UNIVERSITY OF TURKU

On-premises Data Center Load Balancer Service Delivery Model Based on Infrastructure as Code

Cyber Security Engineering

Degree Programme in Information and Communication Technology

Department of Computing, Faculty of Technology

Master of Science in Technology Thesis

Author:

Jere Kankaanpää

Supervisors:

Zahed Iqbal (Telia Finland Oyj)

Seppo Virtanen (University of Turku)

Jouni Isoaho (University of Turku)

May 2024

Master of Science in Technology Thesis
Department of Computing, Faculty of Technology
University of Turku

Subject: Cyber Security
Programme: Master's Degree Programme in Information and Communication Technology
Author: Jere Kankaanpää
Title: On-premises Data Center Load Balancer Service Delivery Model Based on Infrastructure as Code
Number of pages: 63 pages, 14 appendix pages
Date: May 2024

Data center networks foster new innovations and technology and are evolving rapidly. Companies and network operators are struggling to keep up with the required speed of delivery due to lack of automation which affects companies' reputation and can ultimately lead to loss of customers. Furthermore, time pressure and manual processes increases the probability of configuration errors and therefore incidents caused by change which can have worse consequences to companies and society. The solution to the problem is automation.

The thesis uses design science research approach, and the objective is to propose load balancer service delivery model based on Infrastructure as Code (IaC). The proposed delivery model is expected to fulfil all the standard load balancer service requirements and shorten the time of delivery and minimize configuration errors with automation. The proposed delivery model is evaluated with case-based testing and compared with the current delivery model that rely on vendor specific interfaces. The thesis produces a design of the IaC solution and a proof of concept including future development directions and a plan for migration from the current delivery model to the proposed delivery model. The same tools setup during the thesis can be shared with other IaC solutions in the data center networks team.

The proposed load balancer service delivery model with Terraform can be used to successfully deliver all the standard load balancer service request cases and offers the benefits of automation. The proposed delivery model is a valid candidate to be used in the data center networks team.

Keywords: data center networks, infrastructure-as-code, load balancer

Table of contents

List of tables

List of figures

1 Introduction

1.1 Motivation

Data center networks foster new innovations and technology and are evolving and changing rapidly at an increasing speed. Companies and network operators are struggling to keep up with the required speed and meet expected delivery times due to lack of automation which affects the overall company reputation and can ultimately lead to loss of customers. Furthermore, time pressure and manual processes increases the probability of configuration errors and therefore incidents caused by change which can have worse consequences to companies and society.

The solution to the problem is to automate repetitive tasks as much as possible for teams to be more agile and lean which enables companies and network operators to stay ahead of the competition by focusing on the required development work and learning. Studies show that network automation offers several benefits such as increased efficiency, reduced configuration errors, and enhanced performance and security of the network [103]. The data center networks team needs automation in all areas: service development, service requests, and change requests. As automation offers the most value in small and incremental changes that do not have many dependencies [94] the service request and load balancer area was selected. Furthermore, there did not exist any common load balancer automation in the team.

The downside to network automation is that it is difficult to implement due to general complexity of networks which manifests largely from legacy networks overhead. Survey aimed to enterprises, cloud providers, and network service providers reveal that only 23 % of the respondents were fully confident in their data center network automation strategy [95]. The data center network automation challenges are connected to complexity, cross-platform incompatibility, and visualization and feedback limitations [95][96] which the selected network automation specification should consider and address. For these reasons from the available data center network automation options: Command-Line Interface (CLI) based scripts, Representational State Transfer (REST) Application Programming Interface (API) based scripts, and Infrastructure as Code (IaC), IaC was proposed as it offers the most benefits and addresses the data center network automation challenges the most comprehensive from the available options [95][96]. IaC popularity has grown steadily [97] and is expected to continue to be a part of network automation in the future [97]. The IaC tool Terraform was

selected as there is already use and familiarity with Terraform in the team. Terraform is supported by various on-premises tools to complete the delivery model.

1.2 Research objective and questions

The thesis objective is to propose load balancer service delivery model based on IaC with Terraform for the data center networks team. The proposed delivery model is expected to fulfil all the standard load balancer service requirements and reduce the time of delivery and minimize configuration errors with automation. The proposed delivery model is evaluated with case-based testing and compared with the current delivery model that relies on vendor specific interfaces. The thesis produces a design of the IaC solution and a proof of concept including future development directions and a plan for migration from the current delivery model to the proposed delivery model. The same tools setup during the thesis can be shared with other IaC solutions in the team. The objective of the thesis can be formulated into the following research questions which the thesis seeks to answer:

- RQ1: What design choices and tools are required for on-premises IaC based solution?
- RQ2: Does the proposed delivery model based on IaC fulfil the requirements of the standard load balancer service?
- RQ3: Does the proposed delivery model based on IaC offer the benefits of automation?

1.3 Research method

The thesis uses the design science research approach where an innovative artefact is created to solve a specific problem with an expected outcome and evaluation [118]. In the design science methodology, the proposed load balancer service delivery model based on IaC with Terraform is the innovative artefact and the expected outcome is that the delivery model fulfils all the load balancer service requirements and is superior to the current delivery model by offering the benefits of automation that are evaluated during the case-based testing.

1.4 Thesis structure

The rest of the thesis is structured as follows: chapter 2 covers data centers and their networks after which network automation is discussed. Chapter 3 introduces the environment and the data center networks team where the thesis is done and provides development requirements

for automation. Chapter 4 and 5 introduce both delivery models and the way of working with service requests. Chapter 6 compares the current and proposed delivery models with standard load balancer service request cases and presents the results. Chapter 7 provides insight about the proposed delivery model and then finally chapter 8 finishes the thesis. The network drawings are made with Microsoft Visio [119] and Cisco stencil [120].

2 Technical background

2.1 Data centers

Data centers are dedicated spaces to house computing resources efficiently built all around the world and are essential to internet and cloud infrastructure [1]. Every modern business today needs computing resources to host its applications that keep the business running at all such as external and internal web applications and systems that serve customers and employees. These computing resources need power, cooling, redundant components, and physical security and it is inefficient to build them in distributed locations such as remote offices and that is where data centers accessible from anywhere in the world come into play. Data centers can be built by the company itself which is referred to as on-premises deployment or use third party data centers such as Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure referred to as public cloud deployment [2].

Inside a data center there are a varying number of rooms, rows, and racks where a data center infrastructure is installed depending on the size and capacity of the data center. Data center infrastructure components can be divided into three categories: compute, storage, and network.

Compute infrastructure refers to physical servers such as rack and blade servers with varying specifications such as processing power and internal memory. The servers themselves can be dedicated to host specific application only or host a hypervisor that runs virtual servers that host the applications [3].

Storage infrastructure refers to block and file storage devices. Block storage devices are hard drives and solid-state drives that store data in blocks and are connected via Storage Area Networks (SANs). File storage devices are specialized servers to handle data storage and file-sharing requests such as network-attached storage (NAS) [4].

Network infrastructure refers to networking devices that connect the other data center infrastructure components to each other and the outside world. Connections between the components are typically made with single-mode (SM) and multimode (MM) optical fibers excluding management connections that are typically made with copper cables. Networking devices include switches, routers, firewalls, load balancers, and specialized servers that provide the required network services for communication such as Domain Name System

(DNS) [5][6], Network Time Protocol (NTP) [7], and Dynamic Host Configuration Protocol (DHCP) [8][9].

Data centers also include various support components to achieve an uninterrupted environment for the data center infrastructure such as power subsystems, uninterruptible power supplies (UPSs), backup generators, fire suppression systems and building security systems [2].

2.1.1 Architecture

As data centers increased in size and complexity and therefore hosted applications, they have become increasingly critical for societies that depend on their seamless operation. For this reason, it is imperative to have regulations and standards created by governments and other organizations in place that built data centers comply with for safety and quality [10].

The standard TIA-942 [11] by Telecommunications Industry Association (TIA) covers all aspects of data center architecture including topology, environment, power and cooling, telecommunications, fire protection, safety, and physical security. The TIA-942 defines four data center ratings 1 – 4 which indicate the company and customers about the reliability and resiliency of the data center and therefore hosted applications. Data centers rated as 1 (Basic Site Infrastructure) have nonredundant infrastructure components and distribution path and limited protection against physical events. Data centers rated as 2 (Redundant Component Site Infrastructure) have redundant infrastructure components but nonredundant distribution path. Data centers rated as 3 (Concurrently Maintainable Site Infrastructure) have redundant infrastructure components and distribution paths and improved physical security. Data centers rated as 4 (Fault Tolerant Site Infrastructure) have protection against single point of failure scenarios and highest level of physical security.

Similarly, the EN 50600 series of standards [12] by Uptime Institute (UI) covers aspects of data centers such as general concepts, building construction, power, environment, telecommunications, and security systems. UI established well-known concept of tiers I – IV in the industry to compare data center capabilities to better align infrastructure investments to business objectives such as what kind of reliability and resiliency is required for certain applications and then host them at the data center tier that meets these requirements. Tier I data centers have basic infrastructure capacity support and its requirements include dedicated area for infrastructure components, backup generator, UPS, and cooling. Tier II data centers

have redundant power and cooling equipment such as backup generators, chillers, cooling units, pumps, and heat rejection equipment. Tier III data centers have redundant distribution paths and support components such that any part of the data center can be maintained without impacting IT applications. Tier IV data centers have several independent and physically isolated systems that together provide complete redundancy [13]. The choice of data center rating or tier should depend on the business objectives and criticality of the application. Table 1 lists data center tier I – IV features to help guide the decision-making process.

Table 1 Data center tier I – IV features [10].

Tier	Business size	Availability	Annual downtime
I	Small	99.671 %	28.8 hours
II	Medium	99.749 %	22.0 hours
III	Large	99.982 %	1.6 hours
IV	Enterprise	99.995 %	26.3 minutes

2.2 Data center networks

Data center networks follow the same rules as any other network and can be understood by the common reference models such as the Open Systems Interconnection (OSI) model [121]. The OSI model shares the functionality and services of networks into seven layers 1 – 7 and describes the interaction of layers directly above and below. [14] In the event of a failure in a certain layer all the above layers fail which is valuable information for troubleshooting possible issues. Layers of the OSI model are from bottom-up: physical (1), data link (2), network (3), transport (4), session (5), presentation (6), and application (7). [14] Network infrastructure components are responsible for different layers of the OSI model.

Layer 1 protocols such as physical Ethernet describe the physical connections for bit transmission between infrastructure components. [14] The components that are typically responsible for the operation of layer 1 are small form-factor pluggables (SFPs) [15], optical fibers and copper cables.

Layer 2 protocols such as Spanning Tree Protocol (STP), Port Channel (PC), Virtual Port Channel (vPC) [1], and Cisco FabricPath (FP) [16] describe methods for data frame exchange

between infrastructure components over common media. [14] The network devices that are responsible for the operation of layer 2 protocols are switches.

Layer 3 protocols such as Open Shortest Path First (OSPF) [17][18], Intermediate System-to-Intermediate System (IS-IS) [19], Enhance Interior Gateway Routing Protocol (EIGRP) [20], and Border Gateway Protocol (BGP) [21][22] provide services to exchange the individual pieces of data or packets over the network between infrastructure components and the outside world. [14] The network devices that are responsible for the operation of layer 3 protocols are routers.

Layer 4 protocols such as Transmission Control Protocol (TCP) [23] and User Datagram Protocol (UDP) [24] define services such as transfer, segmentation, and reassembly for individual connections between logically the same end devices. [14] The network devices that are responsible for the operation of layer 4 protocols are firewalls and load balancers.

Layer 5 provides services to organize dialogue and data exchange for layer 6 which then provides a common representation for the data transferred. Lastly layer 7 contains the application protocols such as DNS, NTP, DHCP, and Hypertext Transfer Protocol (HTTP) [25] that are used for process-to-process communications [14]. Table 2 summarizes layers of the OSI model and respective protocols and network infrastructure components.

Table 2 Summary of the OSI model and the respective protocols and components [1][14].

Layer	Name	Protocols	Component
1	Physical	Physical Ethernet	SFP, optical fiber, copper cable
2	Data Link	STP, PC, vPC, FP	Switch
3	Network	OSPF, IS-IS, EIGRP, BGP	Router
4	Transport	TCP, UDP	Firewall, load balancer
5	Session	DNS, NTP, DHCP, HTTP	NGFW, WAF, ALB
6	Presentation		
7	Application		

Firewalls and load balancers can also work at layer 7 such as next-generation Firewall (NGFW), web application firewall (WAF) [26][27], and application load balancer (ALB) [28]. Network icons used in the network drawings of the thesis are shown in Figure 1.

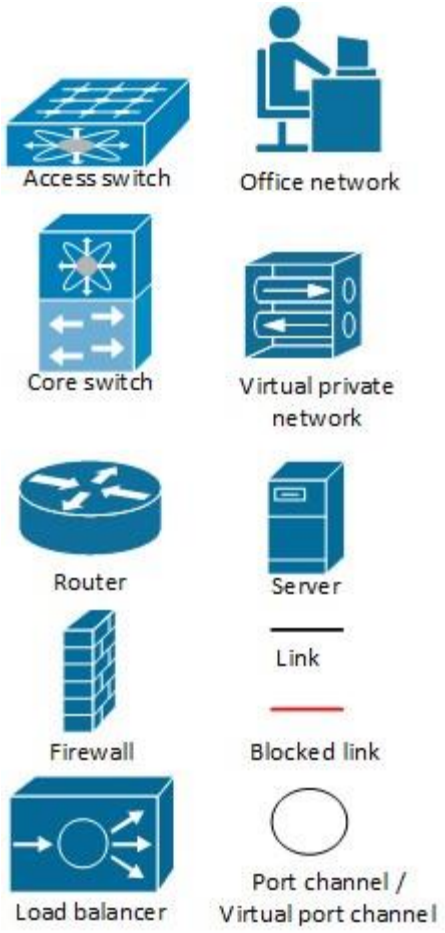


Figure 1 Network icons used in the network drawings of the thesis.

2.2.1 Switching

Switching is a process that works at layer 2 of the OSI model and uses devices' Media Access Control (MAC) addresses to perform forwarding decisions. The network infrastructure component that performs switching is called a switch. Switching topology in data centers consists of interconnected switches using redundant paths and links for high availability [1] as shown in Figure 2.

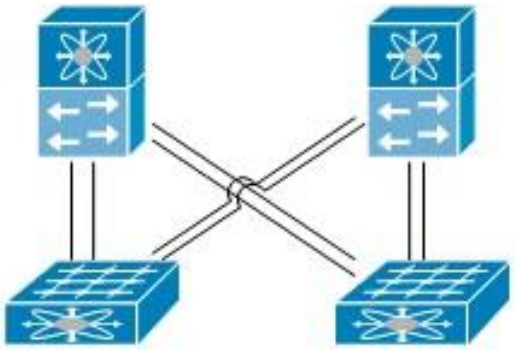


Figure 2 Switching topology with redundant paths and links.

Switching protocols such as STP is used to prevent layer 2 loops and PCs and vPCs are used for link aggregation. Cisco FP is Cisco proprietary protocol that introduces routing concepts at layer 2 [16]. Layer 2 loop refers to a situation where multiple active paths exist in the switching topology and therefore switches learn the same MAC addresses on multiple interfaces and forward duplicate traffic to each of them. This causes a chain reaction where more and more duplicate traffic is forwarded by the switches and ultimately leads to bringing down the whole layer 2 domain. STP operation prevents layer 2 loops by allowing only one active path to exist between any layer 2 devices or switches and blocking the rest of the available paths and links [1] as shown in Figure 3.

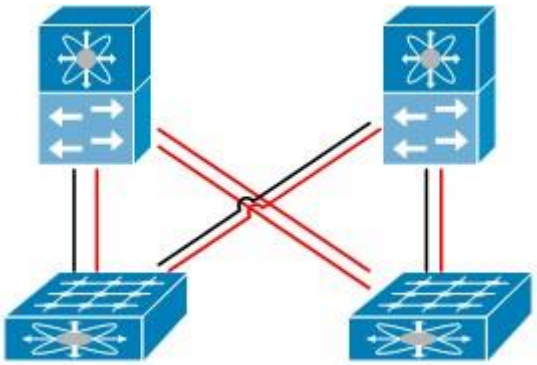


Figure 3 STP allowing only one active path to exist in switching topology.

STP defines a topology in the form of a tree with a root bridge switch from which perspective the STP algorithm calculates the best loop-free path down towards all switches in the network and forces redundant paths into blocked state. In case an active path under STP fails, recalculation is triggered, and a possible redundant path is activated. Switches running STP exchange bridge protocol data units (BPDUs) at regular intervals which are used to create

STP topology and find the best loop-free paths. STP port priority and path cost are used to select the best paths. STP defines different port types such as edge, network, and normal ports and has various extensions such as STP bridge assurance, BPDU guard, BPDU filtering, loop guard, and root guard to optimize the operation of STP and enhance loop prevention [1].

There are several variants of STP that came after the original STP defined in IEEE 802.1D such as Common Spanning Tree (CST), Per-Vlan (Virtual Local Area Network) Spanning Tree Plus (PVST+), Rapid Spanning Tree Protocol (RSTP), Rapid Per-Vlan Spanning Tree Plus (RPVST+), and Multiple Spanning Tree (MST) which have differences in resource utilization, convergence time, and number of instances [30] listed in Table 3.

Table 3 STP variants and respective resource utilization and convergence time [30].

Variant	Standard	Resources	Convergence	Instance
CST	802.1D	Low	Slow	All VLANs
PVST+	Cisco	High	Slow	Per VLAN
RSTP	802.1w	Medium	Fast	All VLANs
RPVST+	Cisco	Very high	Fast	Per VLAN
MST	802.1s	Medium or high	Fast	VLAN list

PCs are used to aggregate multiple physical interfaces into one logical interface. PCs provide several benefits such as redundancy and increased bandwidth between infrastructure components. In addition, PCs are seen as a single path by the STP topology and therefore individual links part of a PC are not put into blocking state [1] as shown in Figure 4.

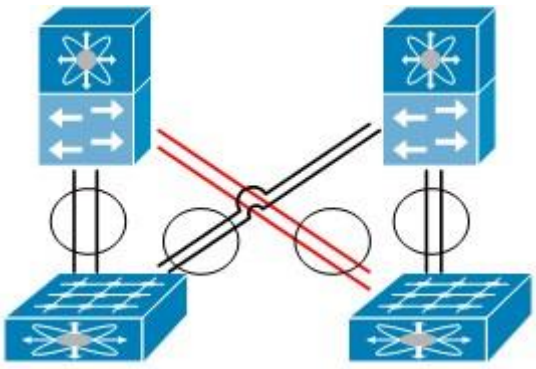


Figure 4 PCs reduce blocked links by STP in switching topology.

Physical port can be a member in only one PC and depending on the hardware and software PCs can typically aggregate up to a maximum of 8 or 16 physical ports. All the members of a PC must be compatible and use the same speed, mode, and other attributes. Full port attribute compatibility check includes network, speed capability and configuration, duplex capability and configuration, mode, and VLAN information.

PCs can be formed statically or by using Link Aggregation Control Protocol (LACP) defined in IEEE 802.3a or Port Aggregation Protocol (PAgP) defined by Cisco. Traffic is load balanced across PC individual links by hashing the addresses in the header which produces a numerical value that selects one of the links in the PC. There are various load balancing methods available that can be used as input to the hashing operation such as source MAC, destination IP, and source and destination TCP or UDP port [1].

Virtual PCs are used to create PCs that span across two switches working as a vPC pair. The vPC pair is connected by a peer link and appears to downstream devices as a single logical switch and provides similar benefits as PCs such as redundancy, bandwidth, and STP non-blocking ports [1] as shown in Figure 5.

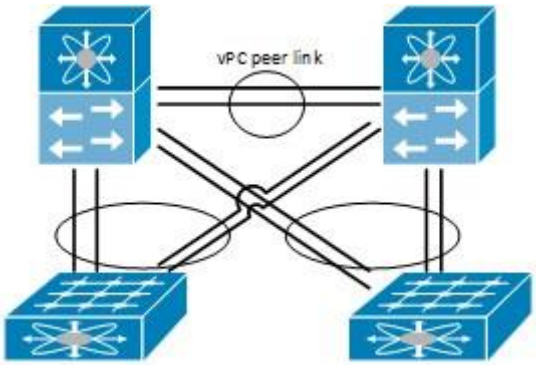


Figure 5 VPCs reduce blocked links by STP to a greater degree in switching topology.

In the pair one switch is the primary and the other switch is the secondary and together they form a vPC domain. Similar to PCs port compatibility, vPC peer switches must be compatible at a global and interface level. Compatible configuration includes MTU, STP global configuration, PC mode, STP interface configuration, and VLAN information [1].

Overlay network technologies such as Cisco FP were developed for workload mobility and address STP limitations in data centers. Cisco FP introduces routing concepts at layer 2 or

MAC routing [1] which allows to build large loop-free layer 2 domains without the need for STP and therefore blocked ports as shown in Figure 6.

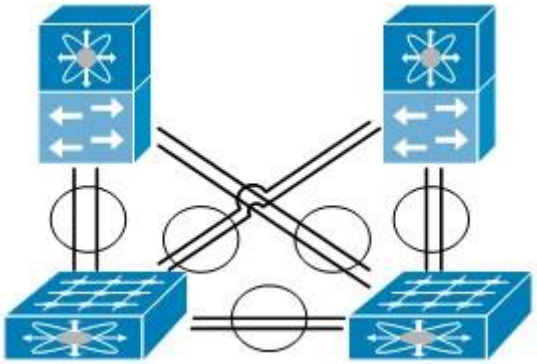


Figure 6 Cisco FP and the absence of STP in switching topology.

Cisco FP encapsulates layer 2 frames entering the fabric at the source switch with a new header that consists of routable switch source and destination addresses. At the destination switch the frame is de-encapsulated and delivered in its original Ethernet format. The switch addresses are automatically assigned and have routing table for all destinations and therefore traffic is never flooded. Cisco FP is simple to configure and is based on IS-IS routing protocol that provides fast convergence and scalability. Loop-free is ensured by time-to-live (TTL) field and reverse-path forwarding (RPF) check [16]. Next generation data centers such as Cisco Application Centric Infrastructure (ACI) typically encapsulate traffic within Virtual Extensible LAN (VXLAN) [1].

2.2.2 Routing

Routing is a process that works at layer 3 of the OSI model and uses hosts' Internet Protocol (IP) addresses to perform forwarding decisions. The network infrastructure component that performs routing is called a router. Similar to switching topology in data centers, routing topology consists of interconnected routers using redundant paths and links for high availability [31] as shown in Figure 7.

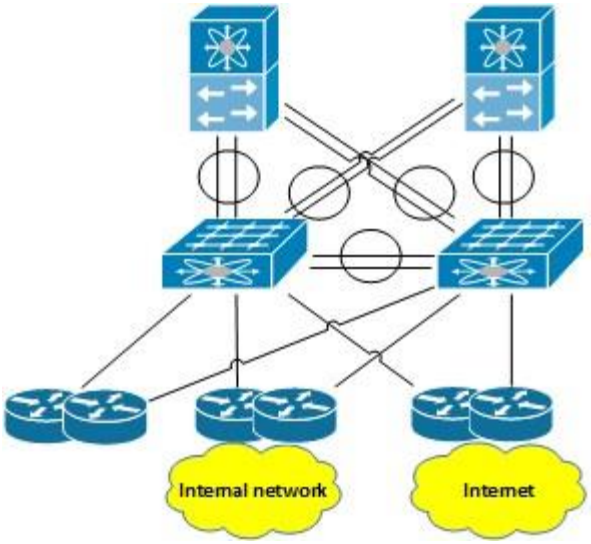


Figure 7 Routing topology with redundant paths and links.

Routing can be configured either statically or by using dynamic routing protocols. Static routing is not scalable and cannot dynamically adapt to network changes and therefore dynamic routing protocols are recommended. Dynamic routing protocols can be categorized into two categories: Interior Gateway Protocols (IGPs) such as OSPF, IS-IS, and EIGRP and Exterior Gateway Protocols (EGPs) such as BGP. Most commonly IGPs are used to exchange routing information within Autonomous System (AS) and EGPs are used to exchange routing information between ASs.

Further categorization of dynamic routing protocols, which are distance vector, link-state, and path vector, are done based on what kind of information they exchange between neighbouring routers about destination networks. Distance vector protocols such as EIGRP exchange a vector that has direction and distance at each signpost or router towards the destination and no further information is available about the path. Link-state protocols such as OSPF and IS-IS exchange the entire network topology and use it to calculate the best path towards the destination. Lastly path vector protocols exchange a vector that has direction and distance towards the destination, but also include information about the path to that destination. Table 4 summarizes the dynamic routing protocols categorization [31].

Table 4 Dynamic routing protocols categorization [31].

Interior Gateway Protocols		Exterior Gateway Protocols
Distance Vector	Link-state	Path Vector
EIGRP	OSPF, IS-IS	BGP

2.2.3 Firewalling

Firewalling is a process that works at layer 4 or 7 of the OSI model and uses information on these layers and below to control traffic between different networks [32][122] such as between data center networks and internal networks and internet. The network infrastructure component that performs firewalling is called a firewall. In addition, firewalls can be implemented in software such as Windows Firewall [33] and Linux iptables [34]. Similar to switching and routing topologies in data centers, firewalling topology consists of interconnected firewalls using redundant paths and links for high availability as shown in Figure 8.

Firewalls typically block all traffic by default and only specifically permitted traffic is allowed. Layer 4 firewalls permit traffic based on information up to layer 4 such as source and destination IP addresses and source and destination TCP and UDP ports such as TCP-22 and TCP-80. Layer 7 firewalls such as NGFWs and WAFs can use traffic information up to layer 7 [26][27] and instead of permitting TCP-22 and TCP-80, they permit the applications Secure Shell (SSH) [35] and HTTP respectively. This adds an additional layer of security as attackers can utilize the ports TCP-22 and TCP-80 for other purposes. By permitting the applications SSH and HTTP the firewall determines that the traffic matches the applications SSH and HTTP alike traffic and blocks all other traffic on the ports. Today most firewalls are stateful meaning it is not specifically needed to allow return traffic by tracking the state of TCP connections which simplifies the firewall ruleset [32].

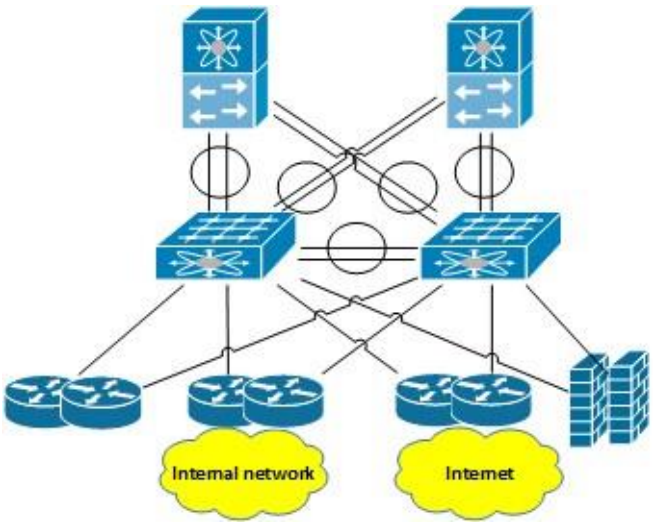


Figure 8 Firewalling topology with redundant paths and links.

2.2.4 Load balancing

Load balancing is a process that works at layer 4 or 7 of the OSI model and uses information on these layers and below to distribute traffic across a number of servers in a pool called pool members or backend servers. Load balancing is used to decrease the burden of individual servers and therefore increase the performance and reliability of applications [36]. The network infrastructure component that performs load balancing is called a load balancer and servers are part of the compute infrastructure. Similar to other topologies in data centers, load balancing and server topology consists of interconnected load balancers and servers using redundant paths and links as shown in Figure 9.

Layer 4 load balancers distribute traffic based on information up to layer 4 such as source and destination IP addresses and source and destination TCP and UDP ports. Layer 7 load balancers can use information up to layer 7 such as HTTP headers [36], cookies [37], and Uniform Resource Identifier (URI) [38].

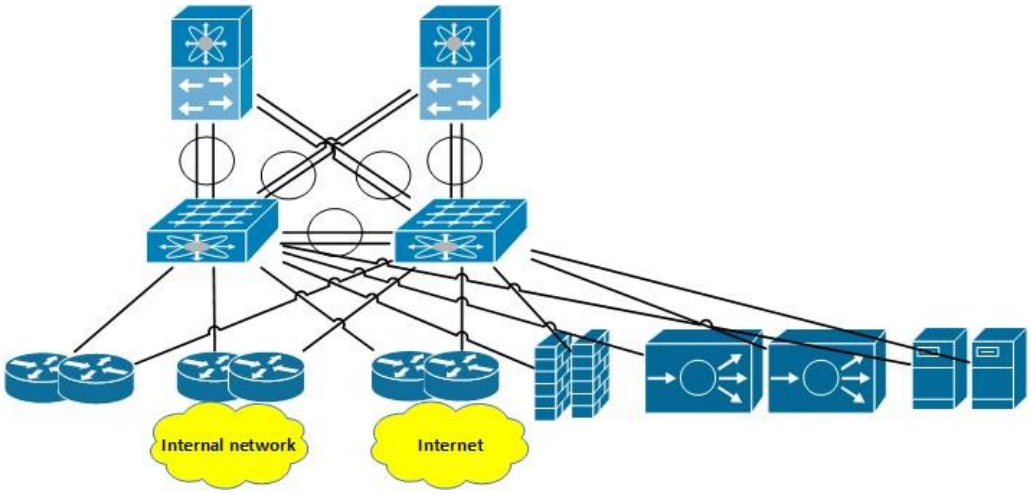


Figure 9 Load balancing and server topology with redundant paths and links.

Load balancers host virtual servers that are the entry point to the application that is load balanced. The virtual servers have virtual IP address and a service such as HTTP and upon receival of traffic, traffic is distributed to the default pool [39]. Traffic can also be distributed to pools other than the default pool or filtered based on layer 4 and 7 information by using scripting [40][41] or advanced configuration options [42] depending on the vendor. Session persistence is used to distribute traffic from the same client to the same pool member during a

session or subsequent sessions that is needed for certain applications such as to keep users' items in a shopping cart or reservations to a concert. The common persistence profiles used are source address and cookie. The source address profile distributes traffic to the same pool member based on the source IP address of the client and cookie profile distributes traffic based on the HTTP cookie header that is inserted by the application [43].

There are various types of load balancing algorithms or methods available for distributing traffic inside a pool. The common load balancing methods used are round robin, least connections, and ratio. The round robin method distributes traffic evenly to each pool member in a round robin fashion. The least connections method distributes traffic to the pool member that has the least number of current connections. The ratio method distributes traffic to each pool member based on a ratio weight that is defined for each pool member. The round robin and least connections methods are most suitable for environments where pool members have roughly equal specifications such as processing power and memory. The ratio method is more suitable for environments where pool members have different specifications [44].

Load balancers use health and performance monitoring of pool members to determine whether the status of a pool member is up or down and therefore capable of handling traffic. The common pool member monitors are TCP, HTTP, HTTP Secure (HTTPS), and Internet Control Message Protocol (ICMP). In addition, custom monitors can be defined such as retrieving a URI from a pool member [45].

