Internship Project Report

STUDY OF HIGH AVAILABILITY APPROACHES IN PUBLIC CLOUD ENVIRONMENT

Ms Tanushree Mondal

Intern

Jio Platforms Limited

24th June, 2024 - 24th September, 2024

tanushreemondal095@gmail.com

TABLE OF CONTENTS

Acknowledgement

The successful completion of this project would not have been possible without the guidance and support of several individuals. I am incredibly fortunate to have received invaluable assistance throughout the entire process, and I extend my deepest gratitude to everyone who contributed to its success.

First and foremost, I would like to express my sincere thanks to **Hardik Bavishi** for granting me the opportunity to work on this project as a part of his team at Jio Platforms Limited. His leadership and vision were instrumental in shaping my work.

I owe a special debt of gratitude to my project guide, **Gaurav Sharma**, for his unwavering dedication and insightful guidance. His keen interest in my project and continuous support were crucial to its successful completion. I deeply appreciate the wealth of information and direction he provided, which helped me develop a well-rounded project.

I am especially grateful to **Kapil M Malviya** for his insights throughout the project. His expertise played a significant role in the development of this work.

I would also like to extend my heartfelt thanks to **Medha Surendran** and **Kundrapu Abhinav Sadwik** for their consistent support and encouragement, despite their busy schedules. Their assistance was invaluable to me throughout this journey.

Lastly, I would like to thank **Sunita Autade** for her unflagging encouragement and timely support. I greatly appreciate her belief in my abilities and her efforts in providing me with this internship opportunity in such a prestigious organization.

I am truly grateful to everyone for their contributions and support.

Background

Journey of telecom: A success story

Initially, telecommunications relied heavily on proprietary hardware, characterized by monolithic architectures and dedicated systems for specific functions. This setup was costly and complex, requiring extensive physical infrastructure and specialized expertise to maintain. As technology progressed, the industry began to adopt Commercial Off-The-Shelf (COTS) servers and virtualization. This shift allowed telecom operators to replace expensive proprietary equipment with more flexible and cost-effective solutions. The next phase involved the integration of cloud computing into telecom operations. This transition has been driven by the need for greater agility, lower costs, and enhanced service offerings. Telecom companies began partnering with major cloud service providers like AWS and Google Cloud Platform to leverage public cloud infrastructures, enabling them to offer services like 5G and edge computing more effectively. Looking ahead, the telecom industry is increasingly adopting cloud-native architectures that utilize microservices, containers, and orchestration tools like Kubernetes. This approach enhances resilience and allows for rapid deployment of new features while maintaining high service levels. The move towards a fully cloud-native environment is seen as essential for supporting emerging technologies such as 5G and AI-driven services. The journey from proprietary hardware to cloud-based solutions marks a transformative era in telecommunications. As operators continue to embrace cloud-native technologies, they will enhance their ability to meet evolving customer needs while driving innovation across the industry.

How is telecom converging in public cloud?

Telecom companies play a crucial role in connecting people and enhancing the lives of millions. The importance of telecom services has become clear, especially in the time of the COVID-19 crisis. Telecom operators have facilitated remote work and studying and helped people around the world stay connected. They have stepped up their support for national healthcare systems, national and local governments, and corporate customers to keep vital processes going. With cloud computing, the telecom industry has become more digital and now can reach more customers, offer more stable internet connections, provide valuable insights, and deliver its services via several devices (not only via mobile or TV but also via a laptop or tablet), etc.

The convergence of telecommunications and public cloud computing is reshaping the industry, enabling telecom operators to enhance their services, improve operational efficiency, and respond more effectively to market demands. Here are the key aspects of this convergence:

- **Migration of Network Functions**
  Telecom companies are increasingly moving their network workloads, including Virtual Network Functions (VNFs) and Cloud-native Network Functions (CNFs), to public cloud platforms.
- **Partnerships with Hyperscalers**
  Telecom operators are forming strategic partnerships with hyperscale cloud providers like AWS, Google Cloud, and Microsoft Azure. These collaborations enable telcos to utilize the extensive infrastructure and resources of hyperscalers, facilitating faster deployment of services such as Multi-Access Edge Computing (MEC).
- **Enhanced Service Offerings**

By migrating to the public cloud, telecom companies can introduce new services more rapidly and efficiently. The programmable nature of cloud-native architectures allows for dynamic scaling and automation, enabling operators to customize solutions for various industries.

- **Cost Efficiency and Resource Optimization**
  Cloud computing reduces capital expenditures associated with maintaining physical infrastructure. This shift not only lowers operational costs but also enhances overall service delivery capabilities.
- **Agility and Innovation**
  The transition to cloud environments fosters greater agility within telecom operations. By adopting microservices architectures and containerization, telecoms can develop, test, and deploy new features more rapidly without disrupting existing services.

Why is the high availability feature important for telecom?

High availability (HA) is crucial in the telecommunications industry for several reasons, primarily revolving around ensuring uninterrupted service, maintaining customer satisfaction, and safeguarding operational integrity.

- **Uninterrupted Service**:
  Telecom networks are expected to provide continuous connectivity and service. High availability ensures that systems can operate without interruption, which is vital for both consumer applications and critical services like emergency communications. Achieving ***"five nines"*** availability (99.999%) means that a service is down for no more than approximately 5.26 minutes per year, which is essential for maintaining reliability in telecommunications
- **Financial Impact**
  Network outages can lead to significant financial losses for both service providers and their customers. For businesses relying on telecom services, even brief interruptions can disrupt operations, resulting in lost revenue and reduced productivity. High availability solutions help mitigate these risks by ensuring that services remain accessible, thus protecting revenue streams.
- **Critical Infrastructure Support**
  Many sectors, including healthcare, finance, and public safety, depend on telecom networks for their operations. High availability is essential in these contexts to ensure that critical communications are maintained at all times. For instance, healthcare systems require reliable connectivity for patient monitoring and emergency response systems.
- **Competitive Advantage**
  In a highly competitive market, telecom providers must differentiate themselves through superior service reliability. Offering high availability can be a key selling point that attracts and retains customers. Organizations that achieve high levels of availability can avoid the negative publicity associated with service outages and enhance their reputation in the market.
- **Technological Resilience**
  High availability systems are designed with redundancy and failover capabilities to quickly recover from hardware or software failures. This resilience allows telecom networks to handle unexpected loads or component failures without significant disruption. Implementing HA strategies helps ensure that telecom infrastructure can adapt to changing conditions while maintaining service quality.
- **User Trust and Reliability**

Consistent service availability fosters trust among users. When customers know they can rely on telecom services without frequent interruptions, it enhances their overall experience and confidence in the provider. This trust is particularly important in sectors where communication is critical, such as finance or healthcare. In summary, high availability is essential for telecom services as it ensures continuous operation, protects against financial losses, supports critical infrastructure, provides a competitive edge, enhances technological resilience, and builds user trust.

In summary, high availability is not just a technical requirement but a strategic necessity in the telecommunications sector. It enhances customer satisfaction, protects financial interests, supports critical services, and safeguards the reputation of service providers

Company Profile

Jio Platforms Limited

Jio Platforms Ltd. (JPL) is an Indian multinational technology company and a subsidiary of Reliance Industries Limited, headquartered in Mumbai, India. Jio Platforms' 5G solutions enable operators to evolve to new 5G capabilities. Its end-to-end 5G portfolio spans radio, core, automation, OSS/BSS, and AI/ML platforms along with network services. The 5G stack is also uniquely positioned to implement innovative 5G use cases for enterprises and private networks, with deployment options at the edge as well in the public/private cloud. In addition, JPL is engaged in active research at its in-house Jio Labs facility to make 6G a reality.

Jio Platforms Limited (Jio) is a subsidiary of Reliance Industries Limited (RIL), India's largest conglomerate, and serves as the technology arm focusing on digital services and telecommunications. Established in 2019, Jio Platforms has quickly become a dominant force in India’s digital ecosystem, revolutionizing the way millions of Indians access the internet, entertainment, and digital services.

Core Business Areas:

1. Telecommunications:

Jio Platforms Limited, a subsidiary of Reliance Industries, is a leading player in the telecommunications sector in India, revolutionizing the digital landscape since its launch in 2016. With a robust LTE network covering all 22 telecom circles, Jio offers a range of services, including 4G and 5G connectivity, alongside innovative solutions like JioFiber and JioAirFiber. The company has rapidly grown to become the largest mobile network operator in India, boasting over 467 million subscribers. Jio's commitment to affordability and accessibility has transformed the telecom market, making high-speed internet accessible to millions. Furthermore, Jio is actively expanding its capabilities with investments in advanced technologies and partnerships with global firms, positioning itself at the forefront of India's digital transformation.

2. Digital Services:

Jio Platforms integrates a wide range of digital services, offering everything from e-commerce, entertainment, and cloud computing to IoT, artificial intelligence, and blockchain solutions. Some notable services include:

- Jio provides high-speed mobile broadband services, including 4G and 5G connectivity, enabling seamless voice and data communication across the country.
- JioFiber: High-speed broadband internet services.
- JioTV and JioCinema: On-demand video streaming platforms (OTT).
- JioMoney: Digital payments.
- Jio offers cloud-based solutions for businesses, including JioBusiness which integrates connectivity with productivity tools like Microsoft Office 365.
- JioSaavn: A music streaming platform.
- JioMart: E-commerce platform in partnership with Reliance Retail.

- JioMeet: Video conferencing service.
- JioChat: Messaging.
- JioGamesCloud: a platform for cloud gaming, allowing users to play games without the need for high-end hardware.
- JioSign: sign documents digitally, streamlining processes for businesses and individuals alike.
- JioSphere: A web browser that enhances users' online experience while ensuring privacy and security.

3. Cloud Computing and Edge Computing:

Through partnerships with tech giants like Google and Microsoft, Jio Platforms is developing cloud infrastructure, edge computing, and AI-based solutions, making significant investments in data centres and cloud services to cater to India's growing digital economy. Jio Platforms Limited is at the forefront of cloud computing and edge computing in India, leveraging its extensive telecommunications infrastructure to provide innovative digital solutions. Through its partnership with Microsoft Azure, Jio has developed JioCloud, a robust cloud platform that offers scalable and secure computing resources tailored for businesses and developers. This cloud infrastructure enables organizations to efficiently manage workloads while benefiting from enhanced data security and reduced operational costs. Additionally, Jio is pioneering edge computing solutions that bring computation and data storage closer to the end-users, significantly reducing latency and improving real-time data processing capabilities. By integrating cloud and edge technologies, Jio Platforms is empowering various sectors—including healthcare, education, and entertainment—to harness the power of advanced computing, thereby accelerating digital transformation across India.

Global Investments:

Jio Platforms Limited has made significant strides in attracting global investments, positioning itself as a leader in the digital services sector.

- **Major Investments from Global Firms**: Jio Platforms has successfully raised substantial capital from prominent global investors. In May 2020 alone, it secured investments totalling approximately **₹78,562 crore** from various firms, including General Atlantic, KKR, and Vista Equity Partners. These investments have been instrumental in enhancing Jio's technological capabilities and expanding its digital ecosystem.
- **Focus on Technology and Innovation**: The investments have enabled Jio to enhance its technological infrastructure, including advancements in broadband connectivity, cloud computing, artificial intelligence, and Internet of Things (IoT) solutions. This focus on innovation positions Jio as a key player not only in India but also in potential international markets.

Recently, several prominent global investors have invested in Jio Platforms Limited, reflecting strong confidence in the company's potential and its impact on the digital landscape in India. Here are some of the key investors:

- **Facebook**: Invested approximately **$5.7 billion** for a **10% stake** in Jio Platforms, marking one of the largest investments by a technology company in India.

- **Google**: Committed around **$4.5 billion** for a **7.7% stake**, further solidifying its partnership with Jio to enhance digital services and connectivity in India.
- **KKR**: The global investment firm invested **₹11,367 crore** (approximately **$1.4 billion**) for a **2.32% stake**, emphasizing its belief in Jio's growth trajectory.
- **Vista Equity Partners**: Also acquired a **2.3% stake** in Jio Platforms, contributing to the overall investment strategy aimed at expanding Jio's digital ecosystem.
- **General Atlantic**: Invested **₹6,598 crore** (around **$790 million**) for a **1.34% stake**, supporting Jio's mission to build a digital society in India.
- **Silver Lake Partners**: Invested around **$1 billion**, acquiring a **2% stake**, which underscores their commitment to backing innovative technology firms.
- **TPG Capital**: Acquired a nearly **1% stake**, reflecting continued interest from private equity firms in Jio's growth potential.
- **Mubadala Investment Company**: The Emirati sovereign fund invested approximately **$1 billion** for a **1.85% stake**, showcasing international interest in sovereign wealth funds.
- **Abu Dhabi Investment Authority (ADIA)**: Acquired a **1.16% stake**, further diversifying the investor base supporting Jio Platforms.

These investments collectively highlight the confidence that global investors have in Jio Platforms as it continues to transform India's telecommunications and digital services landscape, paving the way for significant advancements in connectivity and technology adoption across the country.

Recent Developments & Key Contributions:

Reliance Jio Platforms Limited has made significant contributions to the 5G sector in India, positioning itself as a leader in telecommunications. Here are the key contributions:

- **Subscriber Growth:** Jio has rapidly expanded its 5G subscriber base, reaching 130 million users as of June 2024, with a notable addition of 22 million subscribers in just one quarter. This growth accounts for approximately 31% of Jio's overall wireless data traffic, demonstrating the widespread adoption of its 5G services.
- **Innovative Technology**: Jio has developed a proprietary cloud-native 5G stack that integrates seamlessly with its existing 4G network. This technology allows Jio to manage a substantial portion of its data traffic efficiently and supports the deployment of advanced services.
- **Infrastructure Development**: The company has rolled out its 5G wireless broadband service, JioAirFiber, across nearly 5,900 cities and towns, aiming for pan-India coverage. This extensive infrastructure development supports both mobility and fixed wireless services, significantly enhancing connectivity options for consumers.
- **Global Ambitions**: With over 1,000 patents filed in fiscal year 2024 covering various technologies including 5G and IoT, Jio is positioning itself for international expansion. The company aims to leverage its indigenously developed technologies to capture markets outside India.
- **Partnerships and Collaborations**: Jio has partnered with leading technology firms, such as Qualcomm, to enhance its 5G capabilities. These collaborations focus on integrating advanced technologies like AI and IoT into its network infrastructure, further boosting the quality and reach of its services.

- **Increased Data Consumption**: The introduction of 5G has led to a significant rise in data consumption among users, with per capita usage reaching 30.3 GB per month in Q1 FY25—an increase attributed to the enhanced capabilities of 5G networks.

Through these initiatives, Reliance Jio Platforms Limited is not only transforming the telecommunications landscape in India but also setting a precedent for future technological advancements in the global market.

Mission and Vision:

Reliance Jio Platforms Limited has articulated a clear mission and vision that reflects its commitment to transforming India's digital landscape.

Jio's mission is to **enable digital life for every Indian** by providing affordable access to high-quality digital services. The company aims to meet the growing demand for connectivity and empower individuals and businesses through innovative technologies, ensuring that everyone can participate in the digital revolution.

Jio's vision is to **transform India through the power of digital technology**, connecting everyone and everything, everywhere, at the highest quality and most affordable prices. The company envisions creating a comprehensive digital ecosystem that enhances productivity, fosters economic growth, and bridges gaps in education and healthcare, ultimately shaping a better future for all Indians. These guiding principles underscore Jio's role as a catalyst for change in India's telecommunications sector and its broader commitment to societal development through digital inclusion.

Impact on India's Digital Landscape:

Jio has played a pivotal role in India's digital transformation by:

- Bringing high-speed, low-cost data to millions of users.
- Catalyzing the growth of digital services across education, healthcare, entertainment, and e-commerce.
- Developing a suite of innovative services that empower both businesses and consumers to thrive in the digital economy.

The impact of 5G technology on India's digital landscape is profound, driven significantly by the contributions of the telecom giant Reliance Jio. Here are the key impacts:

- **Enhanced Connectivity and Speed**: The deployment of 5G has dramatically improved internet speeds and connectivity across India. As of early 2024, India ranked 14th globally in 5G median download speeds, achieving impressive speeds of approximately 301.86 Mbps. This leap in performance facilitates better user experiences in streaming, gaming, and other data-intensive applications.
- **Increased Data Consumption**: Jio's rollout of 5G has led to a surge in data consumption, with average monthly data usage per subscriber skyrocketing from 240 MB to 17.4 GB since the

introduction of its 4G services. This shift has transformed India into a "Data Rich" nation, significantly increasing its share of global mobile data traffic from around 3% to 20%.

- **Economic Growth and Digital Services**: The widespread availability of high-speed internet through 5G is set to accelerate economic growth by enabling innovative applications across sectors such as smart cities, telemedicine, and e-education. This technological advancement supports the development of a more connected economy, enhancing overall productivity and service delivery.
- **Improved Consumer Experience**: With faster video start times and reduced latency in mobile gaming, users are experiencing superior digital services compared to previous generations of mobile technology. For instance, Jio's 5G network has achieved video start times as low as 1.14 seconds, significantly enhancing user satisfaction.
- **Increased Accessibility**: The introduction of affordable 5G-capable devices has contributed to the growing adoption of 5G technology in India. As more consumers gain access to these devices, the potential for widespread digital inclusion increases, bridging gaps in connectivity across urban and rural areas.
- **Future Innovations**: The foundation laid by Jio's extensive 5G network enables future technological innovations such as augmented reality (AR), virtual reality (VR), and artificial intelligence (AI). These technologies can further enhance user experiences and drive new business models across various industries.

In summary, the impact of Jio's contributions to the 5G sector is reshaping India's digital landscape by improving connectivity, driving economic growth, enhancing consumer experiences, and laying the groundwork for future technological advancements.

Thus, we can rightly say that Jio Platforms Limited is not just a telecommunications giant but a key player in India’s digital future. Through its broad portfolio of digital services and global partnerships, Jio aims to create a robust digital infrastructure that drives innovation, enhances connectivity, and elevates the quality of life for millions of Indians.

Abstract

In an increasingly digital world, ensuring uninterrupted access to critical applications and services is paramount for organizations. As the telecommunications industry undergoes a significant transformation through the convergence with public cloud environments, the demand for high availability (HA) has become increasingly critical. With telecom operators migrating core network functions to cloud infrastructures, ensuring uninterrupted service delivery is essential for maintaining customer trust and operational integrity. The project, titled **"Study of High Availability Approaches in Public Cloud Environment"** explores various strategies to enhance application redundancy and minimize downtime in a cloud infrastructure. The focus is on the setup of a robust cloud environment using Google Cloud to implement and evaluate high-availability solutions. The project involved the creation and configuration of critical network components and compute instances. These instances were organized into primary and failover instance groups. A key aspect of the project was the implementation of a failover policy to ensure seamless failover during network disruptions. The project also involved conducting failover testing using three techniques: load balancer configuration, alias IP failover, and route modification. The performance of each technique was assessed by comparing average latencies across all three scenarios. This study underscores the significance of automation in achieving high availability and application resilience within cloud environments. The findings provide valuable insights into the efficiency, latency, and overall performance of different failover mechanisms, highlighting the optimization of network reliability and reduction in service downtime.

Tanushree Mondal

Project Mentor:

Gaurav Sharma

Objective

The prime objective of this project, **"Study of High Availability Approaches in Public Cloud Environment"** is to study, investigate and implement high availability mechanisms in a public cloud environment, specifically using Google Cloud, by configuring critical network components and evaluating failover mechanisms to enhance application redundancy and minimize downtime during network disruptions. The project aims to evaluate different failover mechanisms—including load balancer configuration, alias IP failover, and route modification—by comparing their performance, particularly in terms of latency. The overarching goal is to provide insights into optimizing cloud infrastructure for reliability, resilience, and efficient failover handling through HA methods.

Introduction

What is High Availability?

High availability (HA) in cloud computing refers to the ability of a system or service to remain operational and accessible for a high percentage of time. It aims to ensure that cloud resources are consistently available to users, minimizing downtime and service interruptions. High-availability infrastructure is configured to deliver quality performance and handle different loads and failures with minimal or zero downtime.

Source: https://www.filecloud.com/blog/an-introduction-to-high-availability-architecture/

Why do we require High Availability features?

High availability is important for mission-critical systems that cannot tolerate interruption in service, and any downtime can cause damage or result in financial loss.
Highly available systems guarantee a certain percentage of uptime—for example, a system that has 99.9% uptime will be down only 0.1% of the time—0.365 days or 8.76 hours per year.

Basic elements of high availability

The following elements are essential to a highly available system:

- **Redundancy**—ensuring that any elements critical to system operations have an additional, redundant component that can take over in case of failure.
- **Monitoring**—collecting data from a running system and detecting when a component fails or stops responding.
- **Replication**- Data needs to be replicated and shared with the same nodes in a cluster.
- **Failover**—a mechanism that can switch automatically from the currently active component to a redundant component, if monitoring shows a failure of the active component.
- **Fault Tolerance**: Fault tolerance mainly aims to deliver minimal downtimes (ideally zero).

Technical components enabling high availability:

- Data Backup and recovery
- Load Balancing
- Clustering

Methodology

The methodology of the project involves a structured approach to designing, automating, and testing a high-availability cloud infrastructure. The key steps are outlined below:

1. Planning and Design:

- **Defining the Architecture:** The cloud infrastructure was designed to include key components such as a Virtual Private Cloud (VPC), subnets, firewall rules, routers, Network Address Translation (NAT), and compute instances. For redundancy, instances were grouped into primary and failover instance groups.
- **Identifying Failover Strategies:** Three different failover mechanisms, load balancer configuration, alias IP failover, and route modification, were selected for evaluation. The goal was to assess their performance in terms of availability and latency.

2. Script Development:

- **Automating Cloud Resource Creation:** An automation script was developed to streamline the process of creating and configuring the cloud components. The script automated the setup of the VPC, subnets, firewall rules, routers, NAT, and instances.
- **Instance Group Configuration:** The automation also involved organizing the compute instances into primary and failover instance groups, with each group attached to a backend service for load balancing.

3. Implementation of High-Availability Features:

- **Failover Policy Configuration:** A failover policy was implemented to handle traffic redirection during disruptions. This involved setting up health checks to monitor the instances and ensuring that traffic would automatically be rerouted to the failover instance group in case of failures.
- **Load Balancer and Forwarding Rule Setup:** An internal IP was reserved for load balancing purposes, and a forwarding rule was configured to route traffic between instances based on the load balancer's decision.

4. Failover Testing methods:

Failover testing was conducted using three methods.

- **Load Balancer Configuration:** Testing the failover using the load balancer’s built-in mechanism.
- **Alias IP Failover:** Testing how effectively alias IPs can handle traffic redirection during a failure.
- **Route Modification:** Testing manual route modification to ensure traffic is rerouted to healthy instances.

5. Performance Testing:

Performance Testing was done for each method where the average latency during failover was measured and recorded.

- **Comparing Latencies:** The latencies of all three failover mechanisms were compared to assess the efficiency, speed, and reliability of each approach.
- **Analyzing Results:** The results were analyzed to identify the most effective failover strategy in terms of minimizing downtime and reducing latency during network disruptions.

6. Conclusion:

- **Summarizing Findings:** The study provided insights into how different failover mechanisms can impact application availability and network performance in a cloud environment.
- **Optimizing High Availability:** Based on the results, recommendations are made on how to optimize cloud infrastructure for high availability using automation and appropriate failover techniques.

This methodology ensured a systematic approach to creating a resilient cloud application, performing failover testing, and deriving data-driven insights into the performance of high-availability methods.

Tools Used

The following tools and technologies were used during the project work to automate, configure, and test the cloud infrastructure:

1. Google Cloud Platform (GCP):

- **Compute Engine:** For creating and managing virtual machines (VMs) that were organized into primary and failover instance groups.
- **VPC (Virtual Private Cloud):** Used to create isolated network environments, including subnets, firewall rules, routers, and NAT configurations.
- **Cloud Load Balancing:** For distributing traffic across the primary and failover instance groups to ensure high availability and efficient failover.
- **Cloud NAT:** For enabling instances in private subnets to access the internet securely without having public IPs.
- **Cloud Router:** For dynamic routing of traffic between subnets and the internet.

2. Scripting Tools:

- **Google Cloud SDK (gcloud):** Command-line interface (CLI) used to manage and automate the creation of resources in Google Cloud, including VPC, subnets, instances, and load balancers.
- **Bash Scripting:** A shell scripting language used to automate the entire infrastructure setup, including the creation of cloud components, configuration of instances, and failover policies.

