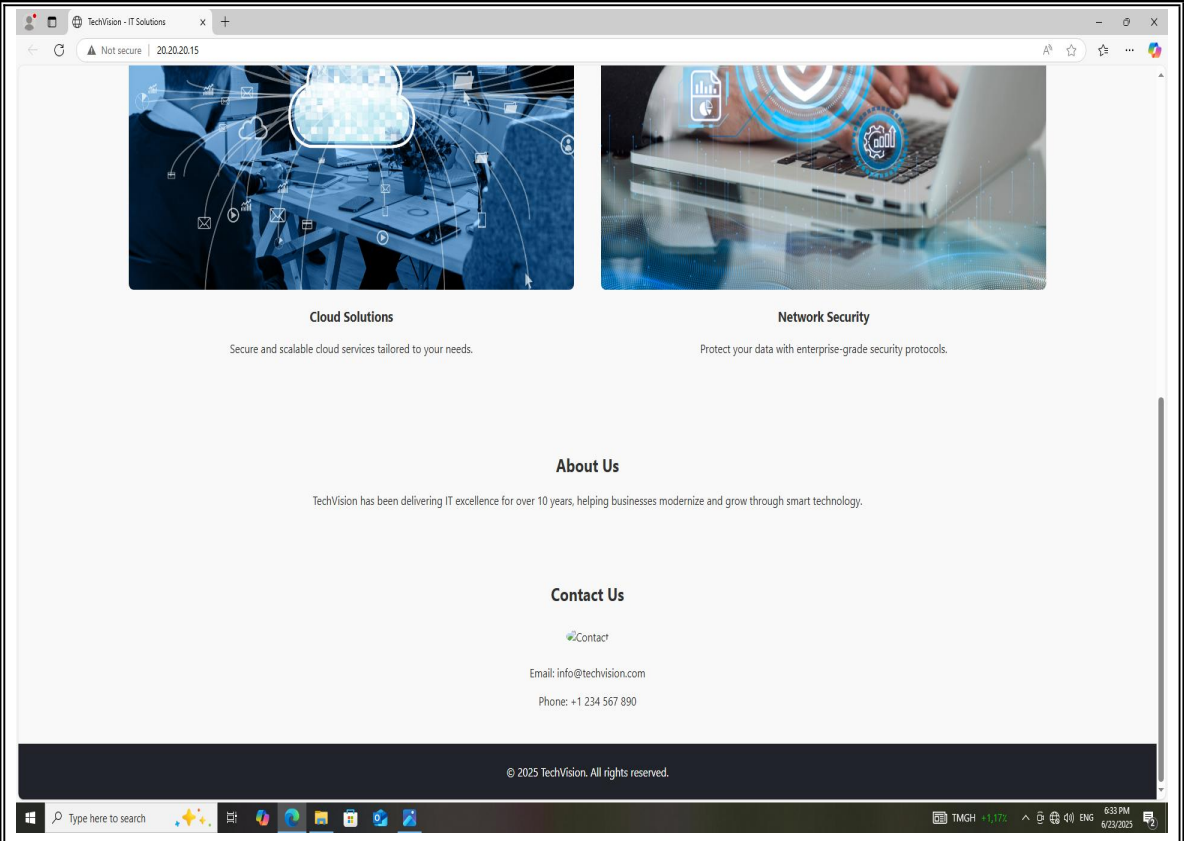
Deploy to dev/staging/prod environments using a branching strategy.

Conclusion

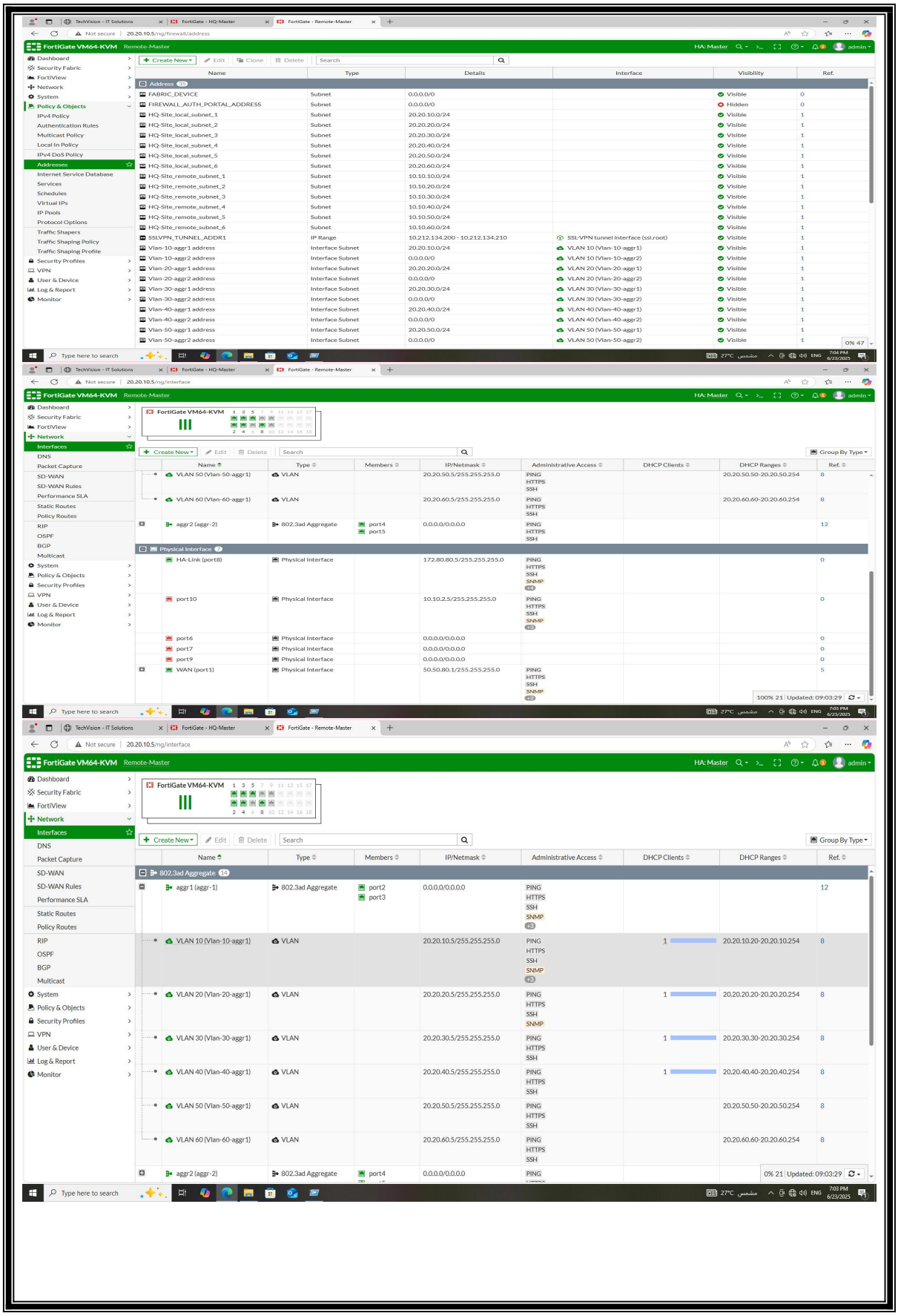
By integrating Dockerized web services, cloud storage, and GitHub-driven automation into our SDN-based graduation project, we created a production-ready DevOps infrastructure. This setup mimics professional workflows used in modern web application delivery and cloud operations. It not only enhances the value of the SDN project but also arms us with practical, in-demand skills for the real world—bridging the gap between network engineering and cloud-native software delivery.

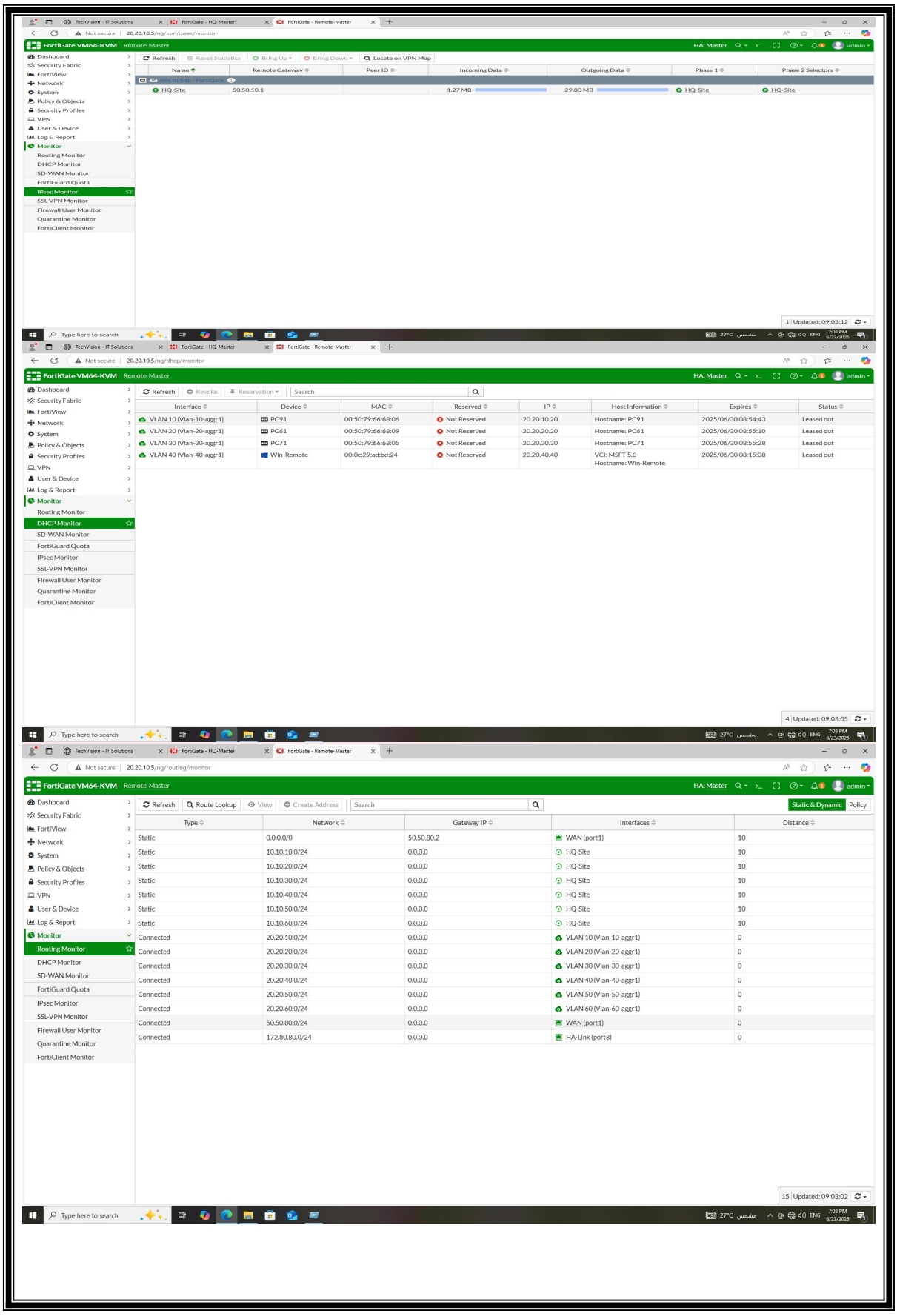
The website shown in the screenshots is hosted using Docker:

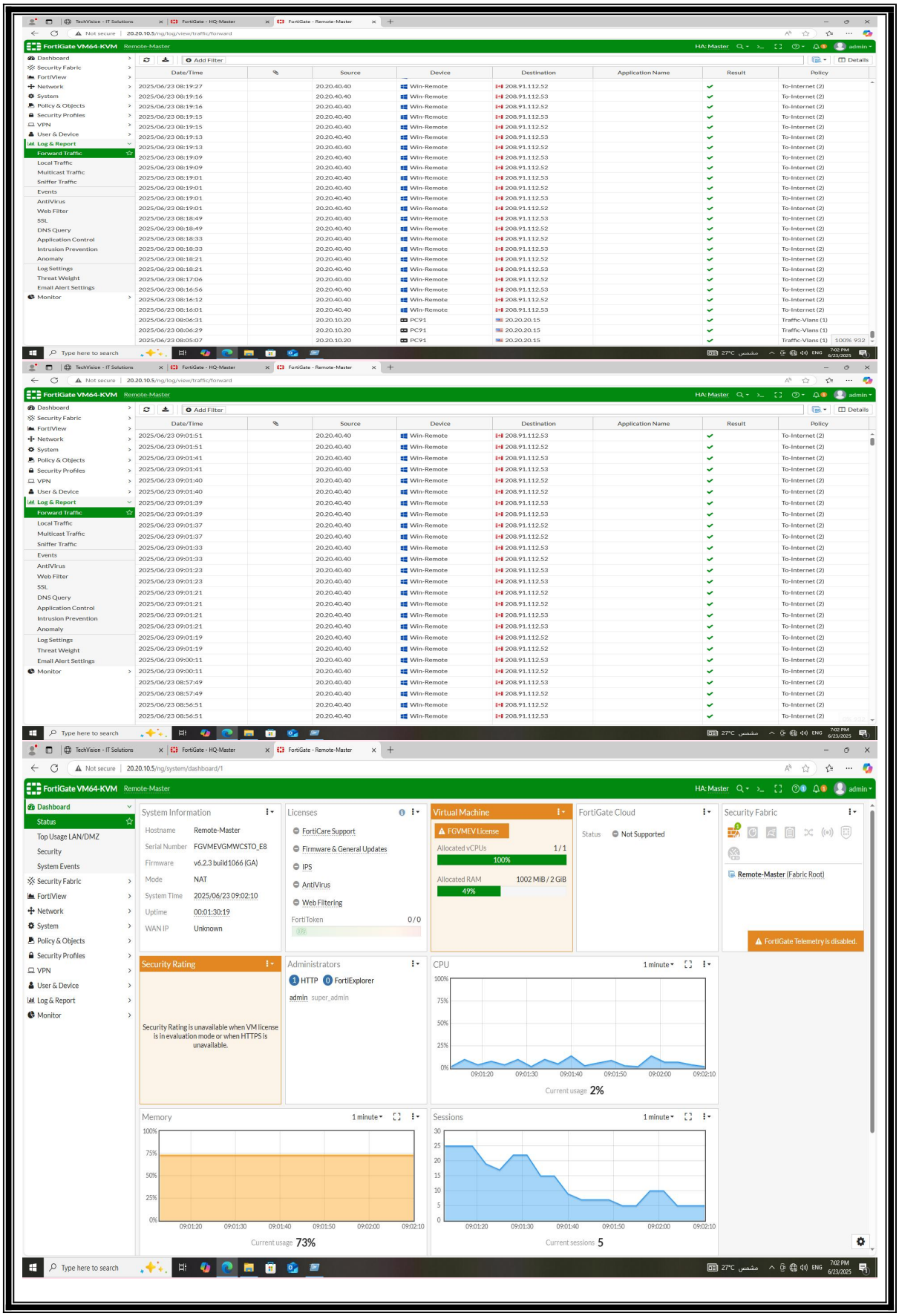


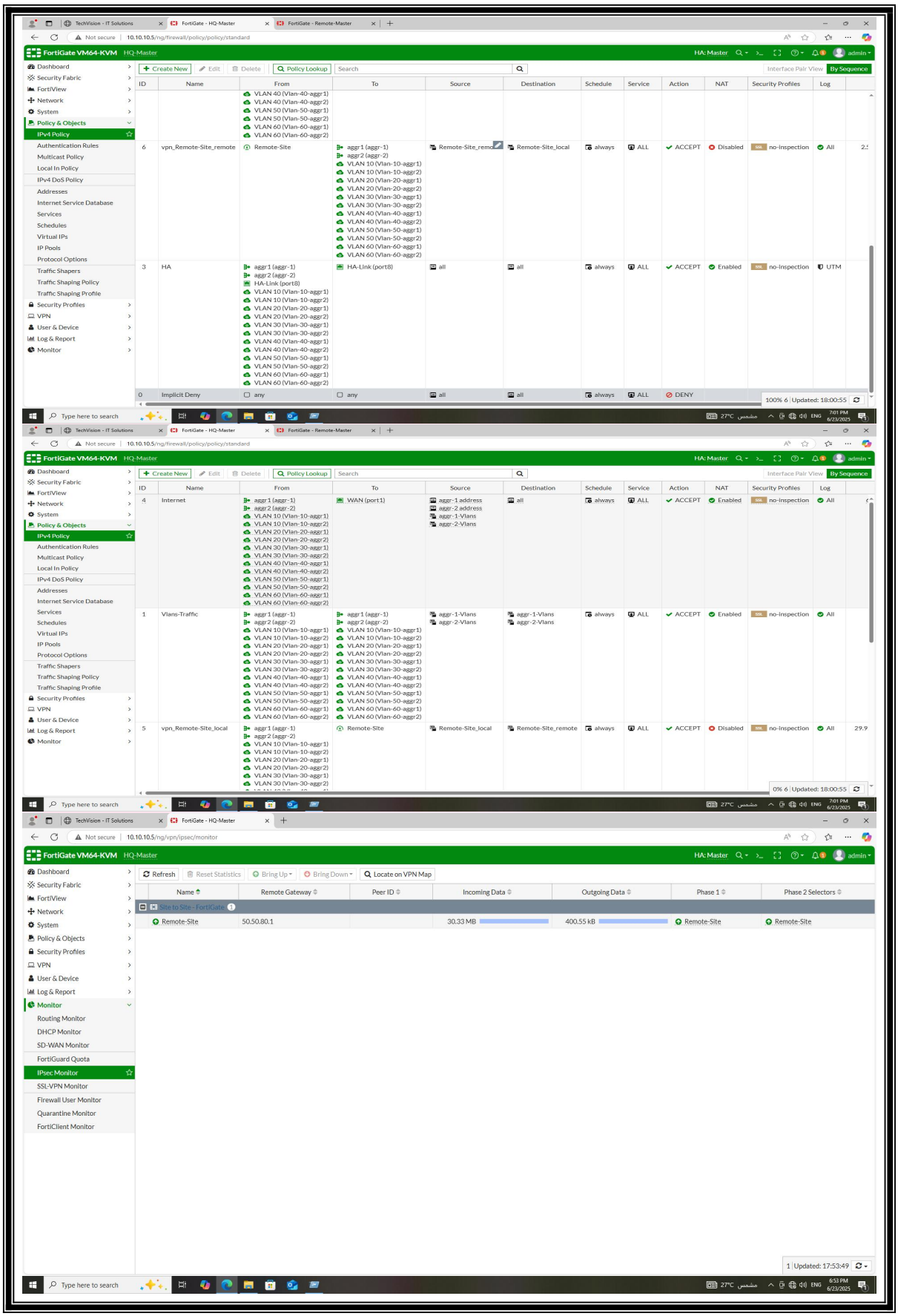


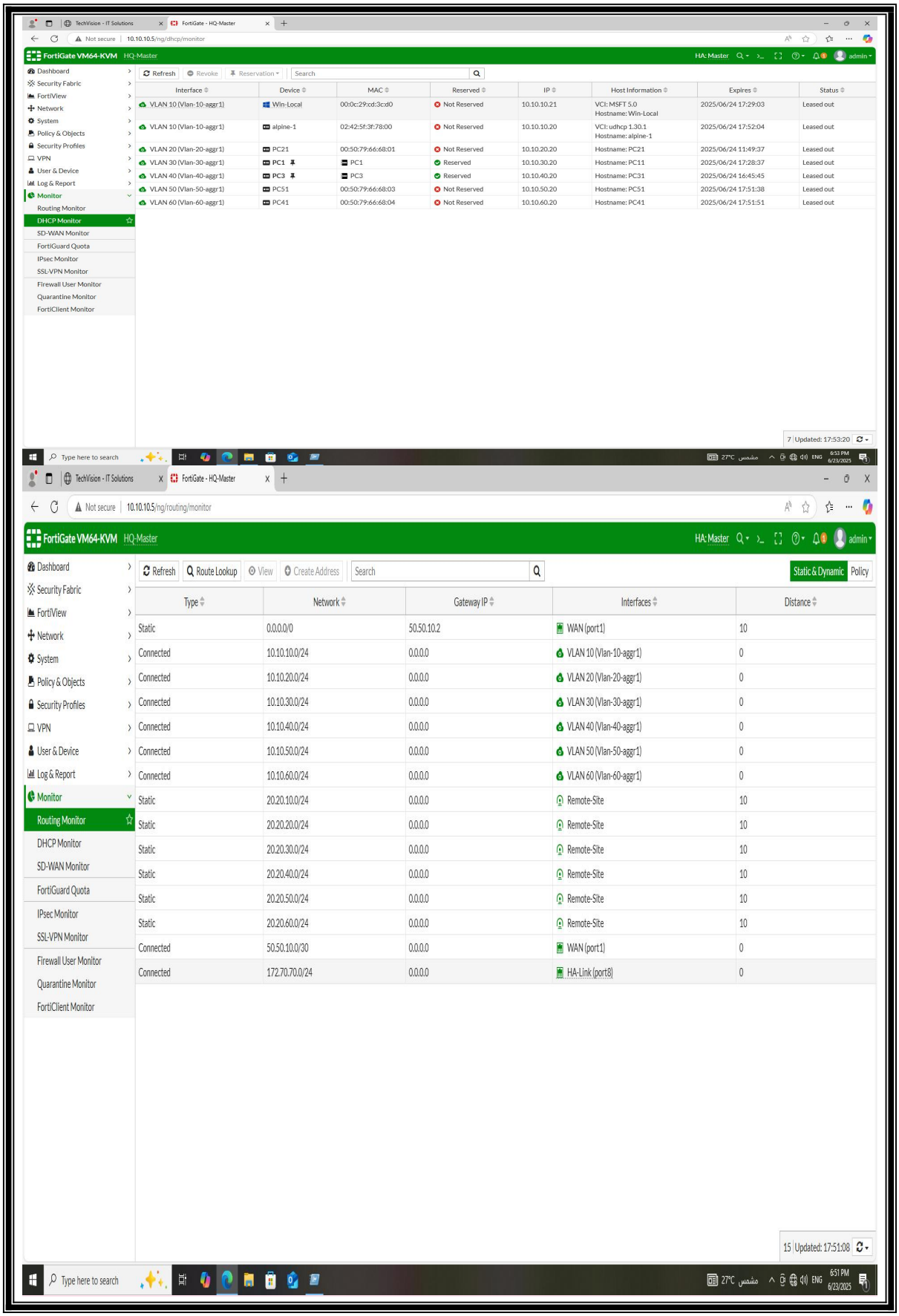
Finally, I'll go through the firewall component in the topology shown in the screenshot above:

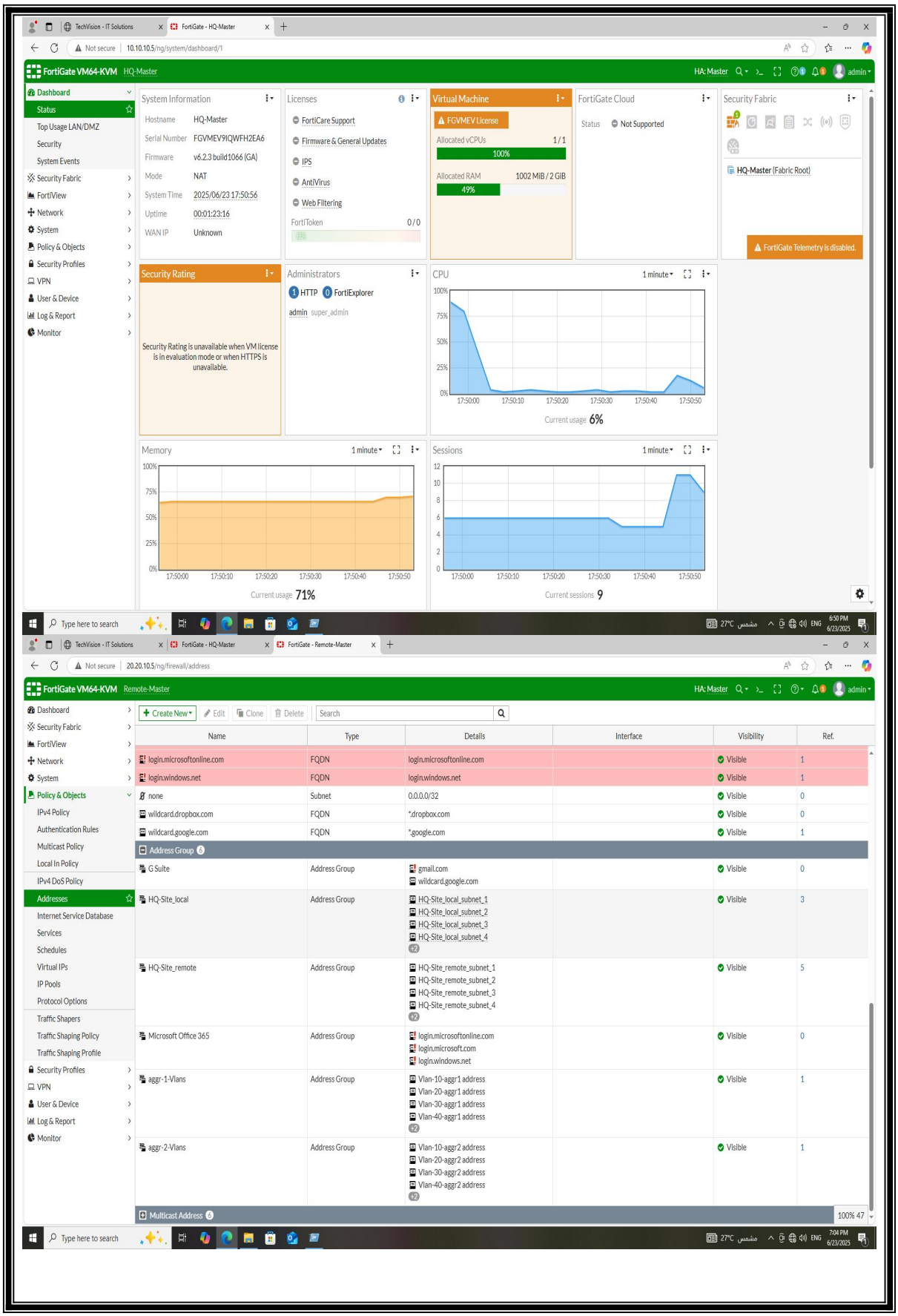

FortiGate VM64-KVM
Remote-Master
HA: Master
admin
Dashboard
Security Fabric
FortiView
Network
System
Policy & Objects
Security Profiles
VPN
User & Device
Log & Report
Monitor
Create New
Edit
Clone
Delete
Search
Name
Type
Details
Interface
Visibility
Ref.
Address
FABRIC_DEVICE
FIREWALL_AUTH_PORTAL_ADDRESS
HQ-Site_local_subnet_1
HQ-Site_local_subnet_2
HQ-Site_local_subnet_3
HQ-Site_local_subnet_4
HQ-Site_local_subnet_5
HQ-Site_local_subnet_6
HQ-Site_remote_subnet_1
HQ-Site_remote_subnet_2
HQ-Site_remote_subnet_3
HQ-Site_remote_subnet_4
HQ-Site_remote_subnet_5
HQ-Site_remote_subnet_6
SSLVPN_TUNNEL_ADDR1
Vlan-10-aggr1 address
Vlan-10-aggr2 address
Vlan-20-aggr1 address
Vlan-20-aggr2 address
Vlan-30-aggr1 address
Vlan-30-aggr2 address
Vlan-40-aggr1 address
Vlan-40-aggr2 address
Vlan-50-aggr1 address
Vlan-50-aggr2 address
Interfaces
DNS
Packet Capture
SD-WAN
SD-WAN Rules
Performance SLA
Static Routes
Policy Routes
RIP
OSPF
BGP
Multicast
Administrative Access
DHCP Clients
DHCP Ranges
VLAN 50 (Vlan-50-aggr1)
VLAN 60 (Vlan-60-aggr1)
aggr2 (aggr-2)
Physical Interface
HA-Link (port8)
port10
port6
port7
port9
WAN (port1)
aggr1 (aggr-1)
VLAN 10 (Vlan-10-aggr1)
VLAN 20 (Vlan-20-aggr1)
VLAN 30 (Vlan-30-aggr1)
VLAN 40 (Vlan-40-aggr1)
Group By Type
Type here to search