Transport Layer Security (TLS) provides a secure channel between two communicating parties, typically client and server or load balancer, to achieve authentication, confidentiality, and integrity [46]. TLS uses digital certificates to verify the identity and ultimately establish a secure channel to another party. The certificates are part of public key infrastructure (PKI) or certificate authority (CA) acting as a third party that both communicating parties trust.

The main principles of TLS certificates are encryption, public key, private key, authentication, and digital signature. Encryption refers to the process of scrambling the original message such that only the recipient can decrypt and understand the message. Public key is a cryptographic key that the server or load balancer sends clients to encrypt messages. Private key is a cryptographic key that only the server or load balancer has and used to decrypt messages. A private key is never sent out of the server or load balancer. Authentication refers to the process of clients verifying the identity of a server or load balancer with the public key by CA. Digital signatures are used to ensure that the certificate

was not altered in transit by external parties [47]. TLS certificates include domain name, CA, CA's digital signature, issuance date, expiration date, public key, and TLS version. HTTPS is the practice of establishing a secure TLS connection over insecure HTTP [48].

The three common TLS methods for load balancer are TLS offloading, TLS pass through, and TLS full proxy. In all the TLS methods load balancer receives HTTPS or encrypted traffic and the method used determines how to handle the encrypted traffic.

The TLS offloading method refers to the setup where load balancer decrypts the traffic, interacts with the traffic, and sends the decrypted traffic to pool members. The TLS offloading method decreases the load of pool members by sparing pool members from encryption and decryption of traffic, but limitation is that the traffic is sent from load balancer to pool members in plaintext.

The TLS pass through method refers to the setup where load balancer passes through the encrypted traffic to pool members without interacting with the traffic. TLS pass through introduces limitations as the load balancer cannot read or interact with the layer 7 information of the message such as HTTP headers, cookies, and URI.

The TLS full proxy refers to the setup where load balancer decrypts the traffic and re-encrypts the traffic and sends the encrypted traffic to pool members. The TLS full proxy address the limitations of TLS offloading and pass through but requires the highest number of resources from all the methods [49]. The TLS methods are depicted in Figure 10.

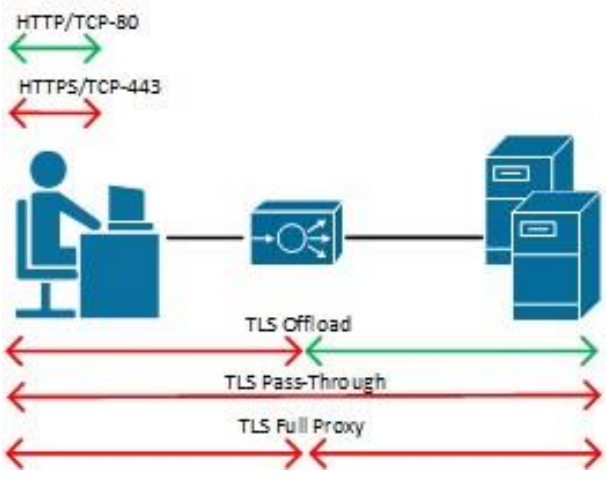


Figure 10 Load balancer common TLS methods [49].

2.2.5 Network services

Network services are processes that work at layer 7 of the OSI model and provide infrastructure components and clients the required services to make communication possible in practice. The network services are provided by specialized servers such as DNS, NTP, and DHCP servers that are part of the compute infrastructure. Similar to other topologies in data centers, DNS, NTP, and DHCP topologies consist of interconnected servers using redundant paths and links.

DNS converts the hosts' numeric IP addresses into a more memorable domain name that can be used by the clients instead of the IP address to reach a host. DNS servers are queried to resolve a fully qualified domain name (FQDN) by the clients and DNS servers return the corresponding IP address for the FQDN. DNS servers store different types of records such as Address (A), NameServer (NS), and Mail Exchange (MX) records. If a DNS server is unable to resolve a name, the DNS servers contacts other DNS servers in hierarchical manner and after a match is found and returned to the original DNS server the corresponding IP address for the name is cached and returned to the client [14].

NTP synchronizes the time among infrastructure components and clients using distributed time servers called NTP servers. NTP uses a stratum to describe the distance to an authoritative time source from which the NTP server receives its time. Stratum 1 NTP server receives its time from directly attached radio or atomic clock and Stratum 2 NTP server receives its time from a stratum 1 NTP server and so on. NTP is critical to correlating events between infrastructure components and effectively troubleshooting possible issues. NTP security features such as access list restriction and authentication are recommended to avoid any accidental or malicious time synchronization [1].

DHCP automates the assignment of hosts' IP information using DHCP servers. When a host that is configured for DHCP and connects to a network that offers DHCP, the host contacts the DHCP server and requests IP information. The DHCP server offers the IP information such as the address that is chosen from a range or pool and leases it to the host based on the server configuration. DHCP is the preferred choice in end user networks such as office networks, and Virtual Private Networks (VPNs) [50], and management networks such as Integrated Dell Remote Access Controller (iDRAC) [51] and Integrated Lights-Out (iLO) [52], where changes are frequent. In data center production networks such as load balancer virtual servers and backend server networks DHCP usage is rarer due to the nature of data

center production networks offering services. In other words, frequent changes to services are not desired.

2.2.6 Network security

Network security refers to securing network access and infrastructure components from unauthorized access, misuse, and theft physically and logically [53]. Traditional logical data center network security is implemented using different network segments or zones that are separated by firewalls that control the traffic between the segments. The different network segments have common security requirements within the company such as internet, demilitarized zone (DMZ) [54] or external, internal, and management [55]. The data center network segments are depicted in Figure 11.

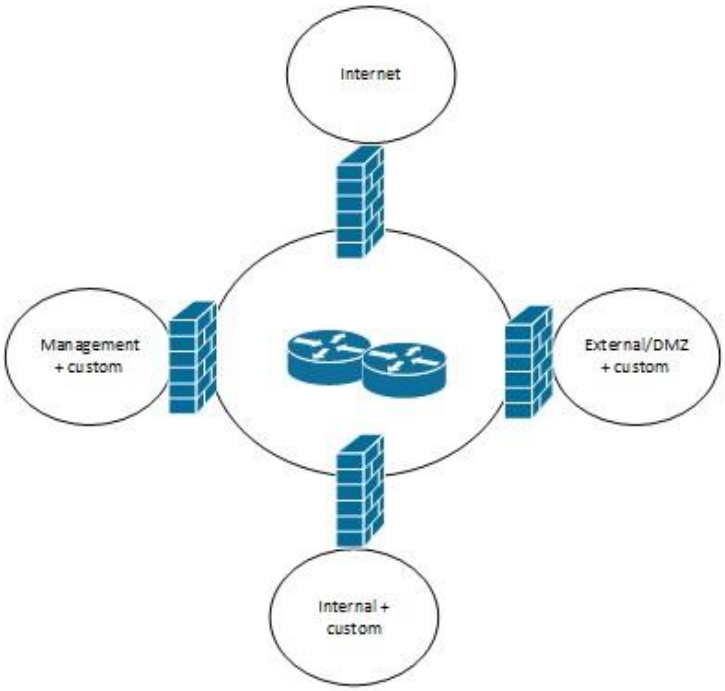


Figure 11 Data center network logical segmentation.

Traditional data center network security using different network segments has shortcomings as traffic within a segment is trusted and not controlled and logged by a network firewall and in case of a compromised server the whole segment is compromised.

The network security zero trust model [56] can be applied to address traditional data center network security shortcomings where the concept of trust is eliminated and all resources and communication are verified, secured, limited, controlled, inspected, and logged [56]. Microsegmentation is part of the zero trust model and architecture that can be accomplished by different technologies. A trivial traditional microsegmentation example is to place each server to its own subnet such as /30 and have a network firewall serving as gateway for the servers but the setup is inefficient and wastes IP addresses. Another traditional microsegmentation example is Private VLANs (PVLANs) [57] that is a mechanism to achieve server isolation in the same subnet. The traditional microsegmentation implementations still have a degree of trust built into them [58] and are hard to accomplish and operate and alternative implementations have been developed to allow for flexibility and automation such as Cisco ACI microsegmentation (uSeg) [1] and VMware NSX Distributed Firewall [59].

2.3 Network automation

Network automation is the process of automating repetitive tasks and standard deliveries in the network domain such as in data center networks. Network automation offers several benefits such as reduced operational costs, redirecting network engineers to focus on development tasks and projects that generate business value rather than doing the same repetitive operational tasks repeatedly. In addition, network automation produces deterministic outcomes and therefore decreases the probability of incidents caused by change as manual and error-prone network configuration processes are minimized. Lastly, network automation enables faster deployment and delivery without requiring manual intervention or network configuration via Graphical User Interface (GUI) or CLI [1].

Network automation code or scripts can be written in various programming languages such as Python [60] and Terraform [61] and run in the network infrastructure components themselves accessing the shell or Bash of the devices or from a remote server acting as a client. Remote server can access the network infrastructure components agentless via CLI [62] and APIs [63] or via an agent that is installed on the devices [1].

2.3.1 CLI based

CLI based network automation refers to scripts that connect to devices' CLI via Telnet or SSH and send commands with input parameters to configure the device and return output. The output from commands such as different show commands is not standardized and can

vary greatly depending on the device such as vendor and software version. The non-standardized output means the network engineer needs to write extra code to parse the data to only include required information which can be time consuming and error prone [64]. CLI based network automation is commonly written in Python and many libraries specifically developed for network automation exist such as Netmiko [65] that can be used to send commands and return output [66] as shown in Figure 12.

The output typically would have to be parsed by the network engineer to search for specific information such as operational state and IP address using filtering and regular expressions (Regex) [67].

```
jere@Jere-Koti:~/code_examples$ cat show_ip_int_brief.py
from netmiko import ConnectHandler
from getpass import getpass
import re

cisco1 = {
    "device_type": "cisco_nxos",
    "host": "example.com",
    "username": "admin",
    "password": getpass(),
}

command = " show interface vlan 1"

with ConnectHandler(**cisco1) as net_connect:
    output = net_connect.send_command(command)

print(output)
jere@Jere-Koti:~/code_examples$ python3 show_ip_int_brief.py
Password:
Vlan1 is up, line protocol is up, autostate enabled
  Hardware is EtherSVI, address is aaaa.bbbb.cccc
  Internet Address is 192.168.1.1/24
  MTU 1500 bytes, BW 1000000 Kbit, DLY 10 usec,
   reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation ARPA, loopback not set
  Keepalive not supported
  ARP type: ARPA
  Last clearing of "show interface" counters never
  L3 in Switched:
    ucast: 0 pkts, 0 bytes
  L3 out Switched:
    ucast: 0 pkts, 0 bytes
```

Figure 12 Cisco NX-OS device CLI based script [66].

2.3.2 JSON and XML

JavaScript Object Notation (JSON) [68] and Extensible Markup Language (XML) [69] are machine-readable and human-readable data formats that structure and standardize data that is input and output to and from a code or script such as in network automation and used for data exchange between infrastructure components.

JSON is a lightweight text-based open standard designed for human-readable data exchange. JSON objects are unordered set of name and value pairs that can be easily passed and consumed programmatically.

XML is a text-based and platform and programming language neutral format to help structure data and is useful for data exchange between infrastructure components. XML documents have a tree structure and consist of root and child elements that contain the data that is structured.

JSON and XML are both machine-readable and human-readable data formats but there are few differences. XML is more verbose than JSON and does not include arrays. In addition, most programming languages contain libraries for parsing JSON and XML but evaluating specific data elements can prove to be more challenging in XML. JSON and XML structures are depicted [1] in Figure 13.

Figure 13 Cisco NX-OS device data in JSON and XML structures [1].

2.3.3 REST API

APIs are offered by applications to provide a way for software to communicate with application. APIs allow the development of rich applications with a wide variety of functionality and services [1]. REST [70] APIs are based on the HTTP request and response

model where for software to communicate with an application via REST API is to make an HTTP request and receive a response in JSON or XML.

To construct a request the required information is method, URL, URL parameters, authentication, additional HTTP headers, and request body. Authentication options are none, basic HTTP, token, and OAuth. Response body includes HTTP status codes, data, format, and attributes [1]. NX-API REST offers improved access to Cisco NX-OS devices via REST APIs [71]. REST API towards Cisco NX-OS device [72][73] is shown in Figure 14.

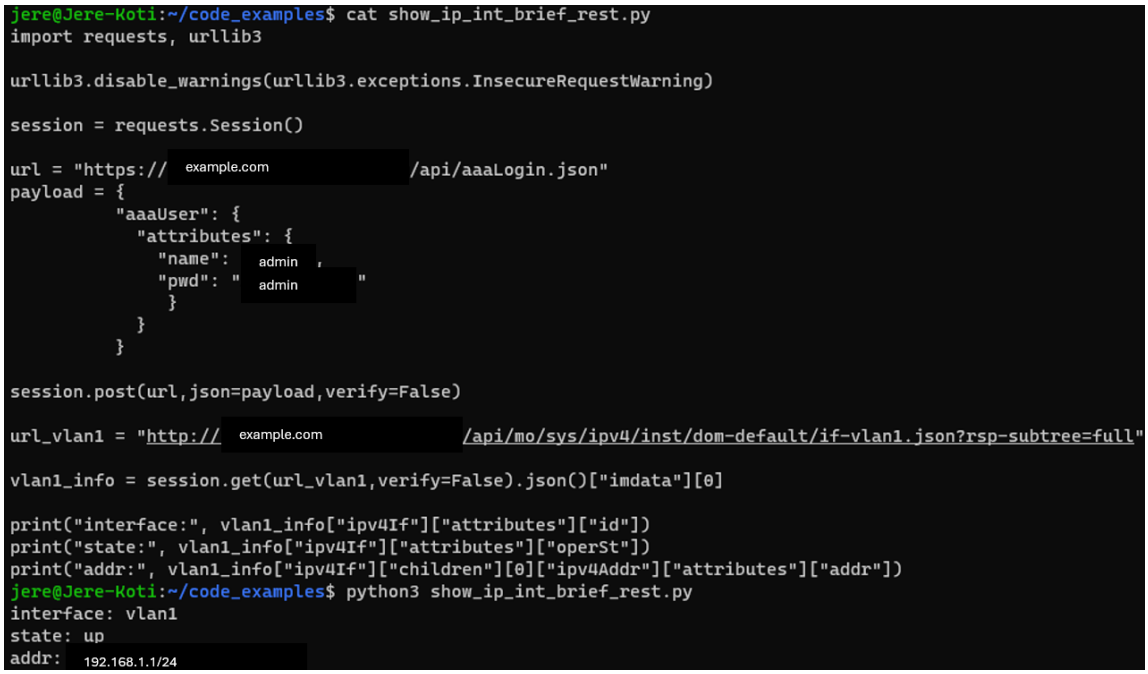


Figure 14 Cisco NX-OS device REST API based script [72][73].

2.3.4 Infrastructure-as-code

IaC is the ability to provision and manage infrastructure components via code [123] instead of manual and error-prone configuration processes via GUI and CLI. IaC defines the desired state of the infrastructure for the environment and automates the process to reach that state [74]. IaC offers all the benefits of automation such as reduced operational costs, deterministic outcomes, and speed. In addition, IaC is highly reusable to create similar environments to support different systems and customers [74].

IaC describes a system architecture that consists of resources such as networking, storage, servers, and operating systems and their configuration in source code which are ultimately

created in the target environment. There are two approaches in creating the resources which are declarative and imperative. Declarative IaC describes the resources and their configuration that make up the desired state regardless of the steps to reach that state. In imperative IaC all the steps are described to set up resources and their configuration to reach the desired state.

IaC can be version controlled like any other code and integrated into continuous integration and continuous deployment (CI/CD) pipelines to reduce development lifecycle and provide continuous delivery of high quality to customers in DevOps fashion [74]. IaC can be written in various programming languages and many IaC tools exist such as Terraform [75], Ansible [76], Puppet [77], AWS CloudFormation [78], and Azure ARM [79].

HashiCorp Terraform is an IaC tool which defines resources in code that can be versioned, reused, and shared to manage infrastructure components throughout their lifecycle. Terraform creates and manages resources via APIs [80]. Providers that are found in Terraform Registry [81] enable Terraform to work with many different types of resources and services such as AWS, Azure, GCP, Cisco, F5, Fortinet, and Citrix.

The Terraform workflow consists of three stages which are write, plan, and apply. Write is where the resources are defined or coded. Terraform plan creates an execution plan based on the defined resources and existing target infrastructure. Terraform apply performs the proposed changes if any in the terraform plan to the target environment [80] Terraform uses a state file to determine the needed changes to match the infrastructure with the configuration. The state file in JSON format can be stored locally or in a remote location [82].

Terraform language is used to define resources that represent the infrastructure components and other features include plugins, data, and dependencies. The syntax of Terraform language consists of blocks, arguments, and expressions. Blocks are containers for content that make up an object such as resource. Arguments appear within blocks and assign a value to a name. Expressions represent a value for arguments.

The Terraform language is declarative and the ordering of blocks in code are not generally significant and only implicit and explicit relationships between resources are considered to determine the order of operations [61]. The Cisco DevNet ACI provider can be used to create resources in ACI environment [83] as shown in Figure 15.

```
jere@Jere-Koti:~/code_examples$ cat main.tf
terraform {
  required_providers {
    aci = {
      source = "ciscodevnet/aci"
    }
  }
}

provider "aci" {
  username = "admin
  password = "admin      "
  url      = "example.com                    "
  insecure = true
}

resource "aci_tenant" "example_tenant" {
  name = "example_tenant"
}

resource "aci_vrf" "example_vrf" {
  tenant_dn = aci_tenant.example_tenant.id
  name      = "example_vrf"
}

resource "aci_bridge_domain" "example_bd" {
  tenant_dn          = aci_tenant.example_tenant.id
  name               = "example_bd"
  relation_fv_rs_ctx = aci_vrf.example_vrf.id
}

resource "aci_application_profile" "example_ap" {
  tenant_dn = aci_tenant.example_tenant.id
  name      = "example_ap"
}

resource "aci_application_epg" "example_epg" {
  application_profile_dn = aci_application_profile.example_ap.id
  name                   = "example_epg"
  relation_fv_rs_bd      = aci_bridge_domain.example_bd.id
}
```

Figure 15 Cisco ACI IaC based code [83].

2.3.5 Code management

Code management is the process of storing version-controlled code centrally to share and collaborate on software development projects within individuals, teams, and organizations [84]. Popular code management platforms are GitHub [85] and GitLab [86] which provide a platform for sharing and collaborating on code stored in repositories. Git [87] is a version control system that tracks changes in files and is the basis of platforms like GitHub and GitLab and is used for operations such as creating repositories, branches, and uploading and editing files. Continuous Git pull and push enables collaboration with others on the same repository at the same time. GitHub and GitLab helps to manage the flow of changes via features such as pull requests and merge requests respectively [84].

GitHub and GitLab offer advanced features such as GitHub Actions and GitLab CI/CD which allow for creation of workflows that trigger on specific events on a repository such as on pull request or merge request to automate the build, test, and deployment pipeline or CI/CD.

GitHub Actions and GitLab CI/CD can be run on Linux, Windows, and macOS virtual machines (VMs) provided by the platform or on self-hosted runners [88].

The GitHub and GitLab enterprise platforms offer all the features and functionality of the public platforms but with additional security, management, and customization options in place [89][90].

3 Data center networks environment

3.1 Data center networks team

Data center networks provide connectivity between infrastructure components and the outside world. The infrastructure components include compute, storage, and network components and the outside world includes connectivity to remote sites, public clouds, and the internet. The thesis environment comprises of several data centers and their networks in several Nordic countries.

The data center networks team is responsible for routing and switching and load balancer services for infrastructure components and applications within the company. Roles and responsibilities of the team include network design, development, architecture, operation, life cycle management (LCM) for IT systems both on-premises and public cloud including service assurance on-call 24/7 and 2nd and 3rd level support. Level 3 support includes delivery, incident, problem and change flow handling, plan and execution of changes according to agreed principles, capacity follow-up, allocations, migrations, expansions, performance follow-up and tuning, upgrades, patching and technical maintenance, improvements and automation, and participation in projects and system development.

The data center networks team manages around 1000 switches, 60 routers, and 180 load balancers from multiple vendors and software versions and handled around 600 service development tasks, 1400 service requests, and 500 change requests in 2023 [91].

3.1.1 Service development

Service development refers to development tasks in the data center networks that are required or requested by stakeholders such as tasks related to roadmap, development, maintenance, operations, capacity management, and LCM. In addition, development of automation and tools belong to service development. Service development task types are analysis, design, implementation, automation, solution design, development, migration, public cloud, security, documentation, and IP version 6 (IPv6) development [91].

SAFe framework [92] is used in the data center networks team for service development and tasks or stories are planned in Program Increments (PIs) and divided between team members and added to sprints by the product owner in collaboration with the team. The sprints are

typically two weeks of length and during the sprints the team works with the planned tasks and ideally closes them before end of the sprint [91].

3.1.2 Service requests

Service requests refers to stakeholder requests to meet their network requirements. There are two different types of service requests: routing and switching and load balancer. Common routing and switching service requests include but are not limited to creating a new VLAN, switch port configuration, routing configuration, general query, security guideline, network selection consultancy, firewall opening consultancy, and troubleshoot of existing network infrastructure. Common load balancer service requests include but are not limited to provision virtual server, adjust virtual server configuration, virtual server troubleshoot, TLS certificate installation and renewal, application deployment consultancy, and application deployment testing and tuning [91].

3.1.3 Change requests

Change requests refers to changes done in the data center networks and are typically implemented in relation to service development task or service request. There are three types of change requests: standard, normal, and emergency. Standard changes are used for repetitive, low-risk, and well-tested changes with no impact to customers, users, and other systems such as creating a new VLAN. Standard changes are pre-approved and can be implemented during office hours without the involvement of change managers. Normal changes are used for changes that must be notified and go through the full change management process: assessment, authorization, prioritization, and possibly change advisory board (CAB) for approval such as router and switch software upgrade. Normal changes are typically implemented outside office hours. Emergency changes are used to restore full production after incidents or urgently prevent incidents from happening [91] such as re-enable a faulty link after fix to secure redundancy or security patch to prevent possible breaches.

3.2 Areas of automation

The data center networks team needs automation in all areas: service development, service requests, and change requests to keep up with the required delivery speed. As automation offers the most value in small and incremental changes that do not have many dependencies

[94] the service request area which fits the description was studied. Areas of automation in service request area marked in orange are shown in Figure 16.

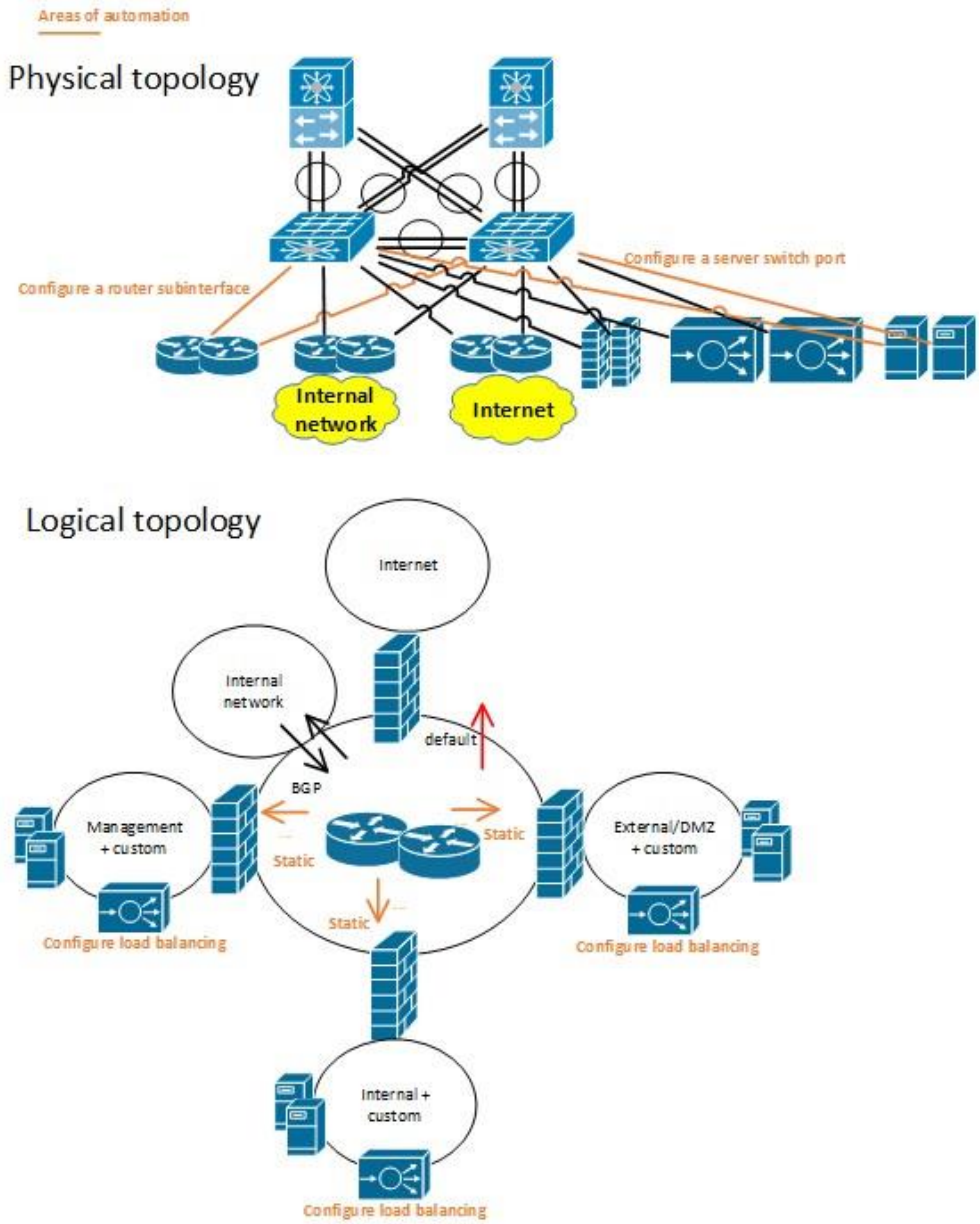


Figure 16 Areas of automation in service request area.

The service request data show that from the routing and switching and load balancer service requests (1400) majority of the service requests were load balancer requests (900) [91]. There did not exist any common load balancer automation in the team, only local CLI based scripts in some of the devices for very specific service requests. The decision was made to focus on load balancer service automation and the goal was to automate all the requests included in the

standard load balancer service request form, such as provision of virtual server including TLS, adjustment of virtual server configuration, and decommission of virtual server.

3.2.1 Challenges

Data center network automation is challenging to implement due to general complexity of networks and based on survey of enterprise, cloud providers, and network service providers only 23 % were fully confident in their data center network automation strategy [95].

Data center network automation planning and evaluation challenges are 1. lack of understanding of the interaction with other tools and management systems due to the multitude of tools used for network management and automation in different teams (39 %) and 2. planning budgets and costs (37 %) [95]. Data center network automation implementation challenges are 3. heterogeneous and legacy equipment that make the automation more difficult and time consuming, such as multitude of vendors and software versions and lack of API support (44 %) and 4. struggle to integrate and contextualize the automation with overall application service delivery (44 %) [95]. Data center network automation use challenges are 5. data authority and quality such as lack of device metrics and traffic flows to understand what automated changes are required and configuration standards and security policies (42 %) and 6. unknown impact of automation to application behaviour and performance (40 %) [95]. The selected data center network automation specification should address and consider the challenges mentioned above.

3.2.2 Requirements

Data center network automation options are CLI based scripts, REST API based scripts, and IaC. CLI based scripts are still commonly used for automation but have several challenges such as complexity (1 and 2), cross-platform incompatibility (3 and 4), and visualization and feedback limitations (5 and 6). REST API based scripts address the complexity via standard API calls and data standardization but still have limitations in cross-platform incompatibility (3 and 4) and visualization and feedback limitations (5 and 6) [95][96]. IaC is an alternative automation option and offers additional benefits of cross-platform support by reusability (3 and 4) and visualization and feedback (5 and 6) before applying changes.

IaC tools provide an abstraction layer between the existing infrastructure and its desired state where focus is more on the desired state instead of the detailed tasks to reach that state which

contributes to simplicity. IaC popularity has grown steadily [97] and is expected to continue to be a part of automation and support emerging trends in technology and DevOps practices. The choice of specific IaC tool such as Terraform, Ansible, Puppet, AWS CloudFormation, and Azure ARM will depend on the organization and its needs and goals [96].

4 Current load balancer service delivery model

4.1 Load balancer service

Load balancer service is offered to share the load of applications between multiple real servers [124]. Load balancer service is provided to both internal and external customers and ordered via standard form in company portal by stakeholders [98]. The load balancer service includes provision, adjustment, and decommission of the service and various options. The required technical information in the form is request type, DNS name, DNS record, virtual server port, list of real server IP addresses and ports, persistence, method, and TLS [99]. Table 5 summarizes the load balancer service request form and the options.

Table 5 Load balancer service request form and the options.

Request type	Provision/Adjust/Decommission
DNS name	x (e.g. example.com)
DNS record	Yes/No
Virtual server port	x (e.g. 443)
List of real server IP addresses and ports	x.x.x.x:x, x.x.x.x:x (e.g. 192.168.1.1:443, 192.168.1.2:443)
Persistence	Cookie/Source address
Method	Least connection/Round robin
TLS	Yes/No

Load balancer service supports applications and businesses in several ways such as by sharing the load between real servers to improve the application performance, providing high availability even if one or more real servers in the pool goes down, and simplifying the real server upgrade and update process by removing real servers one by one from the pool for upgrade and adding them back after without service impact. Additional features include custom real server monitoring up to layer 7, persistence to force sessions of the same client to the same real server, redirection such as from HTTP to HTTPS or based on URI, and scripting to offer highly customized environments. TLS offload, TLS pass-through, and TLS full proxy are all supported. The load balancer service basic setup is shown in Figure 17.

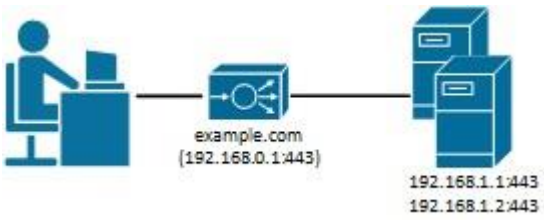


Figure 17 Load balancer service basic setup.

Service level targets for standard load balancer service setup are availability 99.95%, reliability >1 year, and maintainability 1h [98].

4.1.1 Load balancer environment

The data center networks team manages around 180 load balancers either physical or virtual instances from various vendors in several Nordic countries. The load balancer instances are dedicated for different network zones and custom environments that have common security requirements or function within the company. The different load balancer instances have specific IP address blocks allocated for virtual servers and real servers that are used and defined in security policy from which the target load balancer environment is deduced during handling of a load balancer service request [99]. The load balancer instance IP design is shown in Figure 18.