3. Networking Tools:

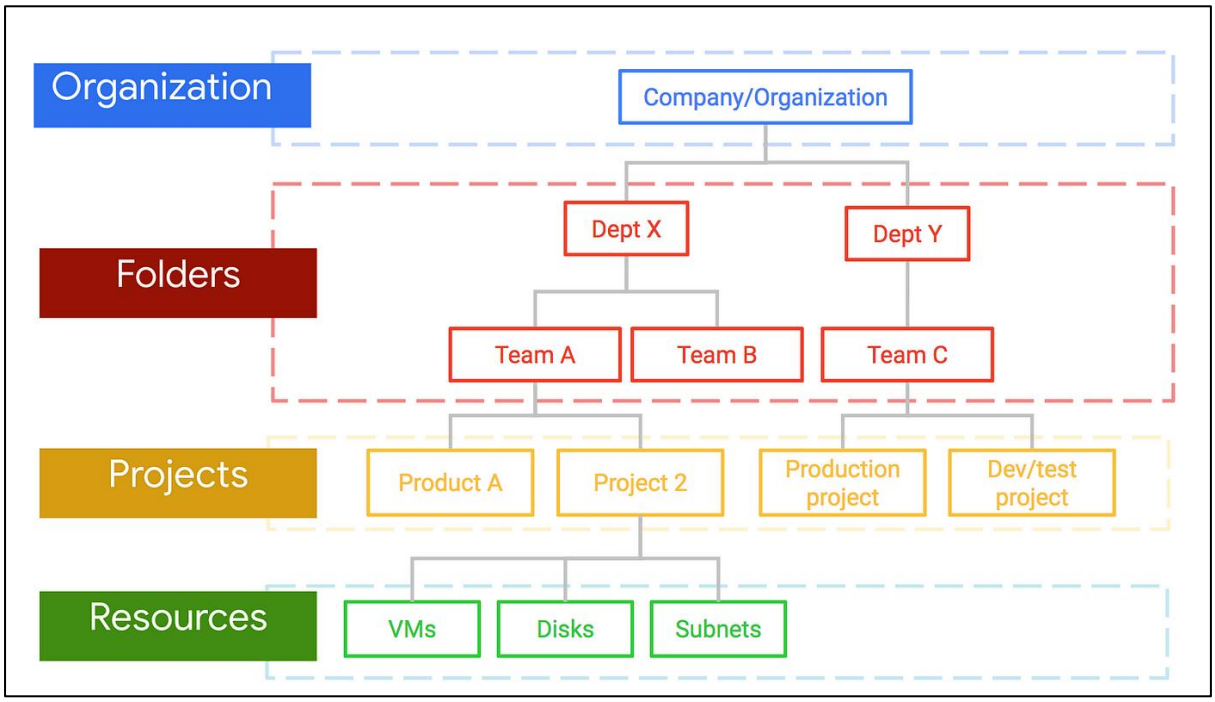
- **Firewall Rules:** Configured to control traffic flow to and from the instances, ensuring only authorized access to the cloud environment.
- **Health Checks:** Configured to monitor the health of instances in both the primary and failover instance groups, ensuring traffic is directed to healthy instances.
- **Forwarding Rules:** Used to define how traffic should be directed to the load balancer and instances based on specific conditions.

4. Performance Testing Tools:

Cloud Monitoring: Used to monitor system performance, instance health, and traffic distribution, providing insights into latency and uptime during failover testing.

These tools were integral in configuring the setup, managing high availability, and performing failover testing in the cloud infrastructure.

HIGH AVAILABILITY – GOOGLE CLOUD PLATFORM



Source: https://cloud.google.com/compute/docs/

Infrastructure

VM availability

In Google Cloud Platform (GCP), VM availability is achieved through a combination of high-availability configurations, managed instance groups, load balancing, and disaster recovery measures.

GCP ensures high VM availability by deploying instances across multiple zones within a region. This zonal redundancy means that if one zone experiences issues, the other zones can continue to operate, providing a safeguard against individual zone failures. For even greater resilience, GCP offers Regional Instances, which span multiple zones, thereby enhancing the reliability of applications.

Managed Instance Groups (MIGs) play a crucial role in maintaining VM availability. MIGs automatically manage and scale instances based on demand and predefined policies. They support auto-healing, which means that if an instance becomes unhealthy, MIGs will replace it with a new one. Additionally, MIGs facilitate rolling updates, allowing for incremental updates to instances with minimal service disruption.

Load balancing is another key feature that contributes to VM availability. GCP's load balancers can distribute traffic across multiple VMs, which can be located in different regions. This global load balancing capability ensures that even during high traffic loads or regional failures, traffic is routed to healthy instances, maintaining service continuity.

To further support VM availability, GCP offers tools for backup and disaster recovery. Persistent disk snapshots allow you to create backups of your VM data, which can be restored if necessary. Additionally, multi-region deployments and cross-region replication provide strategies to ensure that your applications remain operational even in the event of a major regional outage.

Network configuration also plays a role in ensuring VM availability. Private Google Access allows VMs to securely access Google services without relying on public IP addresses, enhancing both security and availability. Firewall rules are used to control access, ensuring that VMs are protected from unauthorized access while remaining accessible to legitimate users.

Overall, GCP's comprehensive approach to VM availability combines strategic deployment, automated management, load balancing, and robust backup and recovery solutions to ensure that your applications remain resilient and reliable.

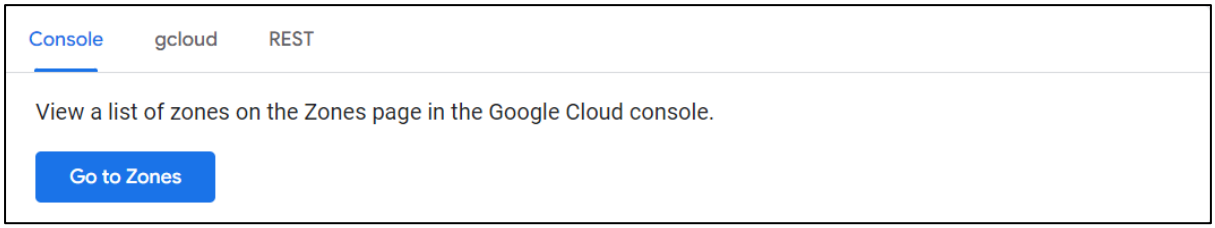
Regions and Zones

- **Region**: *Regions* are collections of zones. To deploy fault-tolerant applications that have high availability, Google recommends deploying applications across multiple zones and multiple regions.
  For example, if you only have customers in the US, or if you have specific needs that require your data to live in the US, it makes sense to store your resources in zones in the US-central1 region or zones in the US-east1 region.
- **Zone**: A *zone* is a deployment area within a region. Zones have high-bandwidth, low-latency network connections to other zones in the same region. The fully-qualified name for a zone is made up of <region>-<zone>. For example, the fully qualified name for zone a in region us-central1 is us-central1-a.

Choosing a region or zone is very important for several reasons:

- **Handling failures:** Google designs zones to minimize the risk of correlated failures caused by physical infrastructure outages like power, cooling, or networking. Thus, if a zone becomes unavailable, you can transfer traffic to another zone in the same region to keep your services running. Similarly, you can mitigate the impact of a region outage on your application by running backup services in a different region.
- **Decreased network latency:** To decrease network latency, you might want to choose a region or zone that is close to your point of service.

View a list of available zones:



View a list of GPU zones:

Search by GPU type

To search by GPU type, use the *gcloud compute accelerator-types list* command with the *--filter* flag to find available zones.

For example, to find all zones with H100 80GB GPUs, run the following command:
gcloud compute accelerator-types list --filter="nvidia-h100-80gb"

```
tanushree1_mondal@cloudshell:~ (jio-cloud-training)$ gcloud compute accelerator-types list --filter="nvidia-h100-80gb"
NAME: nvidia-h100-80gb
ZONE: us-central1-a
DESCRIPTION: NVIDIA H100 80GB

NAME: nvidia-h100-80gb
ZONE: us-central1-b
DESCRIPTION: NVIDIA H100 80GB

NAME: nvidia-h100-80gb
ZONE: us-central1-c
DESCRIPTION: NVIDIA H100 80GB

NAME: nvidia-h100-80gb
ZONE: europe-west1-b
DESCRIPTION: NVIDIA H100 80GB

NAME: nvidia-h100-80gb
ZONE: europe-west1-c
DESCRIPTION: NVIDIA H100 80GB
```

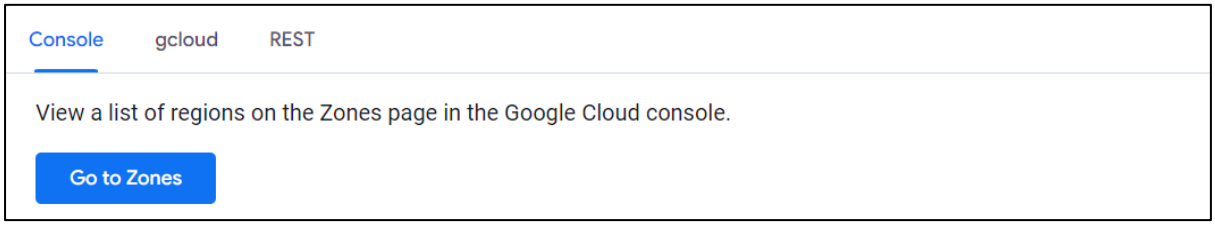
The output returns a list of available GPUs organized by zone. You can then use the *gcloud compute accelerator-types describe* command to get a description of each GPU model returned.

Search by machine type

A3, A2, or G2 accelerator-optimized machine types have GPUs automatically attached to the VMs. For these machine types, you can use the *gcloud compute machine-types list* command with the --filter flag to find available zones. For example, to find all zones with A3 standard machine types, run the following command:
gcloud compute machine-types list --filter="name=a3-highgpu-8g"

```
tanushree1_mondal@cloudshell:~ (jio-cloud-training)$ gcloud compute machine-types list --filter="name=a3-highgpu-8g"
NAME: a3-highgpu-8g
ZONE: us-central1-a
CPUS: 208
MEMORY_GB: 1872.00
DEPRECATED:

NAME: a3-highgpu-8g
ZONE: us-central1-b
CPUS: 208
MEMORY_GB: 1872.00
DEPRECATED:

NAME: a3-highgpu-8g
ZONE: us-central1-c
CPUS: 208
MEMORY_GB: 1872.00
DEPRECATED:

NAME: a3-highgpu-8g
ZONE: europe-west1-b
CPUS: 208
MEMORY_GB: 1872.00
DEPRECATED:
```

View a list of available regions:



We can use the Google Cloud console, the Google Cloud CLI, or REST to see available regions and zones. You can also use the gcloud compute machine-types list command to get a complete list of available machine types in all regions and zones. For example, gcloud compute machine-types list --filter="name=t2d-standard-4" displays all the regions and zones where t2d-standard-4 machine types are available.

Disk

By default, each Compute Engine VM has a single boot disk that contains the operating system. The boot disk data is typically stored on a Persistent Disk or Hyperdisk Balanced volume. When your applications require additional storage space, you can provision one or more of the following storage volumes to your VM.

To learn more about each storage option, review the following table:

	Balanced Persistent Disk	SSD Persistent Disk	Standard Persistent Disk	Extreme Persistent Disk	Hyperdisk Balanced	Hyperdisk ML	Hyperdisk Extreme	Hyperdisk Throughput	Local SSDs
Storage type	Cost-effective and reliable block storage	Fast and reliable block storage	Efficient and reliable block storage	Highest performance Persistent Disk block storage option with customizable IOPS	High performance for demanding workloads with a lower cost	Highest throughput storage optimized for machine learning workloads.	Fastest block storage option with customizable IOPS	Cost-effective and throughput-oriented block storage with customizable throughput	High performance local block storage
Minimum capacity per disk	Zonal: 10 GiB Regional: 10 GiB	Zonal: 10 GiB Regional: 10 GiB	Zonal: 10 GiB Regional: 10 GiB	500 GiB	Zonal and regional: 4 GiB	4 GiB	64 GiB	2 TiB	375 GiB, 3 TiB with Z3
Maximum capacity per disk	64 TiB	64 TiB	64 TiB	64 TiB	64 TiB	64 TiB	64 TiB	32 TiB	375 GiB, 3 TiB with Z3
Capacity increment	1 GiB	1 GiB	1 GiB	1 GiB	1 GiB	1 GiB	1 GiB	1 GiB	Depends on the machine type†
Maximum capacity per VM	257 TiB*	257 TiB*	257 TiB*	257 TiB*	512 TiB*	512 TiB*	512 TiB*	512 TiB*	36 TiB
Scope of access	Zone	Zone	Zone	Zone	Zone	Zone	Zone	Zone	Instance
Data redundancy	Zonal and multi-zonal	Zonal and multi-zonal	Zonal and multi-zonal	Zonal	Zonal and multi-zonal	Zonal	Zonal	Zonal	None
Encryption at rest	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Custom encryption keys	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No

Source: https://cloud.google.com/compute/docs/

Clustering/Managed Instance Groups(MIGs)

An instance group is a collection of virtual machine (VM) instances that you can manage as a single entity. Compute Engine offers two kinds of VM instance groups, managed and unmanaged:

- **Managed instance groups(MIGs)** let you operate apps on multiple identical VMs. You can make your workloads scalable and highly available by taking advantage of automated MIG services, typically associated with clustering, including autoscaling, autohealing, regional (multiple zones) deployment, and automatic updating.
- **Unmanaged instance groups** let you load balance across a fleet of VMs, that need not be identical and can be managed by yourself.

Advantages offered by MIGs:

- **High Availability**

➢ **Automatically repairing failed VMs:** If a VM in the group stops, crashes, gets pre-empted (Spot VMs), or is deleted by an action not initiated by the MIG, the MIG automatically recreates that VM based on its original configuration.
➢ **Application-based autohealing:** If an application is not responding on a VM, the MIG automatically recreates that VM for you.
➢ **Regional(multizone) coverage:** Regional MIGs let you spread the app load across multiple zones. This replication protects against zonal failures.
➢ **Load Balancing:** MIGs work with load balancing services to distribute traffic across all of the instances in the group.

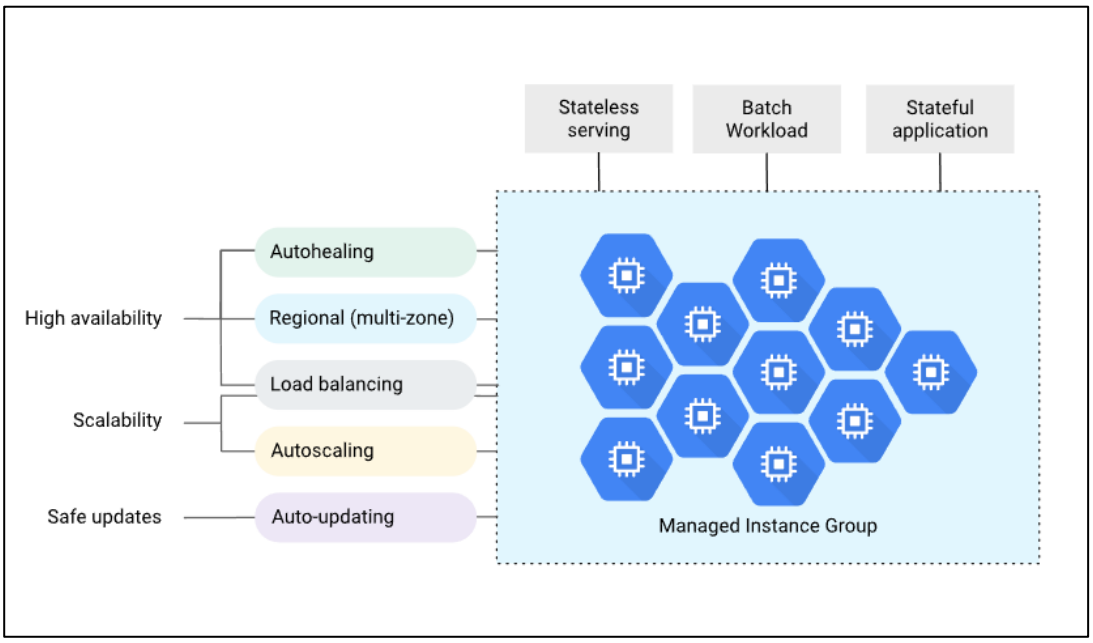
- **Scalability:** When your apps require additional compute resources, auto-scaled MIGs can automatically grow the number of instances in the group to meet demand.

- **Automated updates:** The MIG automatic updater lets you safely deploy new versions of software to instances in your MIG.

- **Support for stateful workloads:** You can use MIGs for building highly available deployments and automating the operation of applications with stateful data or configuration, such as databases, DNS servers, or legacy monolith applications.

- **Create GPU VMs all at once:** When you have a batch job, such as an AI or ML training, that requires an exact number of GPU VMs, then creating a resize request in an MIG can help you to create the VMs all at once.

Overview of MIG capabilities and common workloads: -



Source: https://cloud.google.com/compute/docs/instance-groups

Create a MIG by using the instance template

1. Go to the **Instance groups** page.

   Go to Instance groups

2. Click **Create instance group**, and then perform the following steps:

   a. In the **Name** field, accept the default name or enter `quickstart-instance-group-1`.

   b. In the **Instance template** list, select the instance template that you created earlier.

   c. In the **Location** section, ensure that **Single zone** is selected.

   d. In the **Region** field, select a region where you want to create the MIG.

   e. In the **Autoscaling** section, accept the default settings unless you need to modify them.

3. To create the MIG, click **Create**.

Allow a few minutes for Compute Engine to create the group and its VMs. After the group is ready, it's listed on the **Instance groups** page.

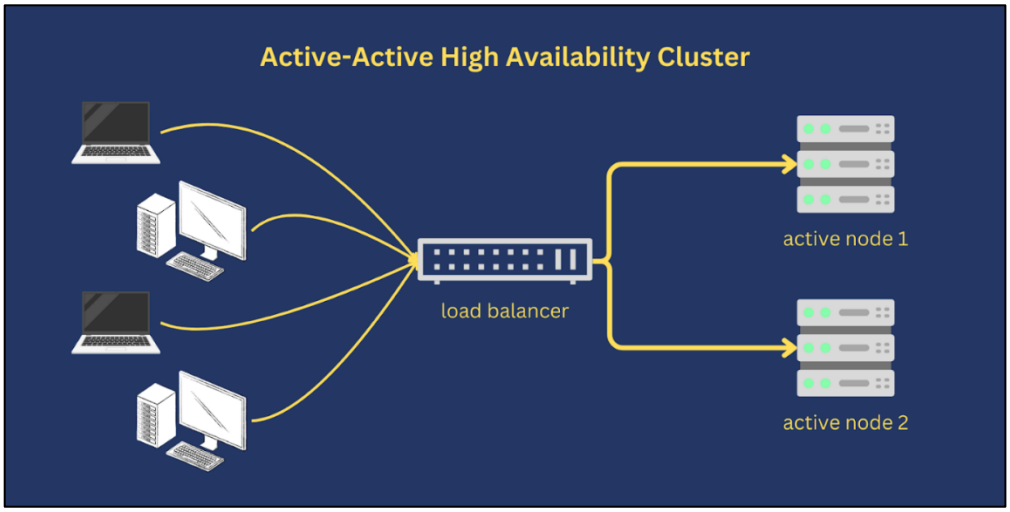
You have successfully created a managed instance group.

Use of multiple application servers

We deploy the same application in multiple VMs, so that if there's an issue in any one of the VMs, then the user will not face any downtime. The load balancer will shift the network traffic between multiple VMs. We need to distribute the load over multiple servers so that none of them is overburdened and the output is optimum. We can also deploy parts of our application on different servers. For instance, there could be a separate server for handling mail or a separate one for processing static files like images (like a Content Delivery Network).

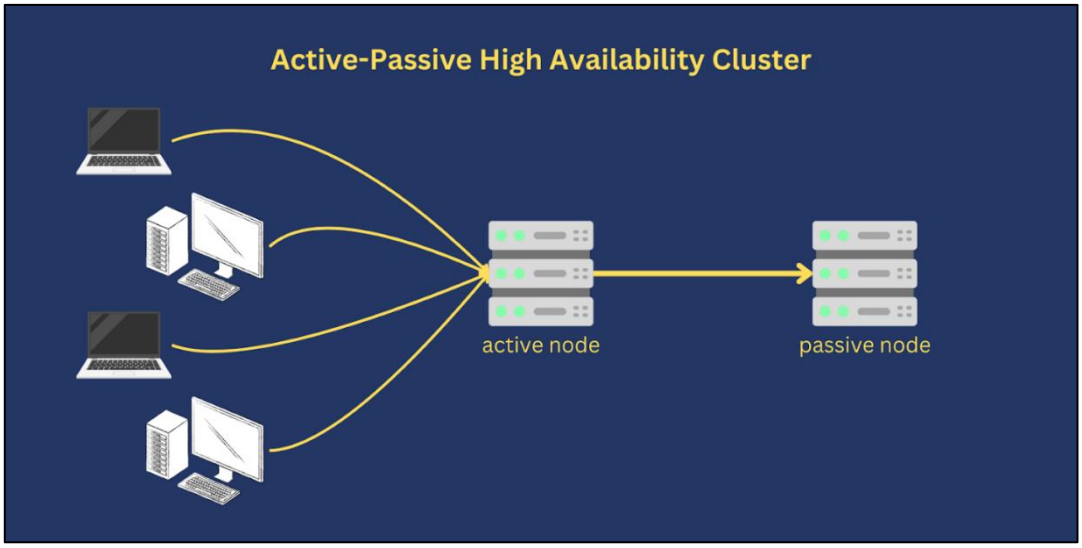
There are different approaches to minimizing downtime. Two common models are:

- **Active-active:** In an active-active high availability cluster, network traffic is shared across two or more servers. The goal is to achieve load balancing. Instead of connecting directly to a specific server, web clients are routed through a load balancer that uses an algorithm to assign the client to one of the servers. This helps any one network node from getting overloaded.



Source: https://www.jscape.com/blog/active-active-vs-active-passive-high-availability-cluster

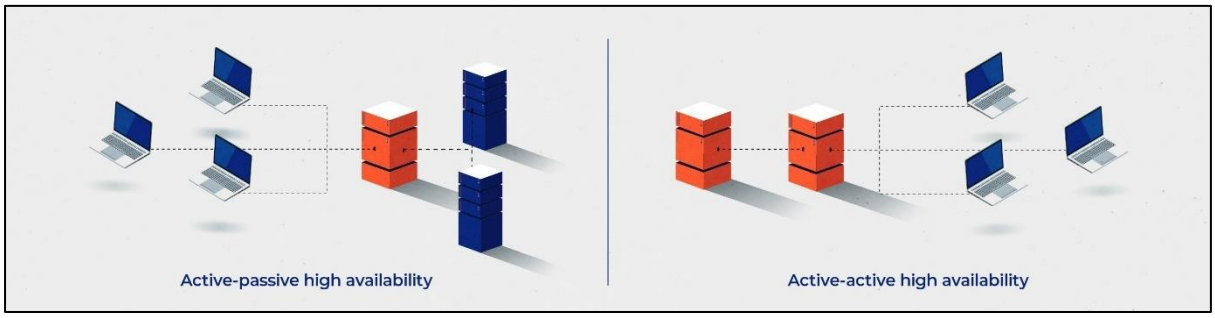
- **Active-passive:** Also known as active-standby, this type of high availability cluster also has at least two servers. When one is active, the other is on standby. All clients connect to the same server. If the primary system fails, the passive server takes over.



Source: https://www.jscape.com/blog/active-active-vs-active-passive-high-availability-cluster

Both models rely on redundancies. Active-passive setups use N+1 redundancy, in which each type of resource has a dedicated backup component so that you always have sufficient resources in place for existing demand. Active-active systems have 1+1 redundancy, in which backup, redundant components are always in operation. If a single point of failure occurs, parallel active resources take its place.

The following diagram shows if we need to support high traffic volume, then an active-active cluster would be more suited for this case:



Source: https://www.jscape.com/blog/active-active-vs-active-passive-high-availability-cluster

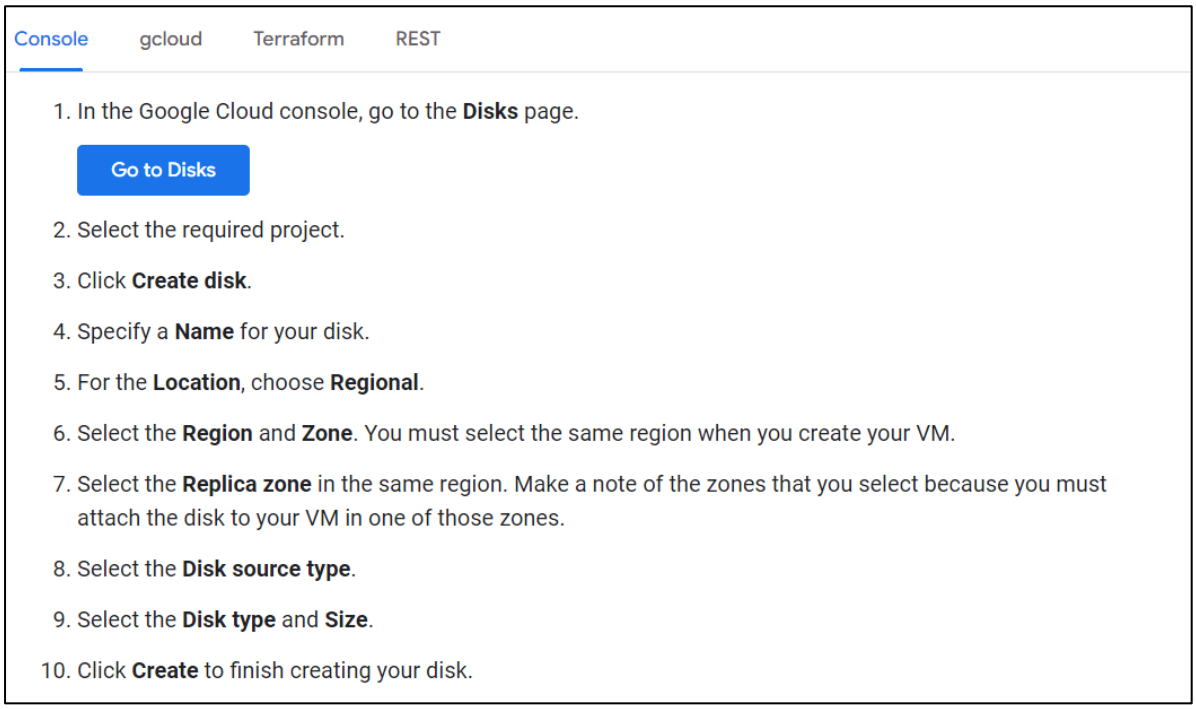
Split brain architecture refers to a scenario in distributed computing where a network partition causes different parts of a distributed system to operate independently, leading to inconsistencies and

potential data loss. In the context of Google Cloud Platform (GCP), mitigating split-brain scenarios involves designing architectures that ensure high availability and consistency despite potential network partitions. Thus, this acts as a drawback while using multiple application servers.