FortiGate VM64-KVM
Remote-Master
HA: Master
admin
Dashboard
Security Fabric
FortiView
Network
System
Policy & Objects
Security Profiles
VPN
User & Device
Log & Report
Monitor
Routing Monitor
DHCP Monitor
SD-WAN Monitor
FortiGuard Quota
IPsec Monitor
SSL-VPN Monitor
Firewall User Monitor
Quarantine Monitor
FortiClient Monitor
Refresh
Reset Statistics
Bring Up
Bring Down
Locate on VPN Map
Name
Remote Gateway
Peer ID
Incoming Data
Outgoing Data
Phase 1
Phase 2 Selectors
Site to Site - FortiGate
HQ-Site
50.50.10.1
1.27 MB
29.83 MB
HQ-Site
HQ-Site
1 Updated: 09:03:12
Revoke
Reservation
Search
Interface
Device
MAC
Reserved
IP
Host Information
Expires
Status
VLAN 10 (Vlan-10-aggr1)
PC91
00:50:79:66:68:06
Not Reserved
20.20.10.20
Hostname: PC91
2025/06/30 08:54:43
Leased out
VLAN 20 (Vlan-20-aggr1)
PC61
00:50:79:66:68:09
Not Reserved
20.20.20.20
Hostname: PC61
2025/06/30 08:55:10
Leased out
VLAN 30 (Vlan-30-aggr1)
PC71
00:50:79:66:68:05
Not Reserved
20.20.30.30
Hostname: PC71
2025/06/30 08:55:28
Leased out
VLAN 40 (Vlan-40-aggr1)
Win-Remote
00:0c:29:ad:bd:24
Not Reserved
20.20.40.40
VCI: MSFT 5.0
Hostname: Win-Remote
2025/06/30 08:15:08
Leased out
4 Updated: 09:03:05
Route Lookup
View
Create Address
Static & Dynamic
Policy
Type
Network
Gateway IP
Interfaces
Distance
Static
0.0.0.0/0
50.50.80.2
WAN (port1)
10
Static
10.10.10.0/24
0.0.0.0
HQ-Site
10
Static
10.10.20.0/24
0.0.0.0
HQ-Site
10
Static
10.10.30.0/24
0.0.0.0
HQ-Site
10
Static
10.10.40.0/24
0.0.0.0
HQ-Site
10
Static
10.10.50.0/24
0.0.0.0
HQ-Site
10
Static
10.10.60.0/24
0.0.0.0
HQ-Site
10
Connected
20.20.10.0/24
0.0.0.0
VLAN 10 (Vlan-10-aggr1)
0
Connected
20.20.20.0/24
0.0.0.0
VLAN 20 (Vlan-20-aggr1)
0
Connected
20.20.30.0/24
0.0.0.0
VLAN 30 (Vlan-30-aggr1)
0
Connected
20.20.40.0/24
0.0.0.0
VLAN 40 (Vlan-40-aggr1)
0
Connected
20.20.50.0/24
0.0.0.0
VLAN 50 (Vlan-50-aggr1)
0
Connected
20.20.60.0/24
0.0.0.0
VLAN 60 (Vlan-60-aggr1)
0
Connected
50.50.80.0/24
0.0.0.0
WAN (port1)
0
Connected
172.80.80.0/24
0.0.0.0
HA-Link (port8)
0
15 Updated: 09:03:02
Type here to search


FortiGate VM64-KVM
Remote-Master
HA: Master
admin
Dashboard
Security Fabric
FortiView
Network
System
Policy & Objects
Security Profiles
VPN
User & Device
Log & Report
Forward Traffic
Local Traffic
Multicast Traffic
Sniffer Traffic
Events
AntiVirus
Web Filter
SSL
DNS Query
Application Control
Intrusion Prevention
Anomaly
Log Settings
Threat Weight
Email Alert Settings
Monitor
Add Filter
Details
Date/Time
Source
Device
Destination
Application Name
Result
Policy
2025/06/23 08:19:27
2025/06/23 08:19:16
2025/06/23 08:19:15
2025/06/23 08:19:13
2025/06/23 08:19:09
2025/06/23 08:19:01
2025/06/23 08:18:49
2025/06/23 08:18:33
2025/06/23 08:18:21
2025/06/23 08:17:06
2025/06/23 08:16:56
2025/06/23 08:16:12
2025/06/23 08:16:01
2025/06/23 08:06:31
2025/06/23 08:06:29
2025/06/23 08:05:07
20.20.40.40
20.20.10.20
Win-Remote
PC91
208.91.112.52
208.91.112.53
20.20.20.15
To-Internet (2)
Traffic-Vlans (1)
2025/06/23 09:01:51
2025/06/23 09:01:41
2025/06/23 09:01:40
2025/06/23 09:01:39
2025/06/23 09:01:37
2025/06/23 09:01:33
2025/06/23 09:01:23
2025/06/23 09:01:21
2025/06/23 09:01:19
2025/06/23 09:00:11
2025/06/23 08:57:49
2025/06/23 08:56:51
Not secure
20.20.10.5/ng/log/view/traffic/forward
20.20.10.5/ng/system/dashboard/1
TechVision - IT Solutions
FortiGate - HQ-Master
FortiGate - Remote-Master
Dashboard
Status
Top Usage LAN/DMZ
Security
System Events
System Information
Hostname
Remote-Master
Serial Number
FGVMEVGMWCSTO_E8
Firmware
v6.2.3 build1066 (GA)
Mode
NAT
System Time
2025/06/23 09:02:10
Uptime
00:01:30:19
WAN IP
Unknown
Licenses
FortiCare Support
Firmware & General Updates
IPS
AntiVirus
Web Filtering
FortiToken
0 / 0
Virtual Machine
FGVMEV License
Allocated vCPUs
1 / 1
100%
Allocated RAM
1002 MiB / 2 GiB
49%
FortiGate Cloud
Status
Not Supported
Security Fabric
Remote-Master (Fabric Root)
FortiGate Telemetry is disabled.
Security Rating
Security Rating is unavailable when VM license is in evaluation mode or when HTTPS is unavailable.
Administrators
1 HTTP
0 FortiExplorer
admin super_admin
CPU
1 minute
100%
75%
50%
25%
0%
09:01:20
09:01:30
09:01:40
09:01:50
09:02:00
09:02:10
Current usage 2%
Memory
1 minute
Current usage 73%
Sessions
1 minute
30
25
20
15
10
5
0
Current sessions 5
Type here to search
27°C
ENG
7:02 PM
6/23/2025


FortiGate VM64-KVM
HQ-Master
IPv4 Policy
vpn_Remote-Site_remote
HA
Implicit Deny
Internet
Vlans-Traffic
vpn_Remote-Site_local
IPsec Monitor
Remote-Site
50.50.80.1
30.33 MB
400.55 kB


TechVision - IT Solutions
FortiGate - HQ-Master
Not secure | 10.10.10.5/ng/dhcp/monitor
FortiGate VM64-KVM HQ-Master
HA: Master admin
Dashboard
Security Fabric
FortiView
Network
System
Policy & Objects
Security Profiles
VPN
User & Device
Log & Report
Monitor
Routing Monitor
DHCP Monitor
SD-WAN Monitor
FortiGuard Quota
IPsec Monitor
SSL-VPN Monitor
Firewall User Monitor
Quarantine Monitor
FortiClient Monitor
Refresh
Revoke
Reservation
Search
Interface | Device | MAC | Reserved | IP | Host Information | Expires | Status
VLAN 10 (Vlan-10-aggr1) | Win-Local | 00:0c:29:cd:3c:d0 | Not Reserved | 10.10.10.21 | VCI: MSFT 5.0 Hostname: Win-Local | 2025/06/24 17:29:03 | Leased out
VLAN 10 (Vlan-10-aggr1) | alpine-1 | 02:42:5f:3f:78:00 | Not Reserved | 10.10.10.20 | VCI: udhcp 1.30.1 Hostname: alpine-1 | 2025/06/24 17:52:04 | Leased out
VLAN 20 (Vlan-20-aggr1) | PC21 | 00:50:79:66:68:01 | Not Reserved | 10.10.20.20 | Hostname: PC21 | 2025/06/24 11:49:37 | Leased out
VLAN 30 (Vlan-30-aggr1) | PC1 | PC1 | Reserved | 10.10.30.20 | Hostname: PC11 | 2025/06/24 17:28:37 | Leased out
VLAN 40 (Vlan-40-aggr1) | PC3 | PC3 | Reserved | 10.10.40.20 | Hostname: PC31 | 2025/06/24 16:45:45 | Leased out
VLAN 50 (Vlan-50-aggr1) | PC51 | 00:50:79:66:68:03 | Not Reserved | 10.10.50.20 | Hostname: PC51 | 2025/06/24 17:51:38 | Leased out
VLAN 60 (Vlan-60-aggr1) | PC41 | 00:50:79:66:68:04 | Not Reserved | 10.10.60.20 | Hostname: PC41 | 2025/06/24 17:51:51 | Leased out
7 Updated: 17:53:20
Type here to search
TechVision - IT Solutions
FortiGate - HQ-Master
Not secure | 10.10.10.5/ng/routing/monitor
FortiGate VM64-KVM HQ-Master
HA: Master admin
Refresh
Route Lookup
View
Create Address
Search
Static & Dynamic
Policy
Type | Network | Gateway IP | Interfaces | Distance
Static | 0.0.0.0/0 | 50.50.10.2 | WAN (port1) | 10
Connected | 10.10.10.0/24 | 0.0.0.0 | VLAN 10 (Vlan-10-aggr1) | 0
Connected | 10.10.20.0/24 | 0.0.0.0 | VLAN 20 (Vlan-20-aggr1) | 0
Connected | 10.10.30.0/24 | 0.0.0.0 | VLAN 30 (Vlan-30-aggr1) | 0
Connected | 10.10.40.0/24 | 0.0.0.0 | VLAN 40 (Vlan-40-aggr1) | 0
Connected | 10.10.50.0/24 | 0.0.0.0 | VLAN 50 (Vlan-50-aggr1) | 0
Connected | 10.10.60.0/24 | 0.0.0.0 | VLAN 60 (Vlan-60-aggr1) | 0
Static | 20.20.10.0/24 | 0.0.0.0 | Remote-Site | 10
Static | 20.20.20.0/24 | 0.0.0.0 | Remote-Site | 10
Static | 20.20.30.0/24 | 0.0.0.0 | Remote-Site | 10
Static | 20.20.40.0/24 | 0.0.0.0 | Remote-Site | 10
Static | 20.20.50.0/24 | 0.0.0.0 | Remote-Site | 10
Static | 20.20.60.0/24 | 0.0.0.0 | Remote-Site | 10
Connected | 50.50.10.0/30 | 0.0.0.0 | WAN (port1) | 0
Connected | 172.70.70.0/24 | 0.0.0.0 | HA-Link (port8) | 0
15 Updated: 17:51:08
Type here to search
27°C مشمس
ENG
6:51 PM
6/23/2025


FortiGate VM64-KVM HQ-Master
HA: Master admin
Dashboard
Status
Top Usage LAN/DMZ
Security
System Events
Security Fabric
FortiView
Network
System
Policy & Objects
Security Profiles
VPN
User & Device
Log & Report
Monitor
System Information
Hostname HQ-Master
Serial Number FGVMEV9IQWFH2EA6
Firmware v6.2.3 build1066 (GA)
Mode NAT
System Time 2025/06/23 17:50:56
Uptime 00:01:23:16
WAN IP Unknown
Licenses
FortiCare Support
Firmware & General Updates
IPS
AntiVirus
Web Filtering
FortiToken 0 / 0
Virtual Machine
FGVMEV License
Allocated vCPUs 1 / 1
100%
Allocated RAM 1002 MiB / 2 GiB
49%
FortiGate Cloud
Status Not Supported
Security Fabric
HQ-Master (Fabric Root)
FortiGate Telemetry is disabled.
Security Rating
Security Rating is unavailable when VM license is in evaluation mode or when HTTPS is unavailable.
Administrators
1 HTTP 0 FortiExplorer
admin super_admin
CPU
1 minute
Current usage 6%
Memory
1 minute
Current usage 71%
Sessions
1 minute
Current sessions 9
FortiGate VM64-KVM Remote-Master
HA: Master admin
Create New
Edit
Clone
Delete
Search
Name
Type
Details
Interface
Visibility
Ref.
login.microsoftonline.com FQDN login.microsoftonline.com Visible 1
login.windows.net FQDN login.windows.net Visible 1
none Subnet 0.0.0.0/32 Visible 0
wildcard.dropbox.com FQDN *.dropbox.com Visible 0
wildcard.google.com FQDN *.google.com Visible 1
Address Group 6
G Suite Address Group gmail.com, wildcard.google.com Visible 0
HQ-Site_local Address Group HQ-Site_local_subnet_1, HQ-Site_local_subnet_2, HQ-Site_local_subnet_3, HQ-Site_local_subnet_4 Visible 3
HQ-Site_remote Address Group HQ-Site_remote_subnet_1, HQ-Site_remote_subnet_2, HQ-Site_remote_subnet_3, HQ-Site_remote_subnet_4 Visible 5
Microsoft Office 365 Address Group login.microsoftonline.com, login.microsoft.com, login.windows.net Visible 0
aggr-1-Vlans Address Group Vlan-10-aggr1 address, Vlan-20-aggr1 address, Vlan-30-aggr1 address, Vlan-40-aggr1 address Visible 1
aggr-2-Vlans Address Group Vlan-10-aggr2 address, Vlan-20-aggr2 address, Vlan-30-aggr2 address, Vlan-40-aggr2 address Visible 1
Multicast Address 6

For our SDN project:

we used the OpenDaylight controller..We executed the implementation using a Python script,This code builds a **virtual network** using **Mininet** and adds **blockchain technology** to make the network **secure and auditable.**

What It Does:

1. **Creates a Network Topology**:

-2 core switches

- 12 access switches (6 in each group)

- 1 router

- 24 hosts (2 per access switch),the routers were not physical devices ,they were represented as host nodes within the simulation

2. **Connects Everything**:

- Access switches connect to both core switches and the router.

- Hosts connect to access switches.

3. **Adds Flow Rules**:

- Flow rules are installed on switches to control traffic.