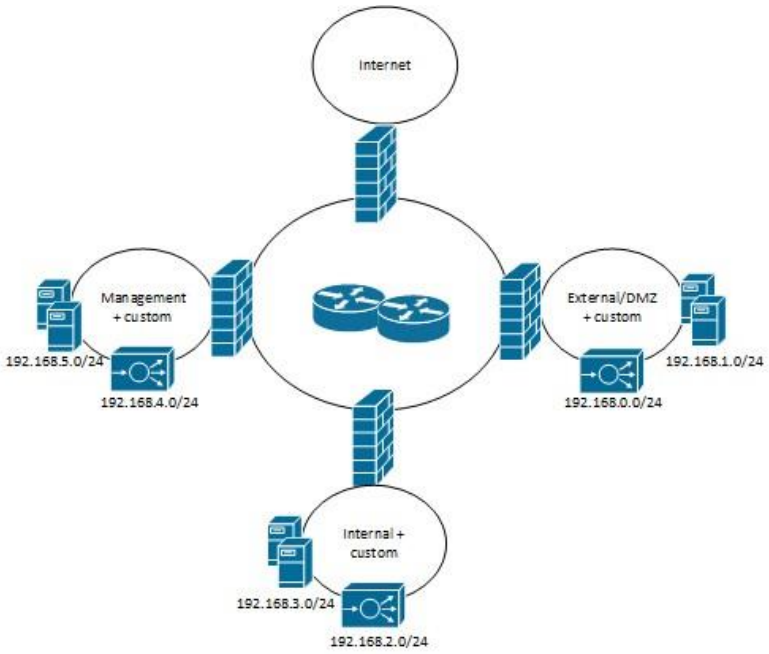


Figure 18 Load balancer instance IP design.

General steps to provision a standard load balancer service include configuration of real servers, a real server pool, and a virtual server. To provision a virtual server with TLS includes additional steps of creating a TLS certificate from a CA and importing it to the load balancer and configuring a TLS profile. Adjustment of the load balancer service can include reconfiguration in any of the steps in provision of a load balancer service. Decommission of a virtual server includes the removal of the configuration added in provision of a load balancer service in reverse direction [99]. The load balancers are configured from jump host in network management zone that has access to load balancers management. Load balancer instance configuration from jump host is shown in Figure 19.

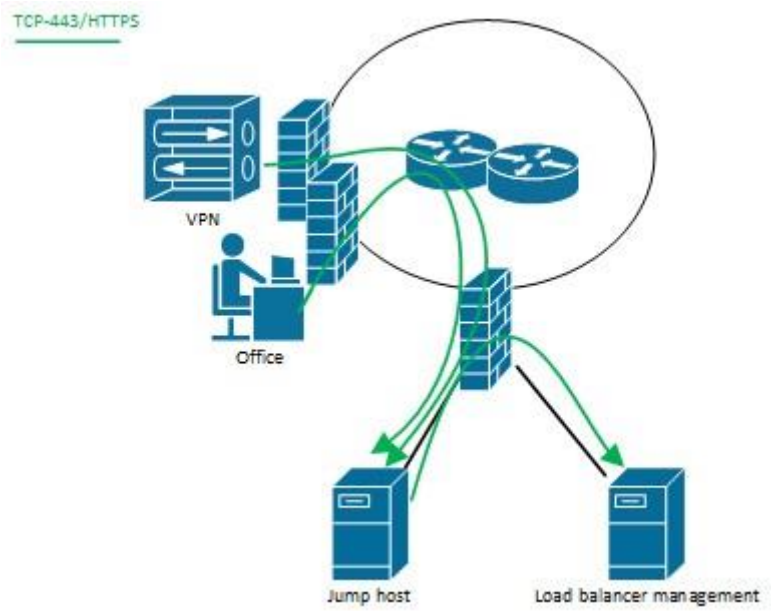


Figure 19 Load balancer instance configuration from jump host.

The load balancer that was selected for the automation development and focus of the thesis is Fortinet Application Delivery Controller (FortiADC) [100]. The decision was made according to the future needs of the company and already existing laboratory environment in place to build a proof of concept. Furthermore, the FortiADC load balancers were relatively new deployments in the environment and therefore there was no automation that have been developed for the FortiADC in the team.

4.2 Current delivery model

FortiADC load balancer optimizes application delivery, enhances performance of applications, and ensures application security available as appliance or VM on-premises or in public cloud. FortiADC offers robust load balancing capabilities up to layer 7 with scripting support and TLS services. FortiADC also has many built-in security features such as WAF, Distributed Denial of Service (DDoS) protection, and Zero Trust Network Access (ZTNA). FortiADC load balancer highlights include application availability, web application protection, global server load balancing (GSLB), data optimization, application access management and artificial intelligence (AI) security. FortiADC load balancer key features include advanced security services, TLS services, scripting and DevOps tools support, analytics and visibility, and security fabric connectors. Table 6 summarizes the FortiADC vendors and tools support [101].

Table 6 FortiADC vendors and tools support [101].

Deployment options	Appliance, Virtual Machine, FortiFlex
Public Cloud	AWS, Azure, GCP, Alibaba, Oracle, IBM
Private Cloud	VMware, Hyper-V, KVM, Citrix, IBM, OpenStack
Connectors	SAP, Kubernetes, RHEL, AWS, Oracle, Splunk
DevOps tools	Ansible, Cloud-init, Terraform, REST API, CloudFormation
Security Fabric	FortiGate, FortiSIEM, FortiAnalyzer, FortiGSLB

FortiADC load balancer use cases are advanced application load balancer, hybrid cloud connectors, web application security, and application automation [101]. The current load balancer service delivery model relies on the FortiADC GUI and CLI for configuration.

4.2.1 GUI and CLI configuration

FortiADC uses virtual domain (VDOM) to create a complete load balancer instance that functions independently to allow support for multitenancy such as based on the network zone or function within the company. VDOM configuration includes all the system and feature configuration options of a full FortiADC load balancer instance. The VDOM feature supports two different virtual domain modes either independent network mode or shared network mode as administrative domains (ADOMs). FortiADC has a predefined VDOM root, and the super and global admin users can add, delete, enable, and disable any non-root VDOM on the system [102]. The configuration of a VDOM in GUI and CLI [102] is shown in Figure 20.

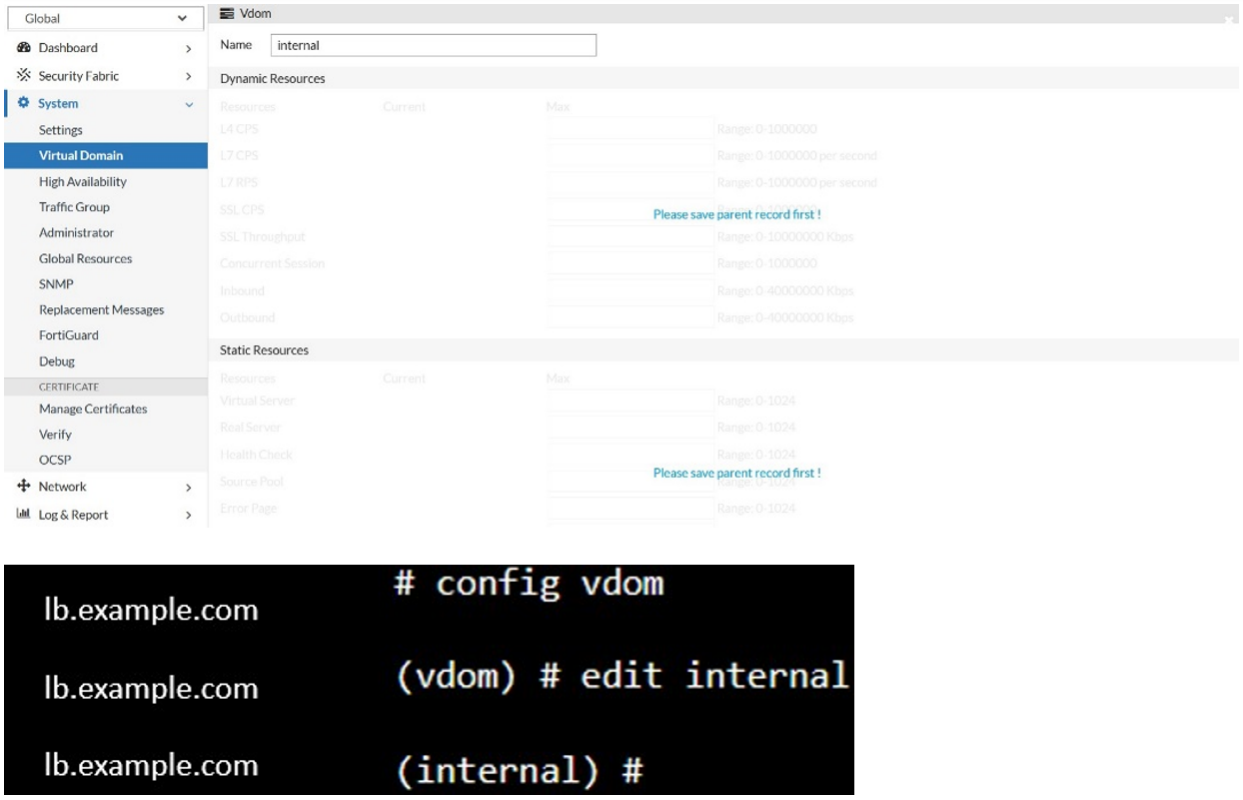


Figure 20 The configuration of a VDOM in GUI and CLI [102].

FortiADC basic load balancer service configuration includes real servers, real server pools, Secure Sockets Layer (SSL) profiles, and virtual servers. Real servers provide services to clients such as HTTP and XML content, audio and video, and file uploads and downloads.

Real server configuration includes name, server type, status, type either IP address or FQDN. The name has valid characters A – Z, a – z, 0 – 9, _, and – with no spaces. The name is

referenced in the real server pool member configuration and cannot be edited after the initial configuration. The server type has options static or dynamic. The server type dynamic options refer to the possibility to configure external real servers from public and private cloud connectors that are provisioned and scaled on-demand. The status options are enable, disable, and maintain. Enabled real server can receive new sessions and serves any current sessions, disabled real server does not receive new sessions and closes any current sessions, and maintain does not receive new sessions but maintains any current sessions. The type either IP address or FQDN are both supported. The IP address support includes IPv6 [102]. The configuration of a real server in GUI and CLI [102] is shown in Figure 21.

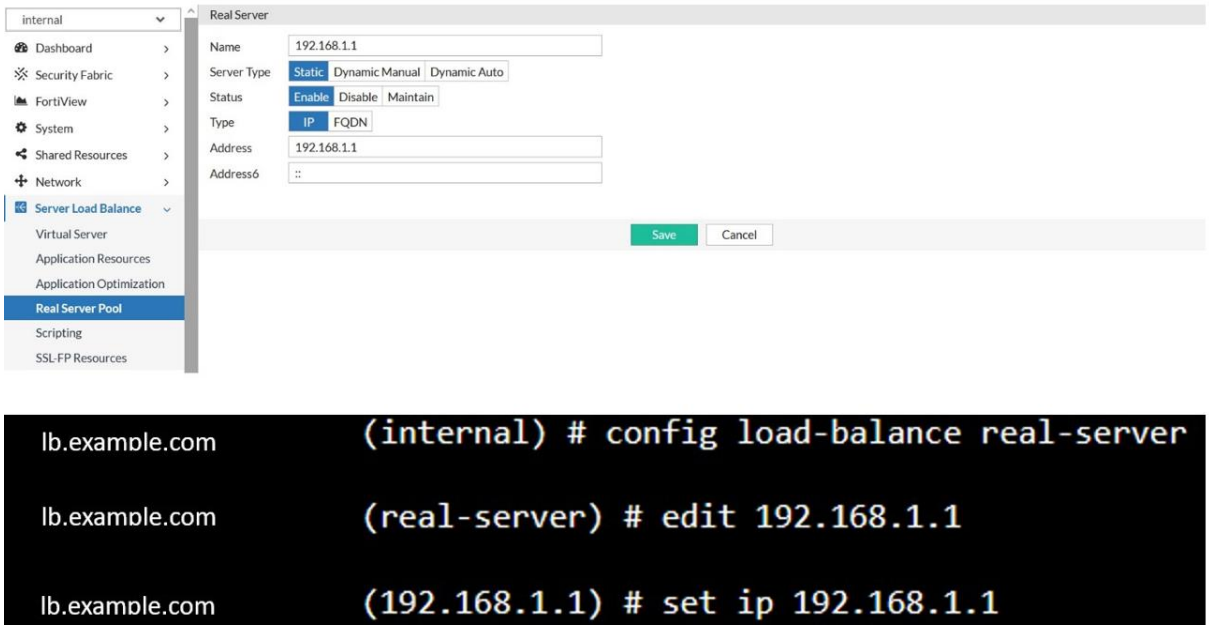


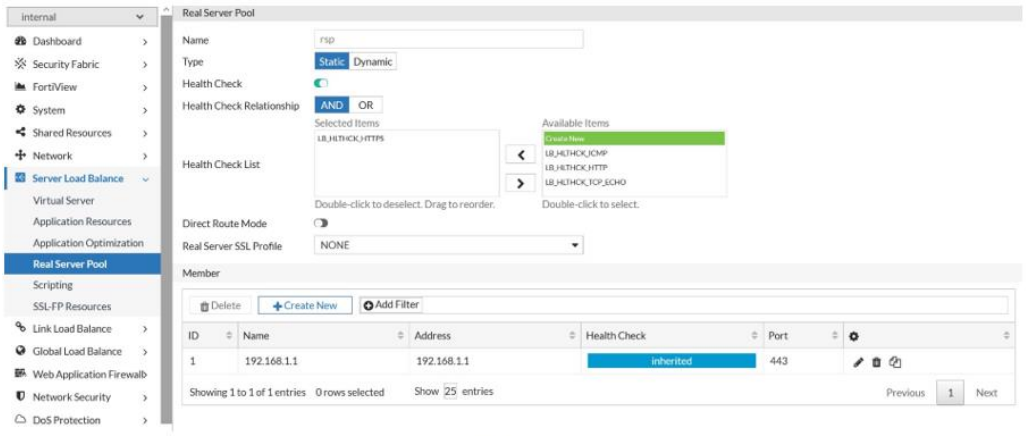
Figure 21 The configuration of a real server in GUI and CLI [102].

Real server pools are groups of real servers that host the applications that are load balanced. Custom health checks and server SSL profiles can be created beforehand and selected in the pool configuration.

Real server pool configuration includes name, address type, type, health check, and real server SSL profile. The name has valid characters A – Z, a – z, 0 – 9, _, and – with no spaces. The name is referenced in the virtual server configuration and cannot be edited after the initial configuration. The address type IP version 4 (IPv4) and IPv6 are both supported. The type has options static or dynamic. The dynamic type refers to external real servers similar to the real server configuration. The health check configuration options enable one or more health

checks for the pool and define on what conditions the pool members are considered available to serve clients. The real server SSL profile is selected to determine settings for communication between the load balancer and pool members and the default is none which is for non-SSL/TLS traffic.

The real server pool member basic configuration includes status, real server, port, weight, cookie, and health check. The status options are enable, disable, and maintain similar to the real server configuration. The real server object is selected from a drop-down list and listening port is selected from the range 0 – 65535 where the port 0 is a wildcard port. The weight option allows to assign a preference among pool members. The weight has a valid range of 1 – 256 where the higher values are preferred, and the default value is 1. The cookie name is set and used when session persistence is enabled on virtual server. The health check options allow to overwrite the health check configuration defined in the pool. The configuration of a real server pool and member in GUI and CLI [102] is shown in Figure 22.




```
lb.example.com          (internal) # config load-balance pool
lb.example.com          (pool) # edit rsp
lb.example.com          (rsp) # set health-check-ctrl enable
lb.example.com          (rsp) # set health-check-list LB_HLTHCK_HTTPS
lb.example.com          (rsp) # set real-server-ssl-profile NONE
lb.example.com          (rsp) # config pool_member
lb.example.com          (pool_member) # edit 1
lb.example.com          (1) # set pool_member_service_port 443
lb.example.com          (1) # set pool_member_cookie rs1
lb.example.com          (1) # set real-server 192.168.1.1
```

Figure 22 The configuration of a real server pool and member in GUI and CLI [102].

SSL profiles client and server determines the communication between the client and load balancer and load balancer and real servers respectively. There exist various predefined SSL profiles for both client and server and custom profiles can be defined. Client SSL profile basic configuration includes name, SSL ciphers, allowed SSL or TLS versions, certificate information, and SSL Diffie-Helman (DH) parameter size. The name is a unique name for the client SSL profile and cannot be edited after the initial configuration. The SSL ciphers options allow to select a predefined list of ciphers or customized set of ciphers. The allowed SSL or TLS versions are SSLv3, TLSv1.0, TLSv1.1, TLSv1.2, and TLSv1.3. The local certificate group needs to be created before creating a client SSL profile. The local certificate group includes the certificate that is created from a CA and imported to the load balancer and offered to clients. The SSL DH parameter size specifies the public key length in DH and the options are 1024 bits, 2048 bits, and 4096 bits. The configuration of a client SSL profile in GUI and CLI [102] is shown in Figure 23.

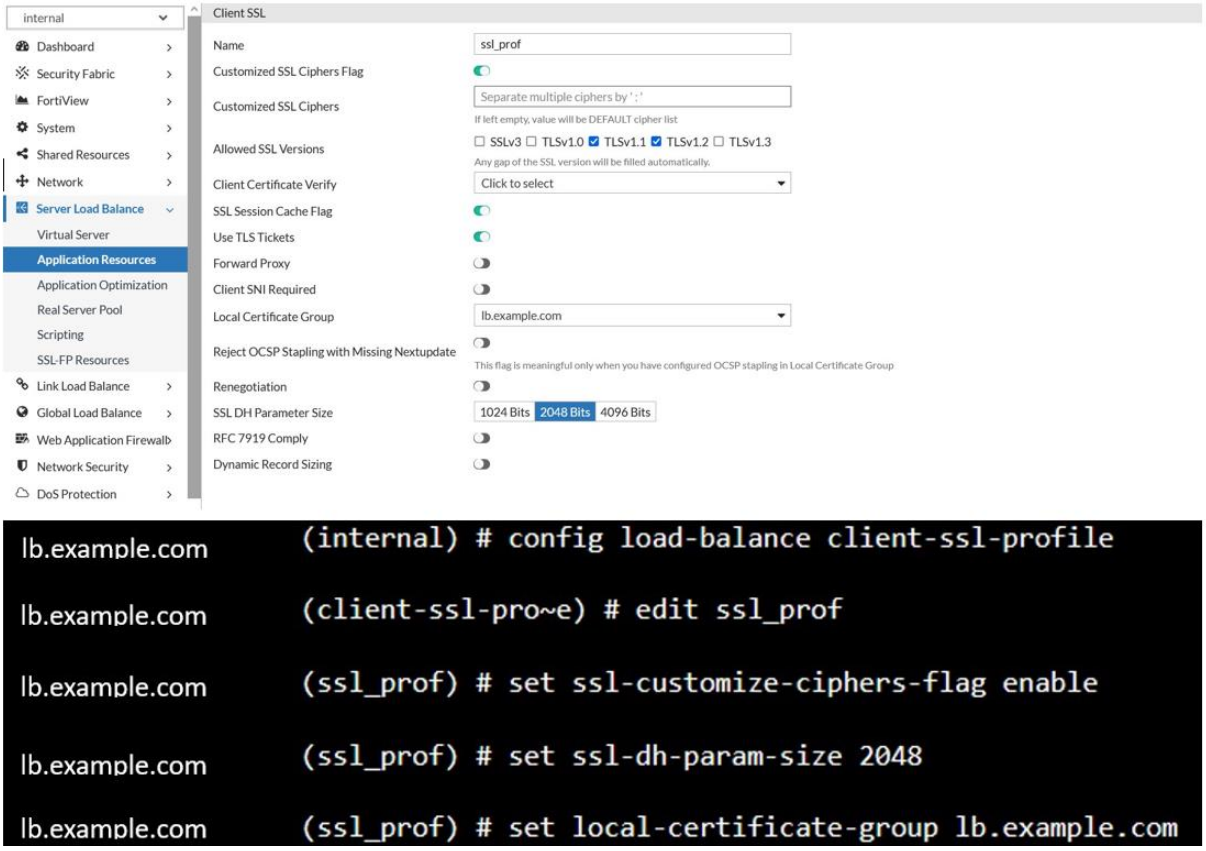
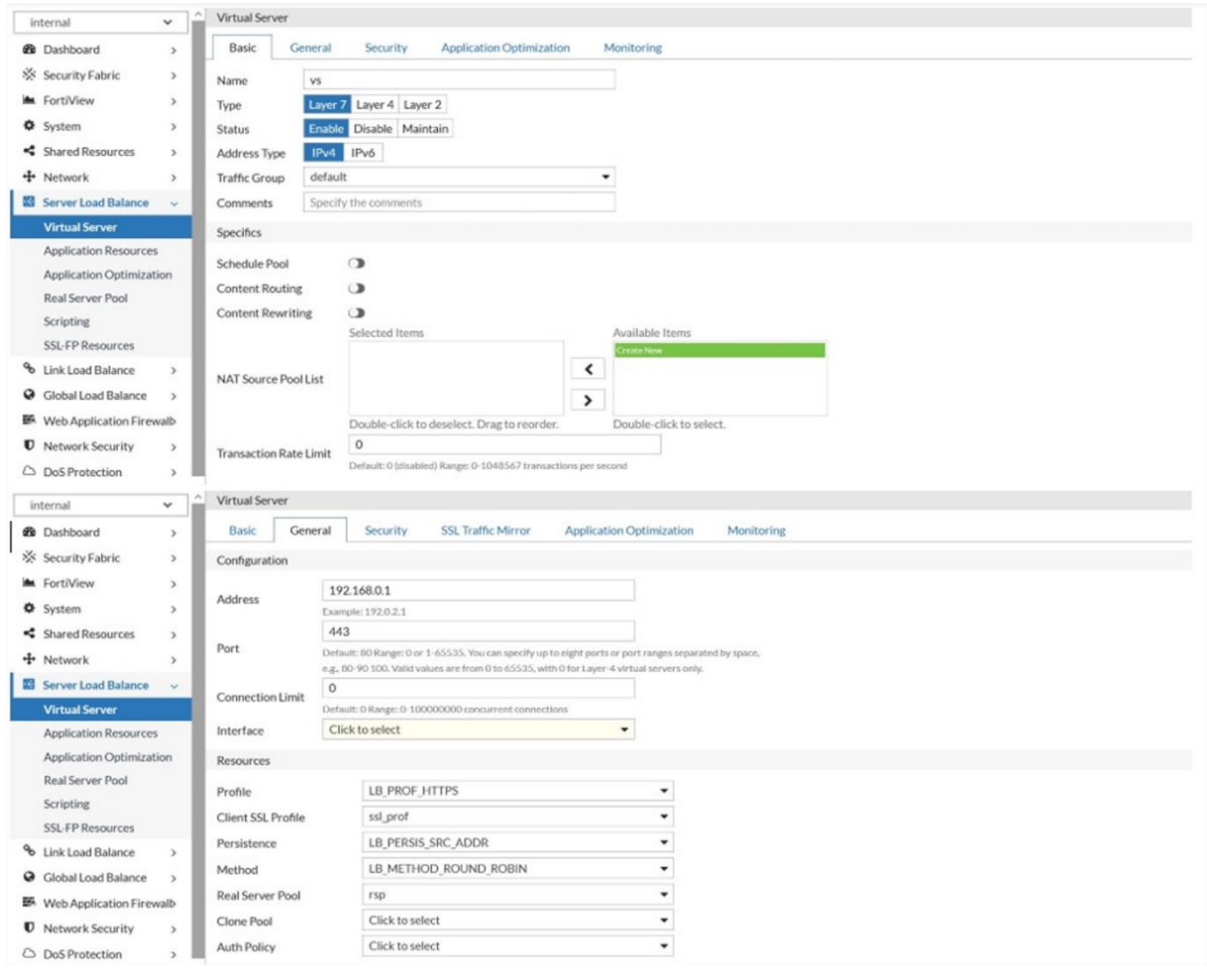


Figure 23 The configuration of a client SSL profile in GUI and CLI [102].

Real server SSL profile basic configuration includes similar options: name, SSL ciphers, allowed SSL or TLS versions, and certificate information [102].

Virtual servers are the entry point to the applications that are load balanced to a number of real servers in a real server pool. The virtual server basic configuration includes name, type, status, address type, address, port, interface, profile, client SSL profile, persistence, method, and real server pool. The name is a unique name for the virtual server and has valid characters A – Z, a – z, 0 – 9, _, and – with no spaces. The virtual server's name cannot be edited after the initial configuration. The type has options layer 2, layer 4, and layer 7 and based on the selected type there are different options and features available for the virtual server. Virtual server type layer 7 offers the most options and features. The status options are enable, disable, and maintain similar to real server and real server pool configuration. The addresses IPv4 and IPv6 are both supported. The virtual server address, port, and interface options describe the virtual server IP address and listening port that clients connect to and from which interface the client traffic is received. The profile option defines how the virtual server should handle the traffic such as predefined profiles for TCP, HTTP, and HTTPS traffic or custom profiles. Client SSL profile option allows to select from previously created client SSL profiles for TLS traffic. The persistence option is used to configure virtual server persistence such as based on source address or cookie. The method and real server pool options select the load balancing method such as round robin or least connection for the real server pool. The configuration of a virtual server in GUI and CLI [102] is shown in Figure 24.




```
lb.example.com      (internal) # config load-balance virtual-server
lb.example.com      (virtual-server) # edit vs
lb.example.com      (vs) # set type l7-load-balance
lb.example.com      (vs) # set ip 192.168.0.1
lb.example.com      (vs) # set port 443
lb.example.com      (vs) # set load-balance-profile LB_PROF_HTTPS
lb.example.com      (vs) # set client-ssl-profile ssl_prof
lb.example.com      (vs) # set load-balance-persistence LB_PERSIS_SRC_ADDR
lb.example.com      (vs) # set load-balance-method LB_METHOD_ROUND_ROBIN
lb.example.com      (vs) # set load-balance-pool rsp
lb.example.com      (vs) # set traffic-group default
```

Figure 24 The configuration of a virtual server in GUI and CLI [102].

4.2.2 Automation

FortiADC supports all the automation options as in data center networks: CLI based scripts, REST API based scripts, and IaC. CLI based scripts can be run to send commands to configure FortiADC and return output in response to a trigger event. The scripts can be entered either manually or as a file. REST API based scripts can be run to communicate with FortiADC and make configuration changes and return standardized output. The REST API administrator account and token is required to communicate with FortiADC REST APIs. FortiADC supports API gateway functionality which allows to create policies that accept only required API calls and return expected output instead of permitting access directly to FortiADC management [102]. FortiADC supports IaC tools such as Terraform and Ansible [101]. None of the supported automation options are used in the data center networks team for FortiADC.

4.3 Development requirements

The current load balancer service delivery model relies on heavy usage of vendor specific GUI and CLI to make configuration changes with weak automation such as local CLI based scripts in some of the devices for very specific service requests and none for the FortiADC.

The development requirement for the current load balancer service delivery model is automation as studies have shown the several benefits of network automation such as increased efficiency, reduced configuration errors, and enhanced performance and security of the network [103]. From the automation options: CLI based scripts, REST API based scripts, and IaC, IaC is proposed as the automation option as IaC offers the most benefits and addresses the data center network automation challenges the most comprehensively from the available options [95][96].

IaC popularity has grown steadily among practitioners and researchers [97] and is expected to continue to be a part of automation and support emerging technology and DevOps practices [96]. The IaC tool Terraform and FortiADC provider [104] are proposed as there is already use and familiarity with Terraform in the data center networks team and the tool is expected to be a part of future automation projects.

Terraform is supported by various on-premises tools: GitHub Enterprise for code management and actions, private cloud for self-hosted runner, and Simple Storage Service (S3) compatible

storage for state data. On-premises tools are selected due to security policy requirements and therefore lack of access from internet towards network management zone where the load balancers management resides.

5 Proposed load balancer service delivery model

5.1 Specification

The proposed load balancer service delivery model is IaC based and the tools selected are Terraform and FortiADC provider. Terraform is supported by various on-premises tools to complete the delivery model such as GitHub Enterprise for code management and actions, private cloud for self-hosted runner and S3 compatible storage for state data. The on-premises tools are selected due to lack of access from internet towards network management zone. The proposed load balancer service delivery model components and design are shown in Figure 25.

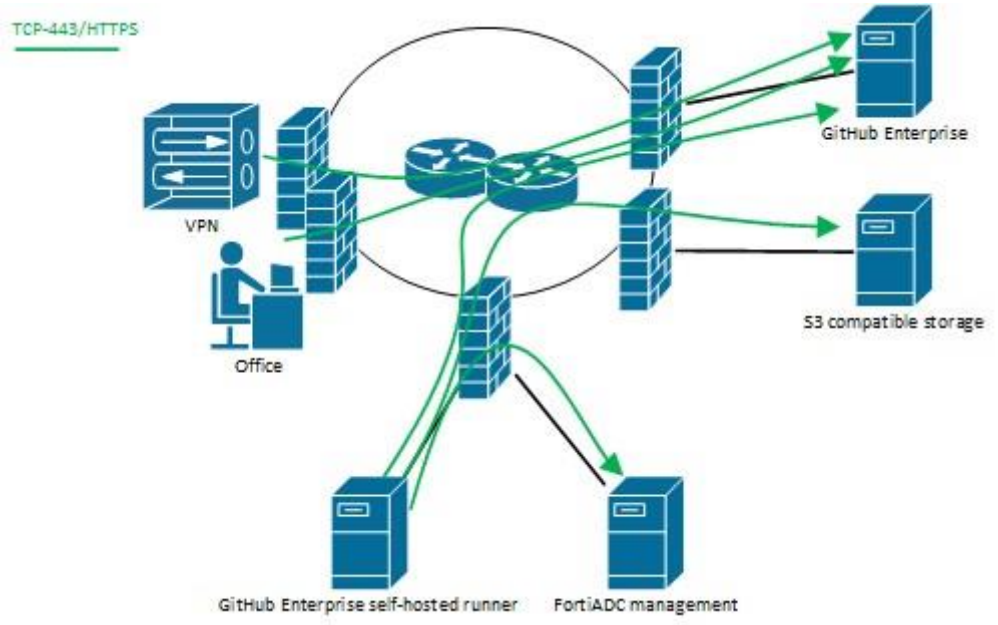


Figure 25 The proposed load balancer service delivery model components and design.

GitHub Enterprise is accessible from the office and VPN networks for users where the delivery model repository resides. The configuration changes are planned via pull requests and applied via merge pull requests that trigger actions on the GitHub Enterprise self-hosted runner to run commands terraform plan and terraform apply respectively. The GitHub Enterprise self-hosted runner is a Red Hat Enterprise Linux (RHEL) [105] VM hosted in the private cloud. The GitHub Enterprise self-hosted runner resides in the network management zone and has access to GitHub Enterprise to clone the repository, FortiADC management to create and manage resources in the FortiADC via APIs, and S3 compatible storage to store the

state file. All the communication is handled in TCP-443/HTTPS and the same design and components can also be used in other IaC projects.