Create and manage synchronously replicated disks

Regional Persistent Disk and Hyperdisk Balanced Disk are storage options that let you implement high availability (HA) services in Compute Engine. Regional Persistent Disk and Hyperdisk Balanced High Availability synchronously replicate data between two zones in the same region and ensure HA for disk data for up to one zonal failure. The synchronously replicated disk can be a boot disk or a non-boot disk.

Create a synchronously replicated disk: The disk must be in the same region as the VM that you plan to attach it to. For regional Persistent Disk, if you create a disk in the Google Cloud console, the default disk type is pd-balanced. If you create a disk using the gcloud CLI or REST, the default disk type is pd-standard.

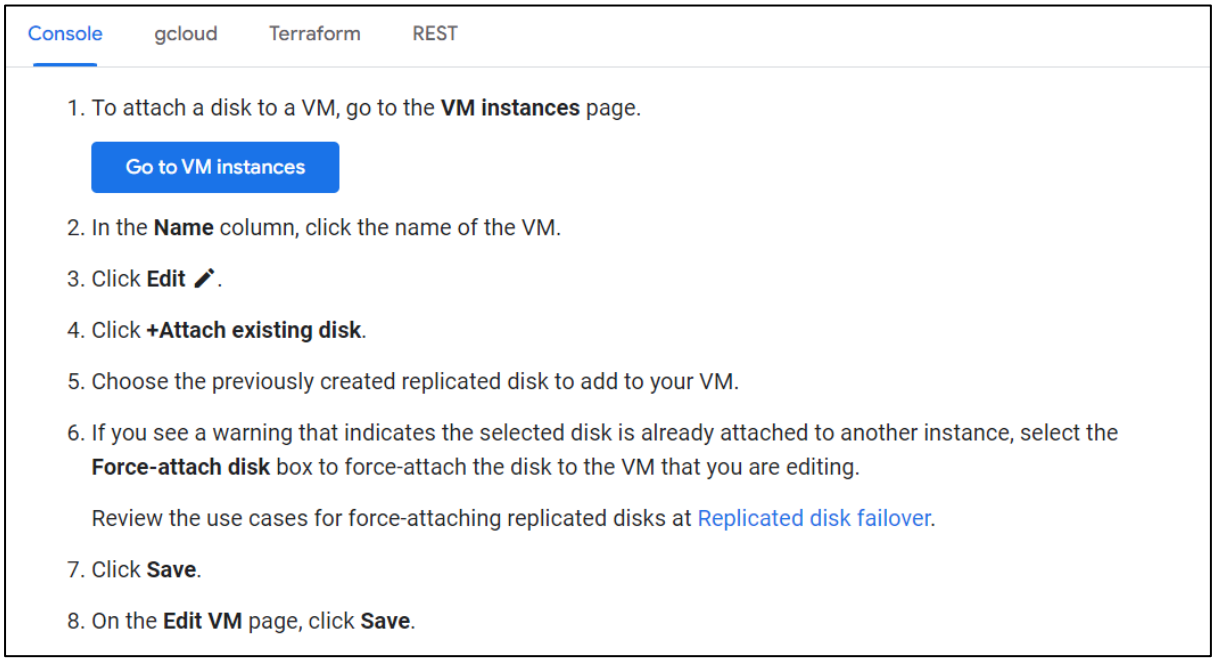


Attach a replicated disk to your VM: For disks that are not boot disks, after you create a regional Persistent Disk or Hyperdisk Balanced High Availability (Preview) volume, you can attach it to a VM. The VM must be in the same region as the disk.

When we avail regional level services in Google Cloud Platform (GCP), we inherently get zone-level redundancy. This ensures higher availability and fault tolerance within a region by distributing your resources across multiple zones.

Compute Engine:

- **Regional Managed Instance Groups:** These allow you to deploy virtual machine instances across multiple zones within a region. This setup ensures that if one zone experiences an outage, instances in other zones can handle the load.
- **Regional Persistent Disks:** These disks are automatically replicated across two zones in the same region, providing redundancy and protecting against zonal failures.



For non-boot disks, after you create and attach a blank replicated disk to a VM, you must format and mount the disk, so that the operating system can use the available storage space.

Create a new VM with a replicated boot disk:

When setting up a highly available VM instance, you can create the primary VM with a replicated boot disk. If a zonal outage occurs, this lets you restart the VM in the secondary zone instead of creating a new VM.

In a high availability setup, where the boot device is a replicated disk, Google recommends that you don't pre-create and start the standby instance. Instead, at the failover stage, attach the existing replicated disk when you create the standby instance by using the forceAttach option.

Use the *gcloud compute instances create* command to create a VM, and the --create-disk flag to specify the replicated disk.

gcloud compute instances create PRIMARY_INSTANCE_NAME \

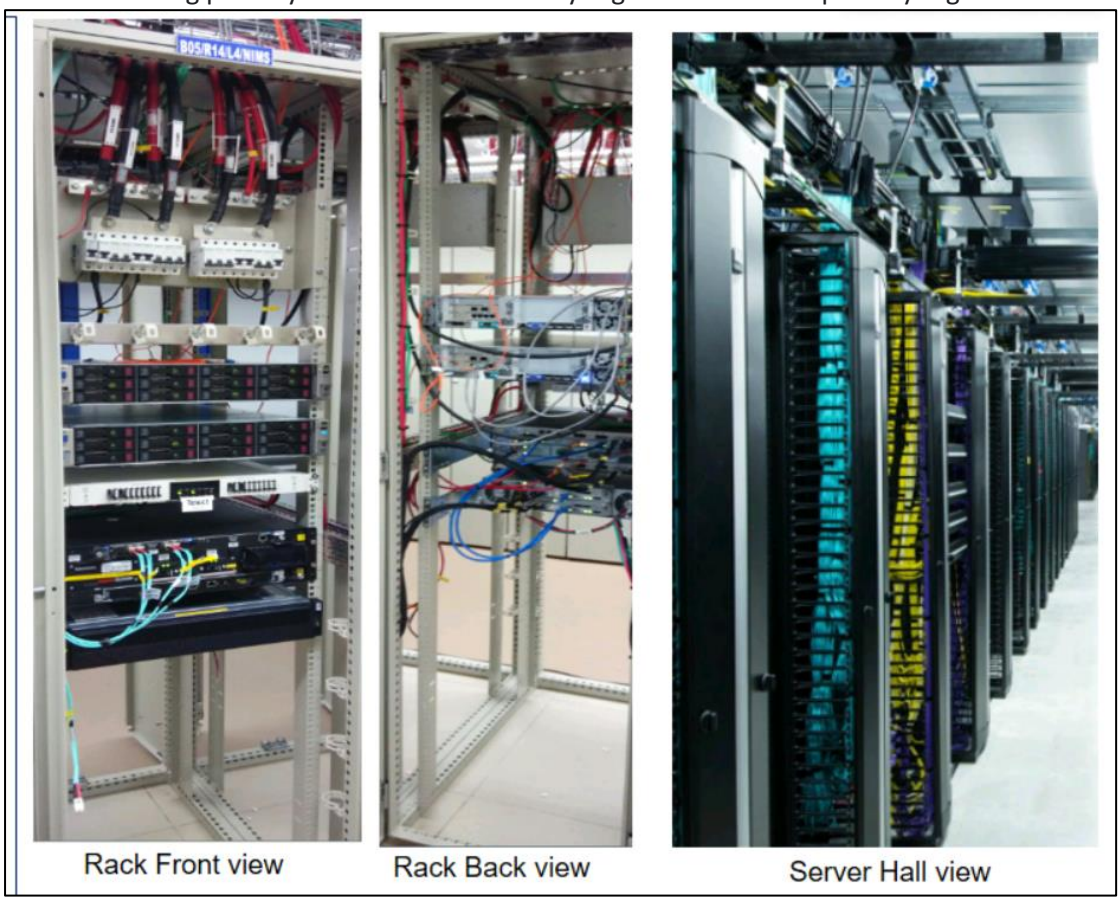
--zone=ZONE \

--create-disk=^:^name=REPLICATED_DISK_NAME*:scope=regional:boot=true:type*=DISK_TYPE*:source-snapshot*=SNAPSHOT_NAME*:replica-zones*=ZONE,REMOTE_ZONE

Failover solutions

Power Backup: In the event of an outage in the primary region, it is your responsibility to identify the outage and failover restart your workload using the secondary disks, in the secondary

region. Following a failover from the primary region to the secondary region, the secondary region becomes the acting primary region.
When you identify that a disaster has occurred, initiate failover to the secondary region. A failover moves the workload from the primary region to the secondary region. After the failover, the secondary disk is the acting primary disk and the secondary region is the action primary region.



- A server will be connected to two power supplies so that if one goes down, it will switch to the other power supply and thus there will be no downtime.
- Telecom hardware and equipment are installed in buildings known as data centres.
- Data centres get their power supply from two or more electricity distributors to ensure high availability.
- Power support backup options in the form of generators are also present.
- The data centre has a cooling system, as the server generates heat due to continuous working.
- Data centres have many “Server Halls” where telecom equipment is present.
- Server Hall has Racks, which are arranged in rows known as “bay”.
- Each rack is 42U with markings from 1 U at the bottom to 42U at the top of the rack.
- As a standard practice, the TOR switch is connected at the top of racks where telecom equipment is connected.
- Each server hall has two ducks below the floor, one for electrical wiring and another for optical/Ethernet cables.
- The data centre has stringent security and access control so that only authorized persons can access the equipment.
- All the cables are properly labelled and routed in the rack and server hall for faster troubleshooting of connectivity.

Switch level redundancy in Google Cloud Platform (GCP) refers to the use of redundant network switches to ensure continuous network connectivity and minimize downtime in case of a hardware failure. This redundancy is crucial for maintaining high availability and reliability of the network infrastructure.

Key Aspects of Switch Level Redundancy in GCP:

Dual Switches:

- Each network component is connected to multiple (typically two) network switches.
- If one switch fails, the other switch continues to provide network connectivity without interruption.

Redundant Paths:

- Network traffic can be rerouted through alternative paths if a switch or link fails.
- This ensures there are no single points of failure in the network path.

Load Balancing:

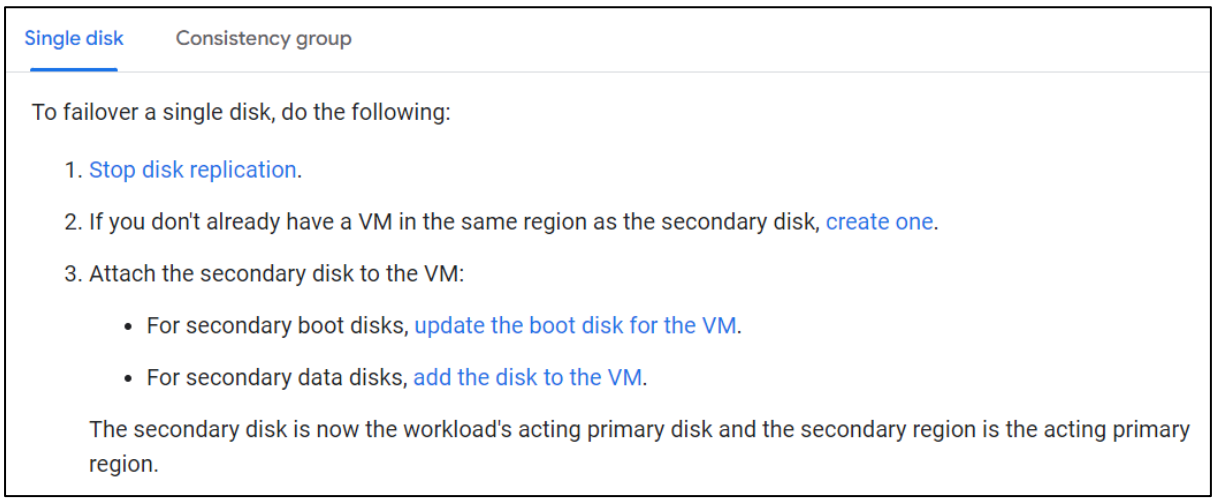
- Network traffic is distributed across multiple switches to balance the load and improve performance.
- This also helps in maintaining service continuity in case of a switch failure.

Failover Mechanisms:

- Automated failover mechanisms detect switch failures and reroute traffic seamlessly.
- This minimizes the impact on applications and services running in GCP.

Thus, Switch Level Redundancy ensures continuous network connectivity and reduces the risk of downtime due to switch failures, thereby providing high availability.

We can failover a single disk, or all disks in a consistency group.



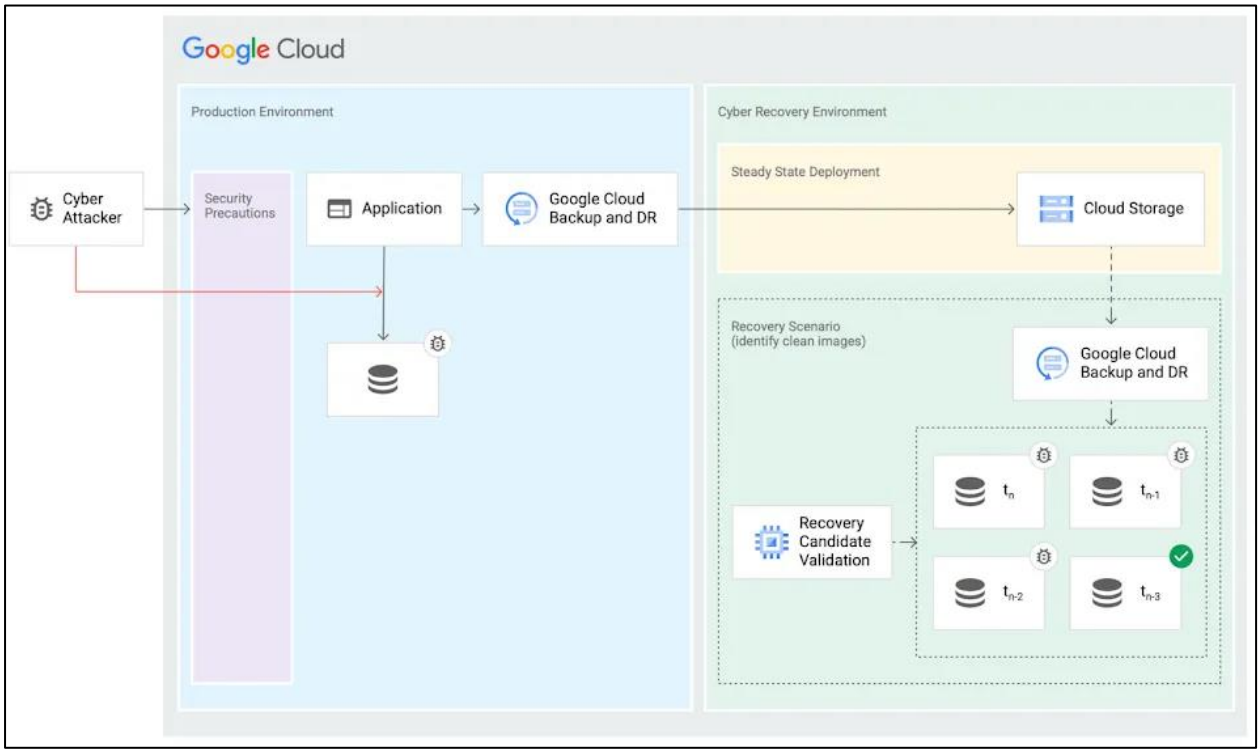
- Data centres get their power supply from 2 or more electricity distributors to ensure high availability.

- Power support back-up options in the form of generators are also present.

Disaster Recovery:
Managed backup and disaster recovery (DR) service for centralized, application-consistent data protection. Protect workloads by backing them up to Google Cloud.

Key Features:

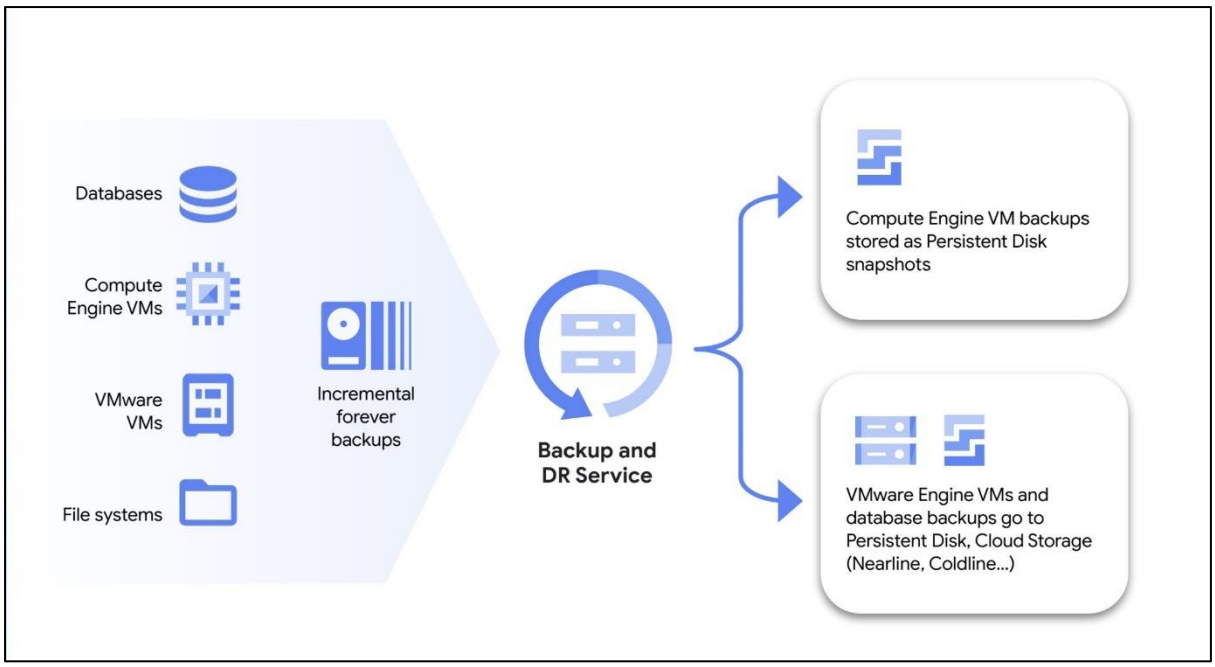
- "Incremental-forever" backups
- Instant mount and recovery
- Application-consistent database backups
- Reduced operational burden
- Broad database support



Source: https://cloud.google.com/compute/docs/

Backup and DR Service is a powerful, centralized, cloud-first backup and disaster recovery solution for cloud workloads. Backup and DR Service lets you recover your data quickly, supporting rapid resumption of critical business operations.

You can use Backup and DR to create, store, and manage backups of the Compute Engine VMs. Backup and DR stores backup data in its original, application-readable format. To store database backups and transaction logs, you can use regional Cloud Storage buckets, which provide the lowest cost backup storage that's redundant across zones. Compute Engine lets us use standard snapshots to capture the point-in-time state of Persistent Disk volumes, to ensure the durability of data. The snapshots are stored redundantly in multiple regions, with automatic checksums to ensure the integrity of your data.

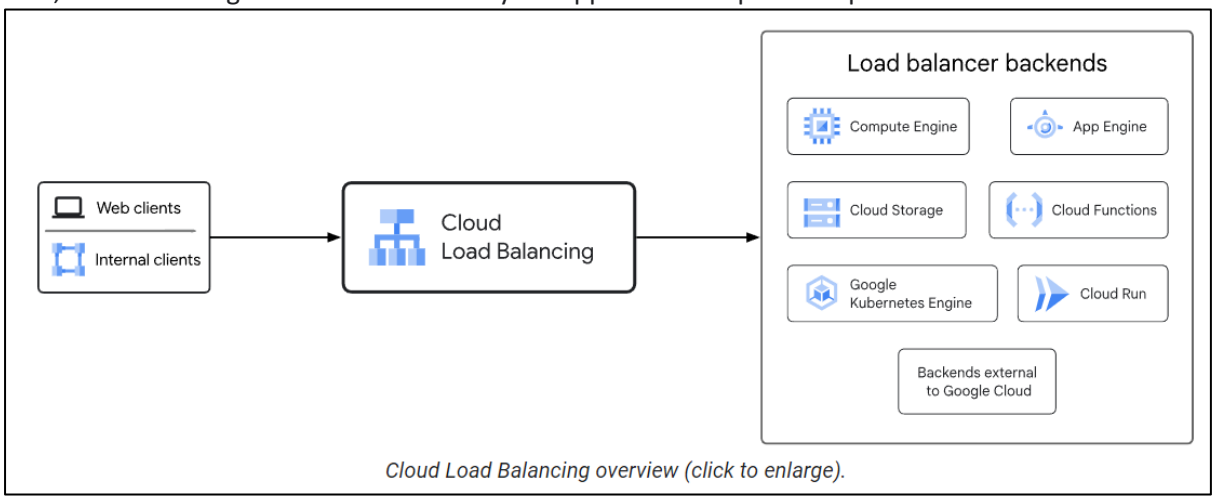


Source: https://cloud.google.com/compute/docs/

Networking

Load Balancing

A load balancer distributes user traffic across multiple instances of your applications. By spreading the load, load balancing reduces the risk that your applications experience performance issues.



Cloud Load Balancing overview (click to enlarge).

Source: https://cloud.google.com/compute/docs/

A **"heartbeat"** refers to the mechanism used to monitor the health and availability of backend servers. This is crucial to ensure that traffic is only directed to servers that are up and running properly. Load Balancer receives heartbeats from the servers periodically, by which it ensures that the servers are active. In case of any downtime, it will not receive the heartbeats.

Source IP Hashing: This method uses the source IP address of the incoming request to generate a hash value. The hash value determines which server in the pool will handle the request. This ensures that requests coming from the same source IP address are always directed to the same server, which can be useful for maintaining session persistence (where all requests from a user session go to the same server).

URL Hashing: In this approach, the URL or a portion of the URL from the incoming request is hashed to determine which server will handle the request. This method is beneficial when you want requests for specific resources (like images or videos) to always go to the same server, optimizing caching and reducing the need for redundant data storage across servers.

Cloud Load Balancing offers the following load balancer features:

- **Single anycast IP address:** With Cloud Load Balancing, a single anycast IP address is the frontend for all of your backend instances in regions around the world. It provides cross-region load balancing, including automatic multi-region failover, which moves traffic to failover backends if your primary backends become unhealthy.
- **Seamless autoscaling:** Cloud Load Balancing can scale as your users and traffic grow, including easily handling huge, unexpected, and instantaneous spikes by diverting traffic to other regions in the world that can take traffic.

- **Software-defined load balancing:** Cloud Load Balancing is a fully distributed, software-defined, managed service for all your traffic.
- **Layer 4 and Layer 7 load balancing:** Use Layer 4-based load balancing to direct traffic based on data from network and transport layer protocols such as TCP, UDP, ESP, GRE, ICMP, and ICMPv6. Use Layer 7-based load balancing to add request routing decisions based on attributes, such as the HTTP header and the uniform resource identifier.
- **External and Internal Load Balancing:** You can use external load balancing when your users reach your applications from the internet. You can use internal load balancing when your clients are inside of Google Cloud.
- **Global and regional load balancing:** You can distribute your load-balanced resources in single or multiple regions to terminate connections close to your users and to meet your high availability requirements.
- **Advanced feature support:** Cloud Load Balancing supports features such as IPv6 load balancing, source IP-based traffic steering, weighted load balancing, WebSockets, user-defined request headers, and protocol forwarding for private virtual IP addresses (VIPs).

It also includes the following integrations:

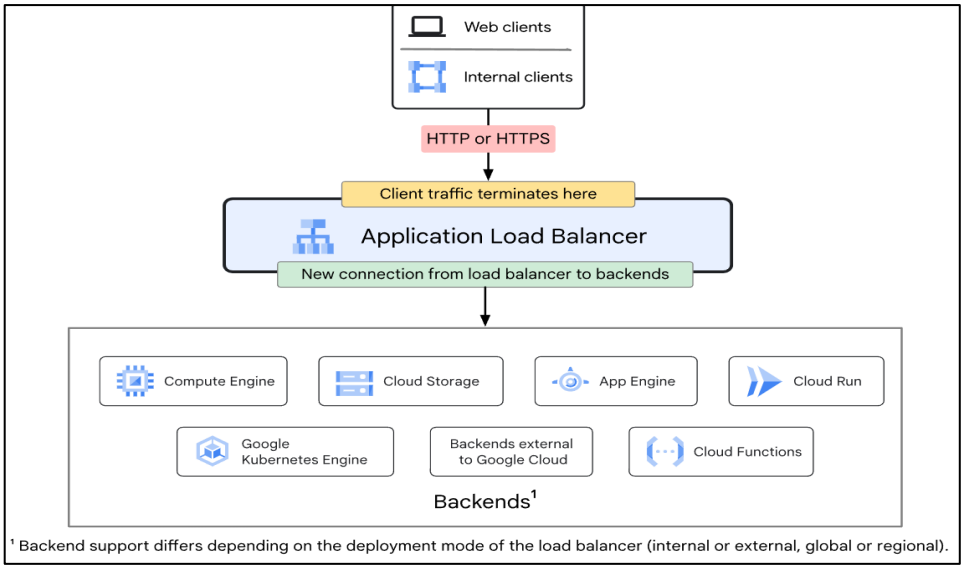
- Integration with *Cloud CDN* for cached content delivery. Cloud CDN is supported with the global external Application Load Balancer and the classic Application Load Balancer.