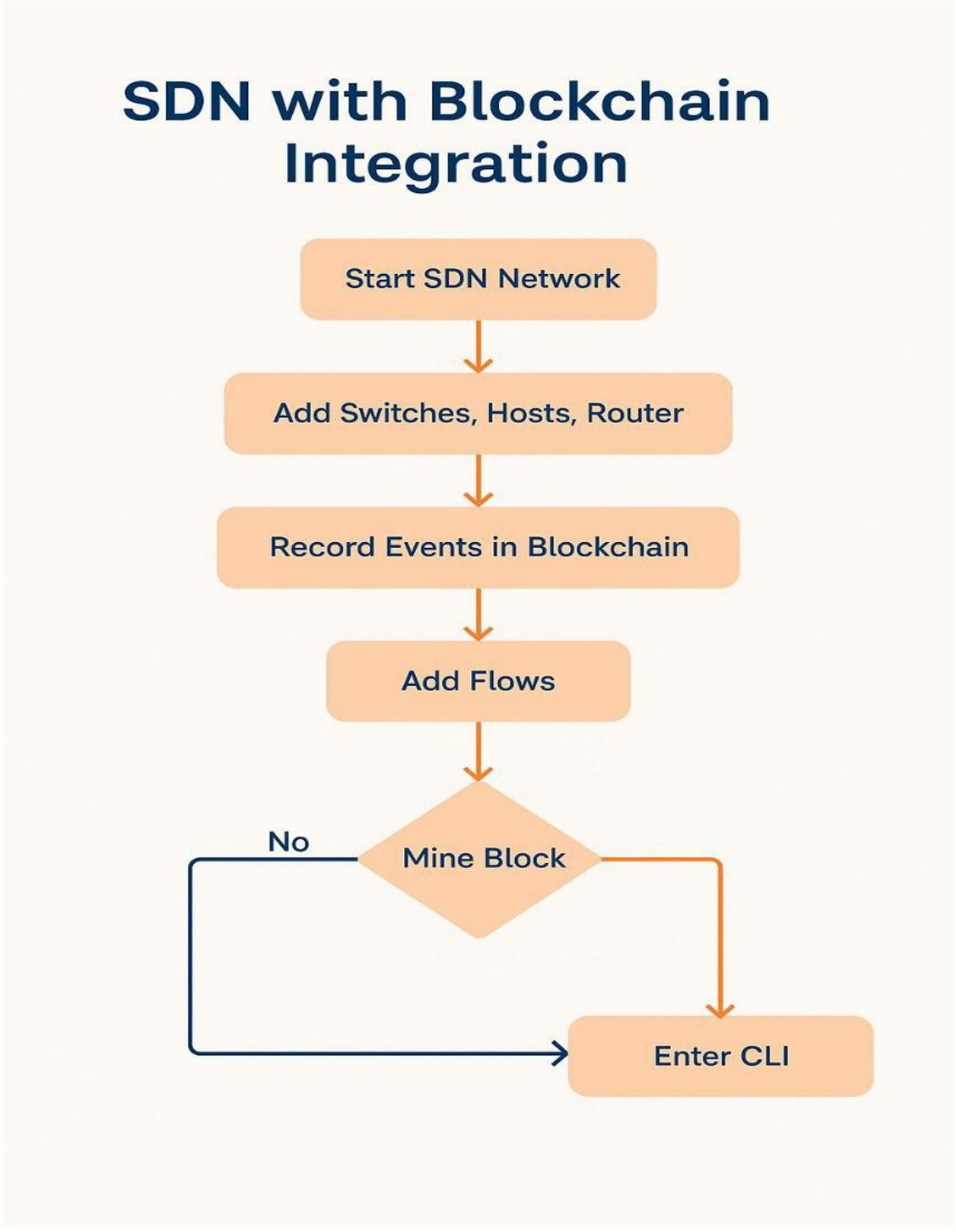
- These rules are sent to an OpenDaylight SDN controller

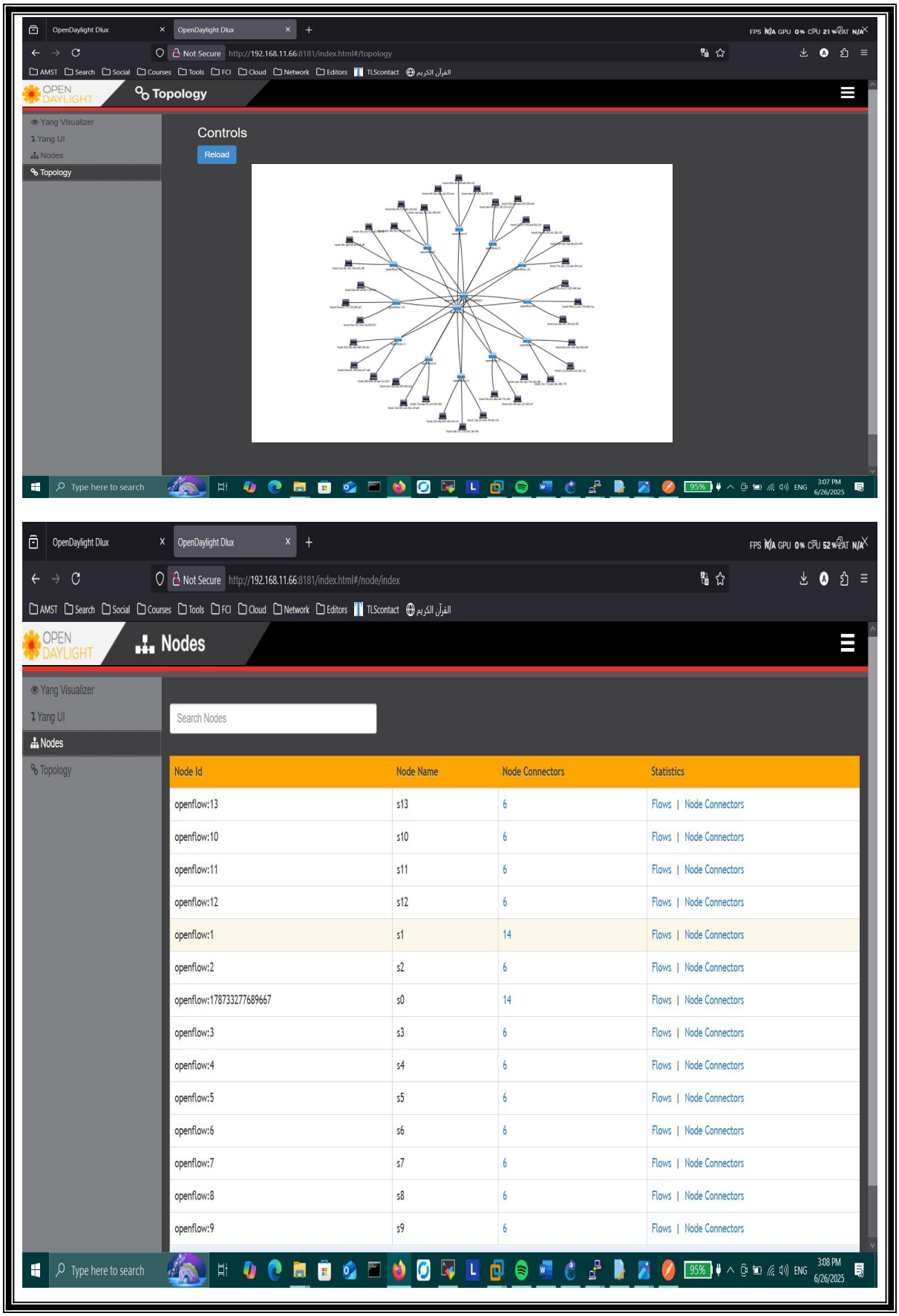
. 4. **Configures Routing**:

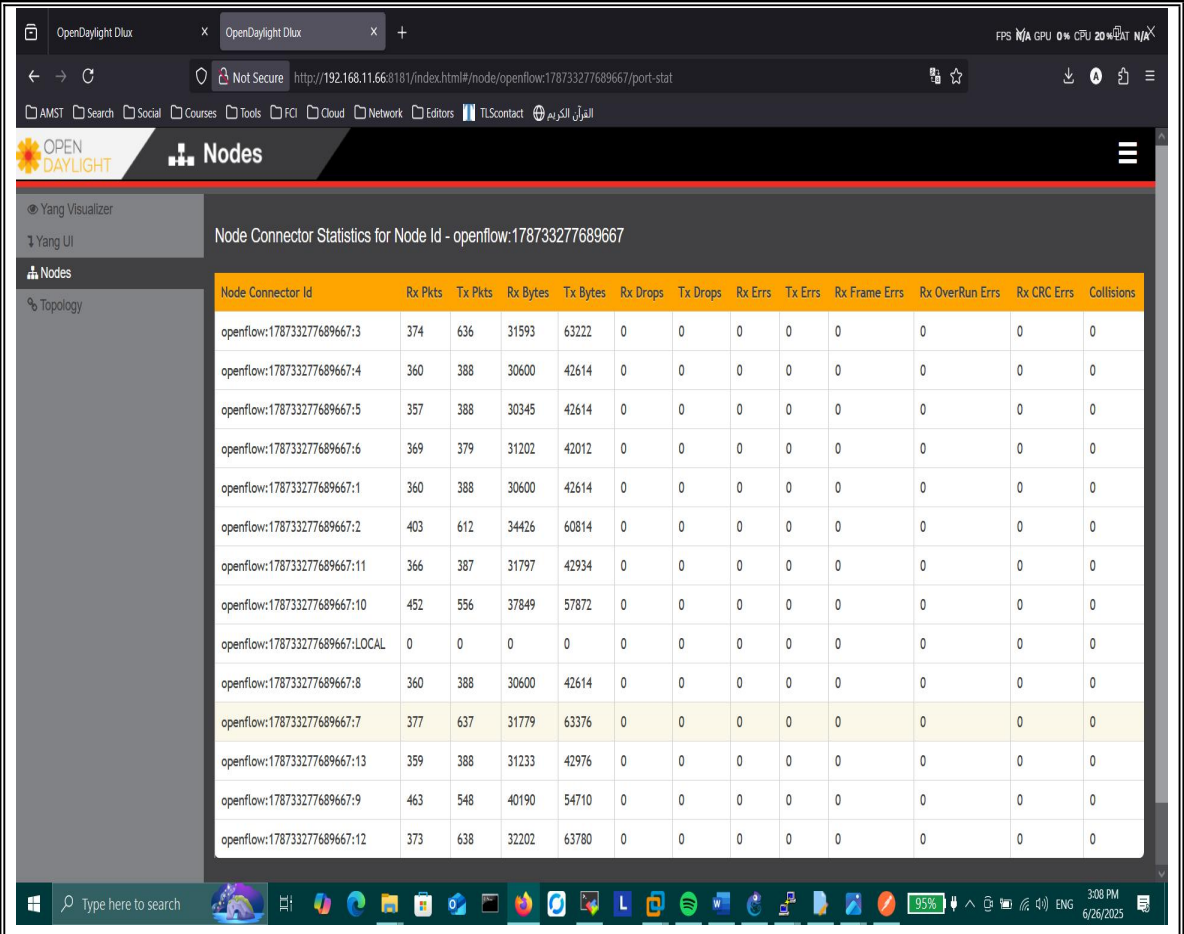
- The router is set up to forward packets. - OSPF (a routing protocol) is configured for dynamic routing.

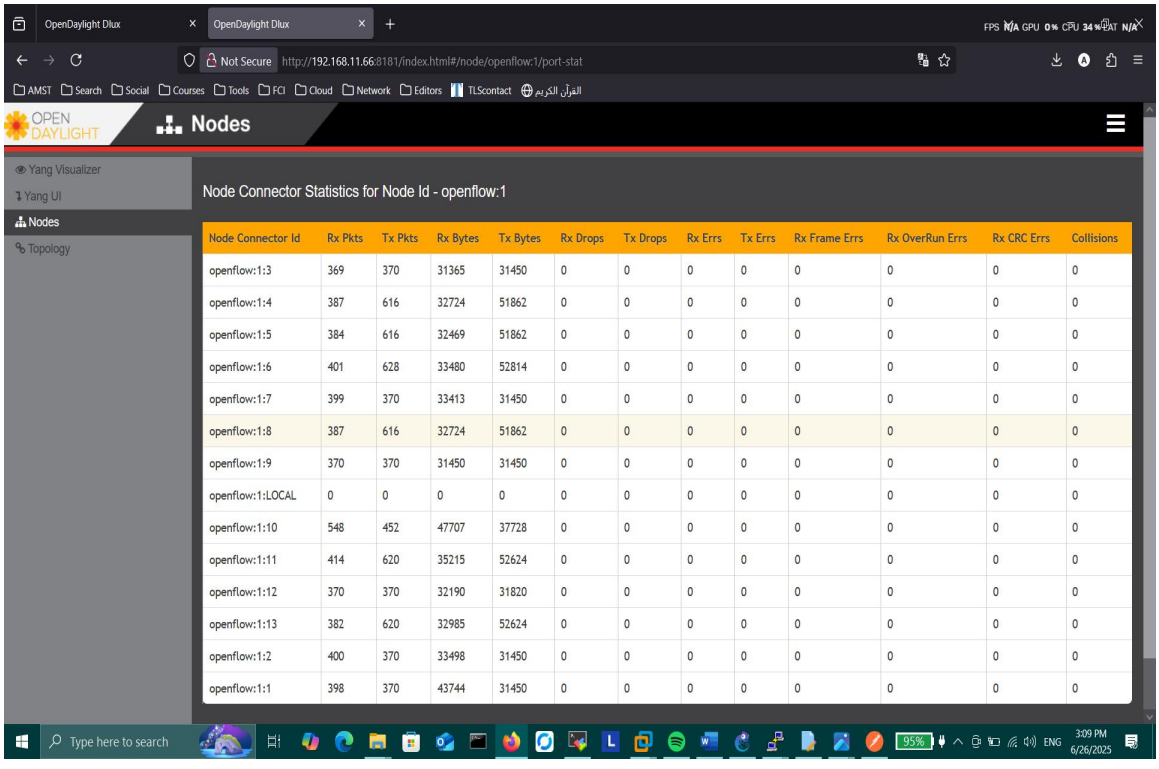
6. **Provides CLI Tools**:

- You can check blockchain status, audit switches, repair or back up the blockchain from the Mininet CLI.