5.1.1 Terraform FortiADC provider

Terraform FortiADC provider is used to create and manage resources in FortiADC. The FortiADC provider requires REST API administrator account and token to be authenticated. The token can be provided as static credential or environment variable methods and in the delivery model the latter method is used. The FortiADC provider example usage [104] is shown in Figure 26.

```
jere@Jere-Koti:~/code_examples$ cat terraform.tf
terraform {
  required_providers {
    fortiadc = {
      source = "fortinetdev/fortiadc"
      version = "1.0.1"
    }
  }
}
```

```
jere@Jere-Koti:~/code_examples$ cat providers.tf
provider "fortiadc" {
  hostname = "lb.example.com"
  insecure = "true"
  vdom = "internal"
}
```

Figure 26 The Terraform FortiADC provider example usage [104].

5.1.2 S3 compatible storage

Amazon Simple Storage Service (Amazon S3) is an object storage service that offers industry-leading data scalability, availability, security, and performance for customers of all sizes and use cases. The Amazon S3 has a rich set of APIs to optimize costs, organize data, and configure granular access controls to meet different business, organizational and compliance requirements [106]. S3 compatible storage is a storage solution that has an S3 compliant interface and offers the same set of APIs to handle the data [107]. The S3 compatible storage is used in the delivery model to store the Terraform state file. The S3 compatible storage example usage [108] is shown in Figure 27.

```
jere@Jere-Koti:~/code_examples$ cat terraform.tf
terraform {
  required_providers {
    fortiadc = {
      source = "fortinetdev/fortiadc"
      version = "1.0.1"
    }
  }
  backend "s3" {
    bucket = "bucket"
    key = "terraform.tfstate"
    region = "us-east-1"
    endpoint "s3.example.com"
    skip_credentials_validation = true
  }
}
```

Figure 27 The S3 compatible storage as backend for Terraform state example usage [108].

5.1.3 GitHub Enterprise

GitHub Enterprise tool is used for code management to store the repository centrally and under version control. The GitHub repository is configured with dedicated self-hosted runner [109] hosted in the private cloud as a RHEL VM in network management zone. The self-hosted runner is used to run actions on pull requests and on merge pull requests. The actions are defined in workflows as YAML files. A simple GitHub Actions workflow [88] is shown in Figure 28.

```
jere@Jere-Koti:~/code_examples$ cat simple.yml
name: Simple

on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

  workflow_dispatch:

jobs:
  build:
    runs-on: [ self-hosted ]

    steps:
      - uses: actions/checkout@v3

      - name: Run a one-line script
        run: echo Simple workflow!
```

Figure 28 A simple GitHub Actions workflow [88].

The required environment variables are configured as encrypted repository secrets and referenced in the workflows [110]. The repository is accessible to the data center networks team and has main branch protection that requires approval before merging a pull request.

5.2 Proposed delivery model

Terraform FortiADC provider can be used to create and manage the same resources as in GUI and CLI of FortiADC such as VDOM, real servers, real server pools, SSL profiles, and virtual servers [104].

5.2.1 Terraform configuration

The VDOM configuration include one argument mkey which is the VDOM name. The VDOM configuration can be inherited from the provider configuration for resources or specifically set for each resource. The configuration of a VDOM in Terraform [104] is shown in Figure 29.

```
jere@Jere-Koti:~/code_examples$ cat vdom.tf
resource "fortiadc_vdom" "vdom_set" {
    mkey = "internal"
}
```

Figure 29 The configuration of a VDOM in Terraform [104].

Real server configuration includes arguments mkey, server_type, status, type, and address. The mkey is real server name. The real server mkey is referenced in the real server pool configuration and cannot be edited after the initial resource configuration. The server_type is to connect external real servers and has valid values 1:static, 2:dynamic_auto, and 3:dynamic_manual. The status valid values are 0:disable, 1:enable, and 2:maintain. The type either 0:ip or 1:fqdn are both supported. The address argument for IPv4 or address6 argument for IPv6 are supported. The configuration of a real server in Terraform [104] is shown in Figure 30.

```
jere@Jere-Koti:~/code_examples$ cat rs.tf
resource "fortiadc_load_balance_real_server" "rs1" {
  mkey        = "192.168.1.1"
  server_type = "static"
  status      = "enable"
  type        = "ip"
  address     = "192.168.1.1"
}
```

Figure 30 The configuration of a real server in Terraform [104].

Real server pool configuration includes mkey, pool_type, type, health check, and rs_profile. The mkey is real server pool name. The real server pool mkey is referenced in the virtual server configuration and cannot be edited after the initial resource configuration. The pool_type 1:ipv4 and 2:ipv6 are both supported. The type is to connect external real servers and has valid values 1:static and 2:dynamic. The health check arguments enable one or more health checks for the pool. The rs_profile is real server SSL profile name.

The real server pool member basic configuration includes mkey, status, real_server_id, port, weight, cookie, health check, and pkey. The mkey is numerical pool member ID. The status options are 0:disable, 1:enable, and 2:maintain. The real_server_id is the mkey of the real server. The port is the listening port from the range 0 – 65535 where the port 0 is a wildcard port. The weight option assigns a preference 1 – 256 where the higher values are preferred. The cookie is the name that is set when session persistence is enabled. Health check options allow to overwrite the health checks defined in the pool. The pkey is parent key or mkey of the real server pool. The configuration of a real server pool and member in Terraform [104] is shown in Figure 31.

```
jere@Jere-Koti:~/code_examples$ cat rsp.tf
resource "fortiadc_load_balance_pool" "rsp" {
    mkey = "rsp"
    type = "static"
    pool_type = "ipv4"
    health_check = "enable"
    health_check_relationship = "AND"
    health_check_list = "LB_HLTHCK_HTTPS"
}

resource "fortiadc_load_balance_pool_child_pool_member" "rsp_rs1" {
    mkey = "1"
    port = "443"
    real_server_id = "192.168.1.1"
    cookie = "rs1"
    pkey = "rsp"
    depends_on = [fortiadc_load_balance_pool.rsp, fortiadc_load_balance_real_server.rs1]
}
```

Figure 31 The configuration of a real server pool and member in Terraform [104].

The depends_on meta-argument is used to explicitly specify dependencies between resources such as between real servers and real server pools and real server pools and virtual servers to instruct Terraform to complete all actions on the dependent resources before performing actions on the resource declaring the dependency [111].

Client SSL profile basic configuration include mkey, SSL ciphers, ssl_allowed_versions local_certificate_group, and ssl_dh_param_size. The mkey is client SSL profile name. SSL

cipher arguments enable predefined list or customized set of ciphers. The ssl_allowed_versions valid values are 2:sslv3, 4:tlsv1.0, 8:tlsv1:1, 16:tlsv1.2, and 32:tlsv1.3. The local_certificate_group is the name of the local certificate group that includes the certificate from a CA. The ssl_dh_param_size options are 1024:1024bit, 2048:2048bit, and 4096:4096bit. The configuration of a client SSL profile in Terraform [104] is shown in Figure 32.

```
jere@Jere-Koti:~/code_examples$ cat client_ssl_prof.tf
resource "fortiadc_load_balance_client_ssl_profile" "ssl_prof" {
    mkey = "ssl_prof"
    customized_ssl_ciphers = "enable"
    ssl_allowed_versions = "tlsv1.1 tlsv1.2"
    local_certificate_group = "lb.example.com"
    ssl_dh_param_size = "2048"
}
```

Figure 32 The configuration of a client SSL profile in Terraform [104].

Real server SSL profile basic configuration include similar options: mkey, SSL ciphers, allow_ssl_versions, and local_cert [104].

Virtual server basic configuration includes mkey, type, status, addr_type, address, port, interface, profile, client_ssl_profile, persistence, method, and pool. The mkey is virtual server name and cannot be edited after the initial resource configuration. The type has valid values 1:l4-load-balance, 2:l7-load-balance, and 3:l2-load-balance. The status options are 0:disable, 1:enable, and 2:maintain similar to real server and real server pool resource configuration. The addr_type 1:ipv4 and 2:ipv6 are both supported. The arguments address, port, and interface describe the virtual server IP address, listening port and interface name the client traffic is received. The profile defines how the virtual server should handle the traffic and predefined profiles and custom profiles can be used. The client_ssl_profile is the client SSL profile name. The persistence enables virtual server persistence. The method and pool arguments select the load balancing method for the real server pool. The configuration of a virtual server in Terraform [104] is shown in Figure 33.

```
jere@Jere-Koti:~/code_examples$ cat vs.tf
resource "fortiadc_load_balance_virtual_server" "vs" {
    mkey = "vs"
    pool = "rsp"
    interface = "port"
    port = "443"
    address = "192.168.0.1"
    type = "l7-load-balance"
    profile = "LB_PROF_HTTPS"
    client_ssl_profile = "ssl_prof"
    persistence = "LB_PERSIS_SRC_ADDR"
    method = "LB_METHOD_ROUND_ROBIN"
    depends_on = [fortiadc_load_balance_pool.rsp, fortiadc_load_balance_client_ssl_profile.ssl_prof]
}
```

Figure 33 The configuration of a virtual server in Terraform [104].

5.2.2 Module design and implementation

The proposed load balancer service delivery model consists of multiple Terraform files to set up the provider, backend, and resources and YAML files to set up the workflows. The root module [112] includes the provider and backend configuration and calls the child module with all the required input variables [113] that is the full load balancer instance configuration managed via Terraform. The child module is configured with for_each meta-arguments [114] to flexibly add, change, and destroy resources based on the input variables without making changes in the child module. The child module allows to package and reuse the configuration [112] for all load balancer instances that is setup in the root module or instance directory. The load balancer service delivery model file structure and Terraform module design are shown in Figure 34.

```
terraform-fortiadc-module
        README.md
        .gitignore
        .github/workflows
                terraform_plan.yml
                terraform_apply.yml
        example
                main.tf
                providers.tf
                terraform.tf
                variables_providers.tf
        instance1
        instance2
        ...
        instanceN
        main.tf
        terraform.tf
        variables.tf
```

```
terraform-fortiadc-module/example/main.tf
module "fortiadc" {
   source = "../"

   fortiadc_x = {
      x1 = {att1 = "x", att2 = "x", ... , attN = "x"}
      x2 = {att1 = "x", att2 = "x", ... , attN = "x"}
      ...
      xN = {att1 = "x", att2 = "x", ... , attN = "x"}
   }
}

terraform-fortiadc-module/main.tf
resource "fortiadc_x" "x" {
   for_each = var.fortiadc_x
   att1 = each.value.att1
   att2 = each.value.att2
   ...
   attN = each.value.attN
}
```

Figure 34 The load balancer service delivery model file structure and Terraform module design.

The load balancer instance configuration changes are only made in specific instance directory main.tf code file which has a configuration section for each resource such as real servers, real server pools, client SSL profiles, and virtual servers. Variables with default values are used where suitable to simplify and increase consistency of the configuration defined in variables.tf code files. The full load balancer instance configuration managed via Terraform is shown in Figure 35.

Figure 35 The full load balancer instance configuration managed via Terraform.

Configuration changes are planned and applied via pull requests and merge pull requests in GitHub Enterprise that trigger actions workflows Terraform plan and Terraform apply respectively on the self-hosted runner. The GitHub Actions workflows Terraform plan and Terraform apply are shown in Figure 36.

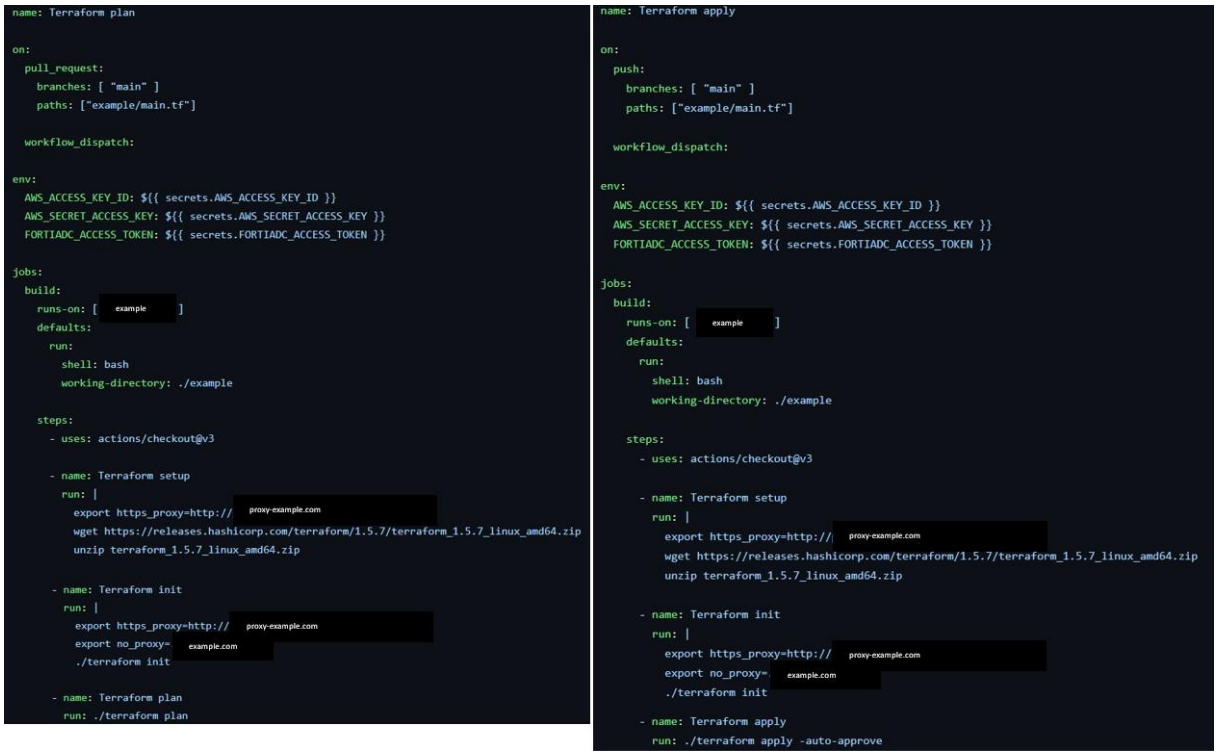


Figure 36 GitHub Actions workflows Terraform plan and Terraform apply.

The load balancer instance is configured by adding, modifying, and removing code lines in the specific instance directory main.tf file. The changes can be done in the personal work computer CLI or GUI by cloning the repository or accessing GitHub Enterprise via web browser. The addition, modification, and removal of a real server configuration and example terraform plan generated in GitHub Actions are shown in the Figure 37.

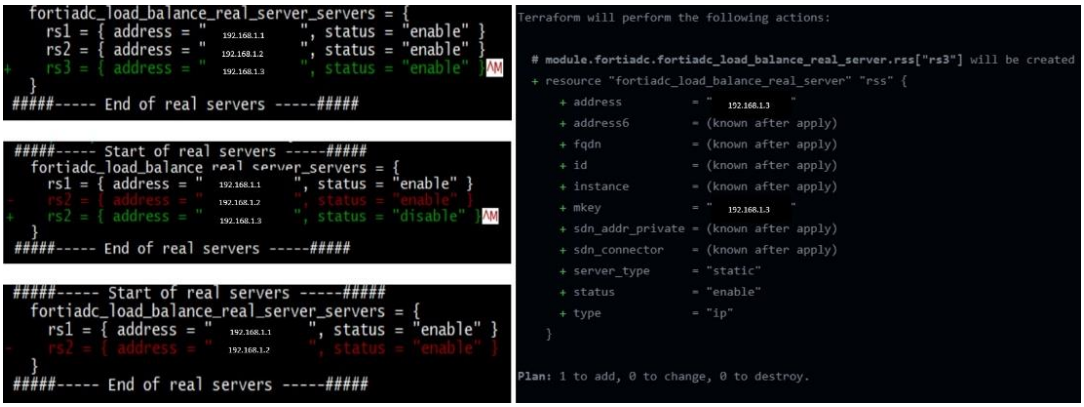


Figure 37 Real server configuration and example terraform plan generated in GitHub Actions.

5.3 Future development directions

The proposed load balancer service delivery model is designed to fulfil all the requirements included in the standard load balancer service request form such as provision virtual server including TLS, adjust virtual server configuration, and decommission virtual server. The future code development directions would be to add more load balancer resources to be managed via Terraform such as custom application profiles and persistence as well as content routing and certificate information possibly with Automatic Certificate Management Environment (ACME) [115] implementation in cooperation with CA.

The future design development directions include possible API gateway configuration to create policies that accept only the required API calls for the delivery model instead of permitting full access directly to the load balancer instance management [102]. Furthermore, change from using the root credentials for S3 in the delivery model and instead create policies, users, and groups with least privileges for different operations and actions [116] and encryption of the state data. Setup Key Management Service (KMS) on-premises solution for all the required credentials in the delivery model.

The proposed load balancer service delivery model is designed to be used by the data center networks team by making changes in the code repository and creating pull requests and merge pull requests in GitHub Enterprise approved by a colleague. The repository could be published to stakeholders for them to create pull requests and have the data center networks team approve the requests for merge. Furthermore, the repository could be integrated with the service request system in use.

6 Case based testing

6.1 Testing procedure

The standard load balancer service delivery includes provision, adjustment, and decommission of the service or virtual server and has various options such as persistence, method, and TLS. For case based testing provision virtual server with TLS, adjust virtual server, and decommission virtual server were selected as the cases for testing and comparing the current and proposed load balancer service delivery models. The comparison is done by analysing the steps required to complete the delivery in each model which gives an indication about time of delivery and probability for configuration errors.

6.2 Provision virtual server with TLS

The steps to provision a virtual server with TLS include configuration of real servers, real server pool, certificate, SSL profile, and virtual server. The creation of certificate from CA and importing it to the load balancer is excluded from testing as the process remains same for both delivery models. Table 7 presents the provision virtual server with TLS service request case.

Table 7 Provision virtual server with TLS service request case.

Request type	Provision
DNS name	example.com (192.168.1.1)
DNS record	Yes
Virtual server port	443
List of real server IP addresses and ports	192.168.1.1:443, 192.168.1.2:443
Persistence	Source address
Method	Round robin
TLS	Yes

The initial step in both delivery models is to locate the load balancer instance and VDOM which is deduced from the IP address blocks used in the service request. In the current delivery model the data center networks team member would connect to the jump host in network management zone and load balancer instance management and start to configure the

real servers, real server pool, SSL profile, and virtual server in the GUI or CLI depending on the preference. In the proposed delivery model the data center networks team member would connect to GitHub Enterprise and navigate to the repository and instance directory and start to add code lines for the real servers, real server pool, SSL profile, and virtual server and create pull request and merge pull request which trigger the actions that plan and apply the changes accordingly. The delivery flowcharts for provision a virtual server with TLS for both delivery models are shown in Figure 38.

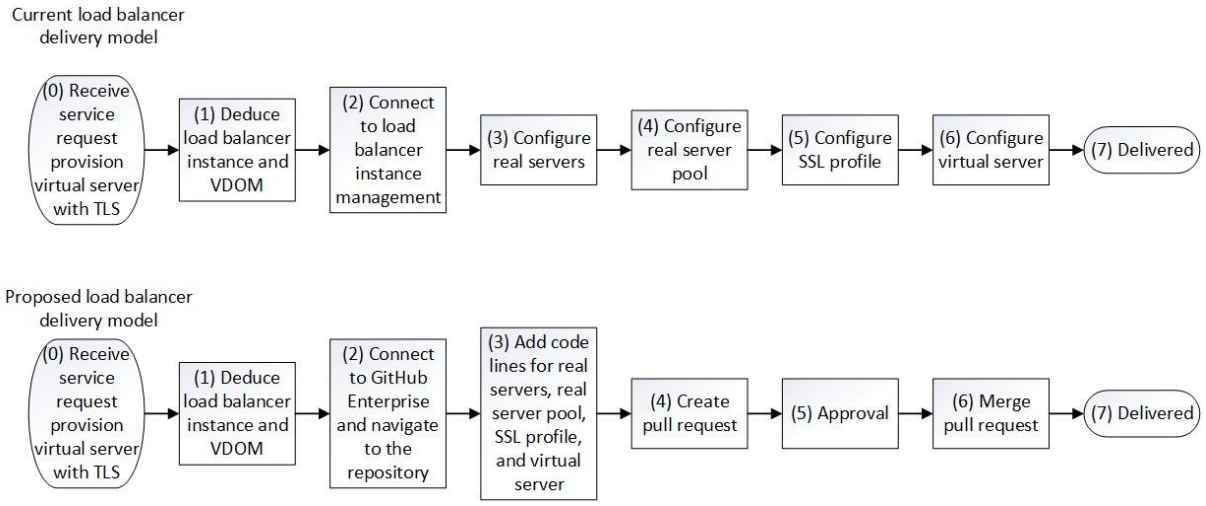


Figure 38 The delivery flowcharts for provision a virtual server with TLS for both delivery models.

In the current delivery model, the steps to (3) configure real servers (Figure 21), (4) real server pool (Figure 22), (5) SSL profile (Figure 23), and (6) virtual server (Figure 24) are individual steps and configured in different sections of the GUI and CLI. In the proposed delivery model, the configuration steps are combined (3), and additional steps required are create pull request (4), approval (5), and merge pull request (6). The provision of virtual server with TLS configuration in the proposed delivery model is shown in Figure 39.

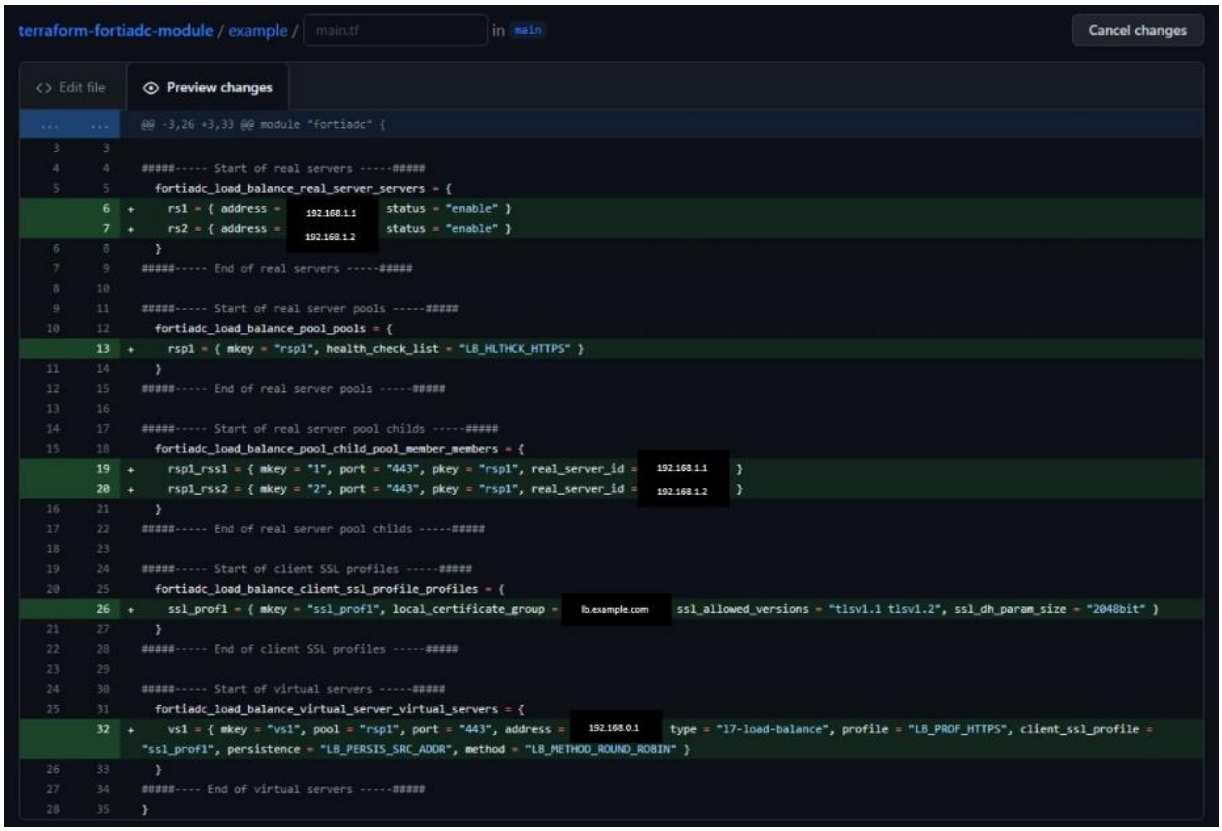


Figure 39 The provision of virtual server with TLS configuration in the proposed delivery model.

6.3 Adjust virtual server

The steps to adjust a virtual server can include reconfiguration in any of the steps in provision of a virtual server. Table 8 presents the adjust virtual server service request case.

Table 8 Adjust virtual server service request case.

Request type	Adjust
IP address	192.168.0.1
Port	443
DNS	example.com
Request	Change virtual server persistence from source address to cookie

In the current delivery model, the data center networks team member would connect to the load balancer instance management following the same steps as in provision a virtual server and start to reconfigure the virtual server. In the proposed delivery model the data center

networks team member would connect to GitHub Enterprise and navigate to the repository and instance directory and reconfigure the virtual server and plan and apply changes following the same process as in provision a virtual server. The delivery flowcharts for adjust virtual server for both delivery models are shown in Figure 40.

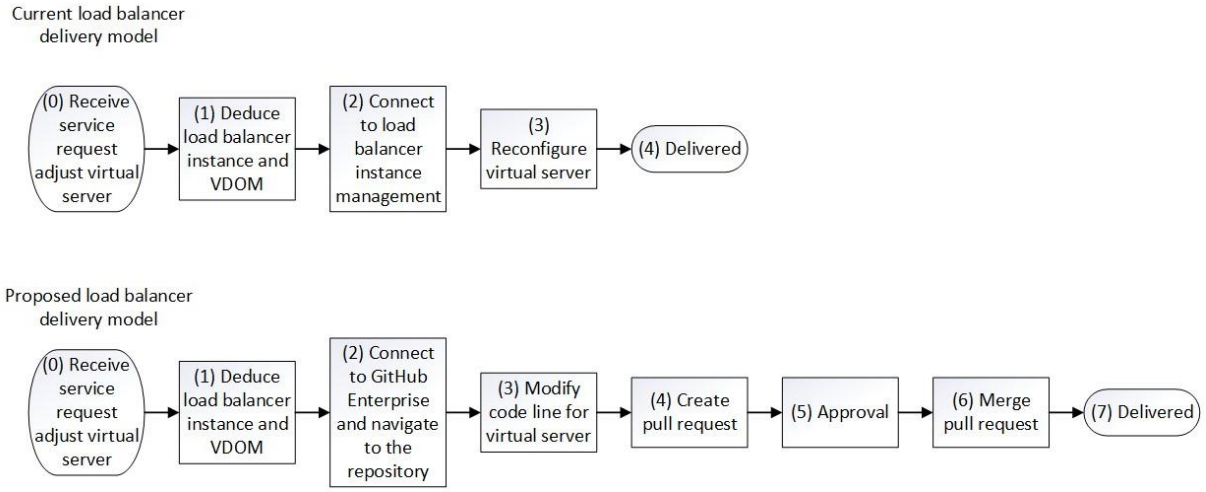


Figure 40 The delivery flowcharts for adjust virtual server for both delivery models.

In the current delivery model, there is one step to reconfigure the (3) virtual server (Figure 24). In the proposed delivery model, there is also one configuration step (3), and similarly additional steps are create pull request (4), approval (5), and merge pull request (6). The adjustment of virtual server configuration in the proposed delivery model is shown in Figure 41.

Figure 41 The adjustment of virtual server configuration in the proposed delivery model.

6.4 Decommission virtual server

The steps to decommission a virtual server include the removal of the configuration added in provision of a virtual server in reverse direction. Table 9 presents the decommission of virtual server service request case.

Table 9 Decommission of virtual server service request case.

Request type	Decommission
IP address	192.168.0.1
Port	443
DNS	example.com

In the current delivery model, the data center networks team member would connect to the load balancer instance management following the same steps as in provision and adjust a virtual server and start to remove the virtual server, SSL profile, real server pool, and real servers. In the proposed delivery model the data center networks team member would connect to GitHub Enterprise and navigate to the repository and instance directory and start to remove code lines for the real servers, real server pool, SSL profile, and virtual server and plan and apply changes following the same process as in provision and adjust a virtual server. The delivery flowcharts for decommission a virtual server for both delivery models are shown in Figure 42.

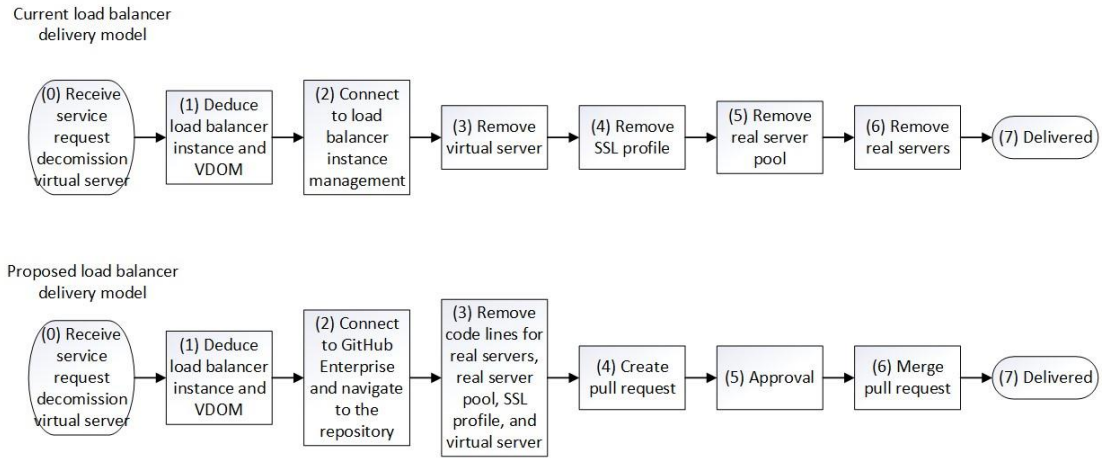


Figure 42 The delivery flowcharts for decommission a virtual server for both delivery models.

In the current delivery model, the steps to remove (3) virtual server (Figure 24), (4) SSL profile (Figure 23), (5) real server pool (Figure 22), and (6) real servers (Figure 21) are individual steps and configured in different sections of the GUI and CLI. In the proposed delivery model, the configuration removal steps are combined (3), and similarly additional steps are create pull request (4), approval (5), and merge pull request (6). The decommission of virtual server configuration in the proposed delivery model is shown in Figure 43.