- Integration with Google Cloud Armor to protect your infrastructure from distributed denial-of-service (DDoS) attacks and other targeted application attacks. Always-on DDoS protection is available for the global external Application Load Balancer, the classic Application Load Balancer, the external proxy Network Load Balancer, and the external passthrough Network Load Balancer.

Types of Load Balancers:

Cloud Load Balancing offers two types of load balancers: Application Load Balancers and Network Load Balancers.

Application Load Balancers: Application Load Balancers are proxy-based Layer 7 load balancers that enable you to run and scale your services behind an anycast IP address. The Application Load Balancer distributes HTTP and HTTPS traffic to backends hosted on a variety of Google Cloud platforms—such as Compute Engine and Google Kubernetes Engine (GKE)—as well as external backends outside Google Cloud. Application Load Balancers can be deployed externally or internally depending on whether your application is internet-facing or internal.

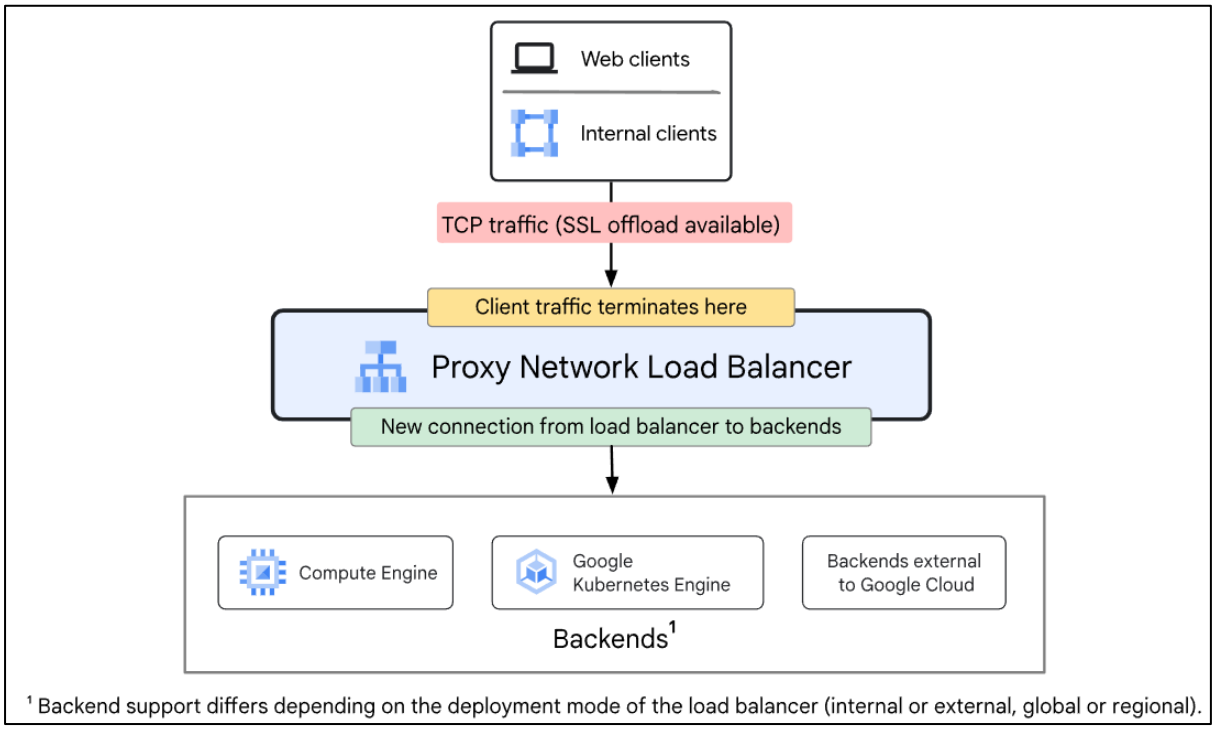


Source: https://cloud.google.com/compute/docs/

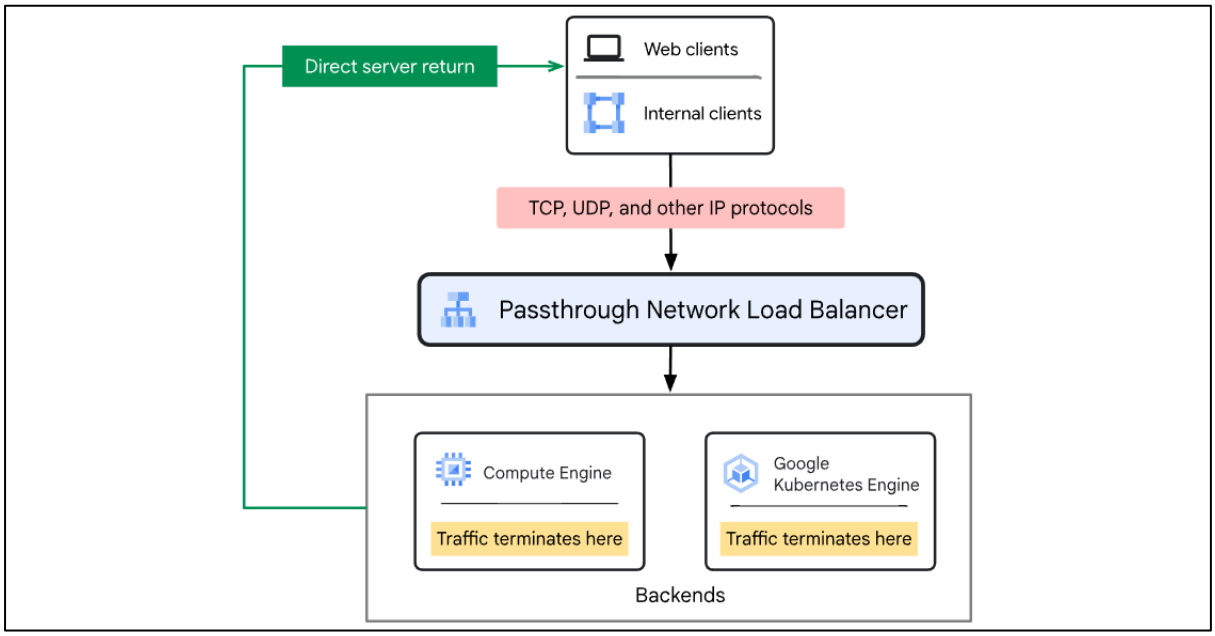
Network Load Balancers: Network Load Balancers are Layer 4 load balancers that can handle TCP, UDP, or other IP protocol traffic. These load balancers are available as either proxy load balancers or passthrough load balancers.

Proxy Network Load Balancers are Layer 4 reverse proxy load balancers that distribute TCP traffic to virtual machine (VM) instances in your Google Cloud VPC network.

Passthrough Network Load Balancers are Layer 4 regional, passthrough load balancers. These load balancers distribute traffic among backends in the same region as the load balancer.

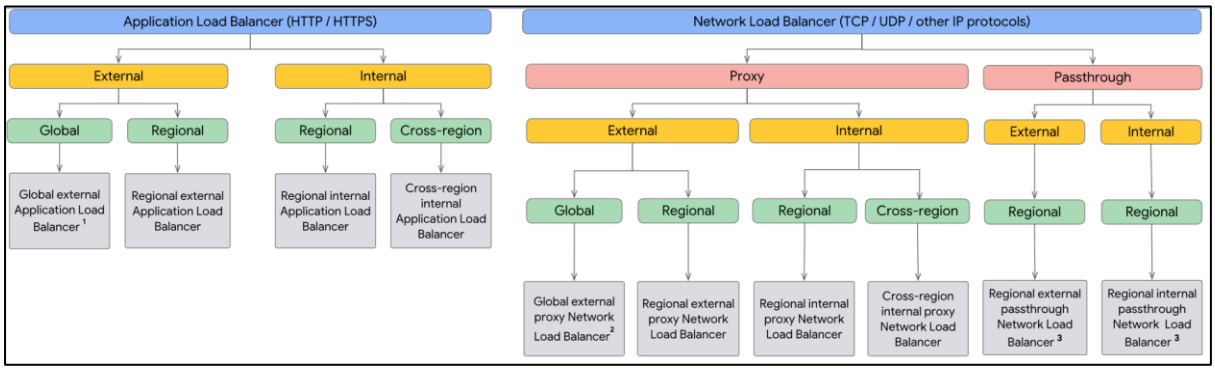


Source: https://cloud.google.com/compute/docs/



Source: https://cloud.google.com/compute/docs/

The following diagram shows all of the available deployment modes for Cloud Load Balancing.



Source: https://cloud.google.com/compute/docs/

Summary of Google Cloud load balancers:

Load balancer	Deployment mode	Traffic type	Network service tier	Load-balancing scheme †
Application Load Balancers	Global external	HTTP or HTTPS	Premium Tier	EXTERNAL_MANAGED
	Regional external	HTTP or HTTPS	Premium or Standard Tier	EXTERNAL_MANAGED
	Classic	HTTP or HTTPS	Global in Premium Tier Regional in Standard Tier	EXTERNAL
	Regional internal	HTTP or HTTPS	Premium Tier	INTERNAL_MANAGED
	Cross-region internal	HTTP or HTTPS	Premium Tier	INTERNAL_MANAGED
Proxy Network Load Balancers	Global external	TCP with optional SSL offload	Premium Tier	EXTERNAL_MANAGED
	Regional external	TCP	Premium or Standard Tier	EXTERNAL_MANAGED
	Classic	TCP with optional SSL offload	Global in Premium Tier Regional in Standard Tier	EXTERNAL
	Regional internal	TCP without SSL offload	Premium Tier	INTERNAL_MANAGED
	Cross-region internal	TCP without SSL offload	Premium Tier	INTERNAL_MANAGED
Passthrough Network Load Balancers	External Always regional	TCP, UDP, ESP, GRE, ICMP, and ICMPv6	Premium or Standard Tier	EXTERNAL
	Internal Always regional	TCP, UDP, ICMP, ICMPv6, SCTP, ESP, AH, and GRE	Premium Tier	INTERNAL

Source: https://cloud.google.com/compute/docs/

Multi-regional deployment

Multi-regional deployment refers to a multi-tier application that runs on Compute Engine VMs in multiple regions in Google Cloud.

Use cases for which a multi-regional deployment on Compute Engine is an appropriate choice:

- Efficient migration of on-premises applications
- High availability for geo-dispersed users
- Low latency for application users

Storage services:

Storage options for multi-regional deployments include regional Persistent disk volumes for all tiers, and Cloud Storage dual-region or multi-region buckets. Objects stored in a dual-region or multi-region bucket are stored redundantly in at least two separate geographic locations.

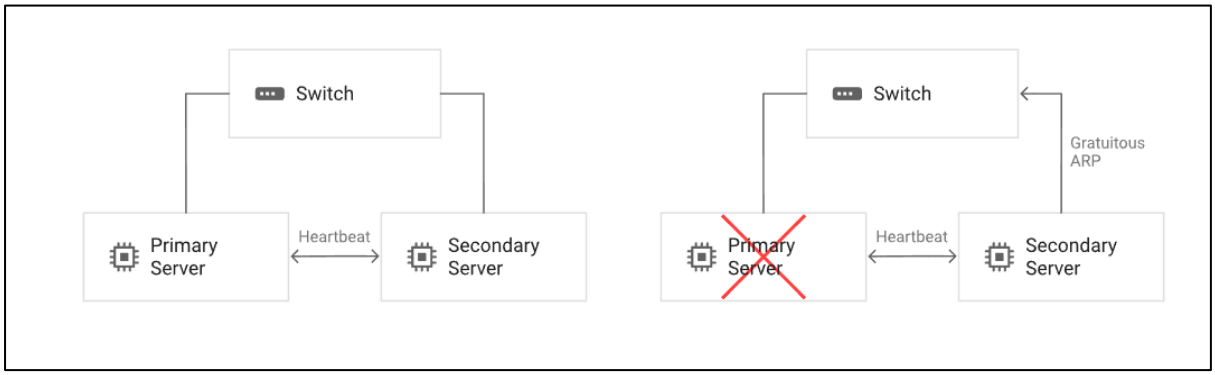
Security and Compliance:

Factors that you should consider when you use this reference architecture to design and build a multi-regional topology in Google Cloud that meets the security and compliance requirements of your workloads:-

- **Protection against threats:** To protect your application against threats like distributed denial of service (DDoS) attacks and cross-site scripting (XSS), you can use Google Cloud Armor security policies.
- **External access for VMs:** The VMs that host the application tier, web tier, and databases don't need inbound access from the internet, so no need to assign external IP addresses to those VMs.
- **VM image security:** To ensure that your VMs use only approved images (that is, images with software that meets your policy or security requirements), you can define an organization policy that restricts the use of images in specific public image projects.
- **Service account privileges:** In Google Cloud projects where the Compute Engine API is enabled, a default service account is created automatically. The default service account is granted the Editor IAM role (roles/editor) unless this behaviour is disabled. By default, the default service account is attached to all VMs that you create by using the Google Cloud CLI or the Google Cloud console.
- **Data residency considerations:** You can use regional load balancers to build a multi-regional architecture that helps you to meet data residency requirements. For example, a country in Europe might require that all user data be stored and accessed in data centres that are located physically within Europe.

Virtual IP based on API

Also referred to as shared or virtual IP addresses, floating IP addresses are often used to make on-premises network environments highly available. Using floating IP addresses, you can pass an IP address between multiple identically configured physical or virtual servers. There are several ways to implement floating IP addresses in an on-premises environment. The following diagram shows a typical setup in an on-premises environment. Most applications that require floating IP addresses use floating internal IP addresses. Few applications use floating external IP addresses, because typically traffic to external applications should be load balanced. Highly available physical appliances, such as a set of firewalls or load balancers, often use floating IP addresses for failovers.



Source: https://cloud.google.com/compute/docs/

The preceding diagram shows how a primary server and a secondary server connected to the same switch exchange responsiveness information through a heartbeat mechanism. If the primary server fails, the secondary server sends a gratuitous ARP frame to the switch to take over the floating IP address.

Monitoring

End-to-end testing

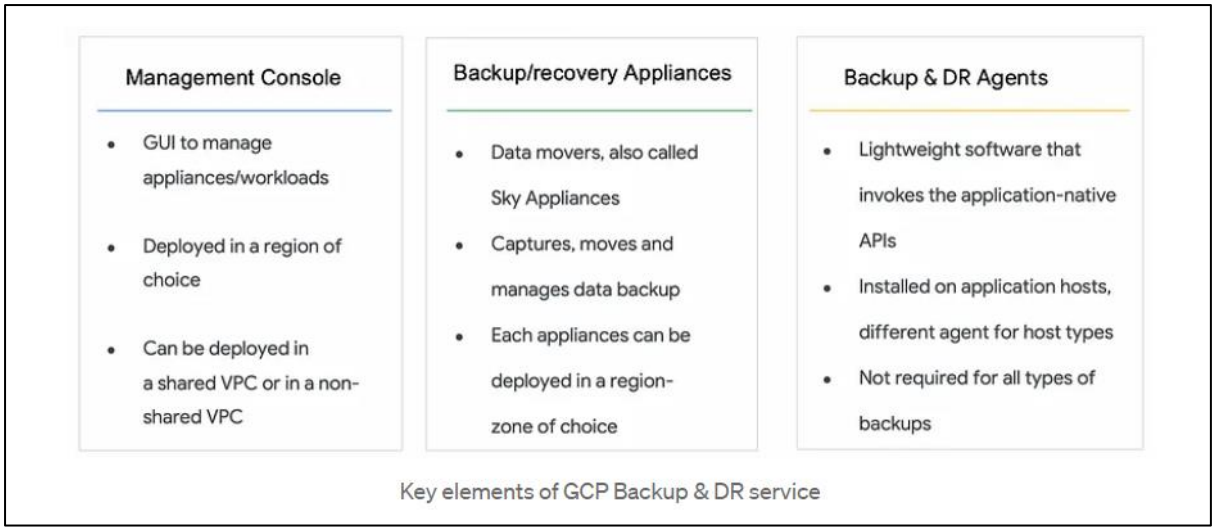
End-to-end testing in Google Cloud Platform (GCP) involves validating the entire workflow of an application, from the frontend to the backend, and ensuring all integrated components work as expected. This process typically includes deploying the application, setting up the necessary infrastructure and a test environment, running comprehensive tests, and analyzing the results. You can ensure your application works as expected across all integrated components, providing a robust and reliable service to your users.

Backup and recovery

Let's start with the salient features of Google Cloud Backup and DR service:

- Software as a Service with a centralised GCP control plane.
- Has a consumption-based pricing model for storage and usage components(monthly).
- Supports application and crash-consistent backups for multiple workloads.
- Stores backup data in its original, application-readable format.
- Data encryption in transit and at rest.
- Employs space-efficient "incremental forever" storage technology.
- Centralised management of backups for Google Cloud and hybrid workloads.
- Cross-region backups and cross-project recoveries.
- Multi-region disaster recovery.
- Supports instant mount from Google Cloud Storage.
- Protection for a broad spectrum of workloads — Compute Engine VMs, VMware VMs, file systems, databases including IBM Db2, Microsoft SQL Server, MongoDB, MySQL, Oracle, SAP ASE, SAP HANA, SAP IQ, SAP MaxDB, and PostgreSQL.

Up next, here are the key components of the Google Cloud Backup and DR service's architecture at a glance:



Key elements of GCP Backup & DR service

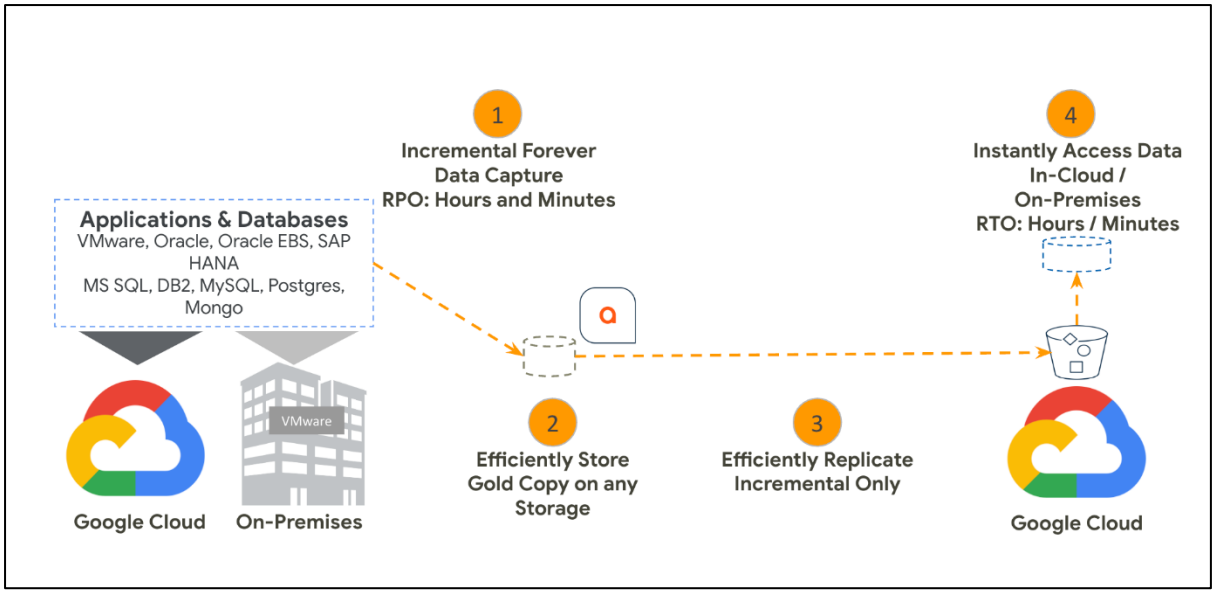
Source: https://cloud.google.com/compute/docs/

Here is how the Google Cloud Backup & DR service operates (their plan in "action"):

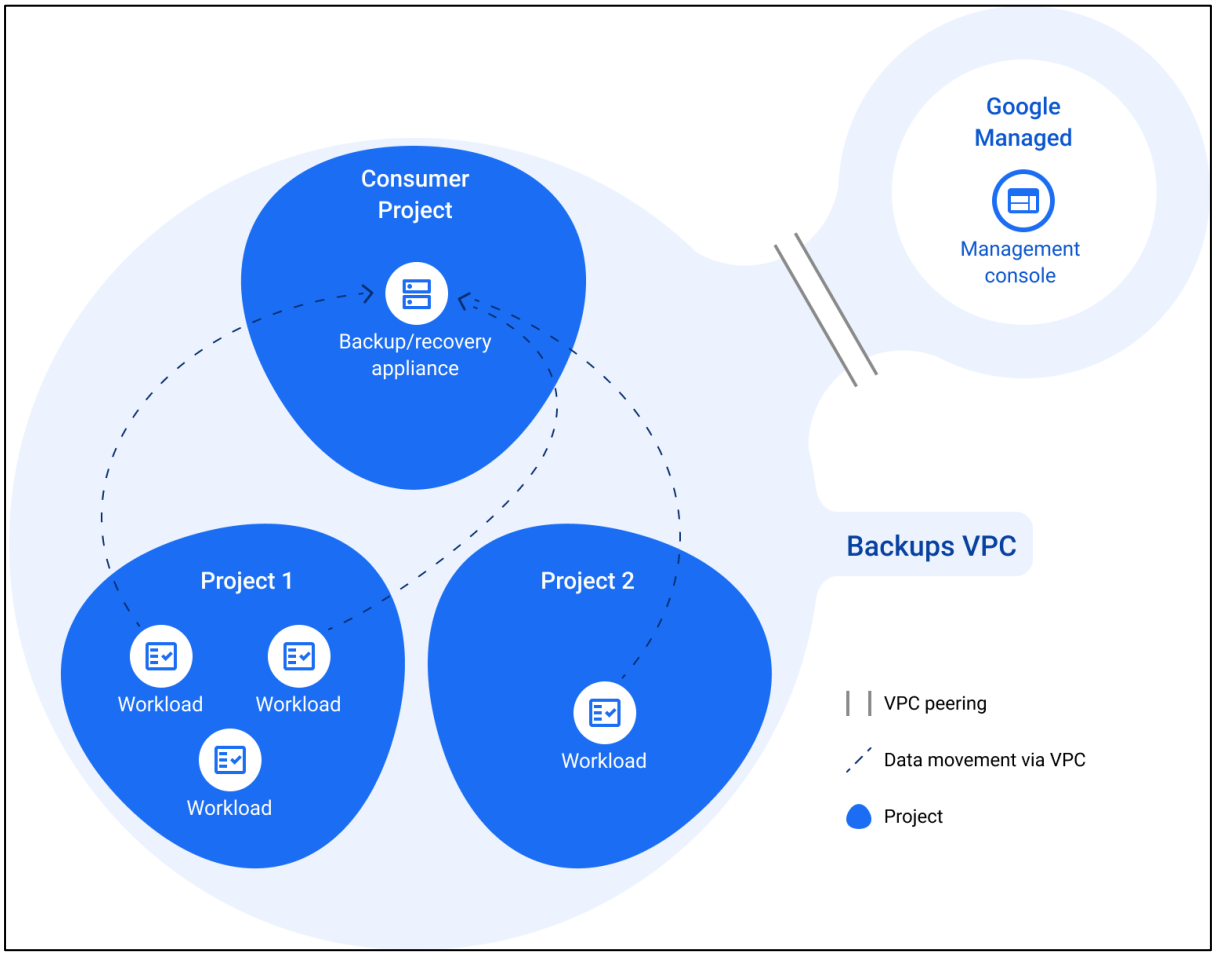
- Identify which applications need backup (VMs, SAP HANA, Oracle, etc).

- Apply backup plans — the data-handling policies, to the applications to manage them.
  — **Backup Templates:** This defines the frequency and retention of data backups.
  — **Resource Profiles:** This specifies the physical location of the storage (snapshot pool or remote appliance) for the data backups.

- Once a managed application has a backup image created, it can be restored to production anytime and the data can then be easily accessed. Given its minimal Recovery Time Objective (RTO) and Recovery Point Objective (RPO), GCP's Backup & DR service provides a one-stop solution for decreasing this downtime gap. It can help an enterprise reach a successful point in its transformational journey in terms of digital maturity. Most significantly, Google Cloud Backup & DR service provides enterprises with the flexibility and peace of mind they need, together with lightning-fast application recovery — whenever they need it.