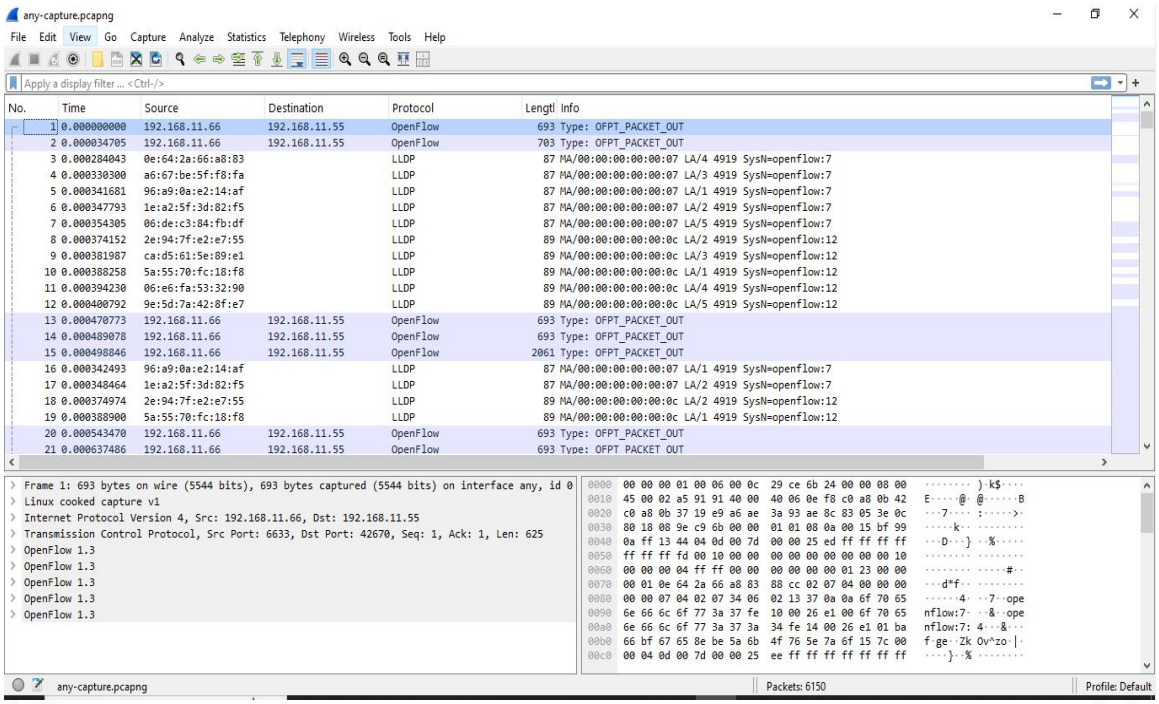

SDN with Blockchain Integration
Start SDN Network
Add Switches, Hosts, Router
Record Events in Blockchain
Add Flows
Mine Block
No
Enter CLI

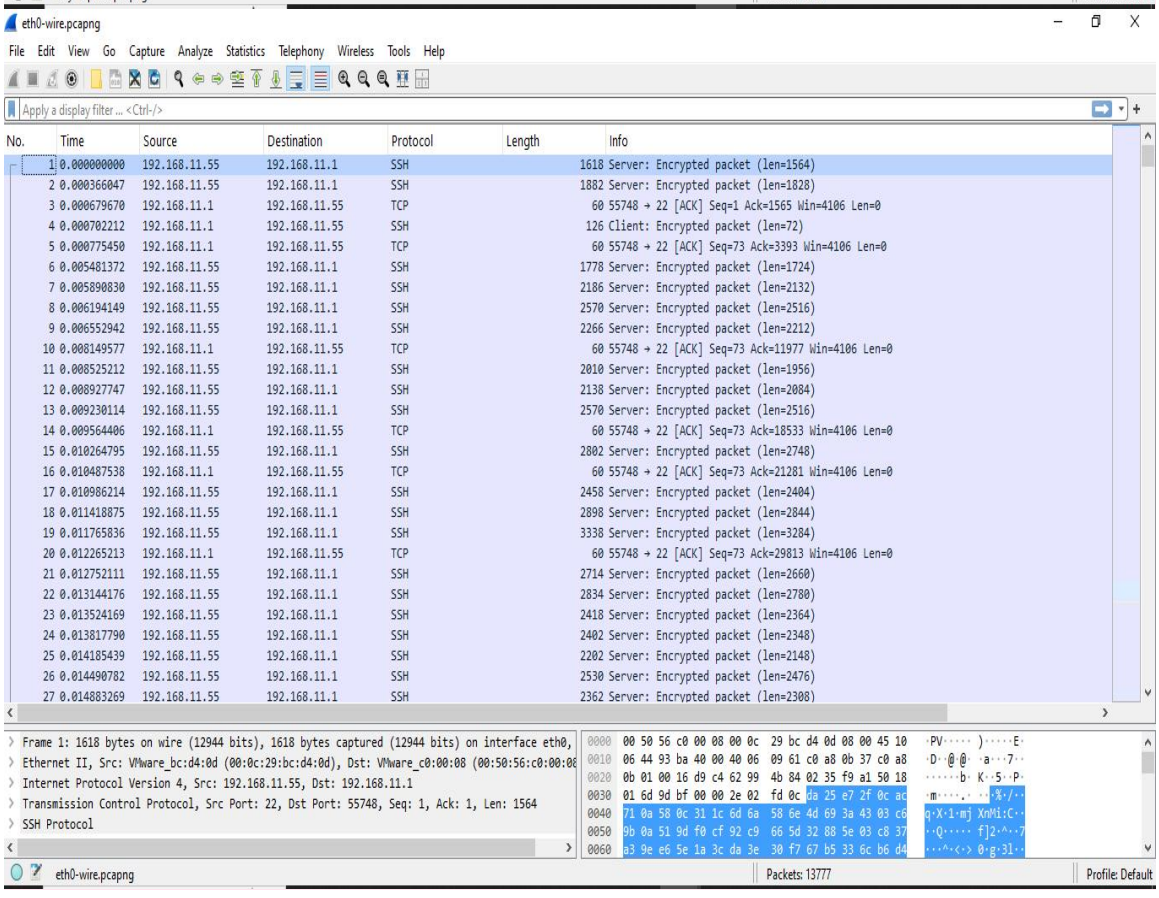

OpenDaylight Dlux
Not Secure
http://192.168.11.66:8181/index.html#/topology
OPEN DAYLIGHT
Topology
Yang Visualizer
Yang UI
Nodes
Topology
Controls
Reload
Type here to search
ENG
3:07 PM
6/26/2025
OpenDaylight Dlux
Not Secure
http://192.168.11.66:8181/index.html#/node/index
OPEN DAYLIGHT
Nodes
Yang Visualizer
Yang UI
Nodes
Topology
Search Nodes
Node Id
Node Name
Node Connectors
Statistics
openflow:13 s13 6 Flows | Node Connectors
openflow:10 s10 6 Flows | Node Connectors
openflow:11 s11 6 Flows | Node Connectors
openflow:12 s12 6 Flows | Node Connectors
openflow:1 s1 14 Flows | Node Connectors
openflow:2 s2 6 Flows | Node Connectors
openflow:17873277689667 s0 14 Flows | Node Connectors
openflow:3 s3 6 Flows | Node Connectors
openflow:4 s4 6 Flows | Node Connectors
openflow:5 s5 6 Flows | Node Connectors
openflow:6 s6 6 Flows | Node Connectors
openflow:7 s7 6 Flows | Node Connectors
openflow:8 s8 6 Flows | Node Connectors
openflow:9 s9 6 Flows | Node Connectors
Type here to search
ENG
3:08 PM
6/26/2025


OpenDaylight Dlux
Not Secure
http://192.168.11.66:8181/index.html#/node/openflow:178733277689667/port-stat
Nodes
Yang Visualizer
Yang UI
Nodes
Topology
Node Connector Statistics for Node Id - openflow:178733277689667
Node Connector Id | Rx Pkts | Tx Pkts | Rx Bytes | Tx Bytes | Rx Drops | Tx Drops | Rx Errs | Tx Errs | Rx Frame Errs | Rx OverRun Errs | Rx CRC Errs | Collisions
openflow:178733277689667:3 | 374 | 636 | 31593 | 63222 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
openflow:178733277689667:4 | 360 | 388 | 30600 | 42614 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
openflow:178733277689667:5 | 357 | 388 | 30345 | 42614 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
openflow:178733277689667:6 | 369 | 379 | 31202 | 42012 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
openflow:178733277689667:1 | 360 | 388 | 30600 | 42614 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
openflow:178733277689667:2 | 403 | 612 | 34426 | 60814 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
openflow:178733277689667:11 | 366 | 387 | 31797 | 42934 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
openflow:178733277689667:10 | 452 | 556 | 37849 | 57872 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
openflow:178733277689667:LOCAL | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
openflow:178733277689667:8 | 360 | 388 | 30600 | 42614 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
openflow:178733277689667:7 | 377 | 637 | 31779 | 63376 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
openflow:178733277689667:13 | 359 | 388 | 31233 | 42976 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
openflow:178733277689667:9 | 463 | 548 | 40190 | 54710 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
openflow:178733277689667:12 | 373 | 638 | 32202 | 63780 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
Type here to search
3:08 PM
6/26/2025


OpenDaylight Dlux
Not Secure
http://192.168.11.66:8181/index.html#/node/openflow:1/port-stat
Nodes
Yang Visualizer
Yang UI
Nodes
Topology
Node Connector Statistics for Node Id - openflow:1
Node Connector Id | Rx Pkts | Tx Pkts | Rx Bytes | Tx Bytes | Rx Drops | Tx Drops | Rx Errs | Tx Errs | Rx Frame Errs | Rx OverRun Errs | Rx CRC Errs | Collisions
openflow:1:3 | 369 | 370 | 31365 | 31450 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
openflow:1:4 | 387 | 616 | 32724 | 51862 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
openflow:1:5 | 384 | 616 | 32469 | 51862 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
openflow:1:6 | 401 | 628 | 33480 | 52814 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
openflow:1:7 | 399 | 370 | 33413 | 31450 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
openflow:1:8 | 387 | 616 | 32724 | 51862 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
openflow:1:9 | 370 | 370 | 31450 | 31450 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
openflow:1:LOCAL | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
openflow:1:10 | 548 | 452 | 47707 | 37728 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
openflow:1:11 | 414 | 620 | 35215 | 52624 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
openflow:1:12 | 370 | 370 | 32190 | 31820 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
openflow:1:13 | 382 | 620 | 32985 | 52624 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
openflow:1:2 | 400 | 370 | 33498 | 31450 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
openflow:1:1 | 398 | 370 | 43744 | 31450 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0
Type here to search
3:09 PM
6/26/2025

And Wireshark was another tool we used." (Good if you're listing several tools:





The following script was used to implement the project:

```
from mininet.net import Mininet

from mininet.node import RemoteController, OVSSwitch, Host

from mininet.topo import Topo

from mininet.cli import CLI

from mininet.log import setLogLevel, info

import requests

import json

import hashlib

import time

from datetime import datetime

import threading


class SecureSDNTopology(Topo):

    def __init__(self):

        self.blockchain = BlockchainSDN()

        super().__init__()


    def build(self):

        """Build network topology with blockchain security"""

        print("=== Building secure SDN topology...")


        self.blockchain.add_network_transaction(

            'topology_build_start',

            'network',

            {'action': 'Starting network topology build'},

            'network_builder'

        )


        core_switch1 = self.addSwitch('s0', cls=OVSSwitch, protocols='OpenFlow13')

        core_switch2 = self.addSwitch('s1', cls=OVSSwitch, protocols='OpenFlow13')


        self.blockchain.add_network_transaction(
```

```
            'core_switches_added',
            'core',
            {'switches': ['s0', 's1'], 'redundancy': True},
            'network_builder'
        )

        self.addLink(core_switch1, core_switch2, bw=1000)

        router = self.addHost('hR1', ip=None)

        self.blockchain.add_network_ransaction(
            'router_added',
            'hR1',
            {'type': 'inter_network_router', 'ip_forwarding': True},
            'network_builder'
        )

        access_switches_group1 = []
        for lan_num in range(2, 8):
            switch = self.addSwitch(f's{lan_num}', cls=OVSSwitch, protocols='OpenFlow13')
            access_switches_group1.append(f's{lan_num}')

            self.addLink(switch, core_switch1)
            self.addLink(switch, core_switch2)

            self.addLink(router, switch)

            hosts_added = []
            for host_num in [10, 20]:
                host_name = f'h{lan_num-1}{host_num}'
                host_ip = f'10.0.{lan_num-1}.{host_num}/24'
                router_ip = f'10.0.{lan_num-1}.1'
```

```
            host = self.addHost(
                host_name,
                ip=host_ip,
                defaultRoute=f'via {router_ip}'
            )
            self.addLink(host, switch)
            hosts_added.append({'name': host_name, 'ip': host_ip})

        self.blockchain.add_network_transaction(
            'access_switch_configured',
            f's{lan_num}',
            {
                'group': 1,
                'network': f'10.0.{lan_num-1}.0/24',
                'hosts': hosts_added,
                'core_connections': ['s0', 's1'],
                'router_connection': 'hR1'
            },
            'network_builder'
        )
```

```
        access_switches_group2 = []
        for lan_num in range(8, 14):
            switch = self.addSwitch(f's{lan_num}', cls=OVSSwitch, protocols='OpenFlow13')
            access_switches_group2.append(f's{lan_num}')

            self.addLink(switch, core_switch1)
            self.addLink(switch, core_switch2)

            self.addLink(router, switch)

            hosts_added = []
            for host_num in [10, 20]:
```

```
            host_name = f'h{lan_num-1}{host_num}'
            host_ip = f'10.1.{lan_num-7}.{host_num}/24'
            router_ip = f'10.1.{lan_num-7}.1'

            host = self.addHost(
                host_name,
                ip=host_ip,
                defaultRoute=f'via {router_ip}'
            )
            self.addLink(host, switch)
            hosts_added.append({'name': host_name, 'ip': host_ip})

        # Record hosts in blockchain
        self.blockchain.add_network_transaction(
            'access_switch_configured',
            f's{lan_num}',
            {
                'group': 2,
                'network': f'10.1.{lan_num-7}.0/24',
                'hosts': hosts_added,
                'core_connections': ['s0', 's1'],
                'router_connection': 'hR1'
            },
            'network_builder'
        )

    # Record topology completion in blockchain
    self.blockchain.add_network_transaction(
        'topology_build_complete',
        'network',
        {
            'total_switches': 14,  # 2 core + 12 access
            'total_hosts': 24,     # 2 hosts per access switch
```

```
            'groups': {
                'group1_switches': access_switches_group1,
                'group2_switches': access_switches_group2
            }
        },
        'network_builder'
    )

    print("? Secure SDN topology built successfully")
```

```
def secure_add_flows(blockchain):
    """Add flows to switches with blockchain security"""
    print("== Adding flows with blockchain security...")

    url = "http://192.168.11.66:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:{}/flow-node-inventory:table/0"
    headers = {
        "Content-Type": "application/json",
        "Authorization": "Basic YWRtaW46YWRtaW4="
    }
```

```
    flows_added = 0
    flows_failed = 0
```

```
    for switch_id in range(2, 14):
        switch_name = f's{switch_id}'

        flow1 = {
            "flow": [
                {
                    "id": "1",
                    "priority": 100,
                    "match": {
```

```
                    "in-port": "1"
                },
                "instructions": {
                    "instruction": [
                        {
                            "order": 0,
                            "apply-actions": {
                                "action": [
                                    {
                                        "order": 0,
                                        "output-action": {
                                            "output-node-connector": "2"
                                        }
                                    },
                                    {
                                        "order": 1,
                                        "output-action": {
                                            "output-node-connector": "3"
                                        }
                                    }
                                ]
                            }
                        }
                    ]
                }
            }
        ]
    }


    blockchain.add_network_transaction(
        'flow_rule_add',
        switch_name,
        {
```

```
            'flow_id': '1',
            'priority': 100,
            'match': 'in-port:1 (hosts)',
            'actions': 'output:2,3 (to core switches)',
            'purpose': 'hosts_to_core_traffic'
        },
        'flow_controller'
    )


    flow2 = {
        "flow": [
            {
                "id": "2",
                "priority": 100,
                "match": {
                    "in-port": "2"
                },
                "instructions": {
                    "instruction": [
                        {
                            "order": 0,
                            "apply-actions": {
                                "action": [
                                    {
                                        "order": 0,
                                        "output-action": {
                                            "output-node-connector": "1"
                                        }
                                    }
                                ]
                            }
                        }
                    ]
```

```
                }
            }
        ]
    }


    blockchain.add_network_transaction(
        'flow_rule_add',
        switch_name,
        {
            'flow_id': '2',
            'priority': 100,
            'match': 'in-port:2 (from core s0)',
            'actions': 'output:1 (to hosts)',
            'purpose': 'core_s0_to_hosts_traffic'
        },
        'flow_controller'
    )


    flow3 = {
        "flow": [
            {
                "id": "3",
                "priority": 100,
                "match": {
                    "in-port": "3"
                },
                "instructions": {
                    "instruction": [
                        {
                            "order": 0,
                            "apply-actions": {
                                "action": [
                                    {
```

```
                                    "order": 0,
                                    "output-action": {
                                        "output-node-connector": "1"
                                    }
                                }
                            ]
                        }
                    }
                ]
            }
        }
    ]
}


blockchain.add_network_transaction(
    'flow_rule_add',
    switch_name,
    {
        'flow_id': '3',
        'priority': 100,
        'match': 'in-port:3 (from core s1)',
        'actions': 'output:1 (to hosts)',
        'purpose': 'core_s1_to_hosts_traffic'
    },
    'flow_controller'
)


try:
    response1 = requests.post(url.format(switch_id), headers=headers, data=json.dumps(flow1), timeout=10)
    response2 = requests.post(url.format(switch_id), headers=headers, data=json.dumps(flow2), timeout=10)
    response3 = requests.post(url.format(switch_id), headers=headers, data=json.dumps(flow3), timeout=10)
    success_count = sum(1 for r in [response1, response2, response3] if r.status_code in [200, 201, 204])
```

```
            if success_count == 3:
                flows_added += 3
                blockchain.add_network_transaction(
                    'flow_installation_success',
                    switch_name,
                    {
                        'flows_installed': 3,
                        'response_codes': [response1.status_code, response2.status_code, response3.status_code]
                    },
                    'flow_controller'
                )
                print(f"? Flows added to {switch_name}: {response1.status_code}, {response2.status_code},
{response3.status_code}")
            else:
                flows_failed += (3 - success_count)
                blockchain.add_network_transaction(
                    'flow_installation_partial_failure',
                    switch_name,
                    {
                        'flows_successful': success_count,
                        'flows_failed': 3 - success_count,
                        'response_codes': [response1.status_code, response2.status_code, response3.status_code]
                    },
                    'flow_controller'
                )
                print(f"? Partial failure for {switch_name}: {response1.status_code}, {response2.status_code},
{response3.status_code}")

        except requests.exceptions.RequestException as e:
            flows_failed += 3
            blockchain.add_network_transaction(
                'flow_installation_error',
                switch_name,
                {
```

```
                    'error': str(e),
                    'flows_failed': 3
                },
                'flow_controller'
            )
            print(f"? Error adding flows to {switch_name}: {e}")

    blockchain.add_network_transaction(
        'flow_installation_summary',
        'network',
        {
            'total_flows_added': flows_added,
            'total_flows_failed': flows_failed,
            'switches_configured': 12,
            'completion_status': 'success' if flows_failed == 0 else 'partial_success'
        },
        'flow_controller'
    )

    print(f"== Flow installation summary: {flows_added} successful, {flows_failed} failed")
```

```
def run_secure_network():
    """Run the secure network with blockchain"""
    print("== Starting Secure SDN Network with Blockchain...")
    print("="*60)

    setLogLevel('info')

    secure_topo = SecureSDNTopology()
    net = Mininet(topo=secure_topo, switch=OVSSwitch, controller=None)
    ctrl = net.addController('odl_ctrl',
                             controller=RemoteController,
                             ip='192.168.11.66',
```

```
                        port=6633,
                        protocol='tcp')

secure_topo.blockchain.add_network_transaction(
    'controller_added',
    'odl_ctrl',
    {
        'type': 'OpenDaylight',
        'ip': '192.168.11.66',
        'port': 6633,
        'protocol': 'tcp'
    },
    'network_manager'
)

ctrl.start()
net.start()

secure_topo.blockchain.add_network_transaction(
    'network_started',
    'network',
    {'controller_status': 'running', 'mininet_status': 'running'},
    'network_manager'
)
```

```
print("== Configuring router...")
router = net.get('hR1')
router.cmd('sysctl -w net.ipv4.ip_forward=1')
secure_topo.blockchain.add_network_transaction(
    'router_ip_forwarding_enabled',
    'hR1',
    {'ip_forward': True},
```

```
    'router_config'
)

router_interfaces_group1 = []
for i in range(6):
    lan_num = i + 1
    interface_ip = f'10.0.{lan_num}.1/24'
    router.cmd(f'ifconfig hR1-eth{i} {interface_ip} up')
    router_interfaces_group1.append({'interface': f'hR1-eth{i}', 'ip': interface_ip})

router_interfaces_group2 = []
for i in range(6, 12):
    lan_num = i - 5
    interface_ip = f'10.1.{lan_num}.1/24'
    router.cmd(f'ifconfig hR1-eth{i} {interface_ip} up')
    router_interfaces_group2.append({'interface': f'hR1-eth{i}', 'ip': interface_ip})

secure_topo.blockchain.add_network_transaction(
    'router_interfaces_configured',
    'hR1',
    {
        'group1_interfaces': router_interfaces_group1,
        'group2_interfaces': router_interfaces_group2,
        'total_interfaces': 12
    },
    'router_config'
)

print("=== Adding default routes for hosts...")
info('*** Adding default routes for internal hosts\n')

routes_added = 0
```

```
    for lan_num in range(1, 7):
        for host_num in [10, 20]:
            host_name = f'h{lan_num}{host_num}'
            gateway_ip = f'10.0.{lan_num}.1'
            host = net.get(host_name)
            host.cmd(f'ip route add default via {gateway_ip}')
            routes_added += 1

    for lan_num in range(7, 13):
        for host_num in [10, 20]:
            host_name = f'h{lan_num}{host_num}'
            gateway_ip = f'10.1.{lan_num-6}.1'
            host = net.get(host_name)
            host.cmd(f'ip route add default via {gateway_ip}')
            routes_added += 1

    secure_topo.blockchain.add_network_transaction(
        'host_default_routes_added',
        'network',
        {
            'total_routes_added': routes_added,
            'hosts_configured': 24
        },
        'routing_config'
    )

    print("== Configuring OSPF...")
    info('*** Configuring OSPF on router\n')

    try:
        router.cmd('zebra -d')
        router.cmd('ospfd -d')
        ospf_interfaces = []
```

```
        for i in range(12):
            interface_name = f'hR1-eth{i}'
            router.cmd(f'vtysh -c "configure terminal" -c "interface {interface_name}" -c "ip ospf area 0"')
            ospf_interfaces.append(interface_name)

        router.cmd('vtysh -c "configure terminal" -c "router ospf" -c "redistribute connected"')
        secure_topo.blockchain.add_network_transaction(
            'ospf_configured',
            'hR1',
            {
                'ospf_area': 0,
                'interfaces': ospf_interfaces,
                'redistribute': 'connected',
                'daemons': ['zebra', 'ospfd']
            },
            'routing_config'
        )
        print("? OSPF configured successfully")

    except Exception as e:
        secure_topo.blockchain.add_network_transaction(
            'ospf_configuration_error',
            'hR1',
            {'error': str(e)},
            'routing_config'
        )
        print(f"? OSPF configuration warning: {e}")

    print("\n== Mining configuration block...")
    secure_topo.blockchain.mine_block()
    secure_add_flows(secure_topo.blockchain)
    print("\n== Mining flows block...")
    secure_topo.blockchain.mine_block()
```

```
secure_topo.blockchain.print_blockchain_status()
```

```
print("\n" + "="*60)
print("== SECURE SDN NETWORK IS READY!")
print("="*60)
print("== All network events are recorded in blockchain")
print("== Network configuration is tamper-proof")
print("== Use 'py net.blockchain_help()' for all blockchain commands")
print("=== Quick commands:")
print("    py net.blockchain_status()    - Check blockchain health")
print("    py net.repair_blockchain()    - Fix corrupted blockchain")
print("    py net.backup_blockchain()    - Create backup")
print("="*60)
```

```
def blockchain_status():
    secure_topo.blockchain.print_blockchain_status()


def audit_switch(switch_name):
    history = secure_topo.blockchain.get_switch_history(switch_name)
    if history:
        print(f"\n== Audit History for {switch_name}:")
        print("-" * 40)
        for record in history:
            print(f"== {record['timestamp'][:19]}")
            print(f"== Type: {record['type']}")
            print(f"== User: {record['user_id']}")
            print(f"== Status: {record['status']}")
            if record['flow_data']:
                print(f"== Data: {record['flow_data']}")
            print("-" * 20)
    else:
        print(f"? No history found for {switch_name}")
```

```
def repair_blockchain():
    """Repair blockchain"""
    repaired = secure_topo.blockchain.repair_blockchain()
    print(f"== Repaired {repaired} blocks")
    secure_topo.blockchain.print_blockchain_status()


def rebuild_blockchain():
    """Rebuild blockchain from scratch"""
    blocks = secure_topo.blockchain.rebuild_blockchain()
    print(f"=== Rebuilt blockchain with {blocks} blocks")
    secure_topo.blockchain.print_blockchain_status()


def backup_blockchain():
    """Export blockchain to file"""
    filename = secure_topo.blockchain.export_blockchain()
    if filename:
        print(f"== Blockchain backed up to: {filename}")


def blockchain_help():
    """Show blockchain commands"""
    print("\n== BLOCKCHAIN COMMANDS:")
    print("=" * 40)
    print("py net.blockchain_status()     - Show blockchain status")
    print("py net.audit_switch('s2')      - Show history for a switch")
    print("py net.repair_blockchain()     - Repair blockchain")
    print("py net.rebuild_blockchain()    - Rebuild blockchain")
    print("py net.backup_blockchain()     - Create backup")
    print("py net.blockchain_help()       - Show this help")
    print("=" * 40)


net.blockchain_status = blockchain_status
net.audit_switch = audit_switch
```

```
    net.repair_blockchain = repair_blockchain
    net.rebuild_blockchain = rebuild_blockchain
    net.backup_blockchain = backup_blockchain
    net.blockchain_help = blockchain_help
    net.blockchain = secure_topo.blockchain
    CLI(net)


    print("== Shutting down secure network...")
    secure_topo.blockchain.add_network_transaction(
        'network_shutdown',
        'network',
        {'shutdown_time': datetime.now().isoformat()},
        'network_manager'
    )

    net.stop()
    print("? Secure SDN network stopped")
```

```
class BlockchainSDN:
    def __init__(self):
        self.chain = []
        self.pending_transactions = []
        self.create_genesis_block()
        print("!! Blockchain initialized for SDN security")

    def create_genesis_block(self):
        """Create the first block in the chain"""
        genesis_block = {
            'index': 0,
            'timestamp': time.time(),
            'transactions': [{
                'type': 'genesis',
                'message': 'SDN Blockchain Network Initialized',
```

```
            'timestamp': datetime.now().isoformat()
        }],
        'previous_hash': '0',
        'nonce': 0
    }
    genesis_block['hash'] = self.calculate_hash(genesis_block)
    self.chain.append(genesis_block)

def calculate_hash(self, block):
    """Calculate hash for a block"""
    import copy
    block_copy = copy.deepcopy(block)
    if 'hash' in block_copy:
        del block_copy['hash']

    block_string = json.dumps(block_copy, sort_keys=True, ensure_ascii=False)
    return hashlib.sha256(block_string.encode('utf-8')).hexdigest()

def add_network_transaction(self, transaction_type, switch_id, flow_data, user_id="sdn_controller"):
    """Add a new network transaction"""
    transaction = {
        'type': transaction_type,
        'switch_id': switch_id,
        'flow_data': flow_data,
        'user_id': user_id,
        'timestamp': datetime.now().isoformat(),
        'status': 'pending'
    }
    self.pending_transactions.append(transaction)
    print(f"!! Transaction added: {transaction_type} on {switch_id}")
    return transaction

def mine_block(self):
```

```
        """Mine a new block"""
        if not self.pending_transactions:
            print("!! No pending transactions to mine")
            return False

        print("!! Mining new block...")
        new_block = {
            'index': len(self.chain),
            'timestamp': time.time(),
            'transactions': self.pending_transactions.copy(),
            'previous_hash': self.chain[-1]['hash'],
            'nonce': 0
        }

        target = "0000"
        while not self.calculate_hash(new_block).startswith(target):
            new_block['nonce'] += 1
            if new_block['nonce'] % 1000 == 0:
                print(f"   Mining... nonce: {new_block['nonce']}")

        new_block['hash'] = self.calculate_hash(new_block)
        self.chain.append(new_block)

        for transaction in self.pending_transactions:
            transaction['status'] = 'confirmed'

        print(f"? Block #{new_block['index']} mined! Hash: {new_block['hash'][:16]}...")
        print(f"   Transactions: {len(self.pending_transactions)}")
        self.pending_transactions = []
        return True

    def repair_blockchain(self):
        """Repair any blockchain inconsistencies"""
```

```
        print("!! Starting blockchain repair process...")
        repaired_blocks = 0

        for i, block in enumerate(self.chain):
            original_hash = block.get('hash', '')
            calculated_hash = self.calculate_hash(block)

            if original_hash != calculated_hash:
                print(f"!! Repairing block {i}...")
                block['hash'] = calculated_hash
                repaired_blocks += 1
                if i + 1 < len(self.chain):
                    self.chain[i + 1]['previous_hash'] = calculated_hash

        print(f"? Blockchain repair completed: {repaired_blocks} blocks repaired")
        return repaired_blocks

    def rebuild_blockchain(self):
        """Completely rebuild the blockchain from transactions"""
        print("=== Rebuilding blockchain from scratch...")

        all_transactions = []
        for block in self.chain[1:]:
            all_transactions.extend(block['transactions'])

        self.chain = []
        self.pending_transactions = []
        self.create_genesis_block()

        batch_size = 20
        for i in range(0, len(all_transactions), batch_size):
            batch = all_transactions[i:i + batch_size]
            self.pending_transactions = batch
```

```
            self.mine_block()


        print("? Blockchain rebuilt successfully!")
        return len(self.chain)


    def export_blockchain(self, filename=None):
        """Export the blockchain to a file"""
        if filename is None:
            filename = f"blockchain_backup_{datetime.now().strftime('%Y%m%d_%H%M%S')}.json"


        try:
            with open(filename, 'w', encoding='utf-8') as f:
                json.dump({
                    'chain': self.chain,
                    'export_time': datetime.now().isoformat(),
                    'total_blocks': len(self.chain),
                    'chain_valid': self.verify_chain()
                }, f, indent=2, ensure_ascii=False)


            print(f"!! Blockchain exported to: {filename}")
            return filename
        except Exception as e:
            print(f"? Export failed: {e}")
            return None


    def verify_chain(self):
        """Verify the integrity of the blockchain"""
        print("!! Verifying blockchain integrity...")


        for i in range(1, len(self.chain)):
            current_block = self.chain[i]
            previous_block = self.chain[i-1]
```

```
            calculated_hash = self.calculate_hash(current_block)
            stored_hash = current_block.get('hash', '')

            if stored_hash != calculated_hash:
                print(f"? Block {i} hash verification failed")
                print(f"   Expected: {calculated_hash[:16]}...")
                print(f"   Got:      {stored_hash[:16]}...")

                print(f"!! Attempting to repair block {i} hash...")
                current_block['hash'] = calculated_hash
                print(f"? Block {i} hash repaired")
                continue

            if current_block['previous_hash'] != previous_block['hash']:
                print(f"? Block {i} previous hash chain broken")
                print(f"   Expected previous: {previous_block['hash'][:16]}...")
                print(f"   Got previous:     {current_block['previous_hash'][:16]}...")
                return False

        print("? Blockchain verification completed")
        return True

    def get_switch_history(self, switch_id):
        """Get all transactions for a specific switch"""
        history = []
        for block in self.chain:
            for transaction in block['transactions']:
                if transaction.get('switch_id') == switch_id:
                    history.append({
                        'block_index': block['index'],
                        'timestamp': transaction['timestamp'],
                        'type': transaction['type'],
                        'flow_data': transaction.get('flow_data', {}),
```

```
                        'user_id': transaction.get('user_id', 'unknown'),
                        'status': transaction.get('status', 'confirmed')
                    })
        return history

    def print_blockchain_status(self):
        """Print status of the blockchain"""
        print("\n" + "="*60)
        print("!! BLOCKCHAIN SDN STATUS")
        print("="*60)
        print(f"!! Total Blocks: {len(self.chain)}")
        print(f"!! Pending Transactions: {len(self.pending_transactions)}")

        is_valid = self.verify_chain()
        print(f"!! Chain Valid: {'Yes' if is_valid else 'No'}")
        total_transactions = sum(len(block['transactions']) for block in self.chain)
        print(f"!! Total Network Transactions: {total_transactions}")
        if len(self.chain) > 1:
            last_block = self.chain[-1]
            print(f"!! Latest Block: #{last_block['index']} ({len(last_block['transactions'])} transactions)")
            print(f"!! Latest Block Time: {datetime.fromtimestamp(last_block['timestamp']).strftime('%Y-%m-
%d %H:%M:%S')}")
            print(f"!! Latest Block Hash: {last_block['hash'][:16]}...")
        switch_stats = {}
        for block in self.chain:
            for transaction in block['transactions']:
                switch_id = transaction.get('switch_id', 'unknown')
                if switch_id not in switch_stats:
                    switch_stats[switch_id] = 0
                switch_stats[switch_id] += 1

        if switch_stats:
            print("\n!! Switch Activity (Top 10):")
            sorted_switches = sorted(switch_stats.items(), key=lambda x: x[1], reverse=True)[:10]
```

```
            for switch_id, count in sorted_switches:
                print(f"  !! {switch_id}: {count} transactions")
        if not is_valid:
            print("\n!! BLOCKCHAIN REPAIR OPTIONS:")
            print("   Use: py net.blockchain.repair_blockchain()")
            print("   Use: py net.blockchain.rebuild_blockchain()")
```

```
if __name__ == '__main__':
    run_secure_network()
```