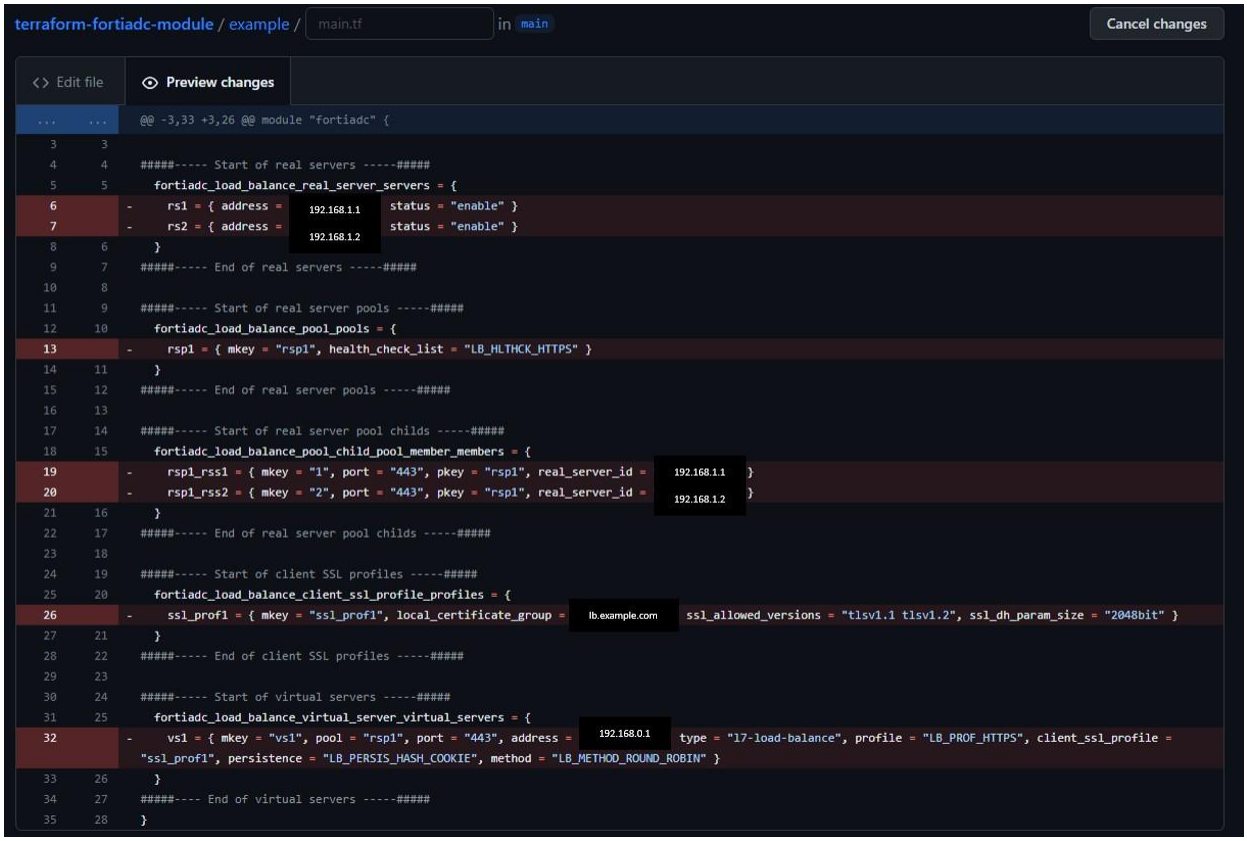


Figure 43 The decommission of virtual server configuration in the proposed delivery model.

6.5 Results and comparison

The proposed IaC based load balancer service delivery model with Terraform can be used to successfully deliver all the standard load balancer service request cases that include provision, adjustment, and decommission of the service with required options which results the proposed delivery model a valid candidate to be used as the next load balancer service delivery model in the data center networks team.

The comparison of the delivery models shows the proposed delivery model reduced time of delivery that is most clearly seen in the service request cases provision virtual server and decommission virtual server and when there are many virtual servers to be added or removed in a single request. The comparison of the delivery models time of delivery includes only the configuration part of the delivery. The comparison of time of delivery to provision and decommission number of virtual servers in the current and proposed delivery is shown in Figure 44.

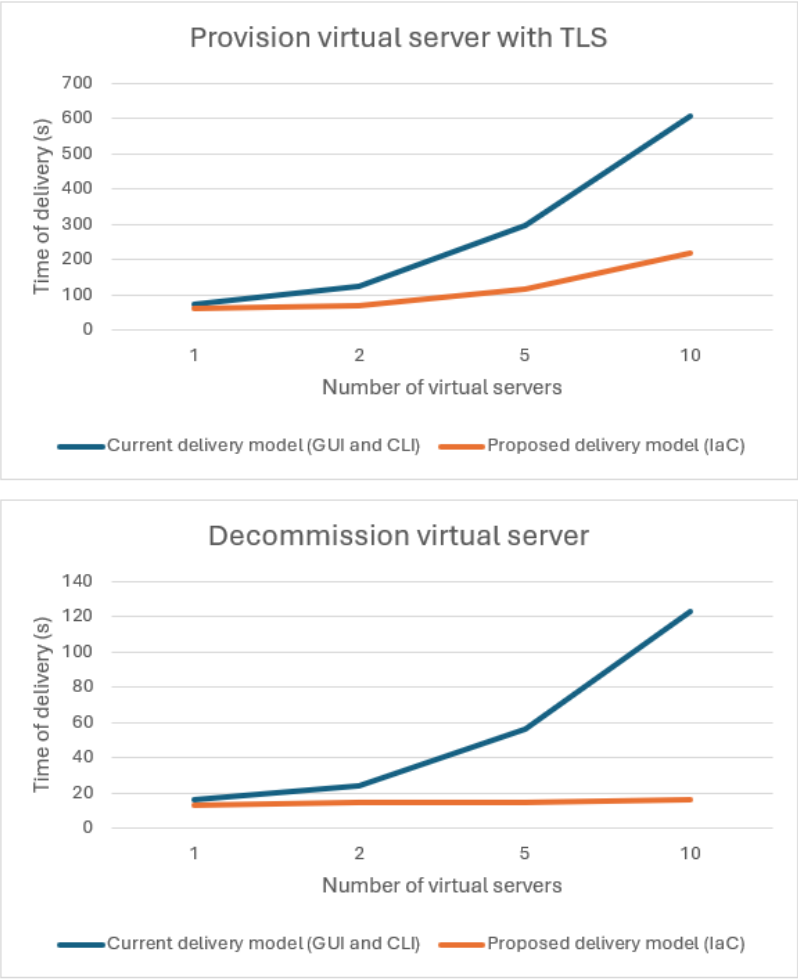


Figure 44 Time of delivery to provision and decommission number of virtual servers with TLS.

The time of delivery is reduced as in the current delivery model the configuration of real servers, real server pool, SSL profile, and virtual server are done in different sections of the vendor specific GUI or CLI in specific order whereas in the proposed delivery model addition or removal of the resources are done by “copy paste” or “cut” in single code file and the

detailed steps to reach the desired state is handled by Terraform. Furthermore, the proposed delivery model uses variables with default values where suitable to simplify and increase the configuration consistency which further reduces the time of delivery and minimizes configuration errors. Additional benefits of the proposed delivery model include feedback from the Terraform commands that is not available in the current delivery model. The proposed delivery model requires additional steps to create pull request, approval, and merge pull request which can introduce slight delay to time of delivery in service request cases where a single resource attribute is reconfigured such as adjust virtual server persistence.

6.6 Discussion

The proposed delivery model Terraform module can be reused with any FortiADC load balancer instance by configuring the provider. Furthermore, the same Terraform module design and way of working can be used with any load balancer vendor that has Terraform provider available. This is made possible with the abstraction layer that Terraform provides and enables network engineers to work with any vendor without the need to learn the specific vendor interface details but rather general load balancer service. In other words, interface and configuration differences between vendors can be abstracted from the Terraform module users' view.

The current delivery model to proposed delivery model migration should happen in stages to gain maturity in the data center networks team such as first migrate small existing and new load balancer instance deployments. The small existing load balancer instances can be migrated to be managed via Terraform with import functionality [117]. The new load balancer instance deployments can be managed via Terraform from the initial deployment. Create and test similar Terraform module with other load balancer vendors.

The proposed delivery model uses software engineering tools and ways of working such as Terraform and GitHub to deliver load balancer services by making changes in code. The same delivery model is expected to expand to also cover routing and switching services in the industry and the data center networks team. The transition from traditional way of working that relies heavily on the vendor specific interfaces GUI and CLI configurations to software configurations in code will require that network engineers are comfortable working with software.

7 Conclusion

The thesis proposed IaC based load balancer service delivery model with Terraform for the data center networks team to reduce the time of delivery and minimize configuration errors with automation compared to the current load balancer service delivery model that relies heavily on the usage of vendor specific interfaces. Terraform is supported with various on-premises tools to complete the delivery model which can be shared with other IaC solutions.

The proposed delivery model was evaluated with case-based testing, and it was concluded that the proposed delivery model can be used to successfully deliver all the standard load balancer service request cases that include provision, adjustment, and decommission of the service with required options and therefore the proposed delivery model fulfils all the requirements of the standard load balancer service. The case-based testing showed that the proposed delivery model benefits are most clearly seen in the service request cases provision and decommission of virtual server and when there are multiple virtual servers to be added or removed in a single service request. Additional benefits include increased configuration consistency and feedback. The additional steps in GitHub part of the proposed delivery model can introduce slight delay to time of delivery in service request cases where a single resource attribute is changed such as adjust virtual server persistence. The proposed delivery model Terraform module can be reused with any FortiADC load balancer instance and the same design and way of working can be used with any vendor that has Terraform provider. Thus, the proposed delivery model offers the benefits of network automation [103] and addresses the studied data center network automation challenges [95][96]. The thesis answers the research questions as follows:

- RQ1: What design choices and tools are required for on-premises IaC based solution?
    - On-premises IaC based solution requires special design choices as access from internet towards on-premises management is typically restricted and not allowed by security policy. Therefore on-premises tools are required for the solution instead of public cloud tools.
- RQ2: Does the proposed delivery model based on IaC fulfil the requirements of the standard load balancer service?

    - The proposed delivery model based on IaC fulfils all the requirements of the standard load balancer service: provision, adjustment, and decommission of the service with required options.

- RQ3: Does the proposed delivery model based on IaC offer the benefits of automation?

    - The proposed delivery model based on IaC offers the benefits of automation such as reduced time of delivery, configuration consistency, and reusability.

The proposed load balancer service delivery model future code development directions are to add more load balancer resources to be managed via Terraform and ACME implementation in cooperation with CA to complete the automation for virtual servers with TLS. The future design development directions include API gateway configuration to accept only the required API calls, S3 policies, users, and groups and encryption of the state data, and KMS solution for all the required credentials. Furthermore, publish the repository for stakeholders and integration with service request system.

The thesis proposed a plan to migrate from the current delivery model to the proposed delivery model in stages to gain maturity in the team by first migrating small existing load balancer instances with Terraform import functionality and start managing new load balancer instance deployments via Terraform from the initial deployment. Create and test similar Terraform module with other load balancer vendors.

References

[1] S. Maloo, F. Ahmed. CCNP and CCIE Data Center Core DCCOR 350-601 Official Cert Guide. San Francisco, CA, USA: Cisco Press, 2020.

[2] Amazon Web Services, “What is a Data Center?”. [Online]. Available: https://aws.amazon.com/what-is/data-center/. Accessed 7.1.2024.

[3] VMware, “What is a hypervisor?”. [Online]. Available: https://www.vmware.com/topics/glossary/content/hypervisor.html.html. Accessed 7.1.2024.

[4] Amazon Web Services, “What is NAS (Network-Attached Storage)?”. [Online]. Available: https://aws.amazon.com/what-is/nas/. Accessed 7.1.2024.

[5] P. Mockapetris. Domain names – Concepts and Facilities. *RFC 1034*, November 1987.

[6] P. Mockapetris. Domain names – Implementation and Specification. *RFC 1035*, November 1987.

[7] D. Mills, J. Martin, J. Burbank, W. Kasch. Network Time Protocol Version 4: Protocol and Algorithms Specification. *RFC 5905*, June 2010.

[8] R. Droms. Dynamic Host Configuration Protocol. *RFC 2131*, March 1997.

[9] T. Mrugalski, M. Siodelski, B. Volz, A. Yourtchenko, M. Richardson, S. Jiang, T. Lemon, T. Winters. Dynamic Host Configuration Protocol for IPv6 (DHCPv6). *RFC 8415*, November 2018.

[10] J. Angara, “Introduction to International Data Center Standards”, 2022. [Online]. Available: https://www.akcp.com/blog/introduction-to-international-data-center-standards/. Accessed 8.1.2024.

[11] Telecommunications Industry Association, “ANSI/TIA-942 STANDARD”. [Online]. Available: https://tiaonline.org/products-and-services/tia942certification/ansi-tia-942-standard/. Accessed 8.1.2024.

[12] European Standard, “European Standards”. [Online]. Available: https://www.en-standard.eu/bs-en-50600-1-2019-information-technology-data-centre-facilities-and-infrastructures-general-concepts/. Accessed 8.1.2024.

[13] Uptime Institute, “Tier Classification System”. [Online]. Available: https://uptimeinstitute.com/tiers. Accessed 8.1.2024.

[14] R. Graziani, A. Johnson. Introduction to Networks v6 Companion Guide. San Francisco, CA, USA: Cisco Press, 2016.

[15] SGRwin, “Navigating the Complex World of Small Form Factor Pluggable Management”, 2023. [Online]. Available: https://www.sgrwin.com/small-form-factor-pluggable-management/. Accessed 10.1.2024.

[16] Cisco, “Cisco FabricPath”. [Online]. Available: https://www.cisco.com/c/dam/en/us/products/collateral/switches/nexus-7000-series-switches/at_a_glance_c45-605626.pdf. Accessed 10.1.2024.

[17] J. Moy. OSPF Version 2. *RFC 2328*, April 1998.

[18] R. Coltun, D. Ferguson, J. Moy, A. Lindem. OSPF for IPv6. *RFC 5340*, July 2008.

[19] M. Shand, L. Ginsberg. Reclassification of RFC 1142 to Historic. *RFC 7142*, February 2014.

[20] D. Savage, J. Ng, S. Moore, D. Slice, P. Paluch, R. White. Cisco's Enhanced Interior Gateway Routing Protocol (EIGRP). *RFC 7868*, May 2016.

[21] Y. Rekhter, T. Li, S. Hares. A Border Gateway Protocol 4 (BGP-4). *RFC 4271*, January 2006.

[22] T. Bates, R. Chandra, D. Katz, Y. Rekhter. Multiprotocol Extensions for BGP-4. *RFC 2283*, February 1998.

[23] W. Eddy. Transmission Control Protocol (TCP). *RFC 9293*, August 2022.

[24] J. Postel. User Datagram Protocol. *RFC 768*, August 1980.

[25] R. Fielding, J. Reschke. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. *RFC 7231*, June 2014.

[26] S. Gold, “The future of the firewall”, Network Security, Vol. 2011, No. 2, pp. 13 – 15, 2011.

[27] K. Neupane, R. Haddad and L. Chen, “Next Generation Firewall for Network Security: A Survey”, In Proceedings of IEEE Southeastern Conference, pp. 1 – 6, 2018.

[28] Amazon Web Services, “What is an Application Load Balancer?”. [Online]. Available: https://docs.aws.amazon.com/elasticloadbalancing/latest/application/introduction.html. Accessed 10.1.2024.

[29] PingPlotter, “A Broadcast Storm Defined & How to Fix It”. [Online]. Available: https://www.pingplotter.com/wisdom/article/weathering-a-broadcast-storm/. Accessed 12.1.2024.

[30] R. Froom, E. Frahim. Implementing Cisco IP Switched Networks (SWITCH) Foundation Learning Guide. San Francisco, CA, USA: Cisco Press, 2015.

[31] D. Teare, B. Vachon, R. Graziani. Implementing Cisco IP Routing (ROUTE) Foundation Learning Guide. San Francisco, CA, USA: Cisco Press, 2015.

[32] K. Scarfone, P. Hoffman, “Guidelines on Firewalls and Firewall Policy”, 2009. [Online]. Available: https://www.nist.gov/publications/guidelines-firewalls-and-firewall-policy. Accessed 20.1.2024.

[33] Microsoft, “Windows Firewall overview”, 2023. [Online]. Available: https://learn.microsoft.com/en-us/windows/security/operating-system-security/network-security/windows-firewall/. Accessed 20.1.2024.

[34] Die, “iptables(8) - Linux man page”. [Online]. Available: https://linux.die.net/man/8/iptables. Accessed 20.1.2024.

[35] F5, “What Is a Load Balancer?”. [Online]. Available: https://www.f5.com/glossary/load-balancer. Accessed 24.1.2024.

[36] T. Ylonen, C. Lonvick. The Secure Shell (SSH) Transport Layer Protocol. *RFC 4253*, January 2006.

[37] A. Barth. HTTP State Management Mechanism. *RFC 6265*, April 2011.

[38] T. Berners-Lee, R. Fielding, L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. *RFC 3986*, January 2005.

[39] F5, “About Virtual Servers”. [Online]. Available: https://techdocs.f5.com/kb/en-us/products/big-ip_ltm/manuals/product/ltm-basics-11-6-0/2.html. Accessed 24.1.2024.

[40] F5, “Introduction to iRules”. [Online]. Available: https://techdocs.f5.com/kb/en-us/products/big-ip_ltm/manuals/product/bigip-system-irules-concepts-11-6-0/1.html. Accessed 24.1.2024.

[41] Fortinet, “Using HTTP scripting”. [Online]. Available: https://docs.fortinet.com/document/fortiadc/7.4.1/handbook/643840/using-https-scripting. Accessed 24.1.2024.

[42] Fortinet, “Configuring content routes”. [Online]. Available: https://help.fortinet.com/fadc/4-5-1/olh/Content/FortiADC/handbook/content_routing.htm. Accessed 24.1.2024.

[43] F5, “Session Persistence Profiles”. [Online]. Available: https://techdocs.f5.com/kb/en-us/products/big-ip_ltm/manuals/product/ltm-profiles-reference-12-1-0/4.html. Accessed 24.1.2024.

[44] F5, “K42275060: There are several Load Balancing Methods. Which one is best for your environment?”, 2023. [Online]. Available: https://my.f5.com/manage/s/article/K42275060. Accessed 24.1.2024.

[45] F5, “Health and Performance Monitoring”. [Online]. Available: https://techdocs.f5.com/kb/en-us/products/big-ip_ltm/manuals/product/ltm-concepts-11-5-1/15.html. Accessed 27.1.2024.

[46] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. *RFC 8446*, August 2018.

[47] Amazon Web Services, “What is an SSL/TLS Certificate?”. [Online]. Available: https://aws.amazon.com/what-is/ssl-certificate/. Accessed 27.1.2024.

[48] Amazon Web Services, “What’s the Difference Between SSL and TLS?”. [Online]. Available: https://aws.amazon.com/compare/the-difference-between-ssl-and-tls/. Accessed 27.1.2024.

[49] F5, “K65271370: Most Common SSL Methods for LTM: SSL Offload, SSL Pass-Through and Full SSL Proxy”, 2023. [Online]. Available: https://my.f5.com/manage/s/article/K65271370. Accessed 27.1.2024.

[50] Kaspersky, “What is a VPN and how does it work?”. [Online]. Available: https://www.kaspersky.fi/resource-center/definitions/what-is-a-vpn. Accessed 29.1.2024.

[51] Dell, “Integrated Dell Remote Access Controller (iDRAC)”. [Online]. Available: https://www.dell.com/en-us/lp/dt/open-manage-idrac. Accessed 29.1.2024.

[52] Hewlett Packard Enterprise, “HPE Integrated Lights-Out (iLO)”. [Online]. Available: https://www.hpe.com/fi/en/hpe-integrated-lights-out-ilo.html. Accessed 29.1.2024.

[53] Cisco, “What Is Network Security?”. [Online]. Available: https://www.cisco.com/c/en/us/products/security/what-is-network-security.html. Accessed 30.1.2024.

[54] Fortinet, “What is a DMZ Network?”. [Online]. Available: https://www.fortinet.com/resources/cyberglossary/what-is-dmz. Accessed 30.1.2024.

[55] Check Point, “What is Network Security?”. [Online]. Available: https://www.checkpoint.com/cyber-hub/network-security/what-is-network-security/. Accessed 30.1.2024.

[56] J. Kindervag, “Build Security Into Your Network’s DNA: The Zero Trust Network Architecture”, 2010. [Online]. Available: https://www.virtualstarmedia.com/downloads/Forrester_zero_trust_DNA.pdf.

[57] S. HomChaudhuri, M. Foschiano. Cisco Systems' Private VLANs: Scalable Security in a Multi-Client Environment. *RFC 5517*, February 2010.

[58] NWN Carousel, “Zero-trust architecture and moving beyond VLANs for segmentation”. [Online]. Available: https://nwncarousel.com/blog/zero-trust-architecture-and-moving-beyond-vlans-for-segmentation/. Accessed 31.1.2024.

[59] VMware, “VMware NSX Distributed Firewall”. [Online]. Available: https://www.vmware.com/products/nsx-distributed-firewall.html. Accessed 31.1.2024.

[60] Python, “Python”. [Online]. Available: https://www.python.org/. Accessed 2.2.2024.

[61] HashiCorp, “Terraform Language Documentation”. [Online]. Available: https://developer.hashicorp.com/terraform/language. Accessed 2.2.2024.

[62] Amazon Web Services, “What is a CLI? (Command Line Interface)”. [Online]. Available: https://aws.amazon.com/what-is/cli/. Accessed 3.2.2024.

[63] Amazon Web Services, “What is an API (Application Programming Interface)?”. [Online]. Available: https://aws.amazon.com/what-is/api/. Accessed 3.2.2024.

[64] Netcode, “CLI vs API network Automation”, 2022. [Online]. Available: https://netcode.rocks/blog/programmability-stack/. Accessed 3.2.2024.

[65] GitHub, “Netmiko”. [Online]. Available: https://github.com/ktbyers/netmiko. Accessed 3.2.2024.

[66] GitHub, “Netmiko Examples”. [Online]. Available: https://github.com/ktbyers/netmiko/blob/develop/EXAMPLES.md. Accessed 3.2.2024.

[67] Mozilla Developer Network, “Regular expressions”. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions. Accessed 3.2.2024.

[68] JavaScript Object Notation, “Introducing JSON”. [Online]. Available: https://www.json.org/json-en.html. Accessed 5.2.2024.

[69] Amazon Web Services, “What is XML?”. [Online]. Available: https://aws.amazon.com/what-is/xml/. Accessed 5.2.2024.

[70] R. Fielding, “Architectural Styles and the Design of Network-based Software Architectures”. Irvine, CA, USA: Doctoral dissertation, University of California, 2000.

[71] Cisco, “NX-API REST”. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus9000/sw/7-x/programmability/guide/b_Cisco_Nexus_9000_Series_NX-

OS_Programmability_Guide_7x/b_Cisco_Nexus_9000_Series_NX-OS_Programmability_Guide_7x_chapter_010001.pdf. Accessed 10.2.2024.

[72] GitHub, “NX-OS code”. [Online]. Available: https://github.com/CiscoDevNet/nxos-code. Accessed 10.2.2024.

[73] Cisco, “Nexus Model Reference”. [Online]. Available: https://pubhub.devnetcloud.com/media/nx-api-dme-model-9-3-1-reference/docs/. Accessed 10.2.2024.

[74] Amazon Web Services, “What is Infrastructure as Code?”. [Online]. Available: https://aws.amazon.com/what-is/iac/. Accessed 10.2.2024.

[75] HashiCorp, “Automate infrastructure on any cloud with Terraform”. [Online]. Available: https://www.terraform.io/. Accessed 10.2.2024.

[76] Ansible, “What is Ansible?”. [Online]. Available: https://www.ansible.com/. Accessed 10.2.2024.

[77] Puppet, “Infrastructure Automation & Compliance at Enterprise Scale”. [Online]. Available: https://www.puppet.com/. Accessed 10.2.2024.

[78] Amazon Web Services, “AWS CloudFormation”. [Online]. Available: https://aws.amazon.com/cloudformation/. Accessed 10.2.2024.

[79] Microsoft, “What is Azure Resource Manager?”, 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/overview. Accessed 10.2.2024.

[80] HashiCorp, “What is Terraform?”. [Online]. Available: https://developer.hashicorp.com/terraform/intro. Accessed 12.2.2024.

[81] HashiCorp, “Terraform Registry”. [Online]. Available: https://registry.terraform.io/. Accessed 12.2.2024.

[82] HashiCorp, “State”. [Online]. Available: https://developer.hashicorp.com/terraform/language/state. Accessed 12.2.2024.

[83] HashiCorp, “ACI”. [Online]. Available: https://registry.terraform.io/providers/CiscoDevNet/aci/latest. Accessed 16.2.2024.

[84] GitHub, “About GitHub and Git”. [Online]. Available: https://docs.github.com/en/get-started/start-your-journey/about-github-and-git. Accessed 18.2.2024.

[85] GitHub, “Let’s build from here”. [Online]. Available: https://github.com/. Accessed 18.2.2024.

[86] GitLab, “Software. Faster.”. [Online]. Available: https://about.gitlab.com/. Accessed 18.2.2024.

[87] Git, “Distributed is the new centralized”. [Online]. Available: https://git-scm.com/. Accessed 18.2.2024.

[88] GitHub, “Understanding GitHub Actions”. [Online]. Available: https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions. Accessed 18.2.2024.

[89] GitHub, “The AI-powered developer platform.”. [Online]. Available: https://github.com/enterprise. Accessed 18.2.2024.

[90] GitLab, “GitLab for Enterprises”. [Online]. Available: https://about.gitlab.com/enterprise/. Accessed 18.2.2024.

[91] Telia, “DC NW Platform – General Information”.

[92] SAFe, “SAFe 6.0”. [Online]. Available https://scaledagileframework.com/. Accessed 22.2.2024.

[93] M. Artac, T. Borovssak, E. Di Nitto, M. Guerriero and D. A. Tamburri, "DevOps: Introducing Infrastructure-as-Code”, IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), Buenos Aires, Argentina: pp. 497-498, 2017.

[94] S. Lee, T. Wong, H. Kim, "To Automate or Not to Automate: On the Complexity of Network Configuration”, IEEE International Conference on Communications, Beijing, China: pp. 5726-5731, 2008.

[95] McGillicuddy S, “Data-center network automation: Its pitfalls and how to avoid them: Automation can streamline data-center networking, but it presents challenges that must be overcome when planning, implementing, and using it”, 2022. [Online]. Available: https://www.proquest.com/trade-journals/data-center-network-automation-pitfalls-how-avoid/docview/2631947148/se-2.

[96] F. Bruschetti, J. Guevara, M. Abeledo, D. Priano, “An Empirical Evaluation of Automated Configuration Tools for Software-Defined Networking: A Usability and Performance Perspective”, Ingénierie des Systèmes d’Information, Vol. 28, No. 5, pp. 1127-1134, 2023.

[97] A. Rahman, R. Mahdavi-Hezaveh, L. Williams, ”A systematic mapping study of infrastructure as code research”, Information and Software Technology, Vol. 108, pp. 65-77, 2019.

[98] Telia, “Load Balancer Service Article”.

[99] Telia, “Load Balancer Service Form”.

[100] Fortinet, “Application Delivery Controller and GSLB”. [Online]. Available: https://www.fortinet.com/products/application-delivery-controller/fortiadc. Accessed 4.3.2024.

[101] Fortinet, “FortiADC”. [Online]. Available: https://www.fortinet.com/content/dam/fortinet/assets/data-sheets/FortiADC.pdf. Accessed 4.3.2024.

[102] Fortinet, “Handbook FortiADC 7.4.1”. [Online]. Available: https://fortinetweb.s3.amazonaws.com/docs.fortinet.com/v2/attachments/07f2746e-996d-11ee-a142-fa163e15d75b/fortiadc-v7.4.1-handbook.pdf. Accessed 8.3.2024.

[103] T. Muhammad, M. Munir, “Network Automation”, European Journal of Technology, Vol. 7, No. 2, pp. 23 – 42, 2023.

[104] HashiCorp, “FortiADC”. [Online]. Available: https://registry.terraform.io/providers/fortinetdev/fortiadc/latest. Accessed 14.3.2024.

[105] Red Hat, “Red Hat Enterprise Linux”. [Online]. Available: https://www.redhat.com/en/technologies/linux-platforms/enterprise-linux. Accessed 17.3.2024.

[106] Amazon Web Services, “Amazon S3”. [Online]. Available: https://aws.amazon.com/s3/. Accessed 18.3.2024.

[107] Scality, “What is S3 compatible storage?”. [Online]. Available: https://www.scality.com/topics/s3-compatible-storage/. Accessed 18.3.2024.

[108] Stan’s blog, “How to use a non-AWS S3 backend with Terraform”, 2018. [Online]. Available: https://stanislas.blog/2018/10/how-to-use-non-aws-s3-backend-terraform/. Accessed 18.3.2024.

[109] GitHub, “About self-hosted runners”. [Online]. Available: https://docs.github.com/en/actions/hosting-your-own-runners/managing-self-hosted-runners/about-self-hosted-runners. Accessed 20.3.2024.

[110] GitHub, “Using secrets in GitHub Actions”. [Online]. Available: https://docs.github.com/en/actions/security-guides/using-secrets-in-github-actions. Accessed 20.3.2024.

[111] HashiCorp, “The depends_on Meta-Argument”. [Online]. Available: https://developer.hashicorp.com/terraform/language/meta-arguments/depends_on. Accessed 21.3.2024.

[112] HashiCorp, “Modules”. [Online]. Available: https://developer.hashicorp.com/terraform/language/modules. Accessed 23.3.2024.

[113] HashiCorp, “Module Blocks”. [Online]. Available: https://developer.hashicorp.com/terraform/language/modules/syntax. Accessed 23.3.2024.

[114] HashiCorp, “The for_each Meta-Argument”. [Online]. Available: https://developer.hashicorp.com/terraform/language/meta-arguments/for_each. Accessed 23.3.2024.

[115] Let’s Encrypt, “ACME Client Implementations”, 2023. [Online]. Available: https://letsencrypt.org/docs/client-options/. Accessed 26.3.2024.

[116] Amazon Web Services, “Root user best practices for your AWS account”. [Online]. Available: https://docs.aws.amazon.com/IAM/latest/UserGuide/root-user-best-practices.html. Accessed 27.3.2024.

[117] HashiCorp, “Import”. [Online]. Available: https://developer.hashicorp.com/terraform/language/import. Accessed 1.4.2024.

[118] C. Wohlin, A. Aurum, “Towards a decision-making structure for selecting a research design in empirical software engineering”, Empirical Software Engineering Vol. 20, pp. 1427–1455, 2015.

[119] Microsoft, “Visio”. [Online]. Available: https://www.microsoft.com/en-us/microsoft-365/visio/flowchart-software. Accessed 3.4.2024.

[120] Cisco, “Network Topology Icons”. [Online]. Available: https://www.cisco.com/c/en/us/about/brand-center/network-topology-icons.html. Accessed 3.4.2024.

[121] J. D. Day and H. Zimmermann, "The OSI reference model”, Proceedings of the IEEE, Vol. 71, No. 2, pp. 1334 – 1340, 1983.