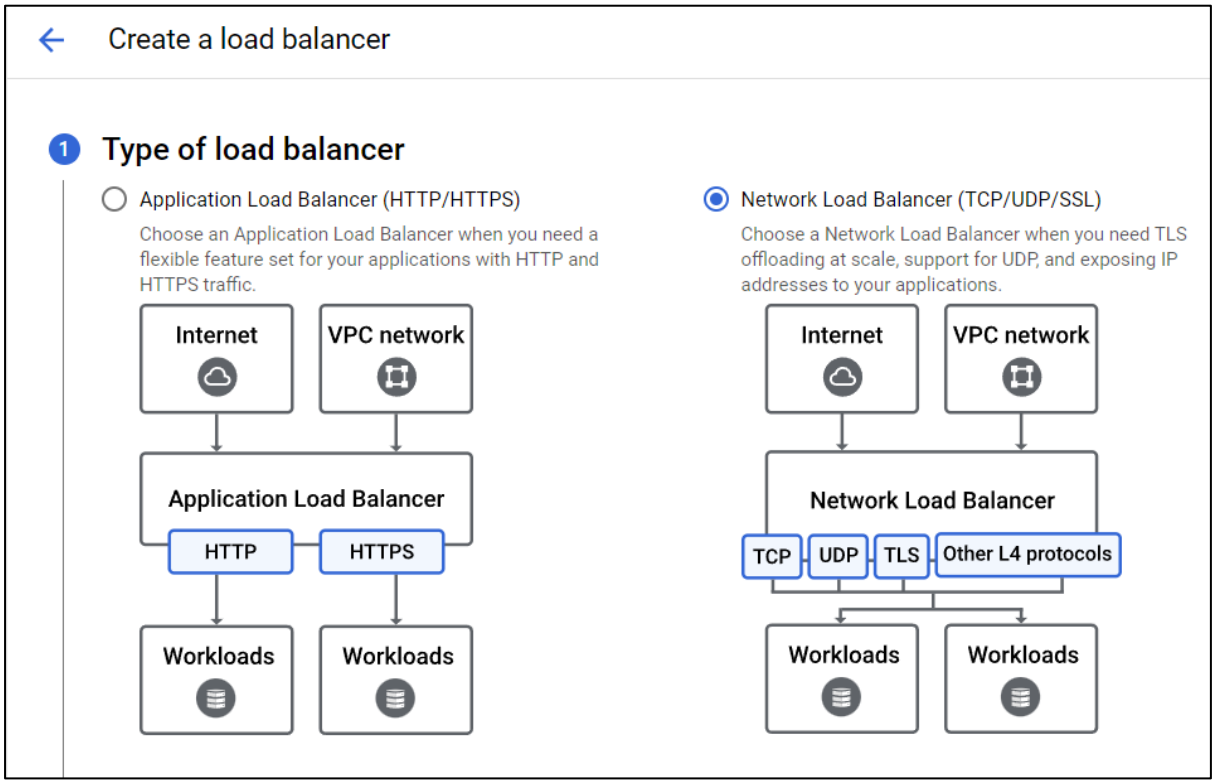
Source: https://cloud.google.com/compute/docs/

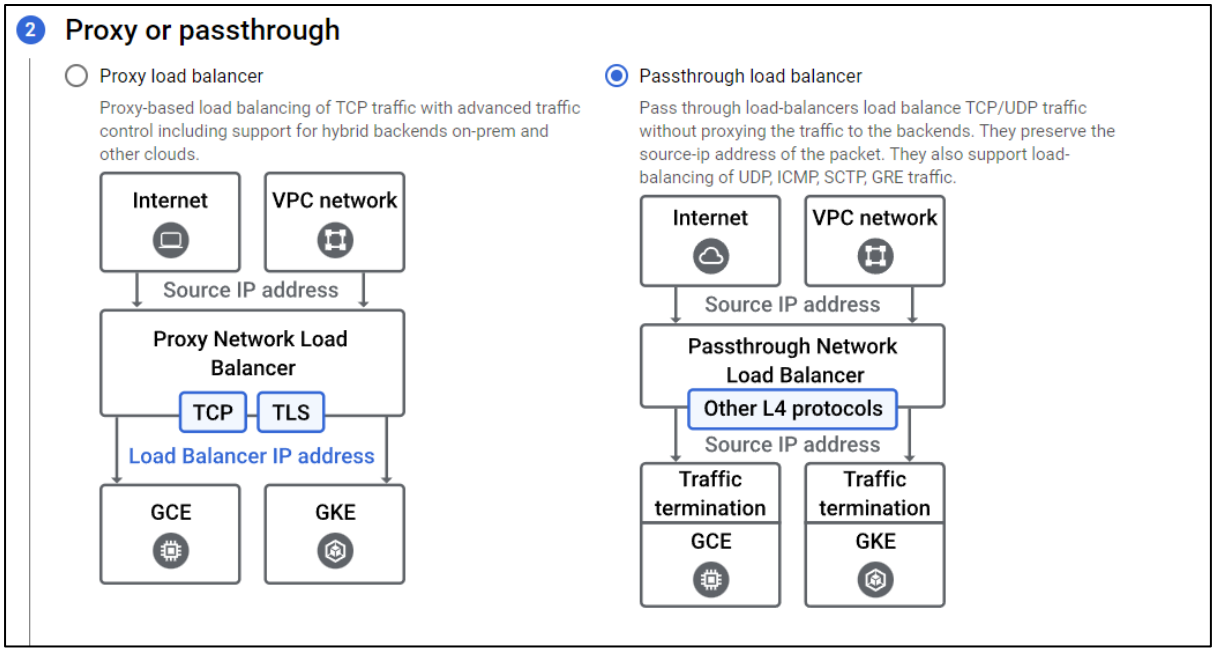


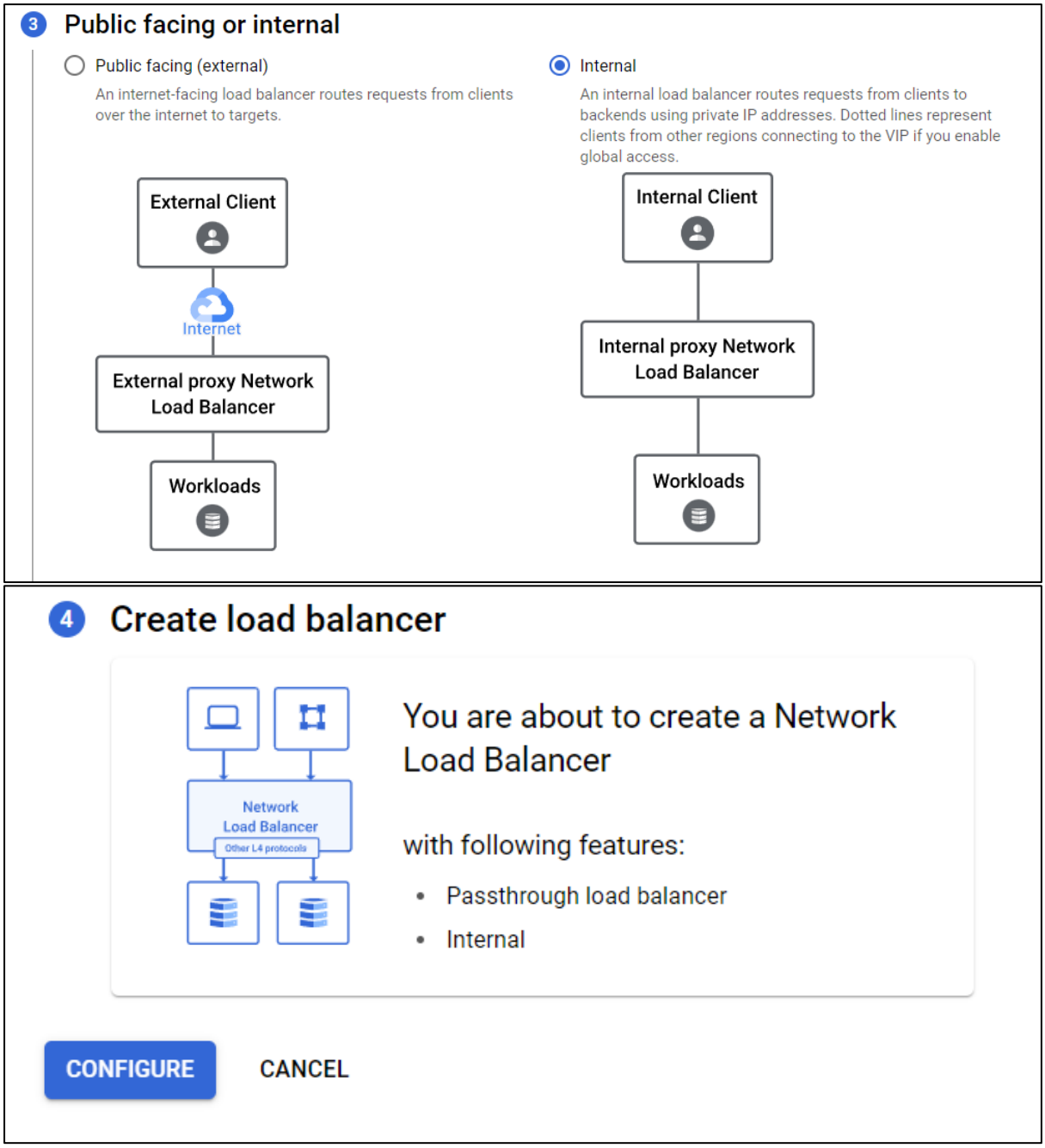
Source: https://cloud.google.com/compute/docs/

Failover Testing (Evaluation of various techniques to achieve application redundancy):

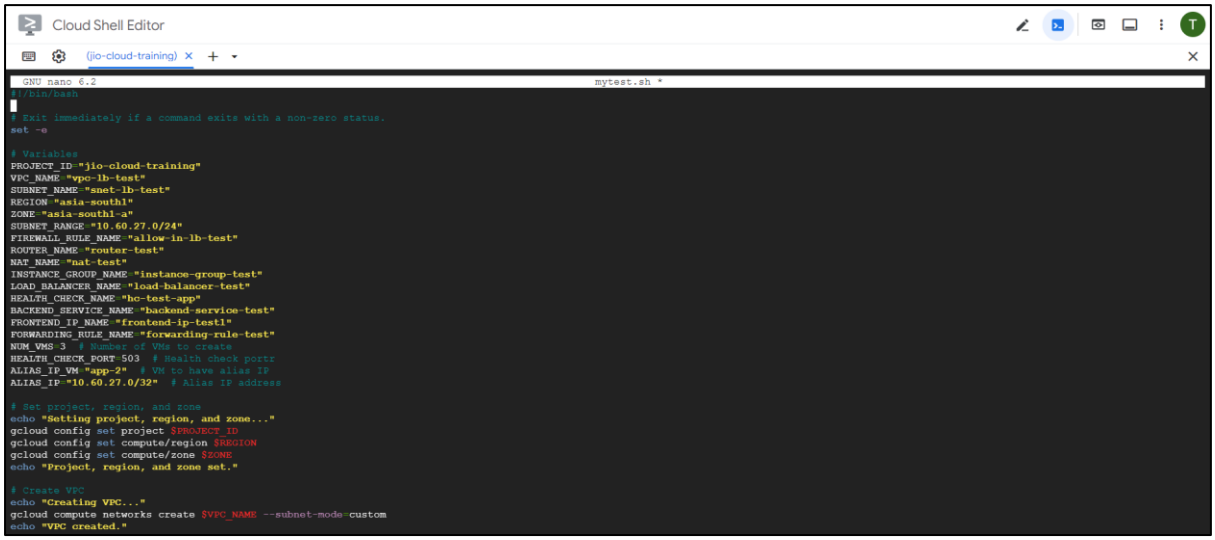
Using Load Balancer

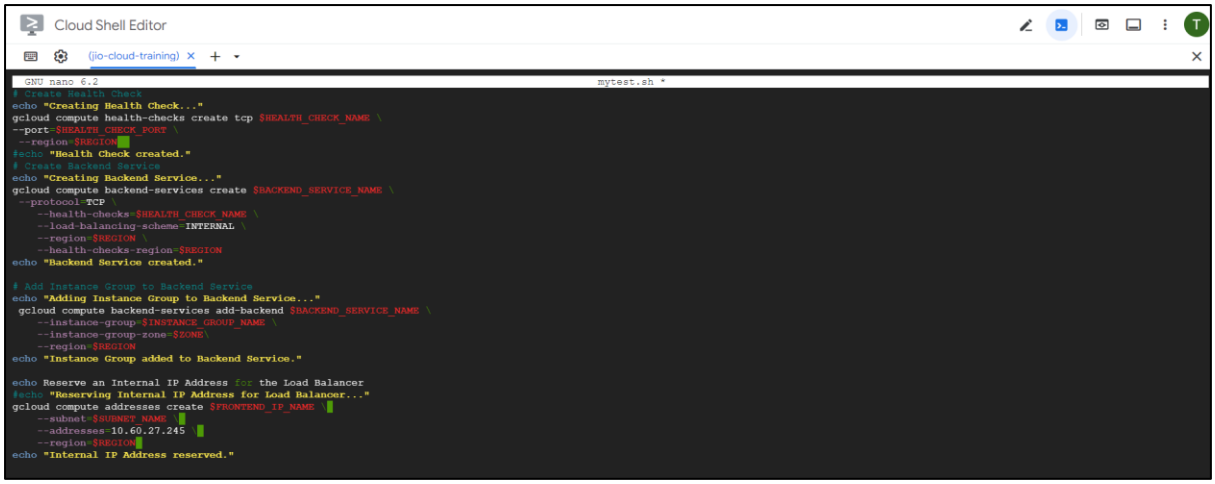


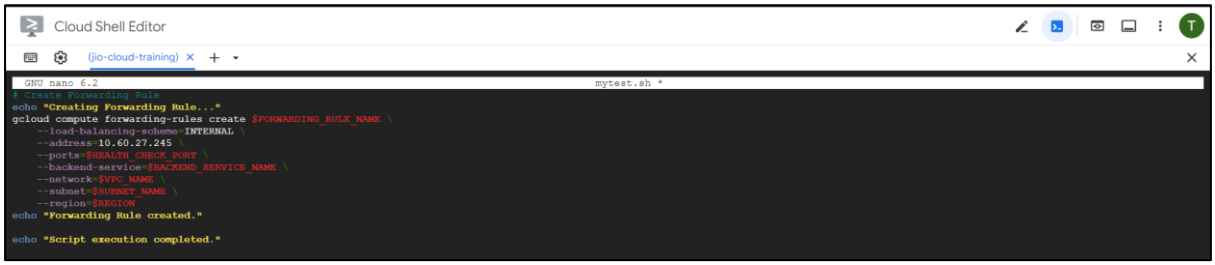



3 Public facing or internal
Public facing (external)
An internet-facing load balancer routes requests from clients over the internet to targets.
External Client
Internet
External proxy Network Load Balancer
Workloads
Internal
An internal load balancer routes requests from clients to backends using private IP addresses. Dotted lines represent clients from other regions connecting to the VIP if you enable global access.
Internal Client
Internal proxy Network Load Balancer
Workloads
4 Create load balancer
Network Load Balancer
Other L4 protocols
You are about to create a Network Load Balancer
with following features:
Passthrough load balancer
Internal
CONFIGURE
CANCEL

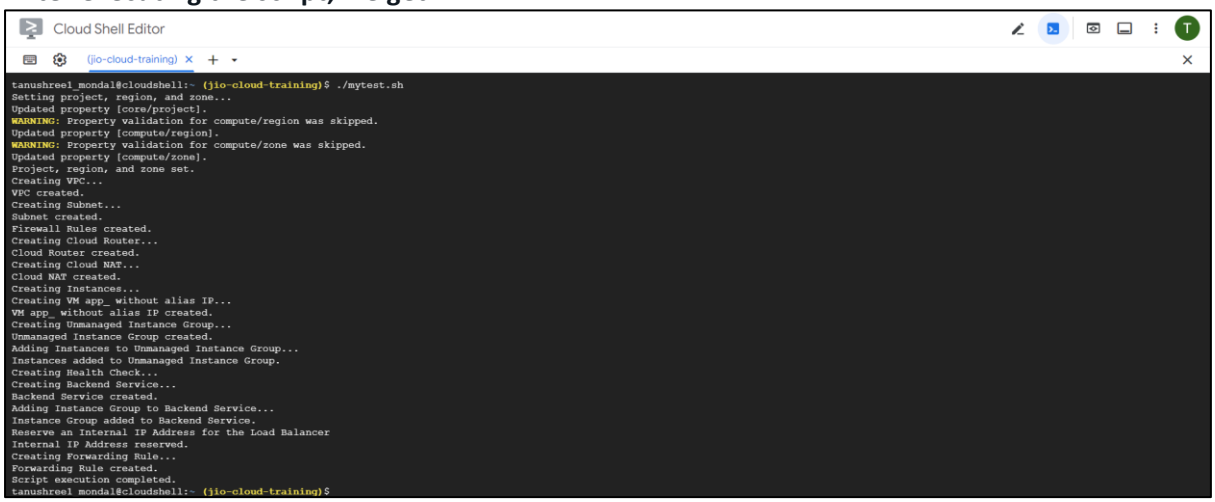
Shell script:







After executing the script, we get:



To check the internet connectivity:

```
tanushree1_mondal@app-2:~$ curl -v www.google.com
*   Trying 142.250.76.196:80...
* TCP_NODELAY set
* Connected to www.google.com (142.250.76.196) port 80 (#0)
```

To switch my user credentials to root user:

```
tanushree1_mondal@app-2:~$ sudo su
root@app-2:/home/tanushree1_mondal#
```

Fetching the latest version of the package list:

```
tanushree1_mondal@app-2:~$ sudo su
root@app-2:/home/tanushree1_mondal# apt update
Hit:1 http://asia-south1.gce.archive.ubuntu.com/ubuntu focal InRelease
Hit:2 http://asia-south1.gce.archive.ubuntu.com/ubuntu focal-updates InRelease
Hit:3 http://security.ubuntu.com/ubuntu focal-security InRelease
Hit:4 http://asia-south1.gce.archive.ubuntu.com/ubuntu focal-backports InRelease
Reading package lists... Done
Building dependency tree
Reading state information... Done
10 packages can be upgraded. Run 'apt list --upgradable' to see them.
root@app-2:/home/tanushree1_mondal#
```

Docker installation:

```
root@app-2:/home/tanushree1_mondal# curl -sSl https://get.docker.com/ | sh
# Executing docker install script, commit: 0d6f72e671ba87f7aa4c6991646a1a5b9f9dae84
+ sh -c apt-get update -qq >/dev/null
+ sh -c DEBIAN_FRONTEND=noninteractive apt-get install -y -qq ca-certificates curl >/dev/null
+ sh -c install -m 0755 -d /etc/apt/keyrings
+ sh -c curl -fsSL "https://download.docker.com/linux/ubuntu/gpg" -o /etc/apt/keyrings/docker.asc
+ sh -c chmod a+r /etc/apt/keyrings/docker.asc
+ sh -c echo "deb [arch=amd64 signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu focal stable" > /etc/apt/sources.list.d/docker.list
+ sh -c apt-get update -qq >/dev/null
+ sh -c DEBIAN_FRONTEND=noninteractive apt-get install -y -qq docker-ce docker-ce-cli containerd.io docker-compose-plugin docker-ce-rootless-extras docker-buildx-plugin >/dev/null
+ sh -c docker version
Client: Docker Engine - Community
 Version:           27.2.0
 API version:       1.47
 Go version:        go1.21.13
 Git commit:        3ab4256
 Built:             Tue Aug 27 14:15:09 2024
 OS/Arch:           linux/amd64
 Context:           default

Server: Docker Engine - Community
 Engine:
  Version:          27.2.0
  API version:      1.47 (minimum version 1.24)
  Go version:       go1.21.13
  Git commit:       3ab5c7d
  Built:            Tue Aug 27 14:15:09 2024
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.7.21
  GitCommit:        472731909fa34bd7bc9c087e4c27943f9835f111
 runc:
  Version:          1.1.13
  GitCommit:        v1.1.13-0-g58aa920
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0

================================================================================
```

Pulling the Python image:

```
root@app-2:/home/tanushree1_mondal# docker pull python:3.9.19-bullseye
3.9.19-bullseye: Pulling from library/python
203e9cf21bd2: Pull complete
9a3438c04e45: Pull complete
6665b6f4bd77: Pull complete
9f6864ced296: Pull complete
2e0da295d452: Pull complete
2f076c620910: Pull complete
c1e672ce867c: Pull complete
90e9e4e248e2: Pull complete
Digest: sha256:f4010aad4cc1b4fd67b36ca54fa17b10055821a8a3b3a7cd735eeaa36f76063f
Status: Downloaded newer image for python:3.9.19-bullseye
docker.io/library/python:3.9.19-bullseye
root@app-2:/home/tanushree1_mondal#
```

Running the Python image:

```
root@app-2:/home/tanushree1_mondal# docker run -it python:3.9.19-bullseye
Python 3.9.19 (main, Aug 13 2024, 02:07:28)
[GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print ("Hello World")
Hello World
>>>
```

Running a http server on container:

```
root@app-2:/home/tanushree1_mondal# docker run -d -p 503:503 python:3.9.19-bullseye python3 -m http.server 503
52e2b448160f6c0ecc601762fd3ab1c0786714f88dfa31e176535dd6456e021f
root@app-2:/home/tanushree1_mondal# docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED          STATUS          PORTS                                       NAMES
52e2b448160f   python:3.9.19-bullseye   "python3 -m http.ser…"   16 seconds ago   Up 15 seconds   0.0.0.0:503->503/tcp, :::503->503/tcp   trusting_newton
root@app-2:/home/tanushree1_mondal#
```

Checking container logs:

```
root@app-2:/home/tanushree1_mondal# docker logs -f 52e2b448160f
10.60.27.242 - - [28/Aug/2024 13:29:16] "GET / HTTP/1.1" 200 -
10.60.27.242 - - [28/Aug/2024 13:30:57] "GET / HTTP/1.1" 200 -
10.60.27.242 - - [28/Aug/2024 13:31:01] "GET / HTTP/1.1" 200 -
```

Interfaces:

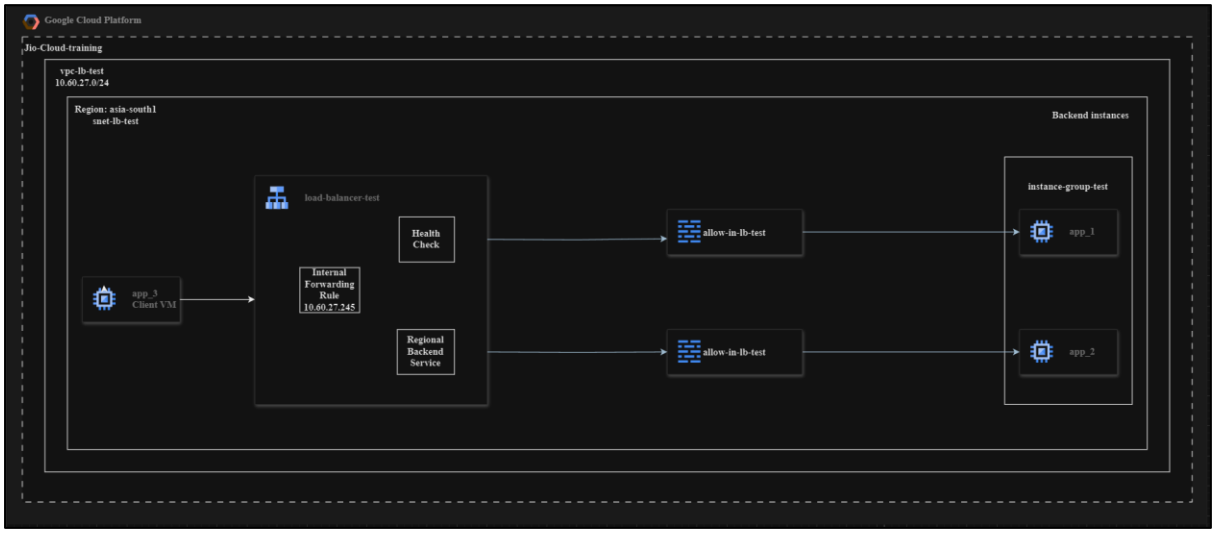
```
root@app-2:/home/tanushree1_mondal# ip -br a
lo               UNKNOWN        127.0.0.1/8 ::1/128
ens4             UP             10.60.27.243/32 fe80::4001:aff:fe3c:1bf3/64
docker0          UP             172.17.0.1/16 fe80::42:f9ff:fec3:4a07/64
vethfa59242@if16 UP             fe80::b028:26ff:fe4e:3cae/64
```

Checking network and their information:

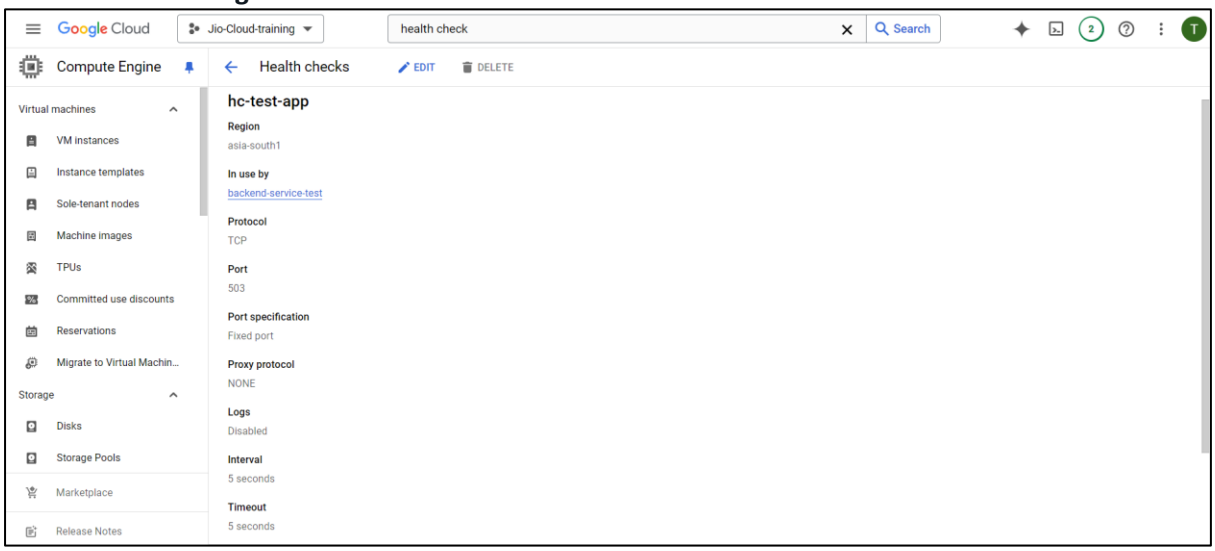
```
root@app-2:/home/tanushree1_mondal# docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
d10c8d25a5f0   bridge    bridge    local
3f622338fdfc   host      host      local
0bcfc1c2e270   none      null      local
root@app-2:/home/tanushree1_mondal# docker network inspect d10c8d25a5f0
[
    {
        "Name": "bridge",
        "Id": "d10c8d25a5f02ae4e555a116077f303d861dfbe5294e871c886b688019a7f1bd",
        "Created": "2024-08-28T13:14:45.054027337Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16",
                    "Gateway": "172.17.0.1"
                }
            ]
```

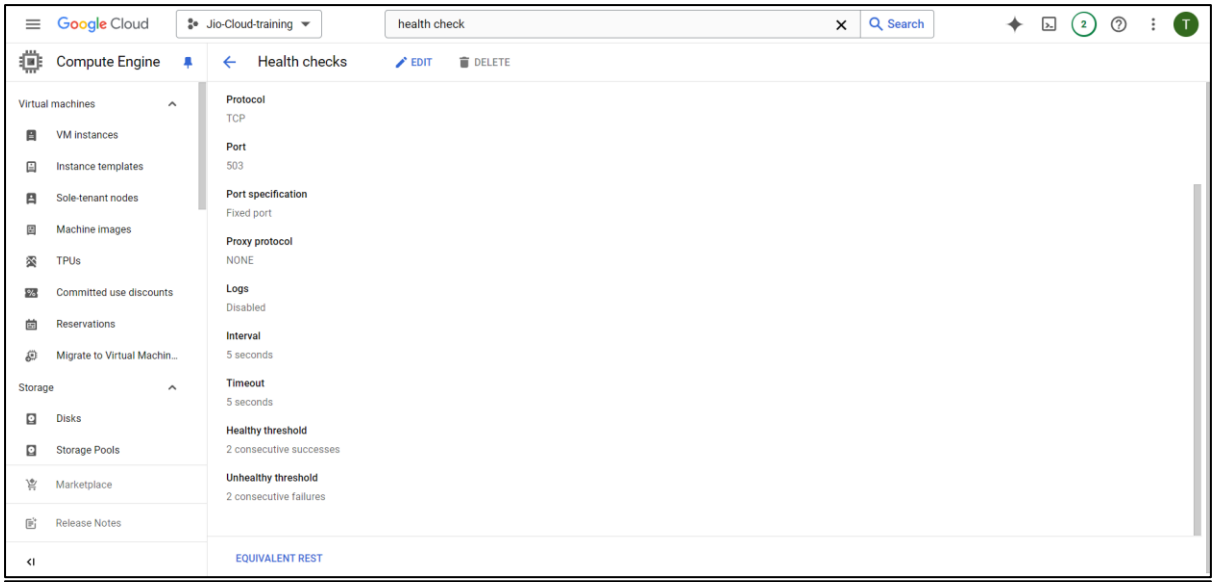
Note: The container installation will be done in both the VMs app-1 and app-2.