It's an in-network aspect of the SDN, and we interact with it using the API." (More conversational):

Network Topology Report

Hosts

Host ID	IP Address	MAC Address	Attachment Point
host:ee:5f:ef:49:49:58	10.1.6.20	ee:5f:ef:49:49:58	openflow:13:5
host:9e:47:7b:ae:e5:0b	10.1.2.10	9e:47:7b:ae:e5:0b	openflow:9:4
host:ee:96:25:cd:3d:3d	10.1.1.10	ee:96:25:cd:3d:3d	openflow:8:4
host:aa:18:ad:a9:bd:8d	10.0.1.1	aa:18:ad:a9:bd:8d	openflow:5:3
host:16:8b:95:b2:c0:1a	10.0.2.20	16:8b:95:b2:c0:1a	openflow:3:5
host:02:c3:a5:93:e1:76	10.0.2.1	02:c3:a5:93:e1:76	openflow:3:3
host:06:52:ad:e9:fd:6e	10.0.3.10	06:52:ad:e9:fd:6e	openflow:4:4
host:ea:88:92:1b:44:3f	10.0.1.1	ea:88:92:1b:44:3f	openflow:13:3
host:1e:f7:4c:25:47:6b	10.1.4.10	1e:f7:4c:25:47:6b	openflow:11:4
host:4a:14:3c:9b:af:a0	10.0.4.10	4a:14:3c:9b:af:a0	openflow:5:4

host:42:13:fd:fc:f4:d5	10.1.3.10	42:13:fd:fc:f4:d5	openflow:10:4
host:66:e2:fd:cd:de:52	10.0.3.1	66:e2:fd:cd:de:52	openflow:4:3
host:4a:14:72:b2:2a:a4	10.1.2.1	4a:14:72:b2:2a:a4	openflow:9:3
host:6e:1f:e1:ad:52:3a	10.1.3.20	6e:1f:e1:ad:52:3a	openflow:10:5
host:8a:a0:d9:a5:f2:a8	10.1.2.20	8a:a0:d9:a5:f2:a8	openflow:9:5
host:7a:11:54:97:53:ff	10.0.6.10	7a:11:54:97:53:ff	openflow:7:4
host:26:40:f2:3f:0f:01	10.0.1.1	26:40:f2:3f:0f:01	openflow:7:3
host:e2:f5:1e:c0:fd:2a	10.0.4.20	e2:f5:1e:c0:fd:2a	openflow:5:5
host:76:e3:b6:63:57:7a	10.1.4.20	76:e3:b6:63:57:7a	openflow:11:5
host:4e:5e:06:11:a6:b2	10.0.1.1	4e:5e:06:11:a6:b2	openflow:2:3
host:b6:6f:96:43:21:d4	10.1.1.1	b6:6f:96:43:21:d4	openflow:8:3
host:ca:ce:85:66:67:77	10.0.5.20	ca:ce:85:66:67:77	openflow:6:5
host:76:25:54:e9:98:8c	10.0.1.20	76:25:54:e9:98:8c	openflow:2:5
host:36:04:57:75:e9:50	10.0.5.10	36:04:57:75:e9:50	openflow:6:4
host:2a:51:a8:32:da:c4	10.1.5.10	2a:51:a8:32:da:c4	openflow:12:4
host:12:ec:95:ee:93:9a	10.0.3.20	12:ec:95:ee:93:9a	openflow:4:5
host:42:f3:89:b1:17:6e	10.1.4.1	42:f3:89:b1:17:6e	openflow:11:3
host:a6:a8:60:e7:f2:eb	10.0.5.1	a6:a8:60:e7:f2:eb	openflow:6:3
host:9a:c7:03:3c:ad:da	10.1.5.20	9a:c7:03:3c:ad:da	openflow:12:5
host:86:e7:c9:24:45:a3	10.0.1.1	86:e7:c9:24:45:a3	openflow:12:3
host:62:10:5c:41:38:34	10.1.6.10	62:10:5c:41:38:34	openflow:13:4
host:7e:5f:e8:35:5a:e9	10.0.6.20	7e:5f:e8:35:5a:e9	openflow:7:5
host:d6:a5:ce:e2:7b:d2	10.0.2.10	d6:a5:ce:e2:7b:d2	openflow:3:4
host:2a:c9:5e:10:d4:2c	10.0.1.10	2a:c9:5e:10:d4:2c	openflow:2:4
host:b2:ed:1c:49:a3:91	10.0.1.1	b2:ed:1c:49:a3:91	openflow:10:3

host:a6:10:83:5c:0e:89	10.1.1.20	a6:10:83:5c:0e:89	openflow:8:5

Switches

Switch ID	Termination Points
openflow:3	openflow:3:3, openflow:3:4, openflow:3:1, openflow:3:2, openflow:3:LOCAL, openflow:3:5
openflow:4	openflow:4:4, openflow:4:5, openflow:4:2, openflow:4:3, openflow:4:LOCAL, openflow:4:1
openflow:1	openflow:1:9, openflow:1:7, openflow:1:8, openflow:1:5, openflow:1:6, openflow:1:3, openflow:1:4, openflow:1:1, openflow:1:2, openflow:1:11, openflow:1:10, openflow:1:LOCAL, openflow:1:13, openflow:1:12
openflow:2	openflow:2:1, openflow:2:4, openflow:2:LOCAL, openflow:2:5, openflow:2:2, openflow:2:3
openflow:218628021663040	openflow:218628021663040:12, openflow:218628021663040:13, openflow:218628021663040:LOCAL, openflow:218628021663040:10, openflow:218628021663040:1, openflow:218628021663040:2, openflow:218628021663040:11, openflow:218628021663040:5, openflow:218628021663040:6, openflow:218628021663040:3, openflow:218628021663040:4, openflow:218628021663040:9, openflow:218628021663040:7, openflow:218628021663040:8
openflow:8	openflow:8:LOCAL, openflow:8:4,

	openflow:8:5, openflow:8:2, openflow:8:3, openflow:8:1
openflow:7	openflow:7:3, openflow:7:4, openflow:7:LOCAL, openflow:7:1, openflow:7:2, openflow:7:5
openflow:6	openflow:6:LOCAL, openflow:6:3, openflow:6:2, openflow:6:5, openflow:6:4, openflow:6:1
openflow:5	openflow:5:2, openflow:5:1, openflow:5:4, openflow:5:3, openflow:5:LOCAL, openflow:5:5
openflow:9	openflow:9:4, openflow:9:3, openflow:9:5, openflow:9:LOCAL, openflow:9:2, openflow:9:1
openflow:10	openflow:10:5, openflow:10:LOCAL, openflow:10:2, openflow:10:1, openflow:10:4, openflow:10:3
openflow:11	openflow:11:LOCAL, openflow:11:4, openflow:11:5, openflow:11:2, openflow:11:3, openflow:11:1
openflow:13	openflow:13:4, openflow:13:LOCAL, openflow:13:5, openflow:13:2, openflow:13:3, openflow:13:1
openflow:12	openflow:12:1, openflow:12:LOCAL, openflow:12:2, openflow:12:5, openflow:12:3, openflow:12:4

Links

Link ID	Source Node	Source TP	Destination Node	Destination TP
host:a6:a8:60:e7:f2:eb/openflow:6:3	host:a6:a8:60:e7:f2:eb	host:a6:a8:60:e7:f2:be	openflow:6	openflow:6:3

host:ca:ce:85:66:67:77/openflow:6:5	host:ca:ce:85:66:67:77	host:ca:ce:85:66:67:77	openflow:6	openflow:6:5
host:76:e3:b6:63:57:7a/openflow:11:5	host:76:e3:b6:63:57:7a	host:76:e3:b6:63:57:7a	openflow:11	openflow:11:5
openflow:5:4/host:4a:14:3c:9b:af:a0	openflow:5	openflow:5:4	host:4a:14:3c:9b:af:a0	host:4a:14:3c:9b:af:a0
openflow:4:4/host:06:52:ad:e9:fd:6e	openflow:4	openflow:4:4	host:06:52:ad:e9:fd:6e	host:06:52:ad:e9:fd:6e
host:7e:5f:e8:35:5a:e9/openflow:7:5	host:7e:5f:e8:35:5a:e9	host:7e:5f:e8:35:5a:e9	openflow:7	openflow:7:5
host:62:10:5c:41:38:34/openflow:13:4	host:62:10:5c:41:38:34	host:62:10:5c:41:38:34	openflow:13	openflow:13:4
openflow:12:2	openflow:12	openflow:12:2	openflow:1	openflow:1:12
openflow:1:1	openflow:1	openflow:1:1	openflow:21862802166304 0	openflow:218628021663040:1
openflow:2:5/host:76:25:54:e9:98:8c	openflow:2	openflow:2:5	host:76:25:54:e9:98:8c	host:76:25:54:e9:98:8c
openflow:6:1	openflow:6	openflow:6:1	openflow:21862802166304 0	openflow:218628021663040:6
host:ee:5f:ef:49:49:58/openflow:13:5	host:ee:5f:ef:49:49:58	host:ee:5f:ef:49:49:58	openflow:13	openflow:13:5
openflow:6:2	openflow:6	openflow:6:2	openflow:1	openflow:1:6
host:9a:c7:03:3c:ad:da/openflow:12:5	host:9a:c7:03:3c:ad:da	host:9a:c7:03:3c:ad:da	openflow:12	openflow:12:5
openflow:5:2	openflow:5	openflow:5:2	openflow:1	openflow:1:5
openflow:10:1	openflow:10	openflow:10:1	openflow:21862802166304	openflow:218628021663040:1

			0	0
openflow:1:4	openflow:1	openflow:1:4	openflow:4	openflow:4:2
openflow:3:3/host:02:c3:a5:93:e1:76	openflow:3	openflow:3:3	host:02:c3:a5:93:e1:76	host:02:c3:a5:93:e1:76
openflow:13:3/host:ea:88:92:1b:44:3f	openflow:13	openflow:13:3	host:ea:88:92:1b:44:3f	host:ea:88:92:1b:44:3f
host:86:e7:c9:24:45:a3/openflow:12:3	host:86:e7:c9:24:45:a3	host:86:e7:c9:24:45:a3	openflow:12	openflow:12:3
openflow:8:5/host:a6:10:83:5c:0e:89	openflow:8	openflow:8:5	host:a6:10:83:5c:0e:89	host:a6:10:83:5c:0e:89
host:a6:10:83:5c:0e:89/openflow:8:5	host:a6:10:83:5c:0e:89	host:a6:10:83:5c:0e:89	openflow:8	openflow:8:5
openflow:1:5	openflow:1	openflow:1:5	openflow:5	openflow:5:2
openflow:11:5/host:76:e3:b6:63:57:7a	openflow:11	openflow:11:5	host:76:e3:b6:63:57:7a	host:76:e3:b6:63:57:7a
openflow:4:1	openflow:4	openflow:4:1	openflow:21862802166304 0	openflow:218628021663040:4
openflow:3:4/host:d6:a5:ce:e2:7b:d2	openflow:3	openflow:3:4	host:d6:a5:ce:e2:7b:d2	host:d6:a5:ce:e2:7b:d2
openflow:11:4/host:1e:f7:4c:25:47:6b	openflow:11	openflow:11:4	host:1e:f7:4c:25:47:6b	host:1e:f7:4c:25:47:6b
host:26:40:f2:3f:0f:01/openflow:7:3	host:26:40:f2:3f:0f:01	host:26:40:f2:3f:0f:01	openflow:7	openflow:7:3
openflow:1:2	openflow:1	openflow:1:2	openflow:2	openflow:2:2
openflow:4:2	openflow:4	openflow:4:2	openflow:1	openflow:1:4
openflow:10:2	openflow:10	openflow:10:2	openflow:1	openflow:1:10
openflow:9:3/host:	openflow:9	openflow:9:3	host:4a:14:72:	host:4a:14:72:b