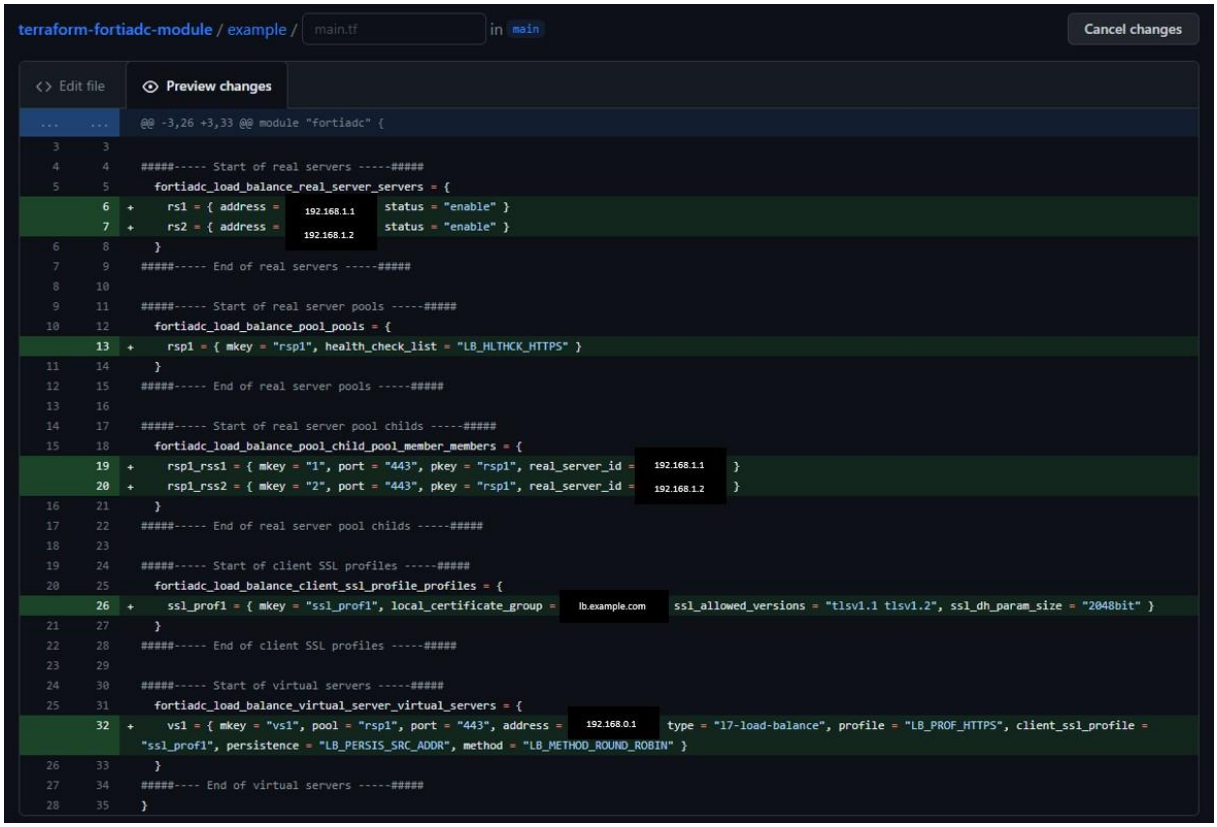
[122] R. Zalenski, “Firewall technologies”, IEEE Potentials, Vol. 21, No. 1, pp. 24 – 29, 2002.

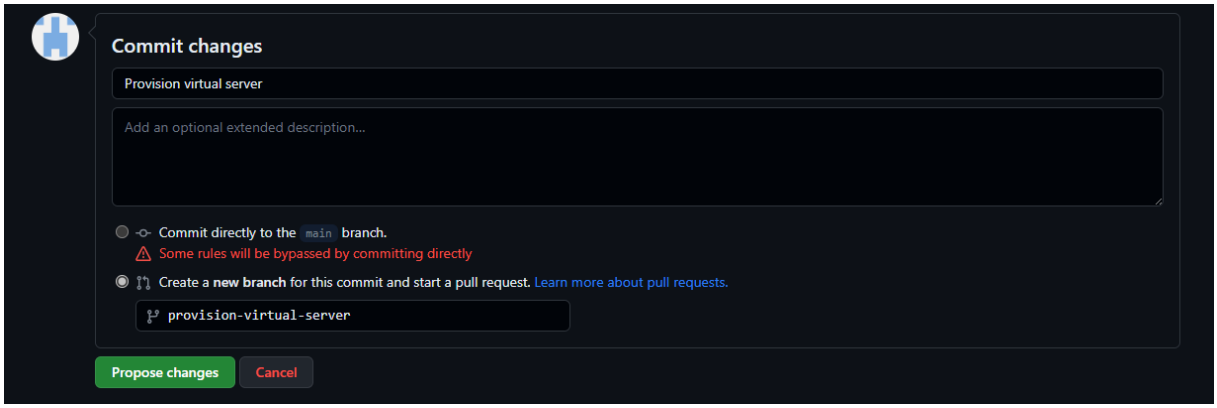
[123] M. Guerriero, M. Garriga, D. A. Tamburri and F. Palomba, “Adoption, Support, and Challenges of Infrastructure-as-Code: Insights from Industry”, In Proceedings of 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 580-589, 2019.

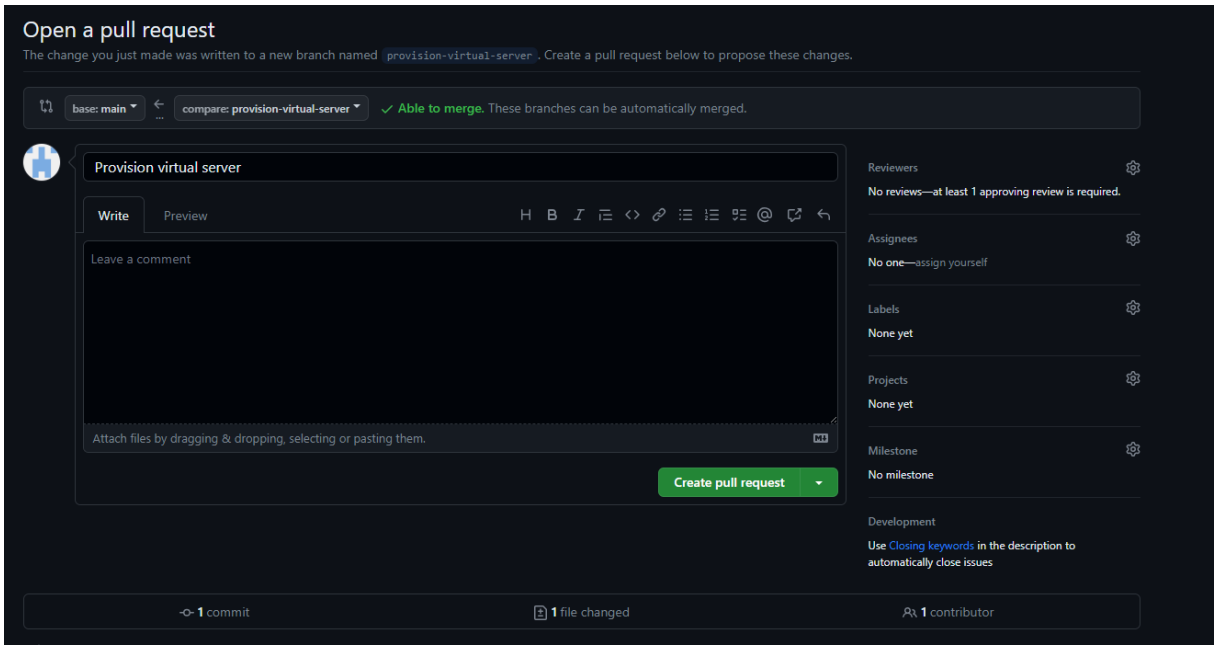
[124] J. Zhang, F. R. Yu, S. Wang, T. Huang, Z. Liu and Y. Liu, “Load Balancing in Data Center Networks: A Survey”, IEEE Communications Surveys & Tutorials, Vol. 20, No. 3, pp. 2324-2352, 2018.

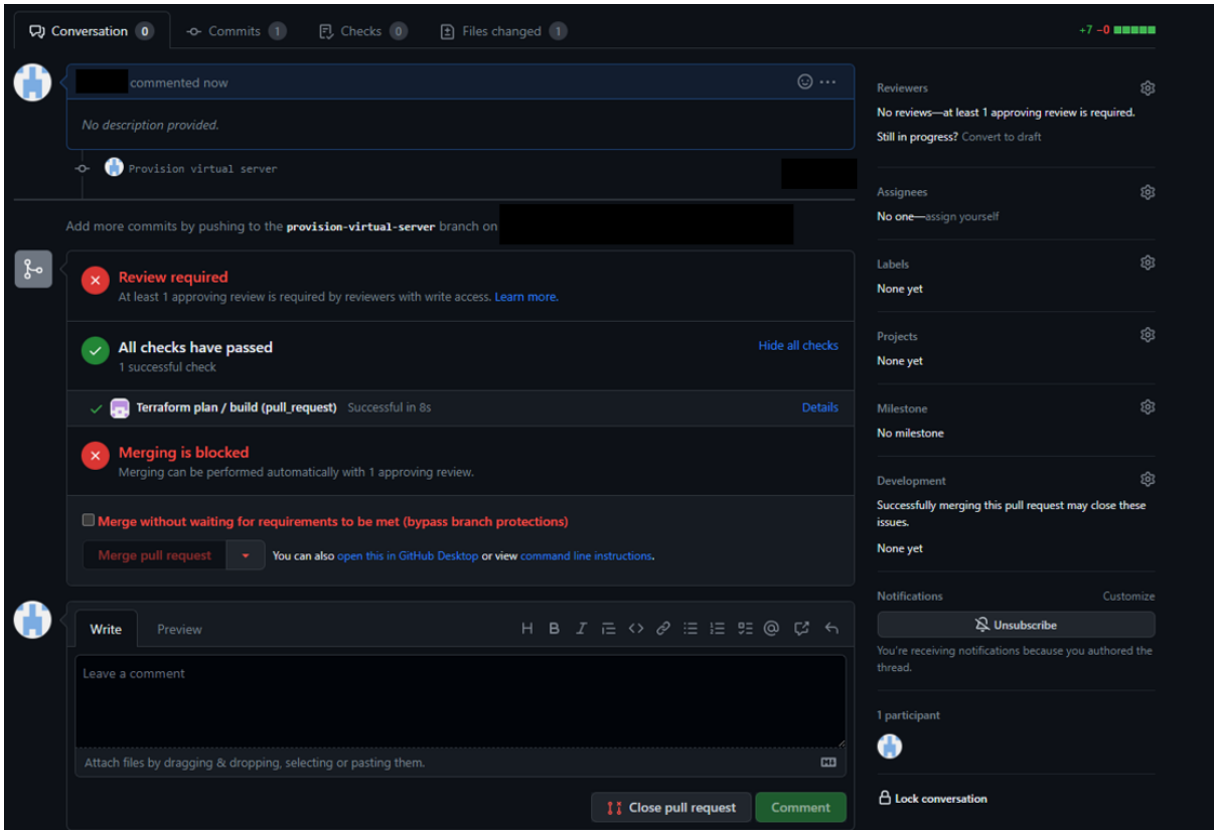
Appendices

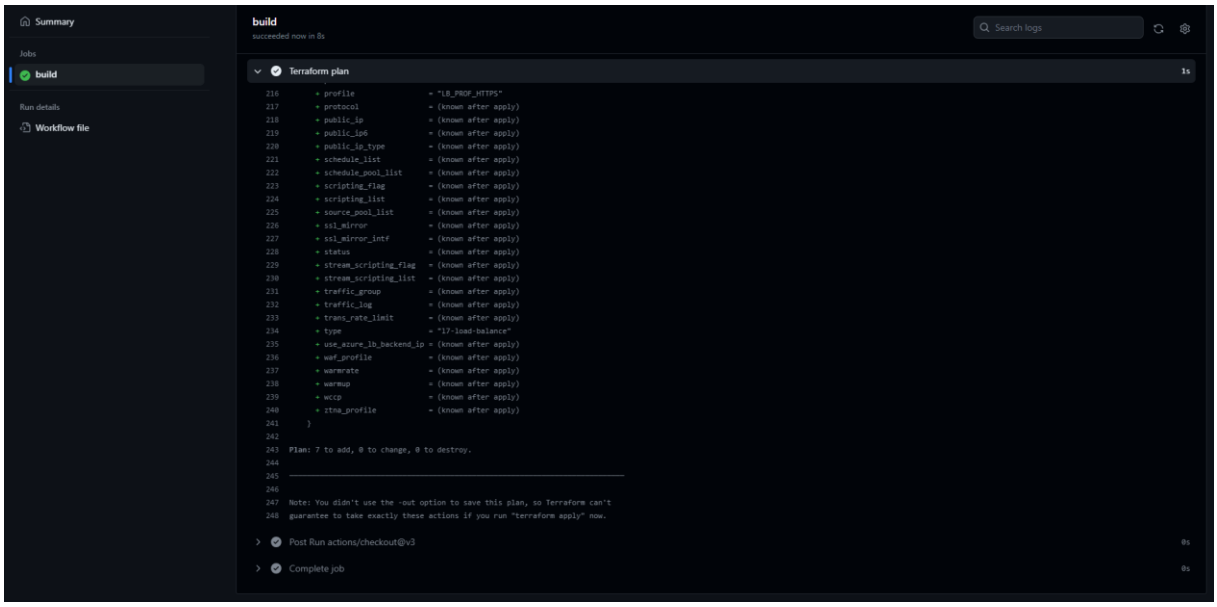
Appendix 1 Proposed load balancer service delivery model provision

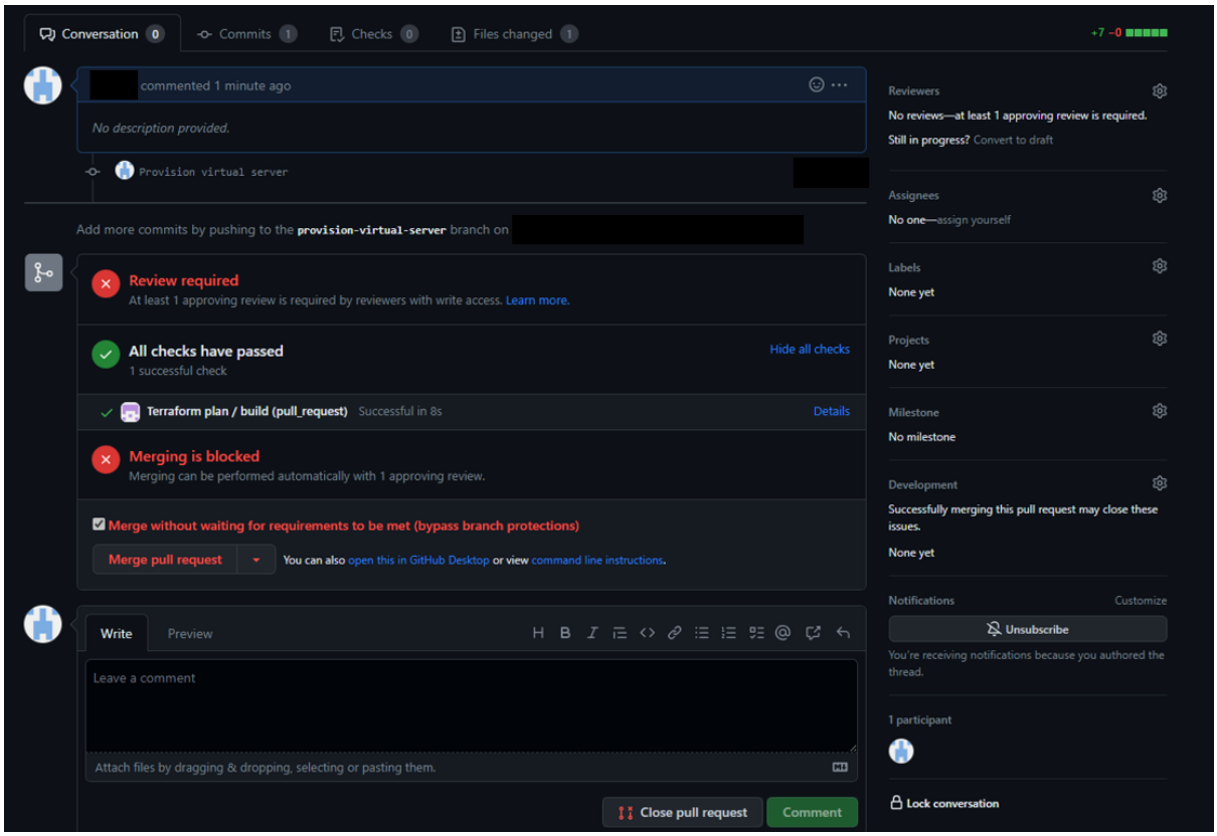


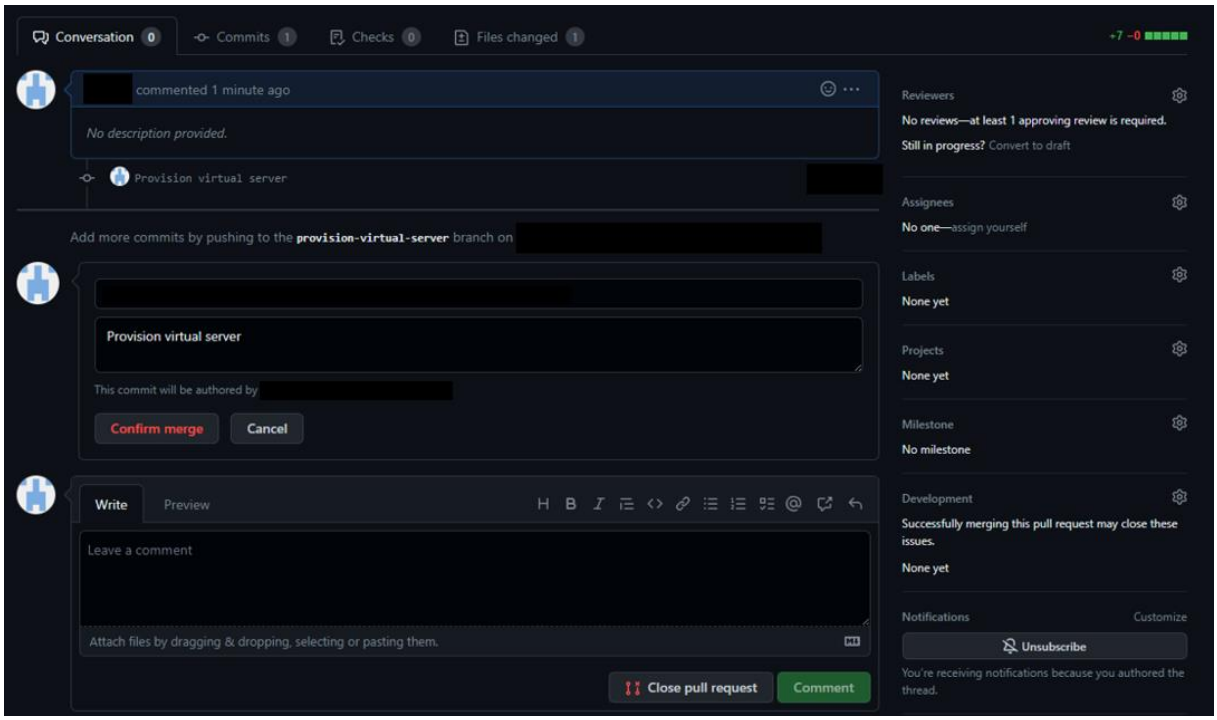


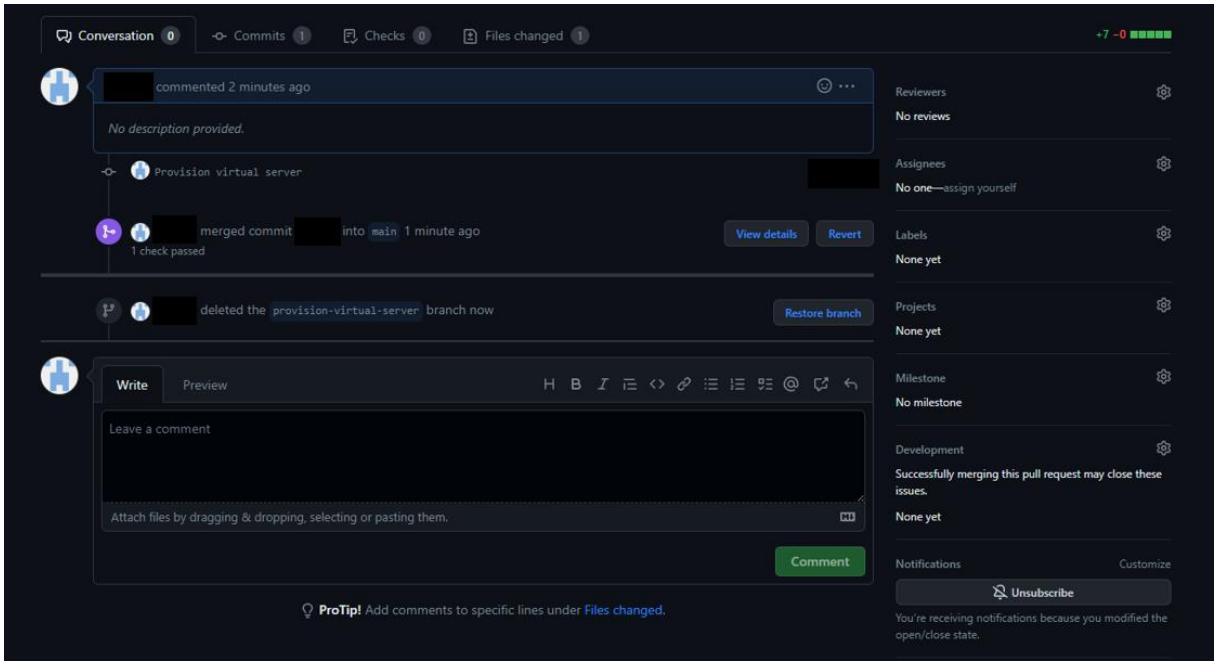

Open a pull request
The change you just made was written to a new branch named provision-virtual-server. Create a pull request below to propose these changes.
base: main
compare: provision-virtual-server
Able to merge. These branches can be automatically merged.
Provision virtual server
Write
Preview
Leave a comment
Attach files by dragging & dropping, selecting or pasting them.
Create pull request
Reviewers
No reviews—at least 1 approving review is required.
Assignees
No one—assign yourself
Labels
None yet
Projects
None yet
Milestone
No milestone
Development
Use Closing keywords in the description to automatically close issues
1 commit
1 file changed
1 contributor

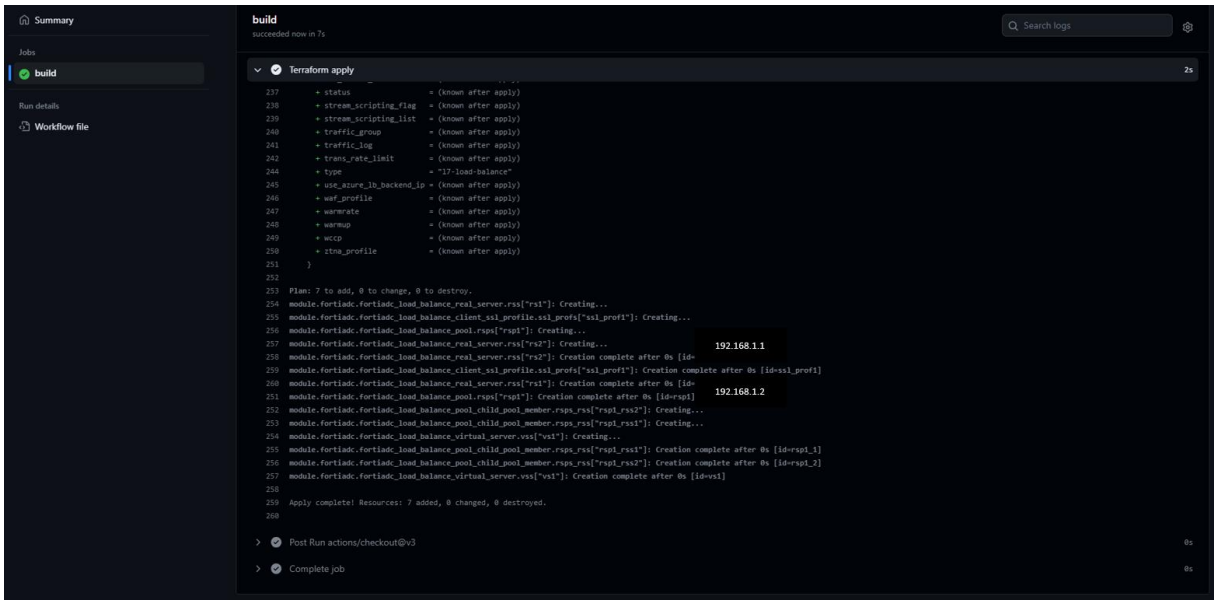

Conversation 0
Commits 1
Checks 0
Files changed 1
+7 −0
commented now
No description provided.
Provision virtual server
Add more commits by pushing to the provision-virtual-server branch on
Review required
At least 1 approving review is required by reviewers with write access. Learn more.
All checks have passed
1 successful check
Hide all checks
Terraform plan / build (pull_request) Successful in 8s
Details
Merging is blocked
Merging can be performed automatically with 1 approving review.
Merge without waiting for requirements to be met (bypass branch protections)
Merge pull request
You can also open this in GitHub Desktop or view command line instructions.
Write
Preview
Leave a comment
Attach files by dragging & dropping, selecting or pasting them.
Close pull request
Comment
Reviewers
No reviews—at least 1 approving review is required.
Still in progress? Convert to draft
Assignees
No one—assign yourself
Labels
None yet
Projects
None yet
Milestone
No milestone
Development
Successfully merging this pull request may close these issues.
None yet
Notifications
Customize
Unsubscribe
You're receiving notifications because you authored the thread.
1 participant
Lock conversation


Summary
Jobs
build
Run details
Workflow file
build
succeeded now in 8s
Search logs
Terraform plan
1s
+ profile = "LB_PROF_HTTPS"
+ protocol = (known after apply)
+ public_ip = (known after apply)
+ public_ip6 = (known after apply)
+ public_ip_type = (known after apply)
+ schedule_list = (known after apply)
+ schedule_pool_list = (known after apply)
+ scripting_flag = (known after apply)
+ scripting_list = (known after apply)
+ source_pool_list = (known after apply)
+ ssl_mirror = (known after apply)
+ ssl_mirror_intf = (known after apply)
+ status = (known after apply)
+ stream_scripting_flag = (known after apply)
+ stream_scripting_list = (known after apply)
+ traffic_group = (known after apply)
+ traffic_log = (known after apply)
+ trans_rate_limit = (known after apply)
+ type = "l7-load-balance"
+ use_azure_lb_backend_ip = (known after apply)
+ waf_profile = (known after apply)
+ warmrate = (known after apply)
+ warmup = (known after apply)
+ wccp = (known after apply)
+ ztna_profile = (known after apply)
}
Plan: 7 to add, 0 to change, 0 to destroy.
Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "terraform apply" now.
Post Run actions/checkout@v3
0s
Complete job
0s

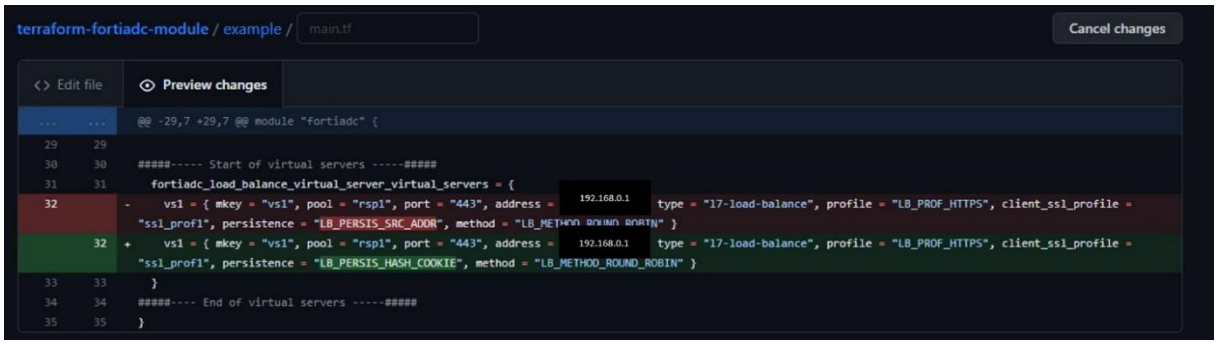

Conversation 0
Commits 1
Checks 0
Files changed 1
+7 −0
[redacted] commented 1 minute ago
No description provided.
Provision virtual server [redacted]
Add more commits by pushing to the provision-virtual-server branch on [redacted]
Review required
At least 1 approving review is required by reviewers with write access. Learn more.
All checks have passed
1 successful check
Hide all checks
Terraform plan / build (pull_request) Successful in 8s
Details
Merging is blocked
Merging can be performed automatically with 1 approving review.
Merge without waiting for requirements to be met (bypass branch protections)
Merge pull request
You can also open this in GitHub Desktop or view command line instructions.
Write
Preview
Leave a comment
Attach files by dragging & dropping, selecting or pasting them.
Close pull request
Comment
Reviewers
No reviews—at least 1 approving review is required.
Still in progress? Convert to draft
Assignees
No one—assign yourself
Labels
None yet
Projects
None yet
Milestone
No milestone
Development
Successfully merging this pull request may close these issues.
None yet
Notifications
Customize
Unsubscribe
You're receiving notifications because you authored the thread.
1 participant
Lock conversation

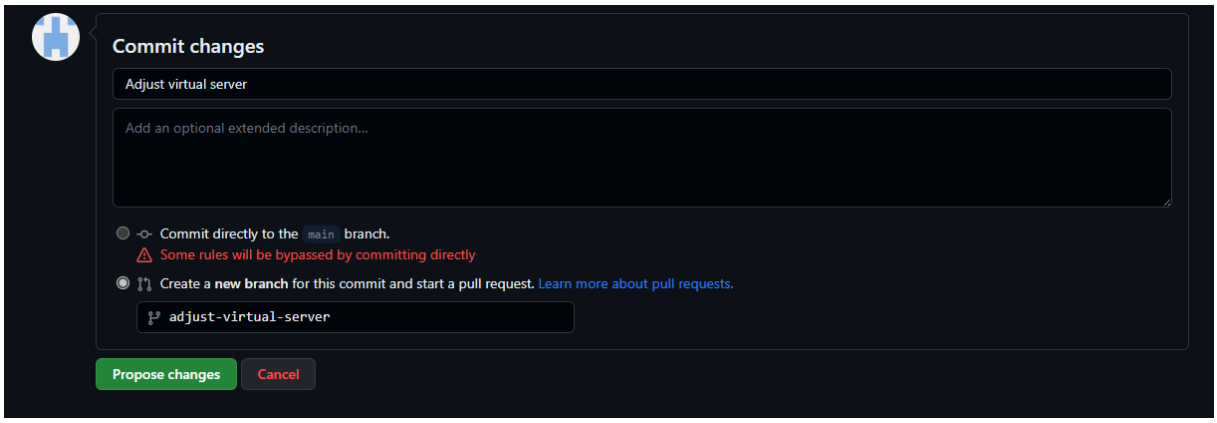

Conversation 0
Commits 1
Checks 0
Files changed 1
+7 −0
[redacted] commented 1 minute ago
No description provided.
Provision virtual server [redacted]
Add more commits by pushing to the provision-virtual-server branch on [redacted]
[redacted]
Provision virtual server
This commit will be authored by [redacted]
Confirm merge
Cancel
Write
Preview
Leave a comment
Attach files by dragging & dropping, selecting or pasting them.
Close pull request
Comment
Reviewers
No reviews—at least 1 approving review is required.
Still in progress? Convert to draft
Assignees
No one—assign yourself
Labels
None yet
Projects
None yet
Milestone
No milestone
Development
Successfully merging this pull request may close these issues.
None yet
Notifications
Customize
Unsubscribe
You're receiving notifications because you authored the thread.