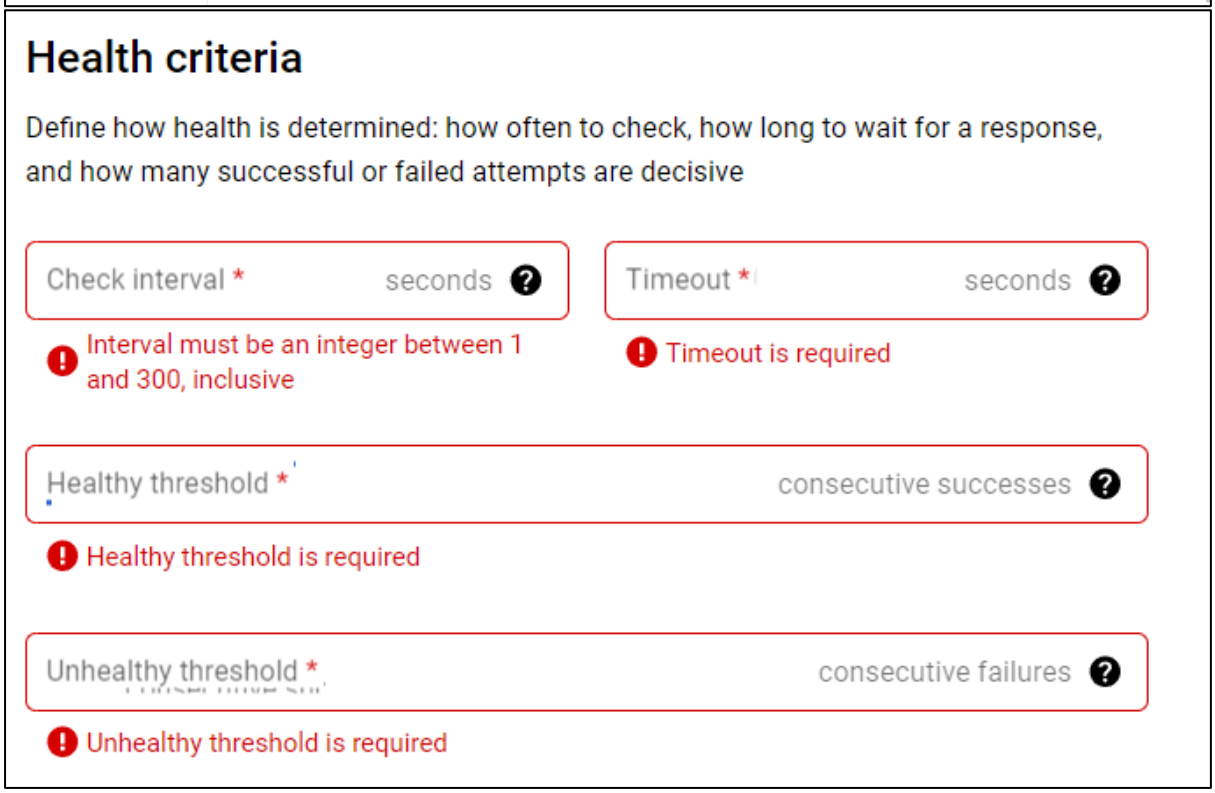
Architectural Diagram:



Health Check configurations:







Explanations of these health check criteria:

- **Health check interval**: How often (in seconds) to send a health check.
- **Timeout:** How long to wait (in seconds) before a request is considered a failure.
- **Healthy threshold**: How many consecutive successes must occur to mark a VM instance healthy?
- **Unhealthy threshold:** How many consecutive failures must occur to mark a VM instance unhealthy?

Further steps to follow:

Step 1: Get inside the container in app-1.

Step 2: Create a html file and write the desired output.

```
tanushree1_mondal@app-1:~$ sudo su
root@app-1:/home/tanushree1_mondal# docker ps
CONTAINER ID   IMAGE                    COMMAND                  CREATED        STATUS        PORTS
          NAMES
3b672dcb5973   python:3.9.19-bullseye   "python3 -m http.ser…"   21 hours ago   Up 21 hours   0.0.0.0:503->503/tcp, :::503
->503/tcp   frosty_rhodes
root@app-1:/home/tanushree1_mondal# docker exec -it 3b672dcb5973 /bin/bash
root@3b672dcb5973:/# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@3b672dcb5973:/# cd home
root@3b672dcb5973:/home# ls
index.html  index1.html
root@3b672dcb5973:/home# nano index1.html
root@3b672dcb5973:/home# cat index1.html
Response from app-1
root@3b672dcb5973:/home#
root@3b672dcb5973:/home#
root@3b672dcb5973:/home#
```

Repeat the same process for app-2.

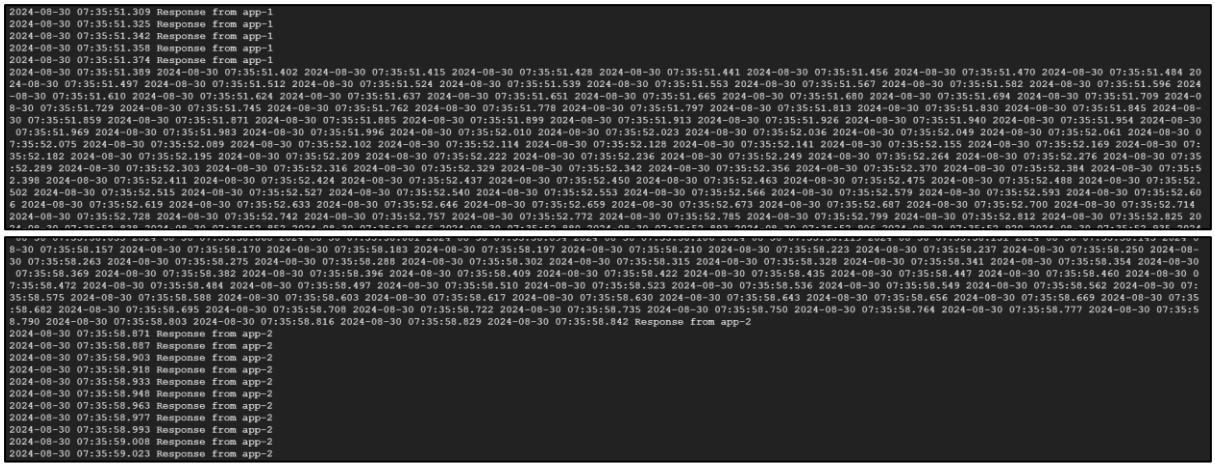
```
tanushree1_mondal@app-2:~$ sudo su
root@app-2:/home/tanushree1_mondal# docker ps
CONTAINER ID   IMAGE                    COMMAND                  CREATED        STATUS        PORTS
          NAMES
5acd92246f0d   python:3.9.19-bullseye   "python3 -m http.ser…"   20 hours ago   Up 20 hours   0.0.0.0:503->503/tcp, :::
503->503/tcp   jolly_hertz
root@app-2:/home/tanushree1_mondal# docker exec -it 5acd92246f0d /bin/bash
root@5acd92246f0d:/# ls
bin   dev  home  lib64  mnt  proc  run   srv  tmp  var
boot  etc  lib   media  opt  root  sbin  sys  usr
root@5acd92246f0d:/# cd home
root@5acd92246f0d:/home# ls
root@5acd92246f0d:/home# touch index1.html
root@5acd92246f0d:/home# nano index1.html
root@5acd92246f0d:/home# cat index1.html
Response from app-2
root@5acd92246f0d:/home#
```

Then, we have to provide the frontend IP of the load balancer in app-3 or the client VM which will divert the traffic to the instance group, which is in active mode (in this case, instance-group-test, app-1).

```
tanushree1_mondal@app-3:~$ while true; do echo -n "$(date '+%Y-%m-%d %H:%M:%S.%3N') "; curl -s http://10.60.27.245:503/home/index1.html;done
2024-08-30 06:52:23.427 Response from app-1
2024-08-30 06:52:23.443 Response from app-1
2024-08-30 06:52:23.458 Response from app-1
2024-08-30 06:52:23.473 Response from app-1
2024-08-30 06:52:23.487 Response from app-1
2024-08-30 06:52:23.503 Response from app-1
2024-08-30 06:52:23.518 Response from app-1
2024-08-30 06:52:23.533 Response from app-1
2024-08-30 06:52:23.549 Response from app-1
2024-08-30 06:52:23.564 Response from app-1
2024-08-30 06:52:23.579 Response from app-1
2024-08-30 06:52:23.594 Response from app-1
2024-08-30 06:52:23.609 Response from app-1
2024-08-30 06:52:23.625 Response from app-1
2024-08-30 06:52:23.640 Response from app-1
2024-08-30 06:52:23.656 Response from app-1
2024-08-30 06:52:23.671 Response from app-1
2024-08-30 06:52:23.686 Response from app-1
2024-08-30 06:52:23.701 Response from app-1
2024-08-30 06:52:23.716 Response from app-1
2024-08-30 06:52:23.731 Response from app-1
2024-08-30 06:52:23.746 Response from app-1
2024-08-30 06:52:23.761 Response from app-1
2024-08-30 06:52:23.776 Response from app-1
2024-08-30 06:52:23.790 Response from app-1
2024-08-30 06:52:23.805 Response from app-1
2024-08-30 06:52:23.820 Response from app-1
2024-08-30 06:52:23.834 Response from app-1
2024-08-30 06:52:23.849 Response from app-1
2024-08-30 06:52:23.864 Response from app-1
2024-08-30 06:52:23.879 Response from app-1
2024-08-30 06:52:23.894 Response from app-1
2024-08-30 06:52:23.909 Response from app-1
2024-08-30 06:52:23.924 Response from app-1
2024-08-30 06:52:23.939 Response from app-1
2024-08-30 06:52:23.954 Response from app-1
2024-08-30 06:52:23.969 Response from app-1
2024-08-30 06:52:23.983 Response from app-1
2024-08-30 06:52:23.998 Response from app-1
2024-08-30 06:52:24.013 Response from app-1
2024-08-30 06:52:24.028 Response from app-1
2024-08-30 06:52:24.043 Response from app-1
2024-08-30 06:52:24.058 Response from app-1
2024-08-30 06:52:24.073 Response from app-1
2024-08-30 06:52:24.087 Response from app-1
```

Now comes the main testing part:-

Iteration 1:



Iteration 2:

Iteration 3:

Iteration 4:

Latency table:

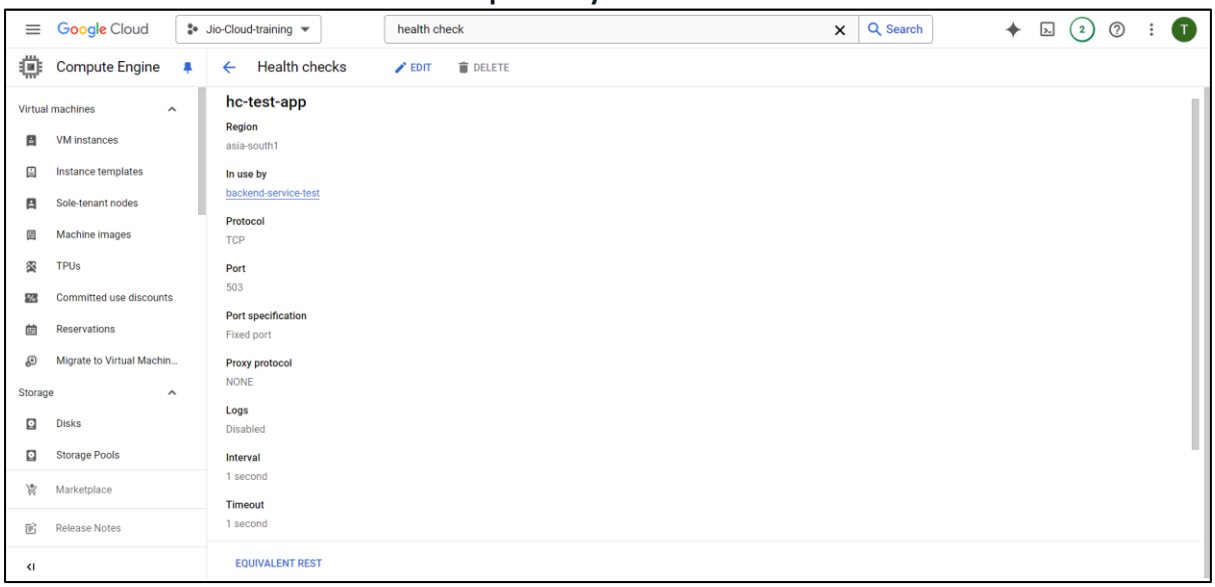
Iteration no.	Stop	Start	Latency
1	07:35:51.374	07:35:58.842	7.468 s
2	07:41:55.235	07:42:03.614	8.379 s
3	07:44:21.985	07:44:28.563	6.578 s
4	07:47:00.706	07:47:08.639	7.933 s

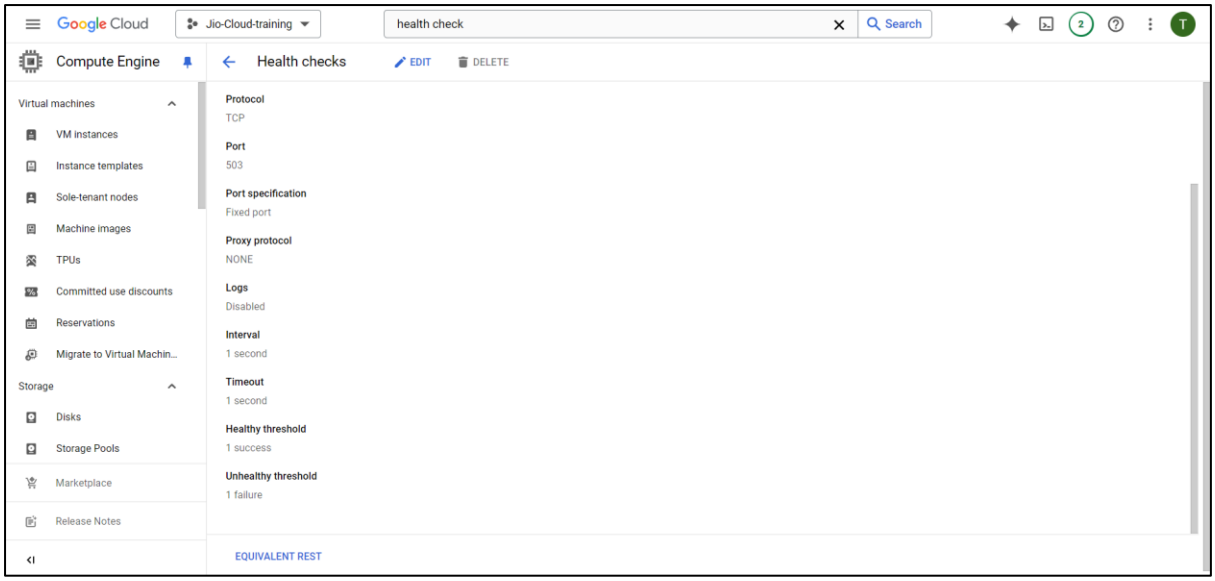
Thus, the average latency is 7.5895 seconds.

A few test case scenarios:

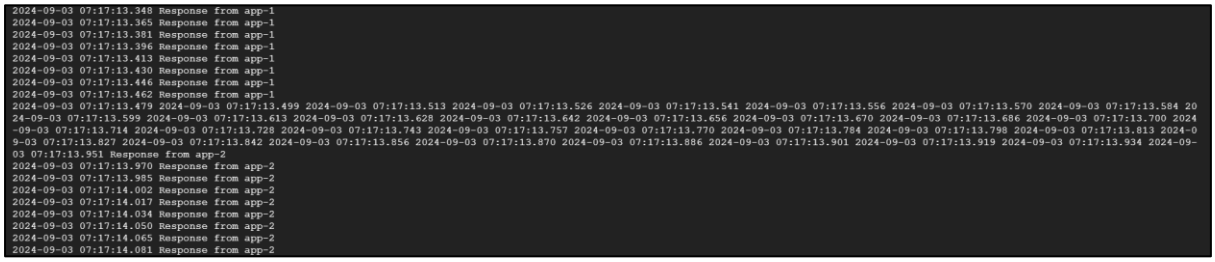
Case 1:

Edit the health check configurations: Interval and Timeout as 1 second, Healthy and Unhealthy threshold as 1 success and 1 failure respectively.





Then perform the failover testing again. We get the iteration as:

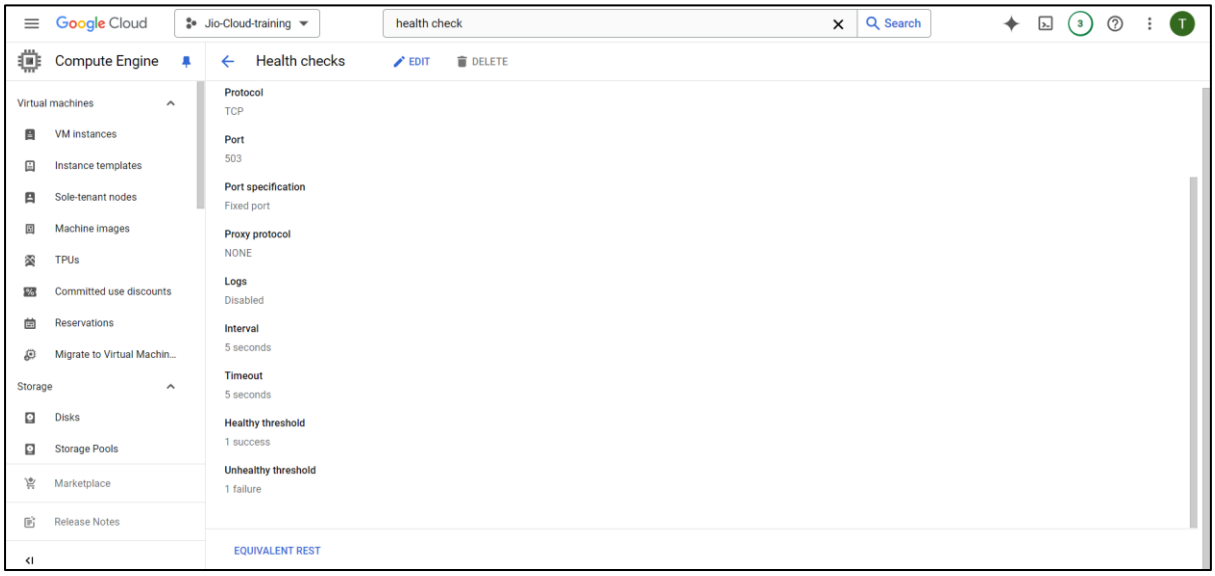


Stop	Start	Latency
07:17:13.462	07:17:13.951	0.489 s

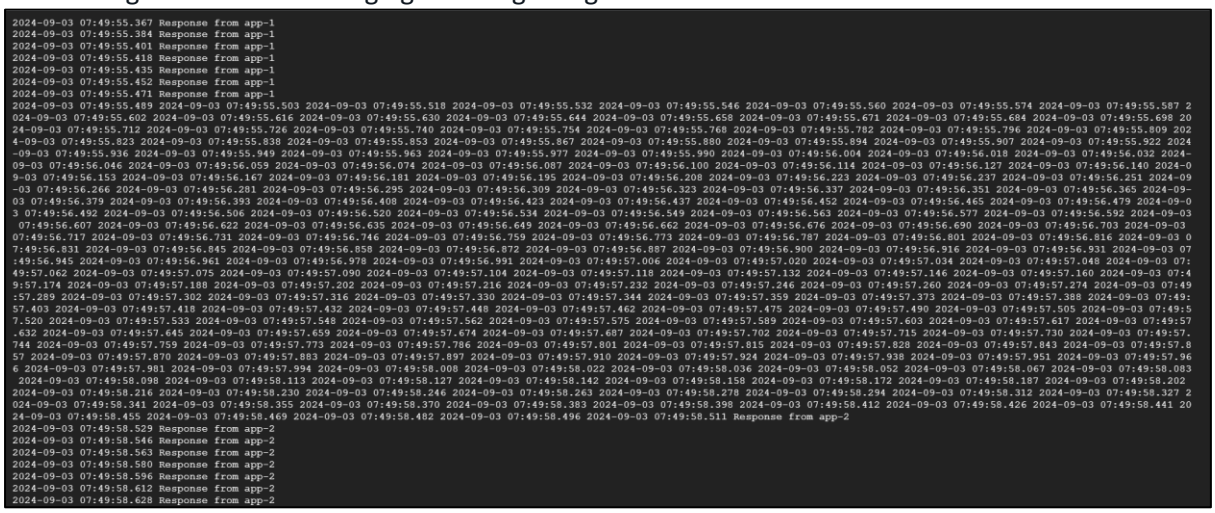
Thus, we can conclude that in this case, the VM was up but the docker container application was down, so the client VM was receiving RST+ACK TCP response from the VM. Due to this, the timeout of 1 second was not observed and the failover time was less than 1 second.

Case 2:

Edit the health check configurations: Interval and Timeout as 5 seconds, Healthy and Unhealthy threshold as 1 success and 1 failure respectively.



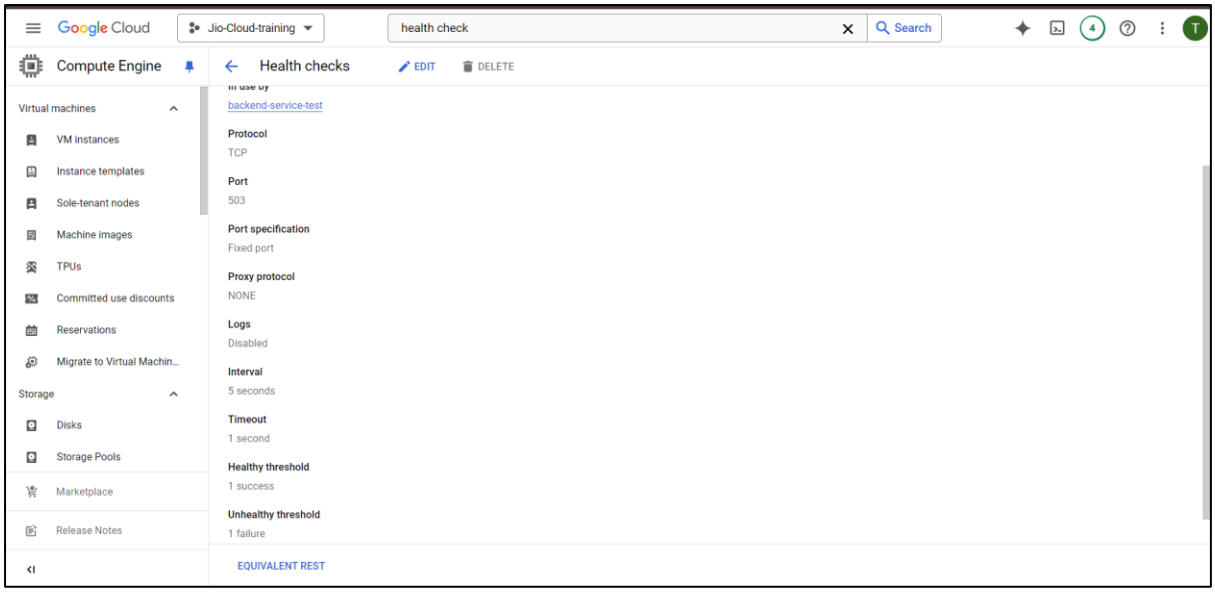
Performing the failover testing again and getting the iteration as:



Stop	Start	Latency
07:49:55.471	**07:49:58.529**	**3.058 s**

Case 3:

Edit the health check configurations: Interval as 5 seconds, Timeout as 1 second, Healthy and Unhealthy threshold as 1 success and 1 failure respectively.