4a:14:72:b2:2a:a4			b2:2a:a4	2:2a:a4
openflow:1:3	openflow:1	openflow:1:3	openflow:3	openflow:3:2
openflow:3:5/host:16:8b:95:b2:c0:1a	openflow:3	openflow:3:5	host:16:8b:95:b2:c0:1a	host:16:8b:95:b2:c0:1a
openflow:9:5/host:8a:a0:d9:a5:f2:a8	openflow:9	openflow:9:5	host:8a:a0:d9:a5:f2:a8	host:8a:a0:d9:a5:f2:a8
openflow:2:4/host:2a:c9:5e:10:d4:2c	openflow:2	openflow:2:4	host:2a:c9:5e:10:d4:2c	host:2a:c9:5e:10:d4:2c
host:36:04:57:75:e9:50/openflow:6:4	host:36:04:57:75:e9:50	host:36:04:57:75:e9:50	openflow:6	openflow:6:4
openflow:4:3/host:66:e2:fd:cd:de:52	openflow:4	openflow:4:3	host:66:e2:fd:cd:de:52	host:66:e2:fd:cd:de:52
openflow:4:5/host:12:ec:95:ee:93:9a	openflow:4	openflow:4:5	host:12:ec:95:ee:93:9a	host:12:ec:95:ee:93:9a
openflow:1:8	openflow:1	openflow:1:8	openflow:8	openflow:8:2
host:9e:47:7b:ae:e5:0b/openflow:9:4	host:9e:47:7b:ae:e5:0b	host:9e:47:7b:ae:e5:0b	openflow:9	openflow:9:4
openflow:7:5/host:7e:5f:e8:35:5a:e9	openflow:7	openflow:7:5	host:7e:5f:e8:35:5a:e9	host:7e:5f:e8:35:5a:e9
host:2a:c9:5e:10:d4:2c/openflow:2:4	host:2a:c9:5e:10:d4:2c	host:2a:c9:5e:10:d4:2c	openflow:2	openflow:2:4
host:b6:6f:96:43:21:d4/openflow:8:3	host:b6:6f:96:43:21:d4	host:b6:6f:96:43:21:d4	openflow:8	openflow:8:3
openflow:1:9	openflow:1	openflow:1:9	openflow:9	openflow:9:2
openflow:10:4/host:42:13:fd:fc:f4:d5	openflow:10	openflow:10:4	host:42:13:fd:fc:f4:d5	host:42:13:fd:fc:f4:d5
openflow:11:2	openflow:11	openflow:11:2	openflow:1	openflow:1:11
host:66:e2:fd:cd:de:52/openflow:4:3	host:66:e2:fd:cd:de:52	host:66:e2:fd:cd:de:52	openflow:4	openflow:4:3

openflow:1:6	openflow:1	openflow:1:6	openflow:6	openflow:6:2
openflow:5:1	openflow:5	openflow:5:1	openflow:21862802166304 0	openflow:218628021663040:5
host:76:25:54:e9:98:8c/openflow:2:5	host:76:25:54:e9:98:8c	host:76:25:54:e9:98:8c	openflow:2	openflow:2:5
openflow:11:1	openflow:11	openflow:11:1	openflow:21862802166304 0	openflow:218628021663040:11
openflow:1:7	openflow:1	openflow:1:7	openflow:7	openflow:7:2
host:d6:a5:ce:e2:7b:d2/openflow:3:4	host:d6:a5:ce:e2:7b:d2	host:d6:a5:ce:e2:7b:d2	openflow:3	openflow:3:4
openflow:7:2	openflow:7	openflow:7:2	openflow:1	openflow:1:7
openflow:13:2	openflow:13	openflow:13:2	openflow:1	openflow:1:13
openflow:13:1	openflow:13	openflow:13:1	openflow:21862802166304 0	openflow:218628021663040:13
openflow:6:4/host:36:04:57:75:e9:50	openflow:6	openflow:6:4	host:36:04:57:75:e9:50	host:36:04:57:75:e9:50
openflow:12:1	openflow:12	openflow:12:1	openflow:21862802166304 0	openflow:218628021663040:12
openflow:6:5/host:ca:ce:85:66:67:77	openflow:6	openflow:6:5	host:ca:ce:85:66:67:77	host:ca:ce:85:66:67:77
openflow:6:3/host:a6:a8:60:e7:f2:eb	openflow:6	openflow:6:3	host:a6:a8:60:e7:f2:eb	host:a6:a8:60:e7:f2:eb
openflow:13:5/host:ee:5f:ef:49:49:58	openflow:13	openflow:13:5	host:ee:5f:ef:49:49:58	host:ee:5f:ef:49:49:58
openflow:7:1	openflow:7	openflow:7:1	openflow:21862802166304 0	openflow:218628021663040:7

host:ee:96:25:cd:3d:3d/openflow:8:4	host:ee:96:25:cd:3d:3d	host:ee:96:25:cd:3d:3d	openflow:8	openflow:8:4
openflow:12:5/host:9a:c7:03:3c:ad:da	openflow:12	openflow:12:5	host:9a:c7:03:3c:ad:da	host:9a:c7:03:3c:ad:da
openflow:3:1	openflow:3	openflow:3:1	openflow:21862802166304 0	openflow:218628021663040:3
host:4a:14:72:b2:2a:a4/openflow:9:3	host:4a:14:72:b2:2a:a4	host:4a:14:72:b2:2a:a4	openflow:9	openflow:9:3
openflow:13:4/host:62:10:5c:41:38:34	openflow:13	openflow:13:4	host:62:10:5c:41:38:34	host:62:10:5c:41:38:34
openflow:218628021663040:1	openflow:21862802166304 0	openflow:218628021663040:1	openflow:1	openflow:1:1
openflow:12:4/host:2a:51:a8:32:da:c4	openflow:12	openflow:12:4	host:2a:51:a8:32:da:c4	host:2a:51:a8:32:da:c4
openflow:5:5/host:e2:f5:1e:c0:fd:2a	openflow:5	openflow:5:5	host:e2:f5:1e:c0:fd:2a	host:e2:f5:1e:c0:fd:2a
host:2a:51:a8:32:da:c4/openflow:12:4	host:2a:51:a8:32:da:c4	host:2a:51:a8:32:da:c4	openflow:12	openflow:12:4
openflow:3:2	openflow:3	openflow:3:2	openflow:1	openflow:1:3
openflow:9:2	openflow:9	openflow:9:2	openflow:1	openflow:1:9
openflow:2:3/host:4e:5e:06:11:a6:b2	openflow:2	openflow:2:3	host:4e:5e:06:11:a6:b2	host:4e:5e:06:11:a6:b2
openflow:9:1	openflow:9	openflow:9:1	openflow:21862802166304 0	openflow:218628021663040:9
openflow:7:3/host:26:40:f2:3f:0f:01	openflow:7	openflow:7:3	host:26:40:f2:3f:0f:01	host:26:40:f2:3f:0f:01

host:4a:14:3c:9b:af:a0/openflow:5:4	host:4a:14:3c:9b:af:a0	host:4a:14:3c:9b:af:a0	openflow:5	openflow:5:4
openflow:10:5/host:6e:1f:e1:ad:52:3a	openflow:10	openflow:10:5	host:6e:1f:e1:ad:52:3a	host:6e:1f:e1:ad:52:3a
host:e2:f5:1e:c0:fd:2a/openflow:5:5	host:e2:f5:1e:c0:fd:2a	host:e2:f5:1e:c0:fd:2a	openflow:5	openflow:5:5
host:ea:88:92:1b:44:3f/openflow:13:3	host:ea:88:92:1b:44:3f	host:ea:88:92:1b:44:3f	openflow:13	openflow:13:3
openflow:1:10	openflow:1	openflow:1:10	openflow:10	openflow:10:2
openflow:9:4/host:9e:47:7b:ae:e5:0b	openflow:9	openflow:9:4	host:9e:47:7b:ae:e5:0b	host:9e:47:7b:ae:e5:0b
host:8a:a0:d9:a5:f2:a8/openflow:9:5	host:8a:a0:d9:a5:f2:a8	host:8a:a0:d9:a5:f2:a8	openflow:9	openflow:9:5
host:42:13:fd:fc:f4:d5/openflow:10:4	host:42:13:fd:fc:f4:d5	host:42:13:fd:fc:f4:d5	openflow:10	openflow:10:4
openflow:1:12	openflow:1	openflow:1:12	openflow:12	openflow:12:2
host:02:c3:a5:93:e1:76/openflow:3:3	host:02:c3:a5:93:e1:76	host:02:c3:a5:93:e1:76	openflow:3	openflow:3:3
openflow:8:2	openflow:8	openflow:8:2	openflow:1	openflow:1:8
host:42:f3:89:b1:17:6e/openflow:11:3	host:42:f3:89:b1:17:6e	host:42:f3:89:b1:17:6e	openflow:11	openflow:11:3
openflow:10:3/host:b2:ed:1c:49:a3:91	openflow:10	openflow:10:3	host:b2:ed:1c:49:a3:91	host:b2:ed:1c:49:a3:91
openflow:218628021663040:8	openflow:218628021663040	openflow:218628021663040:8	openflow:8	openflow:8:1
openflow:1:11	openflow:1	openflow:1:11	openflow:11	openflow:11:2
openflow:8:4/host:	openflow:8	openflow:8:4	host:ee:96:25:	host:ee:96:25:c

ee:96:25:cd:3d:3d			cd:3d:3d	d:3d:3d
host:06:52:ad:e9:fd:6e/openflow:4:4	host:06:52:ad:e9:fd:6e	host:06:52:ad:e9:fd:6e	openflow:4	openflow:4:4
openflow:8:1	openflow:8	openflow:8:1	openflow:21862802166304 0	openflow:2186 28021663040:8
openflow:2186280 21663040:9	openflow:218 62802166304 0	openflow:2186 28021663040:9	openflow:9	openflow:9:1
host:aa:18:ad:a9:bd:8d/openflow:5:3	host:aa:18:ad:a9:bd:8d	host:aa:18:ad:a9:bd:8d	openflow:5	openflow:5:3
host:4e:5e:06:11:a6:b2/openflow:2:3	host:4e:5e:06:11:a6:b2	host:4e:5e:06:11:a6:b2	openflow:2	openflow:2:3
openflow:8:3/host:b6:6f:96:43:21:d4	openflow:8	openflow:8:3	host:b6:6f:96:43:21:d4	host:b6:6f:96:43:21:d4
openflow:2186280 21663040:6	openflow:218 62802166304 0	openflow:2186 28021663040:6	openflow:6	openflow:6:1
openflow:1:13	openflow:1	openflow:1:13	openflow:13	openflow:13:2
openflow:2186280 21663040:10	openflow:218 62802166304 0	openflow:2186 28021663040:1 0	openflow:10	openflow:10:1
openflow:2186280 21663040:7	openflow:218 62802166304 0	openflow:2186 28021663040:7	openflow:7	openflow:7:1
host:16:8b:95:b2:c0:1a/openflow:3:5	host:16:8b:95:b2:c0:1a	host:16:8b:95:b2:c0:1a	openflow:3	openflow:3:5
host:b2:ed:1c:49:a3:91/openflow:10:3	host:b2:ed:1c:49:a3:91	host:b2:ed:1c:49:a3:91	openflow:10	openflow:10:3
openflow:2:2	openflow:2	openflow:2:2	openflow:1	openflow:1:2

openflow:5:3/host:aa:18:ad:a9:bd:8d	openflow:5	openflow:5:3	host:aa:18:ad:a9:bd:8d	host:aa:18:ad:a9:bd:8d
openflow:218628021663040:11	openflow:2186280216630400	openflow:218628021663040:11	openflow:11	openflow:11:1
openflow:218628021663040:4	openflow:2186280216630400	openflow:218628021663040:4	openflow:4	openflow:4:1
openflow:2:1	openflow:2	openflow:2:1	openflow:2186280216630400	openflow:218628021663040:2
openflow:218628021663040:12	openflow:2186280216630400	openflow:218628021663040:12	openflow:12	openflow:12:1
openflow:7:4/host:7a:11:54:97:53:ff	openflow:7	openflow:7:4	host:7a:11:54:97:53:ff	host:7a:11:54:97:53:ff
host:1e:f7:4c:25:47:6b/openflow:11:4	host:1e:f7:4c:25:47:6b	host:1e:f7:4c:25:47:6b	openflow:11	openflow:11:4
openflow:218628021663040:5	openflow:2186280216630400	openflow:218628021663040:5	openflow:5	openflow:5:1
host:7a:11:54:97:53:ff/openflow:7:4	host:7a:11:54:97:53:ff	host:7a:11:54:97:53:ff	openflow:7	openflow:7:4
host:6e:1f:e1:ad:52:3a/openflow:10:5	host:6e:1f:e1:ad:52:3a	host:6e:1f:e1:ad:52:3a	openflow:10	openflow:10:5
openflow:218628021663040:13	openflow:2186280216630400	openflow:218628021663040:13	openflow:13	openflow:13:1
openflow:218628021663040:2	openflow:2186280216630400	openflow:218628021663040:2	openflow:2	openflow:2:1

Reference

1. Start in SDN By Ahmed Abdallah:

https://www.youtube.com/watch?v=wywNDTor0yA&list=PL5ENO5xzfLmKURDfdofLIKsQjmm77ARuG2.

2. Complete, practical SDN and OpenFlow Fundamentals by David Bombal

https://www.udemy.com/course/sdn-and-openflow-fundamentals/

3.mininet:

http://mininet.org/

4.SDN Tools:

https://drive.google.com/drive/folders/1fxyaVCzj3uiJi75Xx u_J7tlgD6sLQnxr

5.Top 5 Software Defined Networking (SDN) Books for Beginners & Experts

https://www.digifloor.com/software-defined-networking-books-14

6.SDN: Software Defined Networks by Thomas D. Nadeau, Ken Gray

https://www.oreilly.com/library/view/sdn-software-defined/9781449342425/ch04.html

7.SDN: Software Defined Networks

https://www.oreilly.com/library/view/sdn-software-defined/9781449542425/

8.SDN Book

https://www.bing.com/videos/search?q=sdn+book&qpvt=sdn+book&FORM=VDRE

9. On the Integration of Blockchain and SDN: Overview, Applications, and Future Perspectives

https://share.google/blLbzpl2kqUW45Lej

10. Towards a blockchain-SDN-based secure architecture for cloud computing in smart industrial IoT

https://www.sciencedirect.com/science/article/pii/S2352864822002449

11.Practical Implementation of a Blockchain-Enabled SDN for Large-Scale Infrastructure Networks

https://share.google/fwS9S6rWsviqjgp87