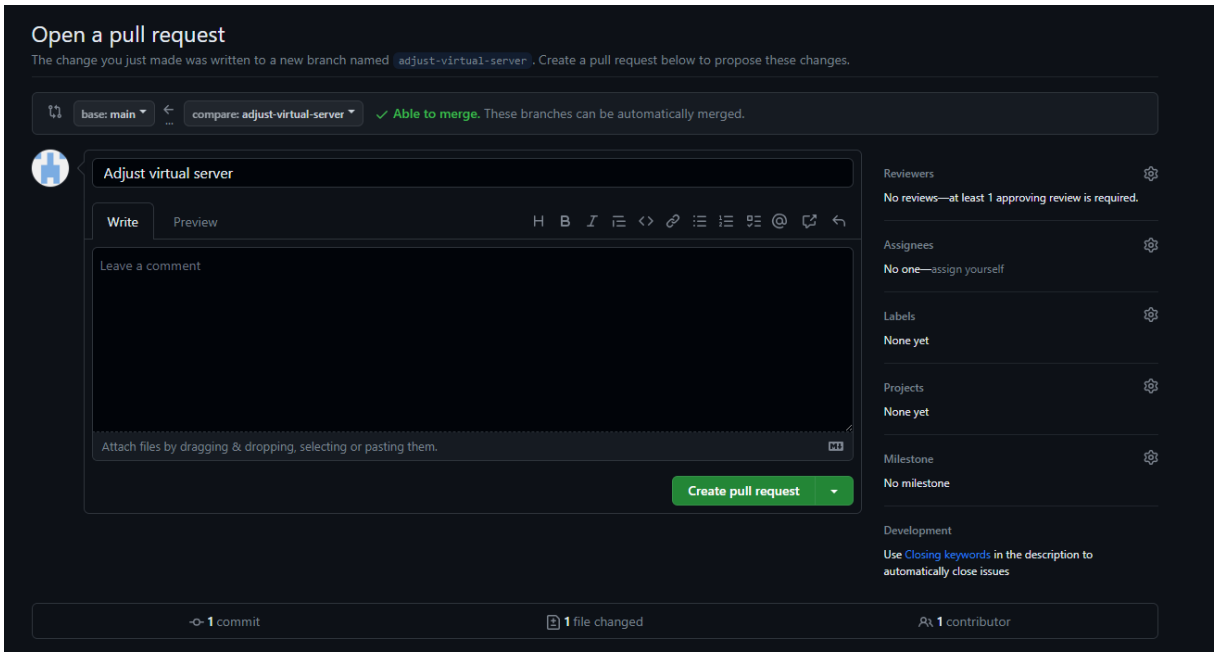

Conversation 0
Commits 1
Checks 0
Files changed 1
+7 −0
[redacted] commented 2 minutes ago
No description provided.
Provision virtual server [redacted]
[redacted] merged commit [redacted] into main 1 minute ago
1 check passed
View details
Revert
[redacted] deleted the provision-virtual-server branch now
Restore branch
Write
Preview
Leave a comment
Attach files by dragging & dropping, selecting or pasting them.
Comment
ProTip! Add comments to specific lines under Files changed.
Reviewers
No reviews
Assignees
No one—assign yourself
Labels
None yet
Projects
None yet
Milestone
No milestone
Development
Successfully merging this pull request may close these issues.
None yet
Notifications
Customize
Unsubscribe
You're receiving notifications because you modified the open/close state.

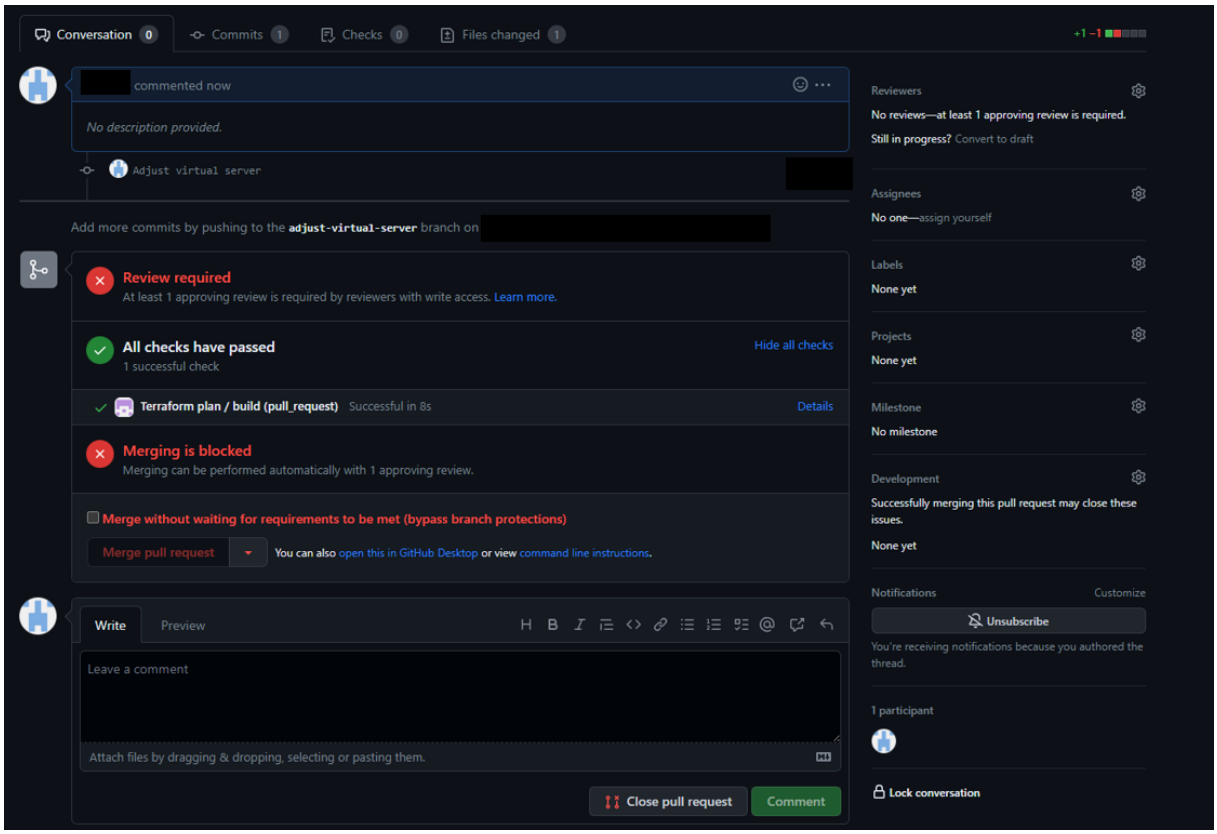

Summary
Jobs
build
Run details
Workflow file
build
succeeded now in 7s
Search logs
Terraform apply
2s
+ status = (known after apply)
+ stream_scripting_flag = (known after apply)
+ stream_scripting_list = (known after apply)
+ traffic_group = (known after apply)
+ traffic_log = (known after apply)
+ trans_rate_limit = (known after apply)
+ type = "l7-load-balance"
+ use_azure_lb_backend_ip = (known after apply)
+ waf_profile = (known after apply)
+ warmrate = (known after apply)
+ warmup = (known after apply)
+ wccp = (known after apply)
+ ztna_profile = (known after apply)
}
Plan: 7 to add, 0 to change, 0 to destroy.
module.fortiadc.fortiadc_load_balance_real_server.rss["rs1"]: Creating...
module.fortiadc.fortiadc_load_balance_client_ssl_profile.ssl_profs["ssl_prof1"]: Creating...
module.fortiadc.fortiadc_load_balance_pool.rsps["rsp1"]: Creating...
module.fortiadc.fortiadc_load_balance_real_server.rss["rs2"]: Creating...
module.fortiadc.fortiadc_load_balance_real_server.rss["rs2"]: Creation complete after 0s [id=
module.fortiadc.fortiadc_load_balance_client_ssl_profile.ssl_profs["ssl_prof1"]: Creation complete after 0s [id=ssl_prof1]
module.fortiadc.fortiadc_load_balance_real_server.rss["rs1"]: Creation complete after 0s [id=
module.fortiadc.fortiadc_load_balance_pool.rsps["rsp1"]: Creation complete after 0s [id=rsp1]
module.fortiadc.fortiadc_load_balance_pool_child_pool_member.rsps_rss["rsp1_rss2"]: Creating...
module.fortiadc.fortiadc_load_balance_pool_child_pool_member.rsps_rss["rsp1_rss1"]: Creating...
module.fortiadc.fortiadc_load_balance_virtual_server.vss["vs1"]: Creating...
module.fortiadc.fortiadc_load_balance_pool_child_pool_member.rsps_rss["rsp1_rss1"]: Creation complete after 0s [id=rsp1_1]
module.fortiadc.fortiadc_load_balance_pool_child_pool_member.rsps_rss["rsp1_rss2"]: Creation complete after 0s [id=rsp1_2]
module.fortiadc.fortiadc_load_balance_virtual_server.vss["vs1"]: Creation complete after 0s [id=vs1]
Apply complete! Resources: 7 added, 0 changed, 0 destroyed.
Post Run actions/checkout@v3
0s
Complete job
0s
192.168.1.1
192.168.1.2

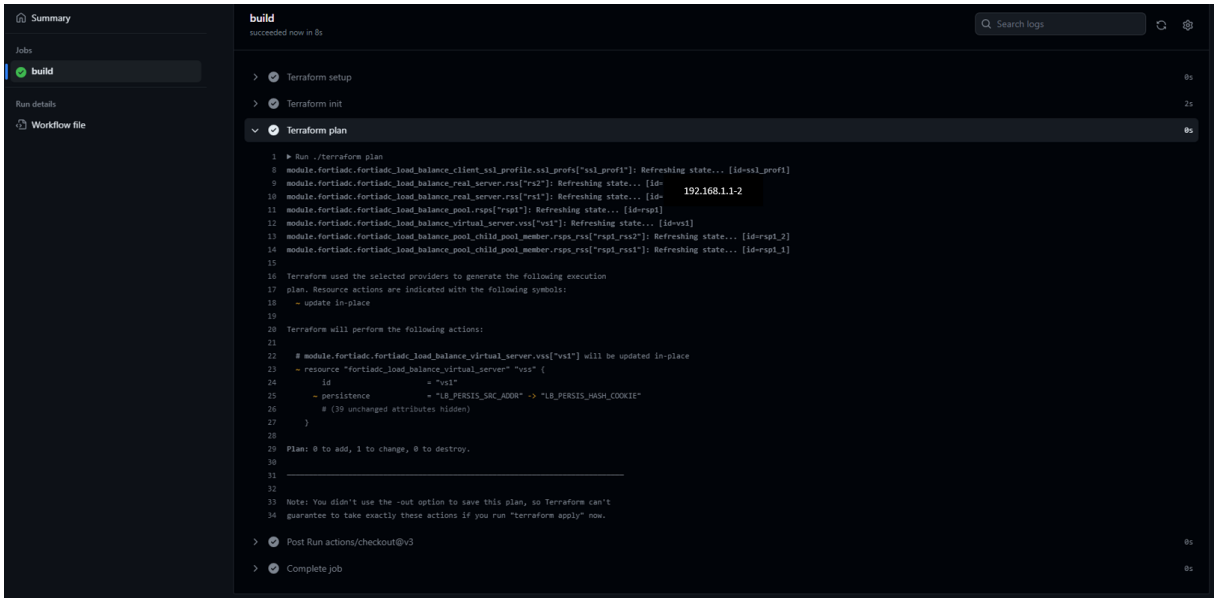
Appendix 2 Proposed load balancer service delivery model adjust

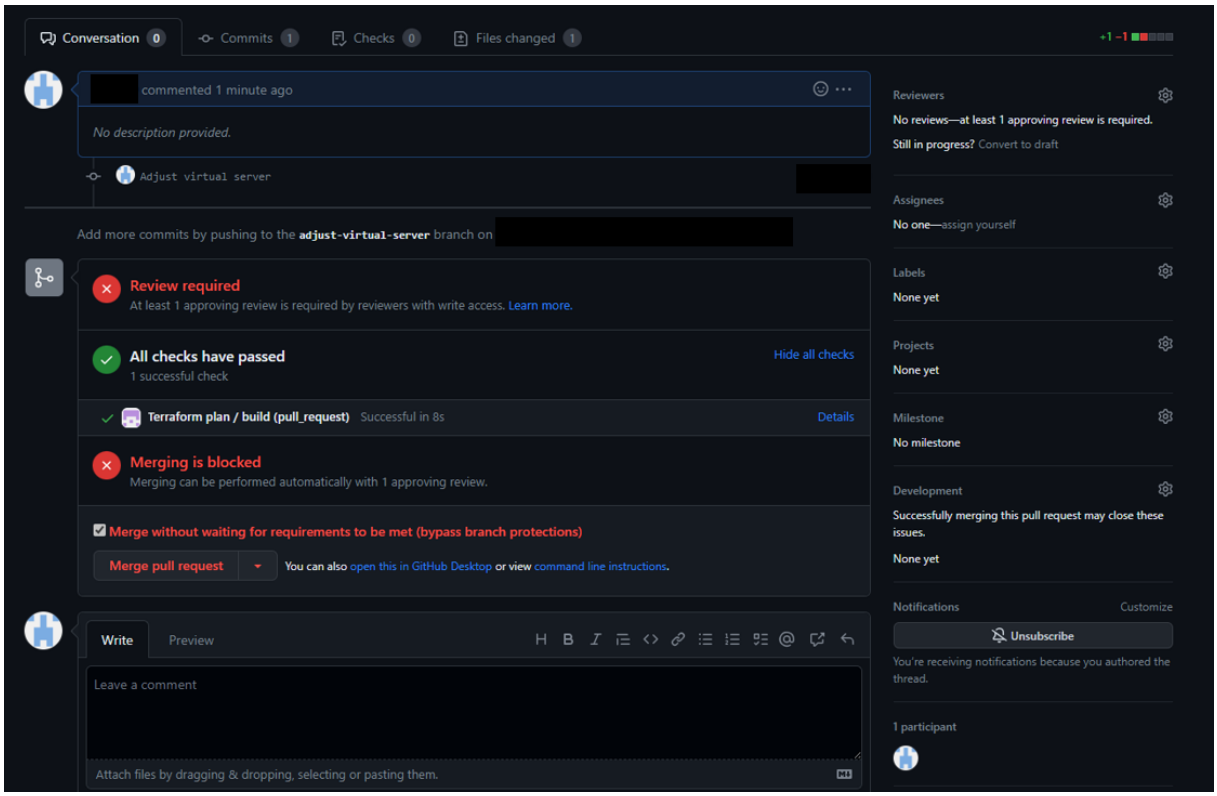


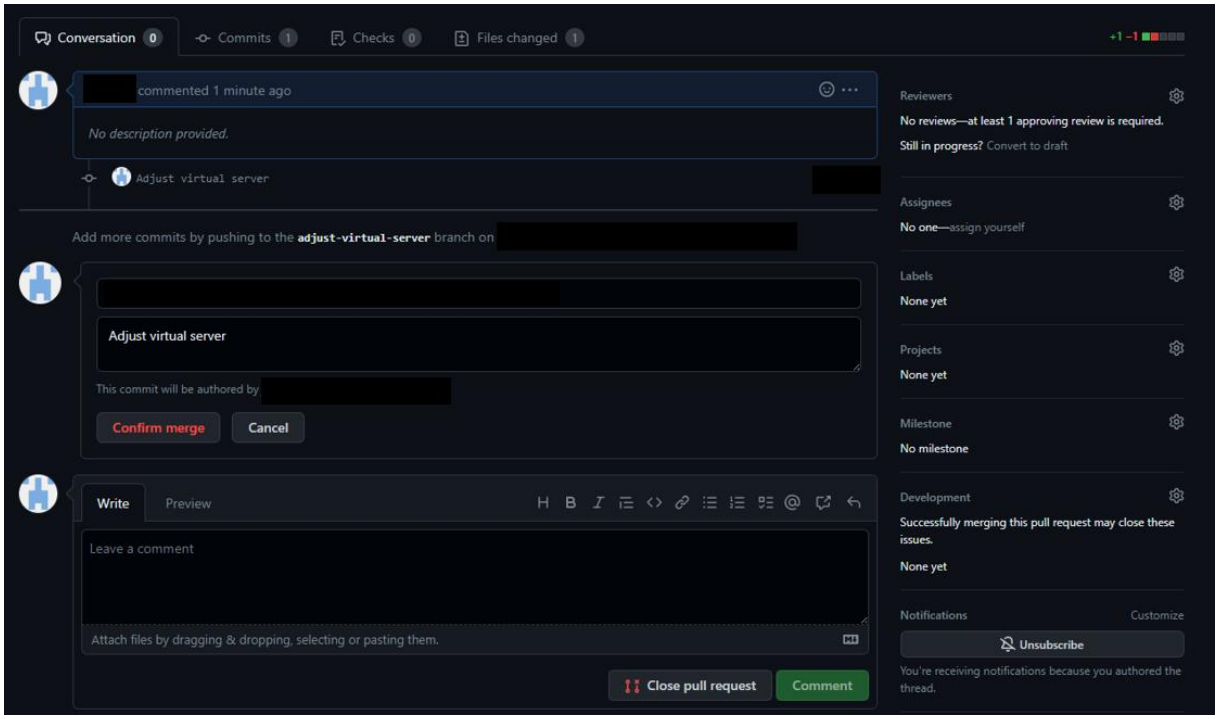


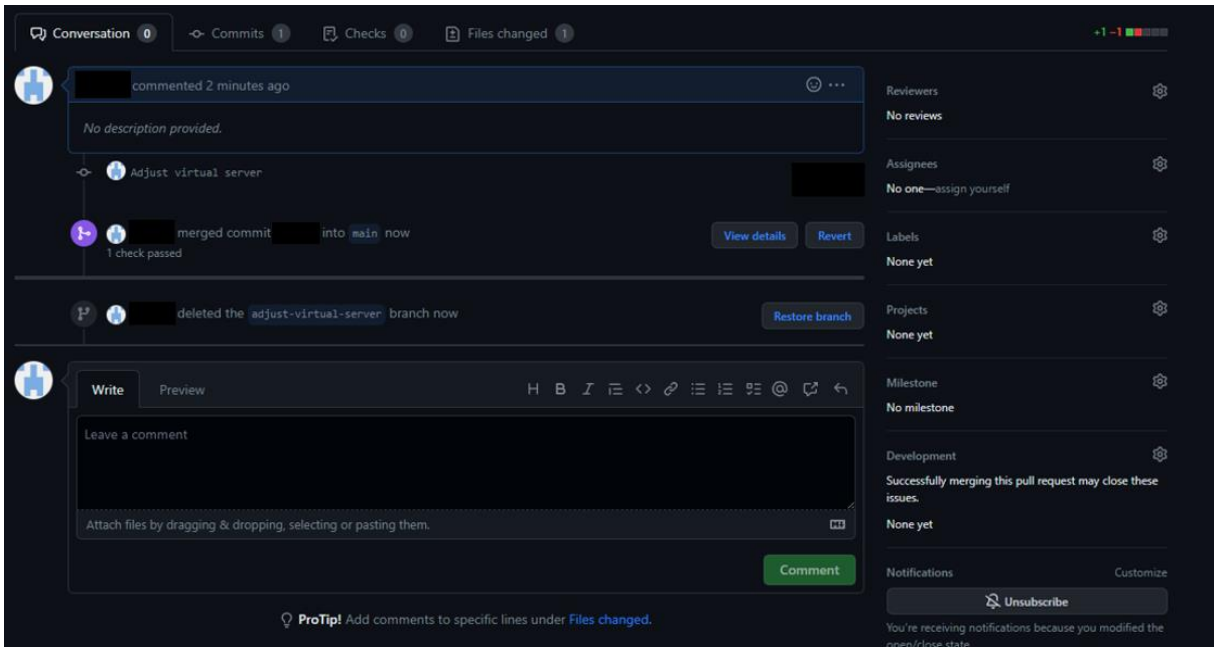


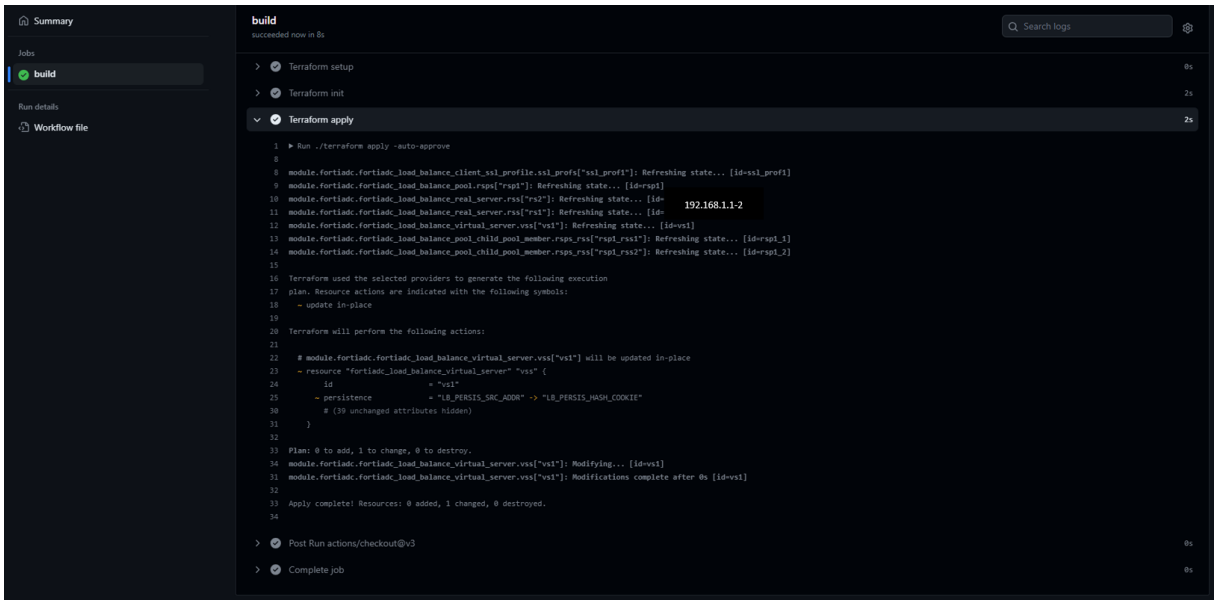

Conversation 0
Commits 1
Checks 0
Files changed 1
+1 −1
[redacted] commented now
No description provided.
Adjust virtual server [redacted]
Add more commits by pushing to the adjust-virtual-server branch on [redacted]
Review required
At least 1 approving review is required by reviewers with write access. Learn more.
All checks have passed
1 successful check
Hide all checks
Terraform plan / build (pull_request) Successful in 8s
Details
Merging is blocked
Merging can be performed automatically with 1 approving review.
Merge without waiting for requirements to be met (bypass branch protections)
Merge pull request
You can also open this in GitHub Desktop or view command line instructions.
Write
Preview
Leave a comment
Attach files by dragging & dropping, selecting or pasting them.
Close pull request
Comment
Reviewers
No reviews—at least 1 approving review is required.
Still in progress? Convert to draft
Assignees
No one—assign yourself
Labels
None yet
Projects
None yet
Milestone
No milestone
Development
Successfully merging this pull request may close these issues.
None yet
Notifications
Customize
Unsubscribe
You're receiving notifications because you authored the thread.
1 participant
Lock conversation


Summary
Jobs
build
Run details
Workflow file
build
succeeded now in 8s
Search logs
Terraform setup
Terraform init
Terraform plan
▶ Run ./terraform plan
module.fortiadc.fortiadc_load_balance_client_ssl_profile.ssl_profs["ssl_prof1"]: Refreshing state... [id=ssl_prof1]
module.fortiadc.fortiadc_load_balance_real_server.rss["rs2"]: Refreshing state... [id=
module.fortiadc.fortiadc_load_balance_real_server.rss["rs1"]: Refreshing state... [id=
192.168.1.1-2
module.fortiadc.fortiadc_load_balance_pool.rsps["rsp1"]: Refreshing state... [id=rsp1]
module.fortiadc.fortiadc_load_balance_virtual_server.vss["vs1"]: Refreshing state... [id=vs1]
module.fortiadc.fortiadc_load_balance_pool_child_pool_member.rsps_rss["rsp1_rss2"]: Refreshing state... [id=rsp1_2]
module.fortiadc.fortiadc_load_balance_pool_child_pool_member.rsps_rss["rsp1_rss1"]: Refreshing state... [id=rsp1_1]
Terraform used the selected providers to generate the following execution
plan. Resource actions are indicated with the following symbols:
~ update in-place
Terraform will perform the following actions:
# module.fortiadc.fortiadc_load_balance_virtual_server.vss["vs1"] will be updated in-place
~ resource "fortiadc_load_balance_virtual_server" "vss" {
id = "vs1"
~ persistence = "LB_PERSIS_SRC_ADDR" -> "LB_PERSIS_HASH_COOKIE"
# (39 unchanged attributes hidden)
}
Plan: 0 to add, 1 to change, 0 to destroy.
Note: You didn't use the -out option to save this plan, so Terraform can't
guarantee to take exactly these actions if you run "terraform apply" now.
Post Run actions/checkout@v3
Complete job

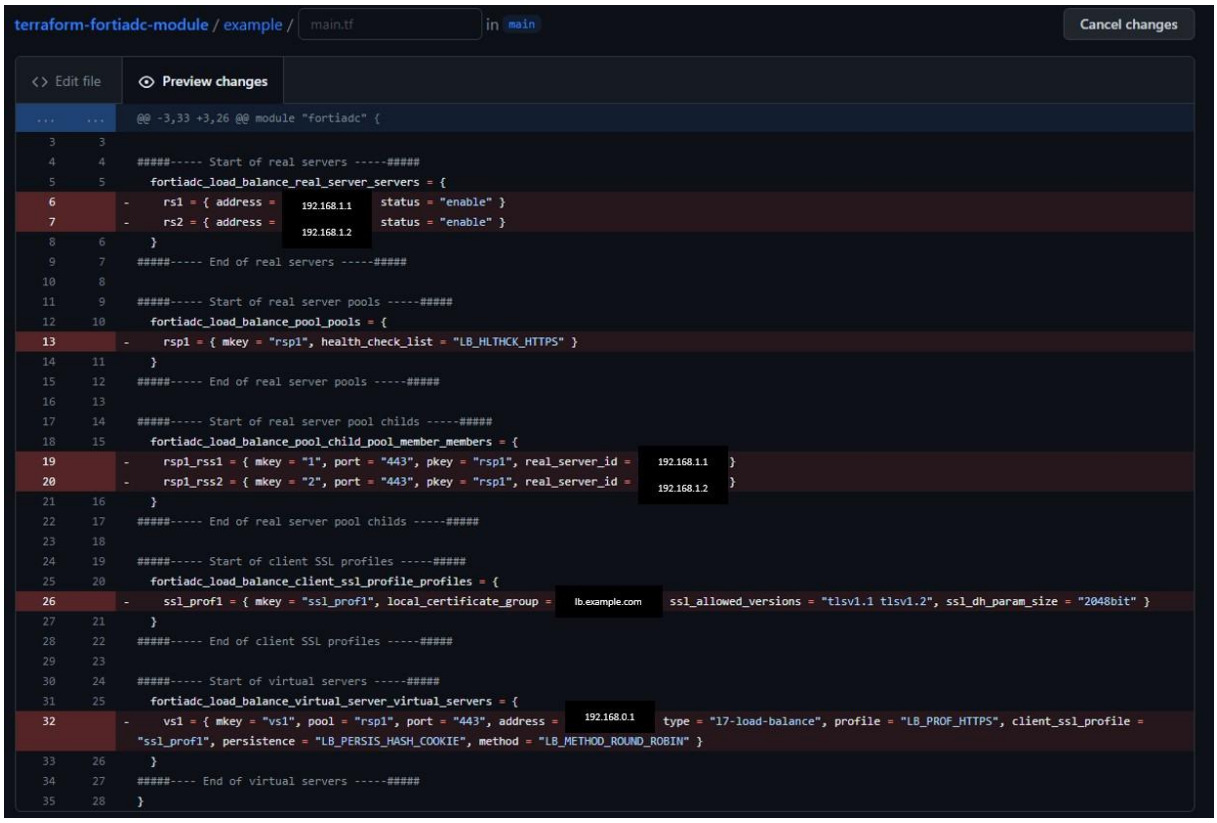

Conversation 0
Commits 1
Checks 0
Files changed 1
+1 −1
[redacted] commented 1 minute ago
No description provided.
Adjust virtual server [redacted]
Add more commits by pushing to the adjust-virtual-server branch on [redacted]
Review required
At least 1 approving review is required by reviewers with write access. Learn more.
All checks have passed
1 successful check
Hide all checks
Terraform plan / build (pull_request) Successful in 8s
Details
Merging is blocked
Merging can be performed automatically with 1 approving review.
Merge without waiting for requirements to be met (bypass branch protections)
Merge pull request
You can also open this in GitHub Desktop or view command line instructions.
Write
Preview
Leave a comment
Attach files by dragging & dropping, selecting or pasting them.
Reviewers
No reviews—at least 1 approving review is required.
Still in progress? Convert to draft
Assignees
No one—assign yourself
Labels
None yet
Projects
None yet
Milestone
No milestone
Development
Successfully merging this pull request may close these issues.
None yet
Notifications
Customize
Unsubscribe
You're receiving notifications because you authored the thread.
1 participant

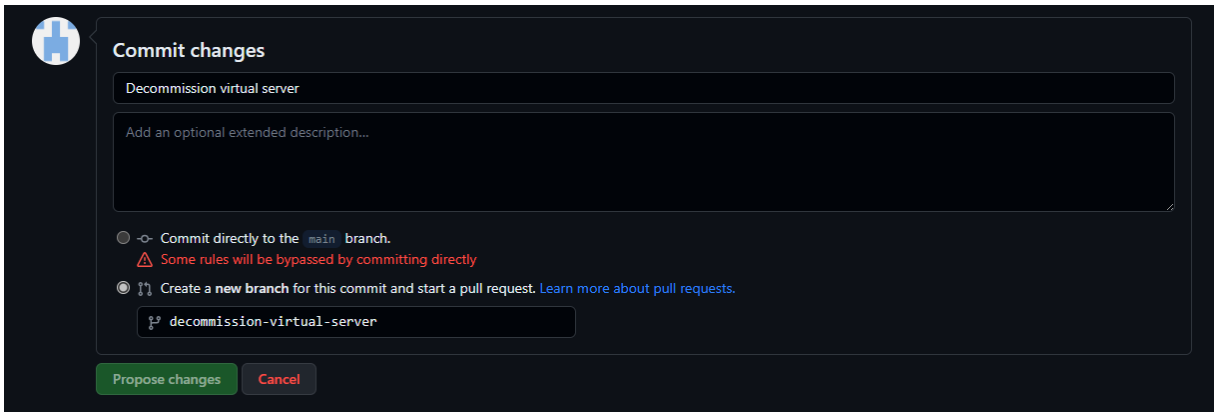

Conversation 0
Commits 1
Checks 0
Files changed 1
+1 −1
[redacted] commented 1 minute ago
No description provided.
Adjust virtual server [redacted]
Add more commits by pushing to the adjust-virtual-server branch on [redacted]
[redacted]
Adjust virtual server
This commit will be authored by [redacted]
Confirm merge
Cancel
Write
Preview
Leave a comment
Attach files by dragging & dropping, selecting or pasting them.
Close pull request
Comment
Reviewers
No reviews—at least 1 approving review is required.
Still in progress? Convert to draft
Assignees
No one—assign yourself
Labels
None yet
Projects
None yet
Milestone
No milestone
Development
Successfully merging this pull request may close these issues.
None yet
Notifications
Customize
Unsubscribe
You're receiving notifications because you authored the thread.