Performing the failover testing and getting the iteration as:

Stop	Start	Latency
09:39:45.689	09:39:49.347	3.658 s

Using Alias IP

What is Alias IP?

Google Cloud alias IP ranges let you assign ranges of internal IP addresses as aliases to a virtual machine's (VM) network interfaces. This is useful if you have multiple services running on a VM and you want to assign each service a different IP address. Using IP aliasing, you can configure multiple internal IP addresses, representing containers or applications hosted in a VM, without having to define a separate network interface. You can assign VM alias IP ranges from either the subnet's primary or secondary ranges. When alias IP ranges are configured, Google Cloud automatically installs Virtual Private Cloud (VPC) network routes for primary and alias IP ranges for the subnet of the primary network interface.

Shell script:

```
  GNU nano 4.8                                                alias_script.sh
#!/bin/bash

# Variables
ALIAS_IP="10.60.27.246/32"
APP_1="app-1"
APP_2="app-2"
PROJECT="jio-cloud-training"
ZONE="asia-south1-a"
NETWORK_INTERFACE="nic0"

APP1_INFO=$(gcloud compute instances describe app-1   --zone=asia-south1-a   --project=jio-cloud-training   --format="json(networkInterfaces)")
APP2_INFO=$(gcloud compute instances describe app-2   --zone=asia-south1-a   --project=jio-cloud-training   --format="json(networkInterfaces)")

remove_alias_ip() {
  VM_NAME=$1
  FINGERPRINT=$2
  echo "Removing alias IP from $VM_NAME..."
  curl -X PATCH \
    -H "Authorization: Bearer $(gcloud auth print-access-token)" \
    -H "Content-Type: application/json" \
    -d '{
      "aliasIpRanges": [],
      "fingerprint": "'$FINGERPRINT'"
    }' \
    "https://compute.googleapis.com/compute/v1/projects/$PROJECT/zones/$ZONE/instances/$VM_NAME/updateNetworkInterface?networkInterface=$NETWORK_INTERFACE"
}

add_alias_ip() {
  VM_NAME=$1
  FINGERPRINT=$2
  echo "Adding alias IP to $VM_NAME..."
  curl -X PATCH \
    -H "Authorization: Bearer $(gcloud auth print-access-token)" \
    -H "Content-Type: application/json" \
    -d '{
      "aliasIpRanges": [
        {
          "ipCidrRange": "'$ALIAS_IP'"
        }
      ],
      "fingerprint": "'$FINGERPRINT'"
    }' \
    "https://compute.googleapis.com/compute/v1/projects/$PROJECT/zones/$ZONE/instances/$VM_NAME/updateNetworkInterface?networkInterface=$NETWORK_INTERFACE"
}

# Check if alias IP is currently in app-1 or app-2 and move it accordingly
if [[ "$APP1_INFO" == *"$ALIAS_IP"* ]]; then
  FINGERPRINT_APP1=$(gcloud compute instances describe $APP_1 --project $PROJECT --zone $ZONE --format="get(networkInterfaces[0].fingerprint)")
  remove_alias_ip $APP_1 $FINGERPRINT_APP1

  FINGERPRINT_APP2=$(gcloud compute instances describe $APP_2 --project $PROJECT --zone $ZONE --format="get(networkInterfaces[0].fingerprint)")
  add_alias_ip $APP_2 $FINGERPRINT_APP2

elif [[ "$APP2_INFO" == *"$ALIAS_IP"* ]]; then
  FINGERPRINT_APP2=$(gcloud compute instances describe $APP_2 --project $PROJECT --zone $ZONE --format="get(networkInterfaces[0].fingerprint)")
  remove_alias_ip $APP_2 $FINGERPRINT_APP2

  FINGERPRINT_APP1=$(gcloud compute instances describe $APP_1 --project $PROJECT --zone $ZONE --format="get(networkInterfaces[0].fingerprint)")
  add_alias_ip $APP_1 $FINGERPRINT_APP1

else
  echo "Alias IP not found on either instance."
fi
```

Now, comes the main testing part:

Iteration 1: Alias IP is currently in app-1, so we'll send the continuous requests from the client VM (or app-3), SSH into app-1 VM, stop the container c_app1 and run the alias_script from app-2 (since the alias IP is present in app-1). Also, we have to check beforehand that while we are stopping c_app1, the container c_app2 must be running inside app-2 to monitor the shifting of traffic using alias IP and thus calculate the latency or the time taken for the failover.

```
tanushree1_mondal@app-3:~$
tanushree1_mondal@app-3:~$ while true; do echo -n "$(date '+%Y-%m-%d %H:%M:%S.%3N') "; curl -s http://10.60.27.246:503/home/index1.html;done
2024-09-11 11:10:23.531 Response from app-1
2024-09-11 11:10:23.555 Response from app-1
2024-09-11 11:10:23.572 Response from app-1
2024-09-11 11:10:23.590 Response from app-1
2024-09-11 11:10:23.608 Response from app-1
```

```
root@app-1:/home/tanushree1_mondal#
root@app-1:/home/tanushree1_mondal# docker stop c_app1
c_app1
root@app-1:/home/tanushree1_mondal#
```

```
tanushree1_mondal@app-2:~$
tanushree1_mondal@app-2:~$ ./alias_script.sh
Removing alias IP from app-1...
{
  "kind": "compute#operation",
  "id": "1283659869807494233",
  "name": "operation-1726053046275-621d60aafccb5-22015f11-f4c3a906",
  "zone": "https://www.googleapis.com/compute/v1/projects/jio-cloud-training/zones/asia-south1-a",
  "operationType": "updateNetworkInterface",
  "targetLink": "https://www.googleapis.com/compute/v1/projects/jio-cloud-training/zones/asia-south1-a/instances/app-1",
  "targetId": "5346838596877 4022",
  "status": "RUNNING",
  "user": "failover-testing-296@jio-cloud-training.iam.gserviceaccount.com",
  "progress": 0,
  "insertTime": "2024-09-11T04:10:46.739-07:00",
  "startTime": "2024-09-11T04:10:46.745-07:00",
  "selfLink": "https://www.googleapis.com/compute/v1/projects/jio-cloud-training/zones/asia-south1-a/operations/operation-1726053046275-621d60aafccb5-22015f11-f4c3a906"
}
Adding alias IP to app-2...
{
  "kind": "compute#operation",
  "id": "8253637487584134230",
  "name": "operation-1726053049180-621d60adc1f7b-ee84a393-561c71bb",
  "zone": "https://www.googleapis.com/compute/v1/projects/jio-cloud-training/zones/asia-south1-a",
  "operationType": "updateNetworkInterface",
  "targetLink": "https://www.googleapis.com/compute/v1/projects/jio-cloud-training/zones/asia-south1-a/instances/app-2",
  "targetId": "1004278200028748696",
  "status": "RUNNING",
  "user": "failover-testing-296@jio-cloud-training.iam.gserviceaccount.com",
  "progress": 0,
  "insertTime": "2024-09-11T04:10:49.627-07:00",
  "startTime": "2024-09-11T04:10:49.634-07:00",
  "selfLink": "https://www.googleapis.com/compute/v1/projects/jio-cloud-training/zones/asia-south1-a/operations/operation-1726053049180-621d60adc1f7b-ee84a393-561c71bb"
}
tanushree1_mondal@app-2:~$
```

```
2024-09-11 11:10:38.710 Response from app-1
2024-09-11 11:10:38.724 2024-09-11 11:10:38.738 2024-09-11 11:10:38.751 2024-09-11 11:10:38.764 2024-09-11 11:10:38.777 2024-09-11 11:10:38.790 2024-09-11 11:10:38.803 2024-09-11 11:10:38.816 20
24-09-11 11:10:38.829 2024-09-11 11:10:38.843 2024-09-11 11:10:38.856 2024-09-11 11:10:38.870 2024-09-11 11:10:38.884 2024-09-11 11:10:38.897 2024-09-11 11:10:38.911 2024-09-11 11:10:38.924 2024
-09-11 11:10:38.937 2024-09-11 11:10:38.951 2024-09-11 11:10:38.964 2024-09-11 11:10:38.977 2024-09-11 11:10:38.990 2024-09-11 11:10:39.004 2024-09-11 11:10:39.017 2024-09-11 11:10:39.030 2024-0
9-11 11:10:39.043 2024-09-11 11:10:39.057 2024-09-11 11:10:39.069 2024-09-11 11:10:39.082 2024-09-11 11:10:39.094 2024-09-11 11:10:39.107 2024-09-11 11:10:39.121 2024-09-11 11:10:39.134 2024-09-
11 11:10:39.148 2024-09-11 11:10:39.161 2024-09-11 11:10:39.174 2024-09-11 11:10:39.187 2024-09-11 11:10:39.201 2024-09-11 11:10:39.214 2024-09-11 11:10:39.227 2024-09-11 11:10:39.241 2024-09-11
 11:10:39.255 2024-09-11 11:10:39.268 2024-09-11 11:10:39.281 2024-09-11 11:10:39.295 2024-09-11 11:10:39.308 2024-09-11 11:10:39.321 2024-09-11 11:10:39.334 2024-09-11 11:10:39.347 2024-09-11 1
1:10:39.361 2024-09-11 11:10:39.373 2024-09-11 11:10:39.386 2024-09-11 11:10:39.399 2024-09-11 11:10:39.412 2024-09-11 11:10:39.425 2024-09-11 11:10:39.438 2024-09-11 11:10:39.451 2024-09-11 11:
10:39.464 2024-09-11 11:10:39.477 2024-09-11 11:10:39.490 2024-09-11 11:10:39.503 2024-09-11 11:10:39.516 2024-09-11 11:10:39.529 2024-09-11 11:10:39.542 2024-09-11 11:10:39.555 2024-09-11 11:10
:39.567 2024-09-11 11:10:39.581 2024-09-11 11:10:39.594 2024-09-11 11:10:39.607 2024-09-11 11:10:39.621 2024-09-11 11:10:39.634 2024-09-11 11:10:39.647 2024-09-11 11:10:39.661 2024-09-11 11:10:3
```

```
7.438 2024-09-11 11:10:47.452 2024-09-11 11:10:47.465 2024-09-11 11:10:47.478 2024-09-11 11:10:47.491 2024-09-11 11:10:47.505 2024-09-11 11:10:47.518 2024-09-11 11:10:47.531 2024-09-11 11:10:47.
545 2024-09-11 11:10:47.559 2024-09-11 11:10:47.572 2024-09-11 11:10:47.585 2024-09-11 11:10:47.598 2024-09-11 11:10:47.611 Response from app-2
2024-09-11 11:10:54.798 Response from app-2
2024-09-11 11:10:54.814 Response from app-2
```

Iteration 2: Alias IP is currently in app-2, so we'll send the continuous requests from the client VM (or app-3), SSH into app-2 VM, stop the container c_app2 and run the alias_script from app-1 (since the alias IP is present in app-2). Also, we have to check from beforehand that while we are stopping c_app2, the container c_app1 must be running inside app-1 to monitor the shifting of traffic using alias IP and thus calculating the latency or the time taken for the failover.

```
tanushree1_mondal@app-3:~$
tanushree1_mondal@app-3:~$ while true; do echo -n "$(date '+%Y-%m-%d %H:%M:%S.%3N') "; curl -s http://10.60.27.246:503/home/index1.html;done
2024-09-11 11:30:40.182 Response from app-2
2024-09-11 11:30:40.199 Response from app-2
2024-09-11 11:30:40.214 Response from app-2
2024-09-11 11:30:40.229 Response from app-2
2024-09-11 11:30:40.245 Response from app-2
2024-09-11 11:30:40.260 Response from app-2
```

```
root@app-2:/home/tanushree1_mondal#
root@app-2:/home/tanushree1_mondal# docker stop c_app2
c_app2
root@app-2:/home/tanushree1_mondal#
```

```
tanushree1_mondal@app-1:~$
tanushree1_mondal@app-1:~$ ./alias_script.sh
Removing alias IP from app-2...
{
  "kind": "compute#operation",
  "id": "5485967850601234838",
  "name": "operation-1726054264930-621d65352ff44-41574c96-2b71ec01",
  "zone": "https://www.googleapis.com/compute/v1/projects/jio-cloud-training/zones/asia-south1-a",
  "operationType": "updateNetworkInterface",
  "targetLink": "https://www.googleapis.com/compute/v1/projects/jio-cloud-training/zones/asia-south1-a/instances/app-2",
  "targetId": "1004278200028748696",
  "status": "RUNNING",
  "user": "failover-testing-296@jio-cloud-training.iam.gserviceaccount.com",
  "progress": 0,
  "insertTime": "2024-09-11T04:31:05.363-07:00",
  "startTime": "2024-09-11T04:31:05.368-07:00",
  "selfLink": "https://www.googleapis.com/compute/v1/projects/jio-cloud-training/zones/asia-south1-a/operations/operation-1726054264930-621d65352ff44-41574c96-2b71ec01"
}
Adding alias IP to app-1...
{
  "kind": "compute#operation",
  "id": "8011400567585554834",
  "name": "operation-1726054269044-621d65391c547-fbf06103-826e9fc0",
  "zone": "https://www.googleapis.com/compute/v1/projects/jio-cloud-training/zones/asia-south1-a",
  "operationType": "updateNetworkInterface",
  "targetLink": "https://www.googleapis.com/compute/v1/projects/jio-cloud-training/zones/asia-south1-a/instances/app-1",
  "targetId": "5346838596877 4022",
  "status": "RUNNING",
  "user": "failover-testing-296@jio-cloud-training.iam.gserviceaccount.com",
  "progress": 0,
  "insertTime": "2024-09-11T04:31:09.481-07:00",
  "startTime": "2024-09-11T04:31:09.486-07:00",
  "selfLink": "https://www.googleapis.com/compute/v1/projects/jio-cloud-training/zones/asia-south1-a/operations/operation-1726054269044-621d65391c547-fbf06103-826e9fc0"
}
tanushree1_mondal@app-1:~$
```

```
2024-09-11 11:30:52.971 Response from app-2
2024-09-11 11:30:52.986 Response from app-2
2024-09-11 11:30:53.002 2024-09-11 11:30:53.015 2024-09-11 11:30:53.028 2024-09-11 11:30:53.041 2024-09-11 11:3
24-09-11 11:30:53.106 2024-09-11 11:30:53.118 2024-09-11 11:30:53.131 2024-09-11 11:30:53.143 2024-09-11 11:30:
```

```
9-11 11:31:06.253 2024-09-11 11:31:06.266 2024-09-11 11:31:06.280 2024-09-11 11:31:06.293 2024-09-11 11:31:06.306 2024-09-11 11:31:06.320 2024-09-11 11:31:06.333 2024-09-11 11:31:06.346 Response
 from app-1
2024-09-11 11:31:13.616 Response from app-1
2024-09-11 11:31:13.633 Response from app-1
2024-09-11 11:31:13.648 Response from app-1
2024-09-11 11:31:13.663 Response from app-1
```

Iteration 3: Alias IP is again in app-1, so we'll send the continuous requests from the client VM (or app-3), SSH into app-1 VM, stop the container c_app1 and run the alias_script from app-2 (since the alias IP is present in app-1). Also, we have to check beforehand that while we are stopping c_app1, the container c_app2 must be running inside app-2 to monitor the shifting of traffic using alias IP and thus calculating the latency or the time taken for the failover.

```
tanushree1_mondal@app-3:~$
tanushree1_mondal@app-3:~$ while true; do echo -n "$(date '+%Y-%m-%d %H:%M:%S.%3N') "; curl -s http://10.60.27.246:503/home/index1.html;done
2024-09-11 11:43:33.122 Response from app-1
2024-09-11 11:43:33.139 Response from app-1
2024-09-11 11:43:33.155 Response from app-1
2024-09-11 11:43:33.170 Response from app-1
2024-09-11 11:43:33.185 Response from app-1
2024-09-11 11:43:33.200 Response from app-1
2024-09-11 11:43:33.216 Response from app-1
```

```
tanushree1_mondal@app-1:~$ sudo su
root@app-1:/home/tanushree1_mondal#
root@app-1:/home/tanushree1_mondal# docker stop c_app1
c_app1
root@app-1:/home/tanushree1_mondal#
```

```
tanushree1_mondal@app-2:~$
tanushree1_mondal@app-2:~$ ./alias_script.sh
Removing alias IP from app-1...
{
  "kind": "compute#operation",
  "id": "5186776060780100755",
  "name": "operation-1726055035797-621d681457e8d-742be8a1-b1cfce84",
  "zone": "https://www.googleapis.com/compute/v1/projects/jio-cloud-training/zones/asia-south1-a",
  "operationType": "updateNetworkInterface",
  "targetLink": "https://www.googleapis.com/compute/v1/projects/jio-cloud-training/zones/asia-south1-a/instances/app-1",
  "targetId": "5346385968774022",
  "status": "RUNNING",
  "user": "failover-testing-296@jio-cloud-training.iam.gserviceaccount.com",
  "progress": 0,
  "insertTime": "2024-09-11T04:43:56.307-07:00",
  "startTime": "2024-09-11T04:43:56.316-07:00",
  "selfLink": "https://www.googleapis.com/compute/v1/projects/jio-cloud-training/zones/asia-south1-a/operations/operation-1726055035797-621d681457e8d-742be8a1-b1cfce84"
}
Adding alias IP to app-2...
{
  "kind": "compute#operation",
  "id": "6251763676278963344",
  "name": "operation-1726055038876-621d68174799c-81225725-988f1ce9",
  "zone": "https://www.googleapis.com/compute/v1/projects/jio-cloud-training/zones/asia-south1-a",
  "operationType": "updateNetworkInterface",
  "targetLink": "https://www.googleapis.com/compute/v1/projects/jio-cloud-training/zones/asia-south1-a/instances/app-2",
  "targetId": "100427820002874869 6",
  "status": "RUNNING",
  "user": "failover-testing-296@jio-cloud-training.iam.gserviceaccount.com",
  "progress": 0,
  "insertTime": "2024-09-11T04:43:59.296-07:00",
  "startTime": "2024-09-11T04:43:59.301-07:00",
  "selfLink": "https://www.googleapis.com/compute/v1/projects/jio-cloud-training/zones/asia-south1-a/operations/operation-1726055038876-621d68174799c-81225725-988f1ce9"
}
tanushree1_mondal@app-2:~$
```

```
2024-09-11 11:43:47.027 Response from app-1
2024-09-11 11:43:47.042 Response from app-1
2024-09-11 11:43:47.058 Response from app-1
2024-09-11 11:43:47.073 2024-09-11 11:43:47.087 2024-09-11 11:43:47.099 2024-09-11 11:43:47.113 2024-09-11 11:43:47.126 2024-09-11 11:43:47.139 2024-09-11 11:43:47.151 2024-09-11 11:43:47.164 20
24-09-11 11:43:47.177 2024-09-11 11:43:47.189 2024-09-11 11:43:47.202 2024-09-11 11:43:47.215 2024-09-11 11:43:47.228 2024-09-11 11:43:47.241 2024-09-11 11:43:47.254 2024-09-11 11:43:47.267 2024
```

```
899 2024-09-11 11:43:56.911 2024-09-11 11:43:56.924 2024-09-11 11:43:56.936 2024-09-11 11:43:56.949 202
1 2024-09-11 11:43:57.014 2024-09-11 11:43:57.026 2024-09-11 11:43:57.040 2024-09-11 11:43:57.053 2024-
2024-09-11 11:43:57.117 2024-09-11 11:43:57.130 2024-09-11 11:43:57.142 Response from app-2
2024-09-11 11:44:04.428 Response from app-2
2024-09-11 11:44:04.445 Response from app-2
2024-09-11 11:44:04.463 Response from app-2
```

Iteration 4: Alias IP is again in app-2, so we'll send the continuous requests from the client VM (or app-3), SSH into app-2 VM, stop the container c_app2 and run the alias_script from app-1 (since the alias IP is present in app-2). Also, we have to check from beforehand that while we are stopping c_app2, the container c_app1 must be running inside app-1 to monitor the shifting of traffic using alias IP and thus calculating the latency or the time taken for the failover.

```
tanushree1_mondal@app-3:~$
tanushree1_mondal@app-3:~$ while true; do echo -n "$(date '+%Y-%m-%d %H:%M:%S.%3N') "; curl -s http://10.60.27.246:503/home/index1.html;done
2024-09-11 11:54:35.495 Response from app-2
2024-09-11 11:54:35.513 Response from app-2
2024-09-11 11:54:35.530 Response from app-2
2024-09-11 11:54:35.547 Response from app-2
```

```
root@app-2:/home/tanushree1_mondal#
root@app-2:/home/tanushree1_mondal# docker stop c_app2
c_app2
root@app-2:/home/tanushree1_mondal#
```

```
tanushree1_mondal@app-1:~$
tanushree1_mondal@app-1:~$ ./alias_script.sh
Removing alias IP from app-2...
{
  "kind": "compute#operation",
  "id": "5456827300656724989",
  "name": "operation-1726055698207-621d6a8c1138d-61b04fff-9f8fe878",
  "zone": "https://www.googleapis.com/compute/v1/projects/jio-cloud-training/zones/asia-south1-a",
  "operationType": "updateNetworkInterface",
  "targetLink": "https://www.googleapis.com/compute/v1/projects/jio-cloud-training/zones/asia-south1-a/instances/app-2",
  "targetId": "1004278200028748696",
  "status": "RUNNING",
  "user": "failover-testing-296@jio-cloud-training.iam.gserviceaccount.com",
  "progress": 0,
  "insertTime": "2024-09-11T04:54:58.653-07:00",
  "startTime": "2024-09-11T04:54:58.658-07:00",
  "selfLink": "https://www.googleapis.com/compute/v1/projects/jio-cloud-training/zones/asia-south1-a/operations/operation-1726055698207-621d6a8c1138d-61b04fff-9f8fe878"
}
Adding alias IP to app-1...
{
  "kind": "compute#operation",
  "id": "5964482865856789498",
  "name": "operation-1726055701319-621d6a8f08ee2-c5addc9e-f829674b",
  "zone": "https://www.googleapis.com/compute/v1/projects/jio-cloud-training/zones/asia-south1-a",
  "operationType": "updateNetworkInterface",
  "targetLink": "https://www.googleapis.com/compute/v1/projects/jio-cloud-training/zones/asia-south1-a/instances/app-1",
  "targetId": "5346838596774022",
  "status": "RUNNING",
  "user": "failover-testing-296@jio-cloud-training.iam.gserviceaccount.com",
  "progress": 0,
  "insertTime": "2024-09-11T04:55:01.750-07:00",
  "startTime": "2024-09-11T04:55:01.755-07:00",
  "selfLink": "https://www.googleapis.com/compute/v1/projects/jio-cloud-training/zones/asia-south1-a/operations/operation-1726055701319-621d6a8f08ee2-c5addc9e-f829674b"
}
tanushree1_mondal@app-1:~$
```

```
2024-09-11 11:54:49.511 Response from app-2
2024-09-11 11:54:49.525 Response from app-2
2024-09-11 11:54:49.540 2024-09-11 11:54:49.553 2024-09-11 11:54:49.567 2024-09-11 11:54:49.580 2024-09-11 11:54:49.593 2024-09-11 11:54:49.607 2024-09-11 1
24-09-11 11:54:49.646 2024-09-11 11:54:49.659 2024-09-11 11:54:49.673 2024-09-11 11:54:49.687 2024-09-11 11:54:49.700 2024-09-11 11:54:49.713 2024-09-11 11:
```

```
376 2024-09-11 11:54:59.389 2024-09-11 11:54:59.402 2024-09-11 11:54:59.4
8 2024-09-11 11:54:59.491 2024-09-11 11:54:59.503 Response from app-1
2024-09-11 11:55:06.708 Response from app-1
2024-09-11 11:55:06.723 Response from app-1
2024-09-11 11:55:06.738 Response from app-1
```

Latency table:

Iteration no.	Stop	Start	Latency
1	11:10:38.710	11:10:47.611	8.901 s
2	11:30:52.986	11:31:06.346	13.360 s
3	11:43:47.058	11:43:57.142	10.084 s
4	11:54:49.525	11:54:59.503	9.978 s

Thus, the average latency is 10.580 seconds.