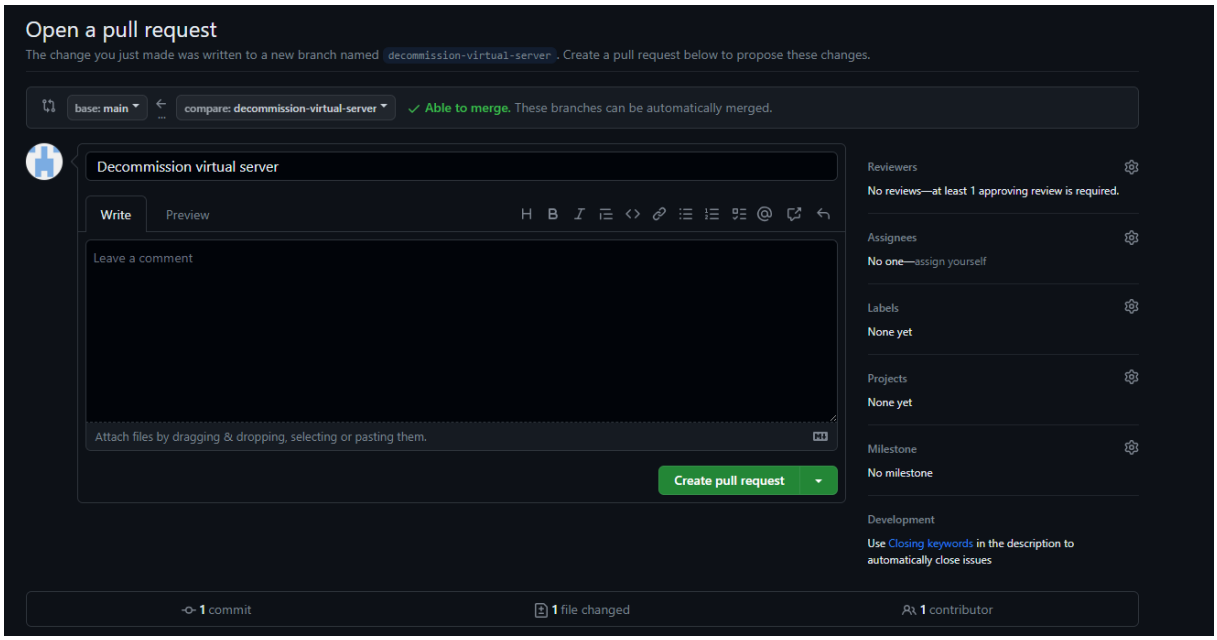

Conversation 0
Commits 1
Checks 0
Files changed 1
+1 −1
[redacted] commented 2 minutes ago
No description provided.
Adjust virtual server
[redacted] merged commit [redacted] into main now
1 check passed
View details
Revert
[redacted] deleted the adjust-virtual-server branch now
Restore branch
Write
Preview
Leave a comment
Attach files by dragging & dropping, selecting or pasting them.
Comment
ProTip! Add comments to specific lines under Files changed.
Reviewers
No reviews
Assignees
No one—assign yourself
Labels
None yet
Projects
None yet
Milestone
No milestone
Development
Successfully merging this pull request may close these issues.
None yet
Notifications
Customize
Unsubscribe
You're receiving notifications because you modified the

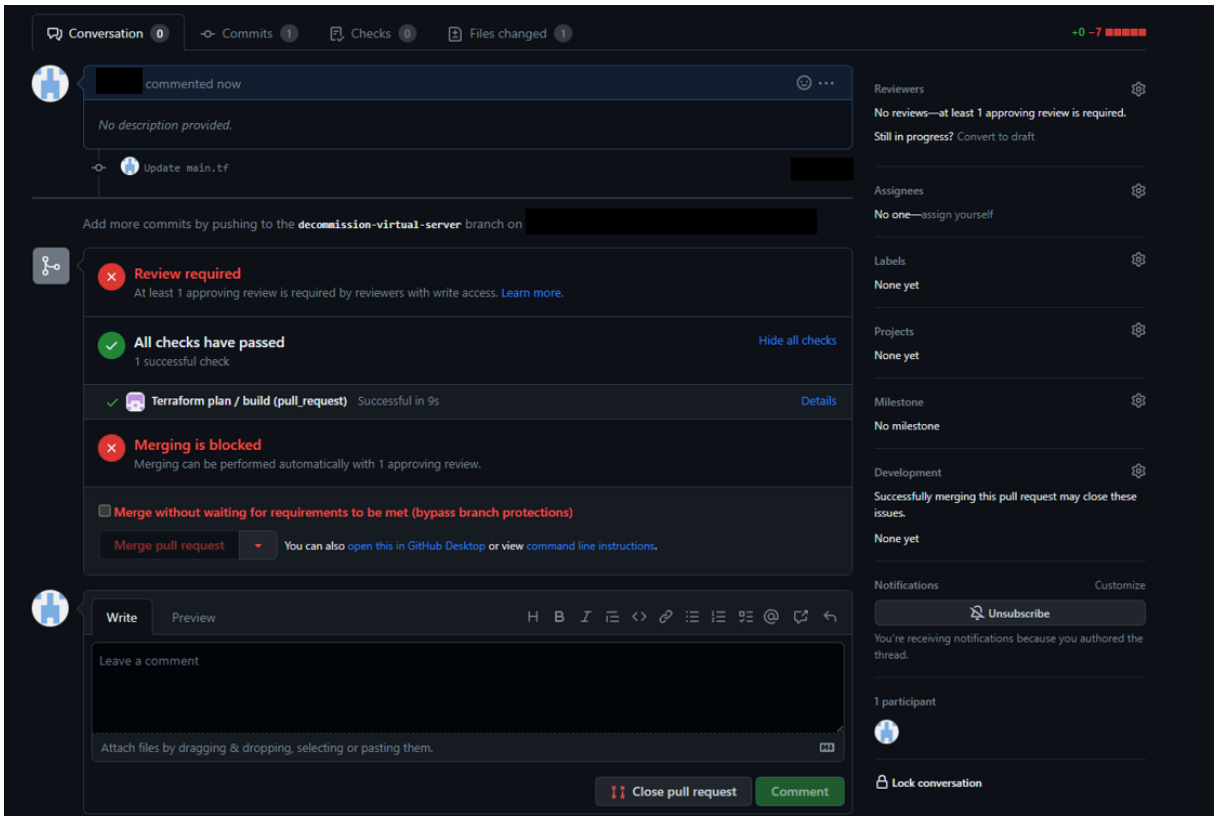

Summary
Jobs
build
Run details
Workflow file
build
succeeded now in 8s
Search logs
Terraform setup
Terraform init
Terraform apply
▶ Run ./terraform apply -auto-approve
module.fortiadc.fortiadc_load_balance_client_ssl_profile.ssl_profs["ssl_prof1"]: Refreshing state... [id=ssl_prof1]
module.fortiadc.fortiadc_load_balance_pool.rsps["rsp1"]: Refreshing state... [id=rsp1]
module.fortiadc.fortiadc_load_balance_real_server.rss["rs2"]: Refreshing state... [id=
module.fortiadc.fortiadc_load_balance_real_server.rss["rs1"]: Refreshing state... [id=
192.168.1.1-2
module.fortiadc.fortiadc_load_balance_virtual_server.vss["vs1"]: Refreshing state... [id=vs1]
module.fortiadc.fortiadc_load_balance_pool_child_pool_member.rsps_rss["rsp1_rss1"]: Refreshing state... [id=rsp1_1]
module.fortiadc.fortiadc_load_balance_pool_child_pool_member.rsps_rss["rsp1_rss2"]: Refreshing state... [id=rsp1_2]
Terraform used the selected providers to generate the following execution
plan. Resource actions are indicated with the following symbols:
~ update in-place
Terraform will perform the following actions:
# module.fortiadc.fortiadc_load_balance_virtual_server.vss["vs1"] will be updated in-place
~ resource "fortiadc_load_balance_virtual_server" "vss" {
id = "vs1"
~ persistence = "LB_PERSIS_SRC_ADDR" -> "LB_PERSIS_HASH_COOKIE"
# (39 unchanged attributes hidden)
}
Plan: 0 to add, 1 to change, 0 to destroy.
module.fortiadc.fortiadc_load_balance_virtual_server.vss["vs1"]: Modifying... [id=vs1]
module.fortiadc.fortiadc_load_balance_virtual_server.vss["vs1"]: Modifications complete after 0s [id=vs1]
Apply complete! Resources: 0 added, 1 changed, 0 destroyed.
Post Run actions/checkout@v3
Complete job

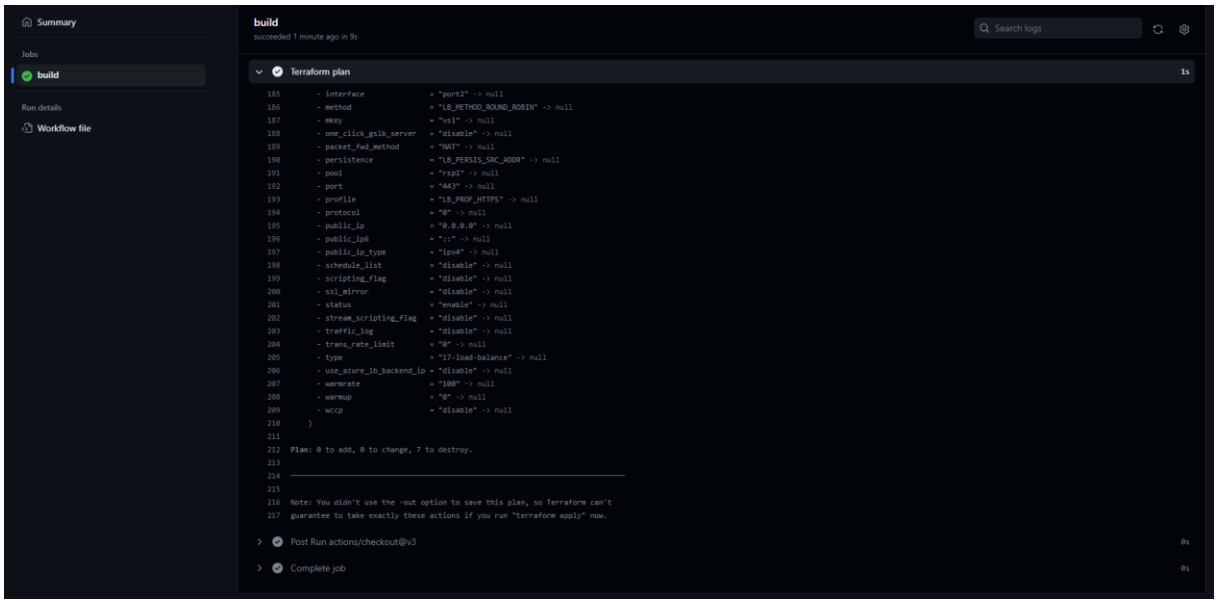
Appendix 3 Proposed load balancer service delivery model decommission

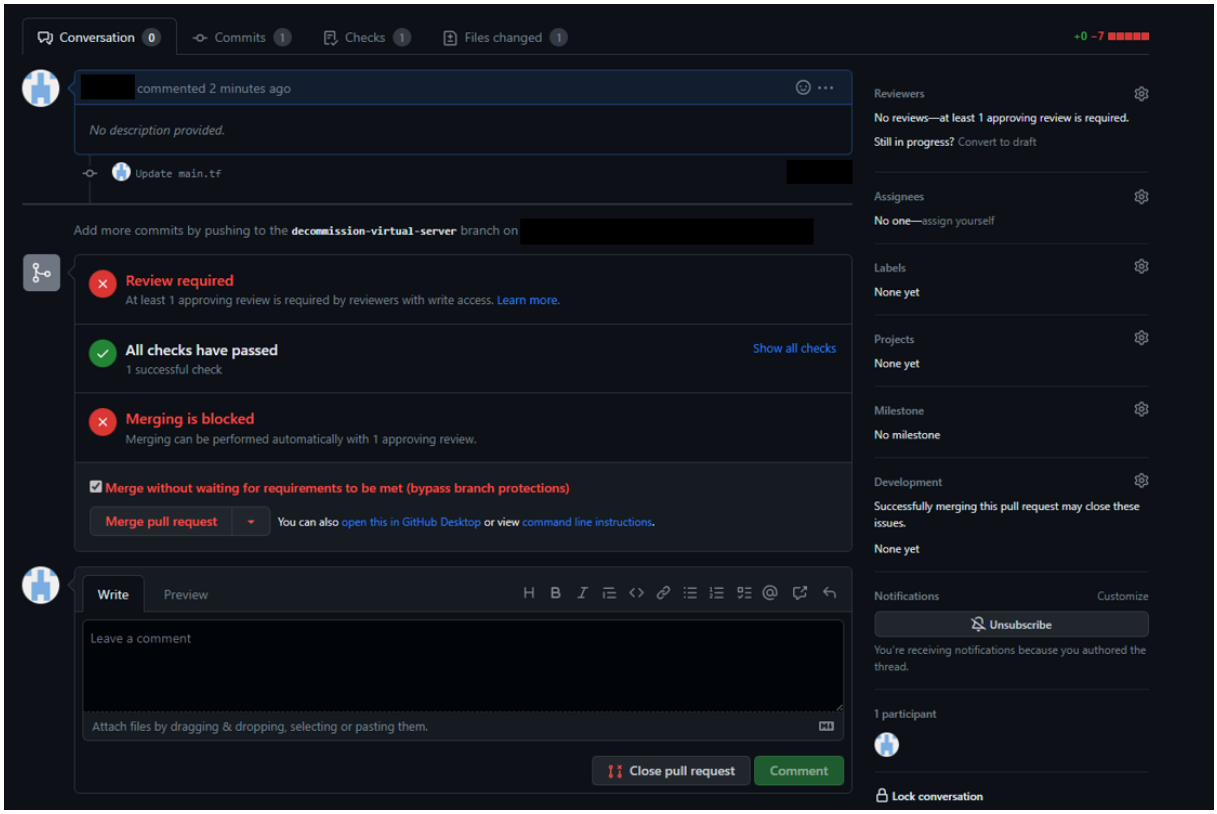


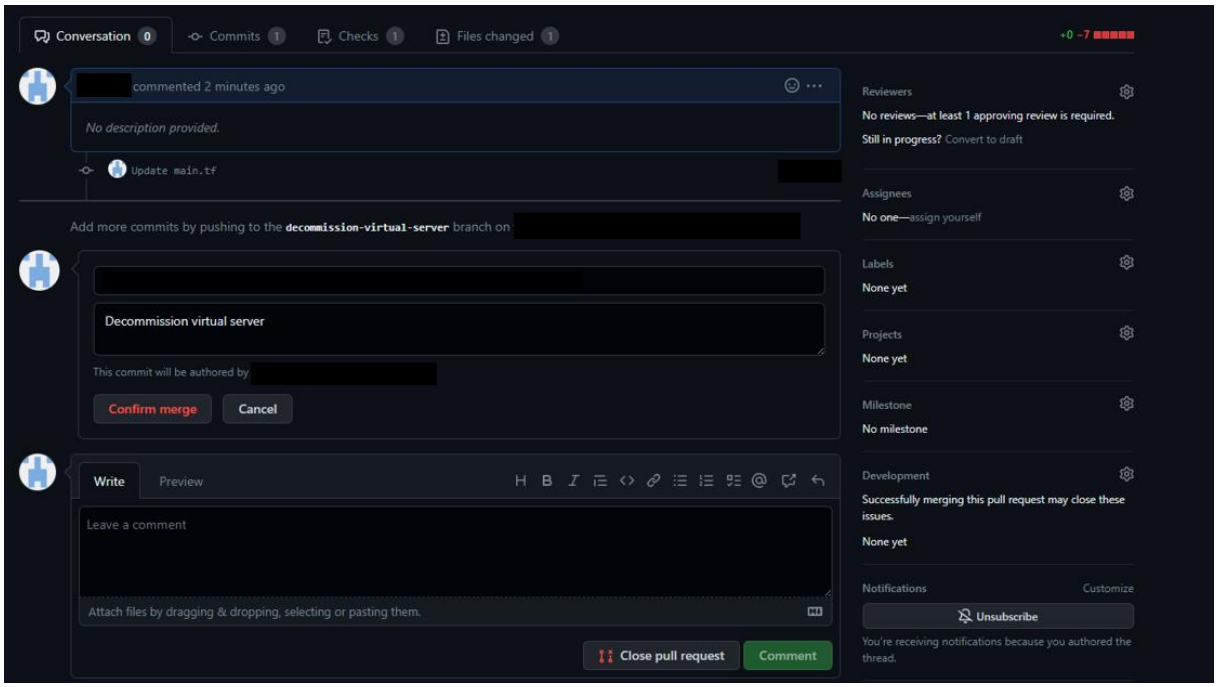


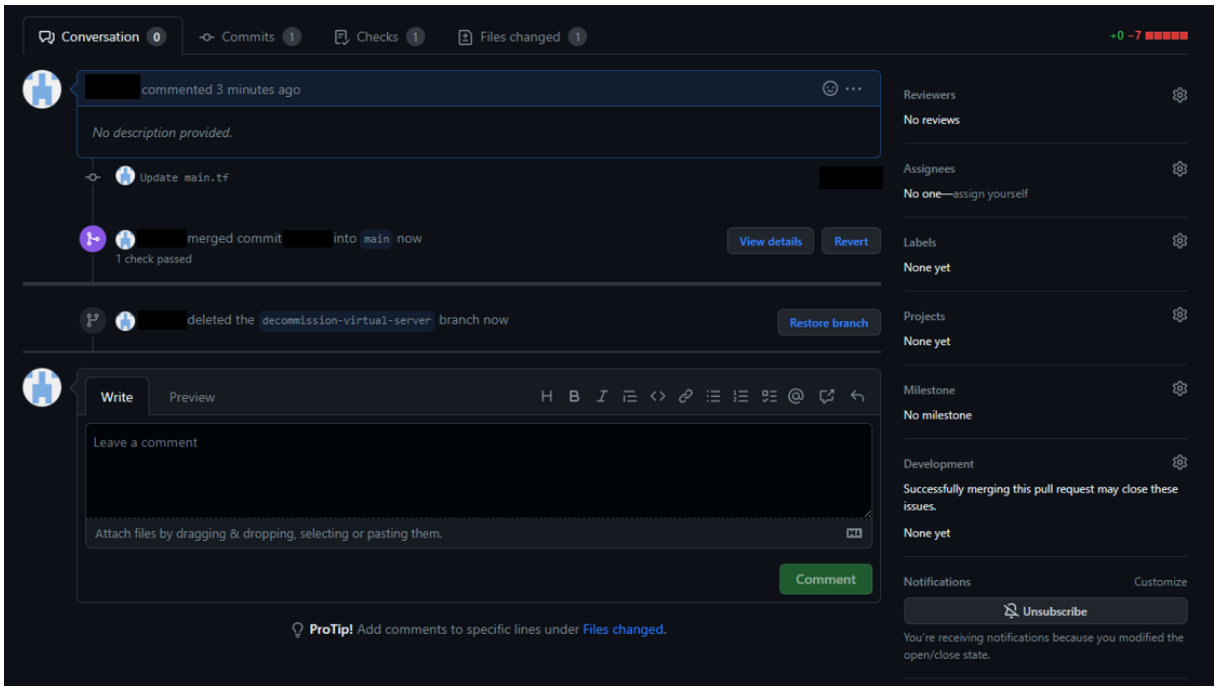

Open a pull request
The change you just made was written to a new branch named decommission-virtual-server . Create a pull request below to propose these changes.
base: main
compare: decommission-virtual-server
Able to merge. These branches can be automatically merged.
Decommission virtual server
Write
Preview
Leave a comment
Attach files by dragging & dropping, selecting or pasting them.
Create pull request
Reviewers
No reviews—at least 1 approving review is required.
Assignees
No one—assign yourself
Labels
None yet
Projects
None yet
Milestone
No milestone
Development
Use Closing keywords in the description to automatically close issues
1 commit
1 file changed
1 contributor

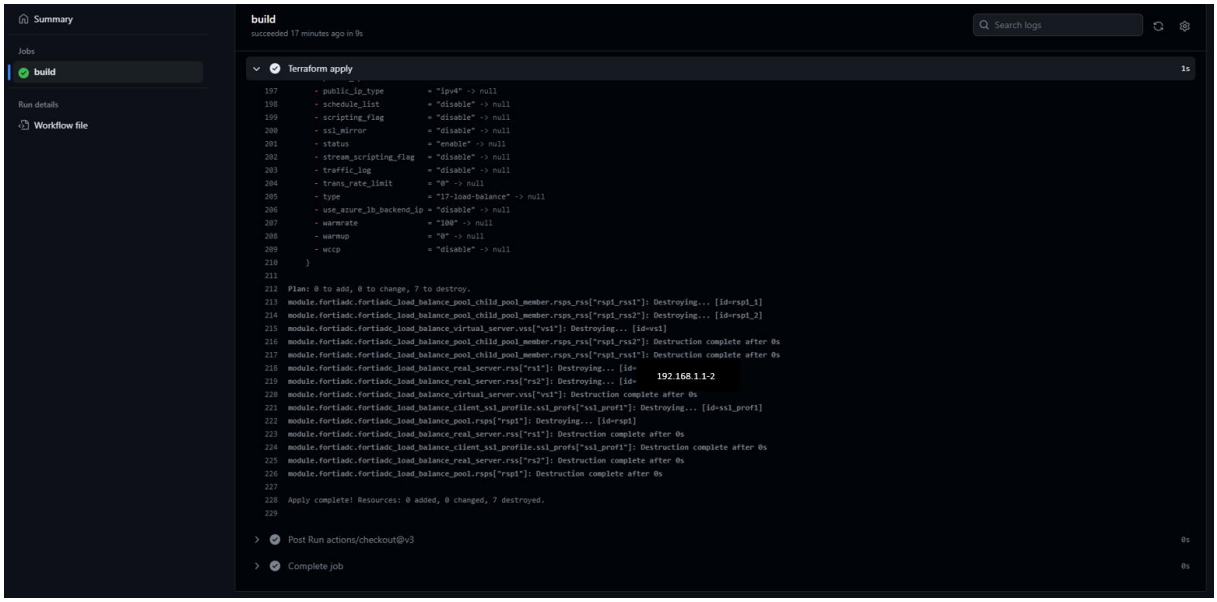

Conversation 0
Commits 1
Checks 0
Files changed 1
+0 −7
commented now
No description provided.
Update main.tf
Add more commits by pushing to the decommission-virtual-server branch on
Review required
At least 1 approving review is required by reviewers with write access. Learn more.
All checks have passed
1 successful check
Hide all checks
Terraform plan / build (pull_request) Successful in 9s
Details
Merging is blocked
Merging can be performed automatically with 1 approving review.
Merge without waiting for requirements to be met (bypass branch protections)
Merge pull request
You can also open this in GitHub Desktop or view command line instructions.
Write
Preview
Leave a comment
Attach files by dragging & dropping, selecting or pasting them.
Close pull request
Comment
Reviewers
No reviews—at least 1 approving review is required.
Still in progress? Convert to draft
Assignees
No one—assign yourself
Labels
None yet
Projects
None yet
Milestone
No milestone
Development
Successfully merging this pull request may close these issues.
None yet
Notifications
Customize
Unsubscribe
You're receiving notifications because you authored the thread.
1 participant
Lock conversation


Summary
Jobs
build
Run details
Workflow file
build
succeeded 1 minute ago in 9s
Search logs
Terraform plan
1s
- interface = "port2" -> null
- method = "LB_METHOD_ROUND_ROBIN" -> null
- mkey = "vs1" -> null
- one_click_gslb_server = "disable" -> null
- packet_fwd_method = "NAT" -> null
- persistence = "LB_PERSIS_SRC_ADDR" -> null
- pool = "rsp1" -> null
- port = "443" -> null
- profile = "LB_PROF_HTTPS" -> null
- protocol = "0" -> null
- public_ip = "0.0.0.0" -> null
- public_ip6 = "::" -> null
- public_ip_type = "ipv4" -> null
- schedule_list = "disable" -> null
- scripting_flag = "disable" -> null
- ssl_mirror = "disable" -> null
- status = "enable" -> null
- stream_scripting_flag = "disable" -> null
- traffic_log = "disable" -> null
- trans_rate_limit = "0" -> null
- type = "17-load-balance" -> null
- use_azure_lb_backend_ip = "disable" -> null
- warmrate = "100" -> null
- warmup = "0" -> null
- wccp = "disable" -> null
}
Plan: 0 to add, 0 to change, 7 to destroy.
Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "terraform apply" now.
Post Run actions/checkout@v3
0s
Complete job
0s


Conversation 0
Commits 1
Checks 1
Files changed 1
+0 −7
[redacted] commented 2 minutes ago
No description provided.
Update main.tf
[redacted]
Add more commits by pushing to the decommission-virtual-server branch on [redacted]
Review required
At least 1 approving review is required by reviewers with write access. Learn more.
All checks have passed
1 successful check
Show all checks
Merging is blocked
Merging can be performed automatically with 1 approving review.
Merge without waiting for requirements to be met (bypass branch protections)
Merge pull request
You can also open this in GitHub Desktop or view command line instructions.
Write
Preview
Leave a comment
Attach files by dragging & dropping, selecting or pasting them.
Close pull request
Comment
Reviewers
No reviews—at least 1 approving review is required.
Still in progress? Convert to draft
Assignees
No one—assign yourself
Labels
None yet
Projects
None yet
Milestone
No milestone
Development
Successfully merging this pull request may close these issues.
None yet
Notifications
Customize
Unsubscribe
You're receiving notifications because you authored the thread.
1 participant
Lock conversation


Conversation 0
Commits 1
Checks 1
Files changed 1
+0 −7
[redacted] commented 2 minutes ago
No description provided.
Update main.tf
[redacted]
Add more commits by pushing to the decommission-virtual-server branch on [redacted]
[redacted]
Decommission virtual server
This commit will be authored by [redacted]
Confirm merge
Cancel
Write
Preview
Leave a comment
Attach files by dragging & dropping, selecting or pasting them.
Close pull request
Comment
Reviewers
No reviews—at least 1 approving review is required.
Still in progress? Convert to draft
Assignees
No one—assign yourself
Labels
None yet
Projects
None yet
Milestone
No milestone
Development
Successfully merging this pull request may close these issues.
None yet
Notifications
Customize
Unsubscribe
You're receiving notifications because you authored the thread.


Conversation 0
Commits 1
Checks 1
Files changed 1
+0 −7
[redacted] commented 3 minutes ago
No description provided.
Update main.tf
[redacted] merged commit [redacted] into main now
1 check passed
View details
Revert
[redacted] deleted the decommission-virtual-server branch now
Restore branch
Write
Preview
Leave a comment
Attach files by dragging & dropping, selecting or pasting them.
Comment
ProTip! Add comments to specific lines under Files changed.
Reviewers
No reviews
Assignees
No one—assign yourself
Labels
None yet
Projects
None yet
Milestone
No milestone
Development
Successfully merging this pull request may close these issues.
None yet
Notifications
Customize
Unsubscribe
You're receiving notifications because you modified the open/close state.


Summary
Jobs
build
Run details
Workflow file
build
succeeded 17 minutes ago in 9s
Search logs
Terraform apply
1s
- public_ip_type = "ipv4" -> null
- schedule_list = "disable" -> null
- scripting_flag = "disable" -> null
- ssl_mirror = "disable" -> null
- status = "enable" -> null
- stream_scripting_flag = "disable" -> null
- traffic_log = "disable" -> null
- trans_rate_limit = "0" -> null
- type = "l7-load-balance" -> null
- use_azure_lb_backend_ip = "disable" -> null
- warmrate = "100" -> null
- warmup = "0" -> null
- wccp = "disable" -> null
}
Plan: 0 to add, 0 to change, 7 to destroy.
module.fortiadc.fortiadc_load_balance_pool_child_pool_member.rsps_rss["rsp1_rss1"]: Destroying... [id=rsp1_1]
module.fortiadc.fortiadc_load_balance_pool_child_pool_member.rsps_rss["rsp1_rss2"]: Destroying... [id=rsp1_2]
module.fortiadc.fortiadc_load_balance_virtual_server.vss["vs1"]: Destroying... [id=vs1]
module.fortiadc.fortiadc_load_balance_pool_child_pool_member.rsps_rss["rsp1_rss2"]: Destruction complete after 0s
module.fortiadc.fortiadc_load_balance_pool_child_pool_member.rsps_rss["rsp1_rss1"]: Destruction complete after 0s
module.fortiadc.fortiadc_load_balance_real_server.rss["rs1"]: Destroying... [id=
192.168.1.1-2
module.fortiadc.fortiadc_load_balance_real_server.rss["rs2"]: Destroying... [id=
module.fortiadc.fortiadc_load_balance_virtual_server.vss["vs1"]: Destruction complete after 0s
module.fortiadc.fortiadc_load_balance_client_ssl_profile.ssl_profs["ssl_prof1"]: Destroying... [id=ssl_prof1]
module.fortiadc.fortiadc_load_balance_pool.rsps["rsp1"]: Destroying... [id=rsp1]
module.fortiadc.fortiadc_load_balance_real_server.rss["rs1"]: Destruction complete after 0s
module.fortiadc.fortiadc_load_balance_client_ssl_profile.ssl_profs["ssl_prof1"]: Destruction complete after 0s
module.fortiadc.fortiadc_load_balance_real_server.rss["rs2"]: Destruction complete after 0s
module.fortiadc.fortiadc_load_balance_pool.rsps["rsp1"]: Destruction complete after 0s
Apply complete! Resources: 0 added, 0 changed, 7 destroyed.
Post Run actions/checkout@v3
0s
Complete job
0s