Using Route Modification

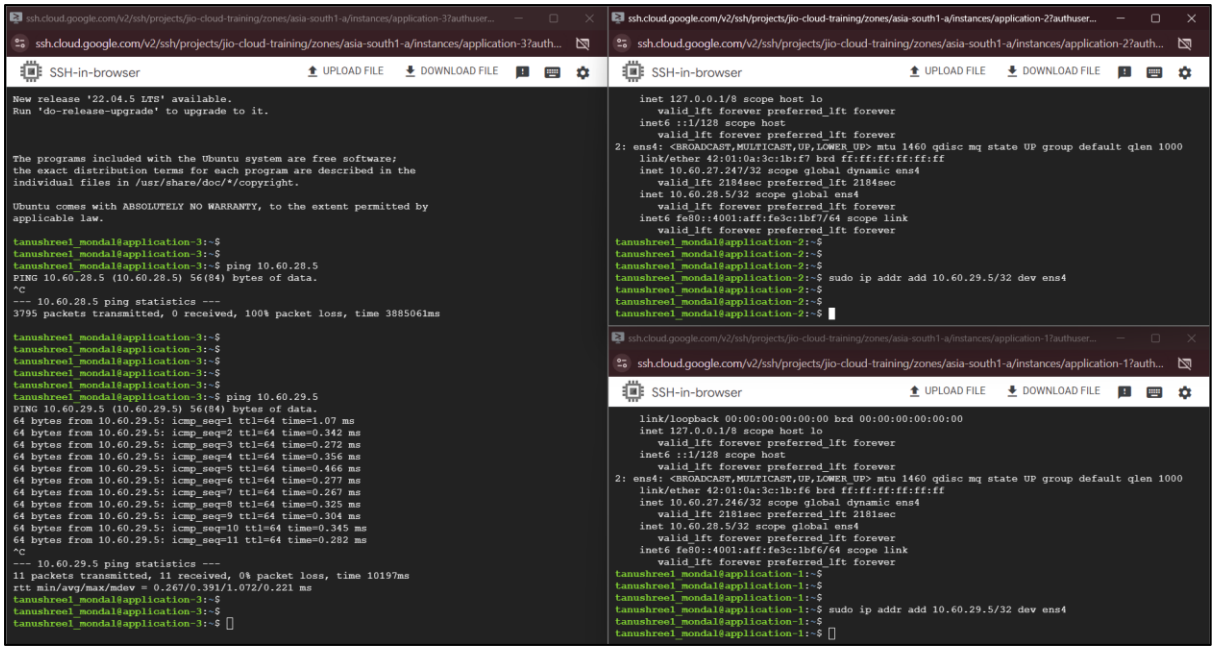
Google Cloud routes define the paths that network traffic takes from a virtual machine (VM) instance to other destinations. These destinations can be inside your Google Cloud Virtual Private Cloud (VPC) network (for example, in another VM) or outside it. In a VPC network, a route consists of a single *destination* prefix in CIDR format and a single *next hop*. When an instance in a VPC network sends a packet, Google Cloud delivers the packet to the route's next hop if the packet's destination address is within the route's destination range.

Every VPC network uses a scalable, distributed virtual routing mechanism. There is no physical device that's assigned to the network. Some routes can be applied selectively, but the routing table for a VPC network is defined at the VPC network level. Each VM instance has a controller that is kept informed of all applicable routes from the network's routing table. Each packet leaving a VM is delivered to the appropriate next hop of an applicable route based on a routing order. When you add or delete a route, the set of changes is propagated to the VM controllers by using an eventually consistent design.

We created three new VMs:

```
# Create Instances in a loop
echo "Creating Instances..."
for i in $(seq 1 $NUM_VMS); do
    #Create VM with IP forwarding enabled
    echo "Creating VM application-$i with IP forwarding enabled..."
   gcloud compute instances create application-$i \
        --machine-type=e2-micro \
        --network=$VPC_NAME \
        --subnet=$SUBNET_NAME \
        --image-project="ubuntu-os-cloud" \
        --image-family="ubuntu-2004-lts"  \
        --boot-disk-device-name="test-os-$i" \
        --boot-disk-size="20GB" \
        --boot-disk-type="pd-balanced" \
        --private-network-ip="10.60.27.$((245 + $i))" \
        --no-address \
        --can-ip-forward
    echo "VM application-$i with IP forwarding enabled created."

done
```

Adding the virtual IP address 10.60.29.5/32 in both of the VMs application-1 and application-2 and pinging them from application-3 or the client VM.



Then, we created two routes namely testroute-application-1 and testroute-application-2 with priority set as 0 and 1 respectively.

```
root@application-1:/home/tanushreel_mondal# ./route_script.sh
{
  "kind": "compute#operation",
  "id": "2854672764578652 71",
  "name": "operation-1726566103783-6224d7f4c193e-fc0a6ec3-5896bacc",
  "operationType": "insert",
  "targetLink": "https://www.googleapis.com/compute/v1/projects/jio-cloud-training/global/routes/testroute-application-1",
  "targetId": "6094603070501875767",
  "status": "RUNNING",
  "user": "failover-testing-296@jio-cloud-training.iam.gserviceaccount.com",
  "progress": 0,
  "insertTime": "2024-09-17T02:41:44.065-07:00",
  "startTime": "2024-09-17T02:41:44.097-07:00",
  "selfLink": "https://www.googleapis.com/compute/v1/projects/jio-cloud-training/global/operations/operation-1726566103783-6224d7f4c193e-fc0a6ec3-5896bacc"
}
Route testroute-application-1 created successfully.
{
  "kind": "compute#operation",
  "id": "5777464403352154167",
  "name": "operation-1726566104587-6224d7f585d23-af5f3bdc-a242a7f1",
  "operationType": "insert",
  "targetLink": "https://www.googleapis.com/compute/v1/projects/jio-cloud-training/global/routes/testroute-application-2",
  "targetId": "5154793651234467895",
  "status": "RUNNING",
  "user": "failover-testing-296@jio-cloud-training.iam.gserviceaccount.com",
  "progress": 0,
  "insertTime": "2024-09-17T02:41:44.980-07:00",
  "startTime": "2024-09-17T02:41:45.016-07:00",
  "selfLink": "https://www.googleapis.com/compute/v1/projects/jio-cloud-training/global/operations/operation-1726566104587-6224d7f585d23-af5f3bdc-a242a7f1"
}
Route testroute-application-2 created successfully.
Routes for application-1 and application-2 created successfully.
root@application-1:/home/tanushreel_mondal#
```

Shell script:

Script in application-2: This script is used to automate the management of routes between two instances (application-1 and application-2) in a Google Cloud environment. It involves stopping a Docker container on one instance, deleting existing routes, and creating new routes for both instances. The ID_RSA_PATH is the local file path to the SSH private key (id_rsa) used to access the remote instance with the remote IP address 10.60.27.246.

ssh -i "$ID_RSA_PATH" scriptadmin@"$REMOTE_IP" -t 'sudo docker stop c_application1'

This line connects to the remote server (REMOTE_IP) via SSH using the private key (ID_RSA_PATH). It stops the Docker container c_application1 on the remote server by executing the docker stop command.

This script automates route management between instances in a Google Cloud environment, ensuring proper failover and routing logic for application instances.

```
  GNU nano 4.8                                                    final4.sh
#!/bin/bash

# Variables
PROJECT_ID="jio-cloud-training"
NETWORK_NAME="vpc-lb-test"
ZONE="asia-south1-a"
DEST_RANGE="10.60.29.0/24"
ID_RSA_PATH="/home/kundrapu_sadwik/id_rsa"
REMOTE_IP="10.60.27.246"
ACCESS_TOKEN=$(gcloud auth application-default print-access-token)

# SSH and stop Docker container on the remote server
ssh -i "$ID_RSA_PATH" scriptadmin@"$REMOTE_IP" -t 'sudo docker stop c_application1'

# Function to create a route
create_route() {
  local ROUTE_NAME=$1
  local PRIORITY=$2
  local INSTANCE_NAME=$3

  # Create the new route
  curl -s -X POST \
    -H "Authorization: Bearer $ACCESS_TOKEN" \
    -H "Content-Type: application/json" \
    -d '{
      "name": "'"$ROUTE_NAME"'",
      "network": "https://www.googleapis.com/compute/v1/projects/'"$PROJECT_ID"'/global/networks/'"$NETWORK_NAME"'",
      "destRange": "'"$DEST_RANGE"'",
      "priority": '"$PRIORITY"',
      "nextHopInstance": "https://www.googleapis.com/compute/v1/projects/'"$PROJECT_ID"'/zones/'"$ZONE"'/instances/'"$INSTANCE_NAME"'"
    }' \
    https://compute.googleapis.com/compute/v1/projects/$PROJECT_ID/global/routes

  echo "Route $ROUTE_NAME created with priority $PRIORITY."
}
```

```
# Function to delete a route
delete_route() {
  local ROUTE_NAME=$1

  curl -s -X DELETE \
    -H "Authorization: Bearer $ACCESS_TOKEN" \
    https://compute.googleapis.com/compute/v1/projects/$PROJECT_ID/global/routes/$ROUTE_NAME

  echo "Route $ROUTE_NAME deleted."
}

# Delete routes for application-1 and application-2
delete_route "testroute-application-1"
delete_route "testroute-application-2"

sleep 5

# Create new routes for application-1 and application-2
create_route "testroute-application-2" 0 "application-2"

sleep 3

create_route "testroute-application-1" 1 "application-1"

echo "Routes for application-1 and application-2 have been processed."
```

Script in application-1:

```
  GNU nano 4.8                                                    final4.sh
#!/bin/bash

# Variables
PROJECT_ID="jio-cloud-training"
NETWORK_NAME="vpc-lb-test"
ZONE="asia-south1-a"
DEST_RANGE="10.60.29.0/24"
ID_RSA_PATH="/home/kundrapu_sadwik/id_rsa"
REMOTE_IP="10.60.27.247"
ACCESS_TOKEN=$(gcloud auth application-default print-access-token)

# SSH and stop Docker container on the remote server
ssh -i "$ID_RSA_PATH" scriptadmin@"$REMOTE_IP" -t 'sudo docker stop c_application2'

# Function to create a route
create_route() {
  local ROUTE_NAME=$1
  local PRIORITY=$2
  local INSTANCE_NAME=$3

  # Create the new route
  curl -s -X POST \
    -H "Authorization: Bearer $ACCESS_TOKEN" \
    -H "Content-Type: application/json" \
    -d '{
      "name": "'"$ROUTE_NAME"'",
      "network": "https://www.googleapis.com/compute/v1/projects/'"$PROJECT_ID"'/global/networks/'"$NETWORK_NAME"'",
      "destRange": "'"$DEST_RANGE"'",
      "priority": '"$PRIORITY"',
      "nextHopInstance": "https://www.googleapis.com/compute/v1/projects/'"$PROJECT_ID"'/zones/'"$ZONE"'/instances/'"$INSTANCE_NAME"'"
    }' \
    https://compute.googleapis.com/compute/v1/projects/$PROJECT_ID/global/routes

  echo "Route $ROUTE_NAME created with priority $PRIORITY."
}
```

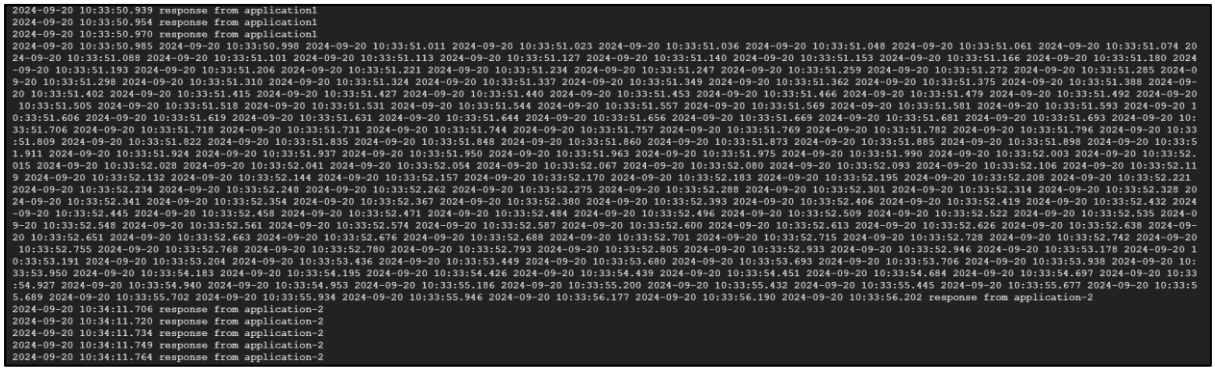
```
# Function to delete a route
delete_route() {
  local ROUTE_NAME=$1

  curl -s -X DELETE \
    -H "Authorization: Bearer $ACCESS_TOKEN" \
    https://compute.googleapis.com/compute/v1/projects/$PROJECT_ID/global/routes/$ROUTE_NAME

  echo "Route $ROUTE_NAME deleted."
}

# Delete routes for application-1 and application-2
delete_route "testroute-application-1"
delete_route "testroute-application-2"

sleep 5

# Create new routes for application-1 and application-2
create_route "testroute-application-1" 0 "application-1"

sleep 3

create_route "testroute-application-2" 1 "application-2"

echo "Routes for application-1 and application-2 have been processed."
```

This is the while command that we have to run in application-3.

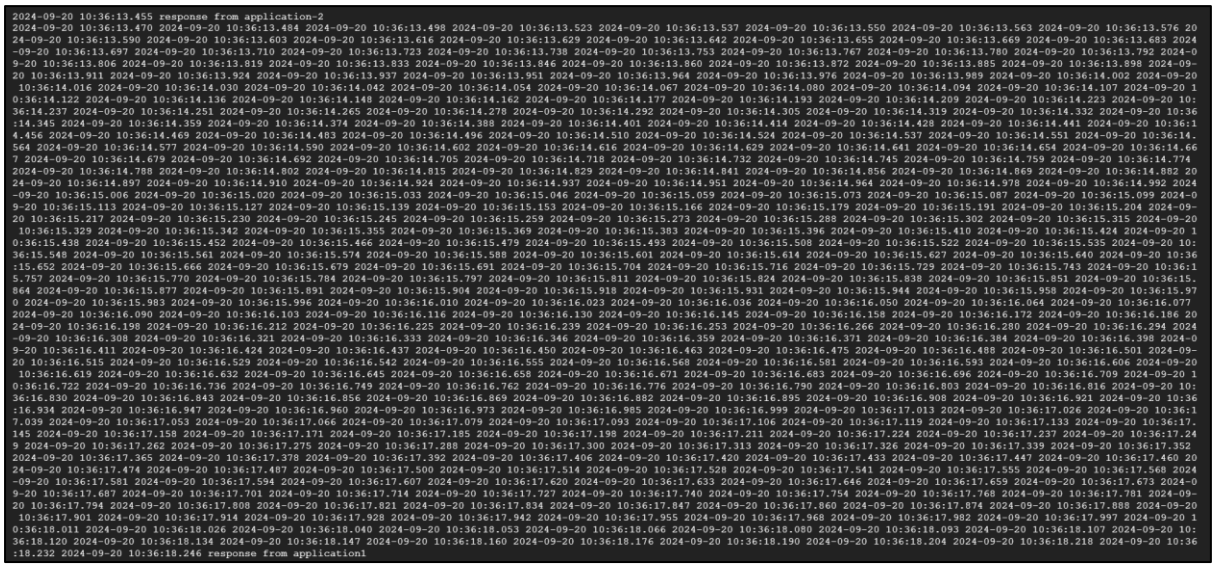
```
tanushree1_mondal@application-3:~$
tanushree1_mondal@application-3:~$
tanushree1_mondal@application-3:~$
tanushree1_mondal@application-3:~$
tanushree1_mondal@application-3:~$ while true; do echo -n "$(date '+%Y-%m-%d %H:%M:%S.%3N') "; curl -s http://10.60.29.5:503/home/index.html;done
```

Now, comes the main testing part:

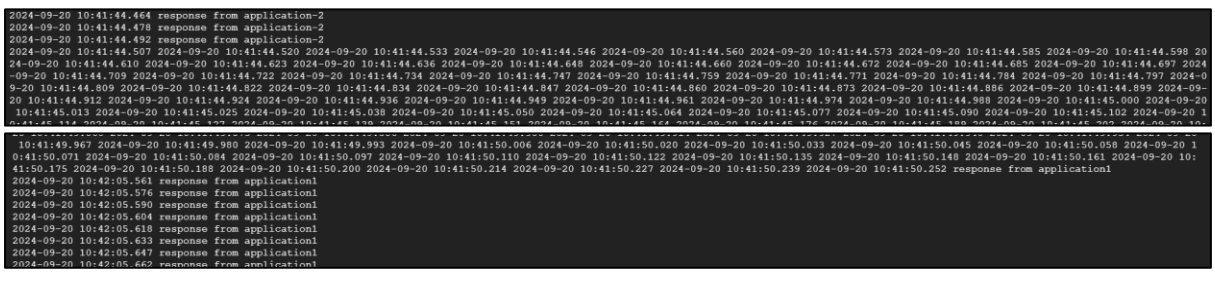
Iteration 1:



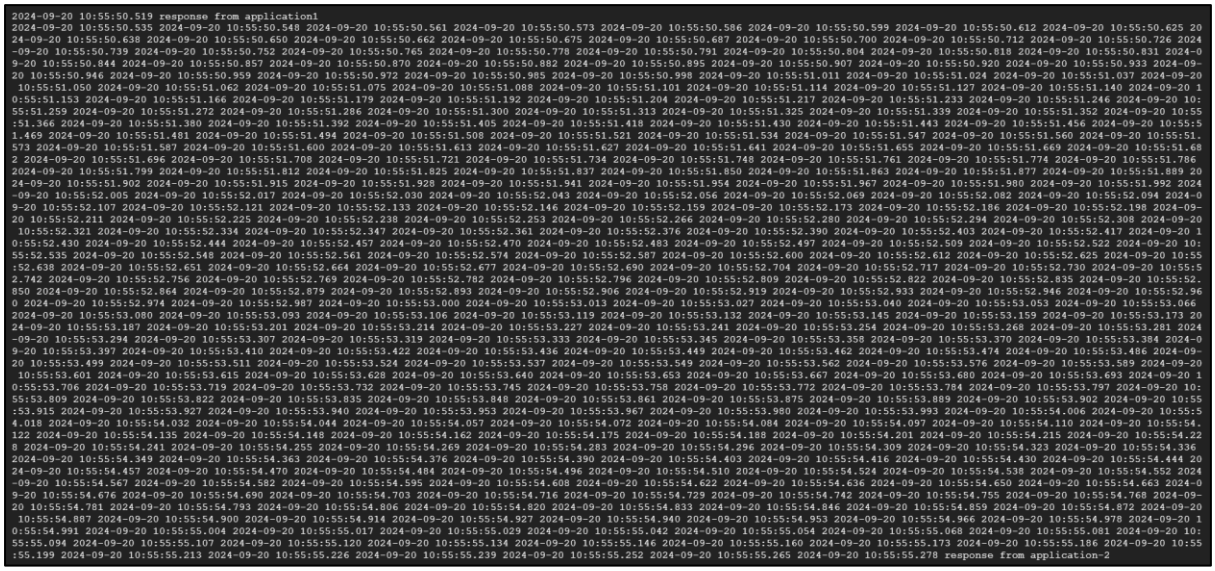
Iteration 2:



Iteration 3:



Iteration 4:



Latency table:

Iteration no.	Stop	Start	Latency
1	10:33:50.970	10:33:56.202	5.232 s
2	10:36:13.455	10:36:18.246	4.791 s
3	10:41:44.492	10:41:50.252	5.760 s
4	10:55:50.535	10:55:55.278	4.743 s

Thus, the average latency is 5.132 seconds.

Comparison of the three Failover Testing methods for Application Redundancy:

Data Table:

Failover Technique	Average Latency (seconds)
Load Balancer	**0.489**
Alias IP	**10.580**
Route Modification	**5.132**

Summary of Techniques:

1. **Using Load Balancer:**

- **Description:** Automatically distributes incoming traffic across multiple instances and reroutes traffic when an instance fails.
- **Pros:** Fastest failover, minimal manual intervention, low latency.
- **Cons:** Requires load balancing setup and may incur additional costs.

2. **Using Alias IP:**

- **Description:** IP addresses are dynamically moved between instances to route traffic in case of failure.
- **Pros:** Maintains static IP addressing.
- **Cons:** Involves more overhead than load balancers, relatively higher latency.

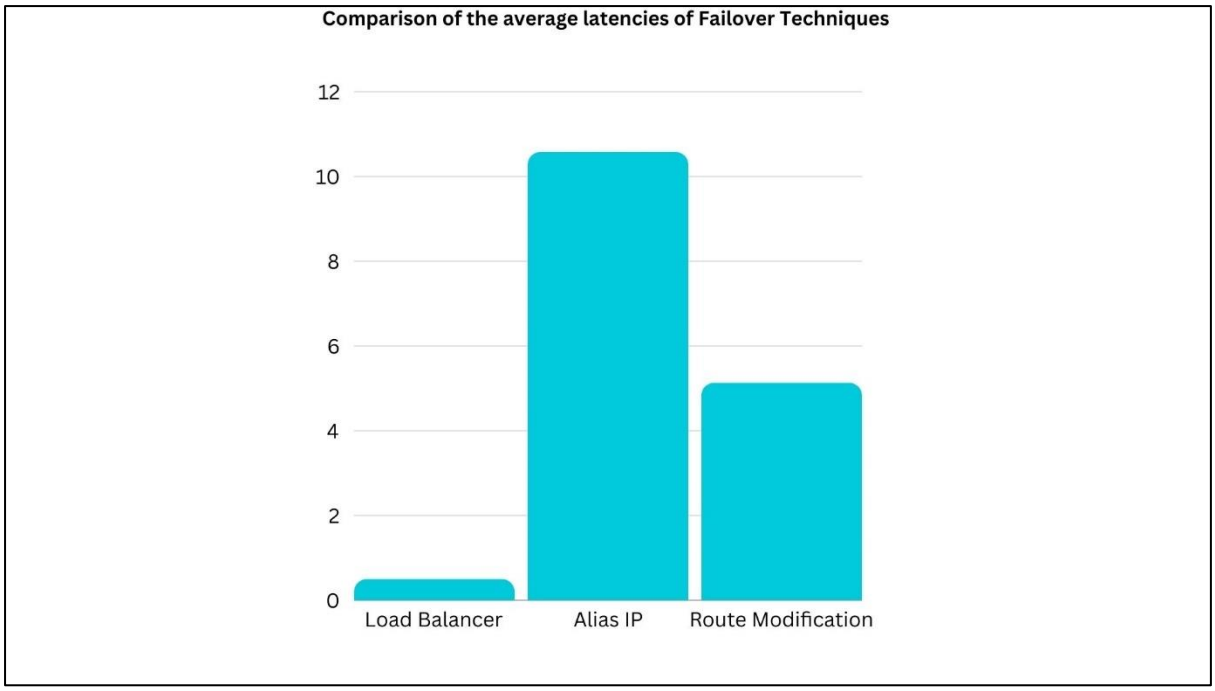
3. **Using Route Modification:**

- **Description:** Modifies the routing table to direct traffic to a different instance during failover.
- **Pros:** Can be automated in some environments, but typically slower.
- **Cons:** High latency due to manual or semi-automated route changes, less suitable for real-time dynamic traffic.

Conclusion:

- **Best Technique:** *Using Load Balancer*
- **Reason:** The load balancer offers the fastest and most efficient failover with the lowest average latency of 0.489 seconds. Its automation and low overhead make it ideal for dynamic and large-scale applications, ensuring minimal downtime and a seamless user experience.

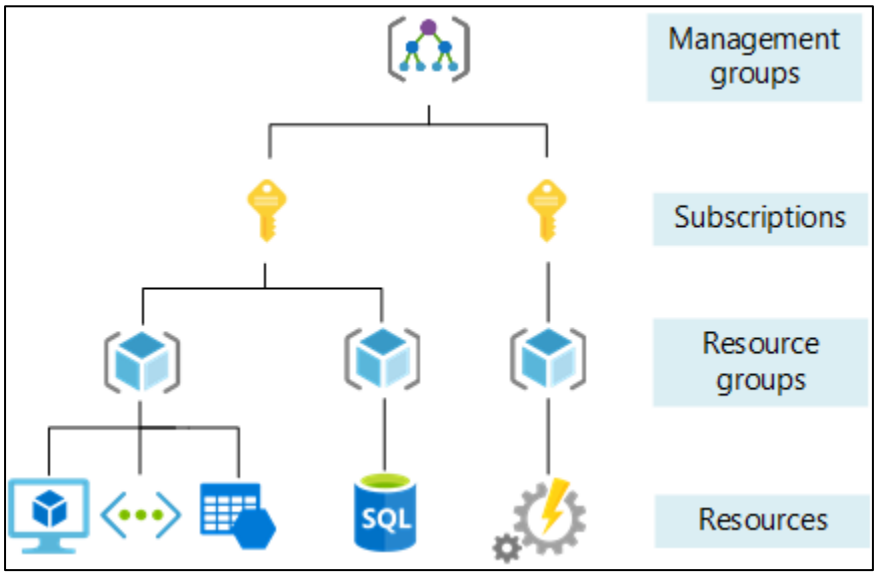
Visual Comparison (at a glance):



Here is a bar chart for the visual comparison of the average latencies across the three failover techniques: Load Balancer, Alias IP, and Route Modification. This chart clearly shows that:

- Load Balancer has the lowest latency (0.489 seconds), making it the fastest failover option.
- Alias IP has a moderate latency (10.58 seconds), which is slightly higher but still reasonable.
- Route Modification exhibits the highest latency (5.132 seconds), which makes it the slowest technique for failover.

HIGH AVAILABILITY – MICROSOFT AZURE



Source: https://docs.microsoft.com/en-us/azure/

Infrastructure

Azure Storage Redundancy

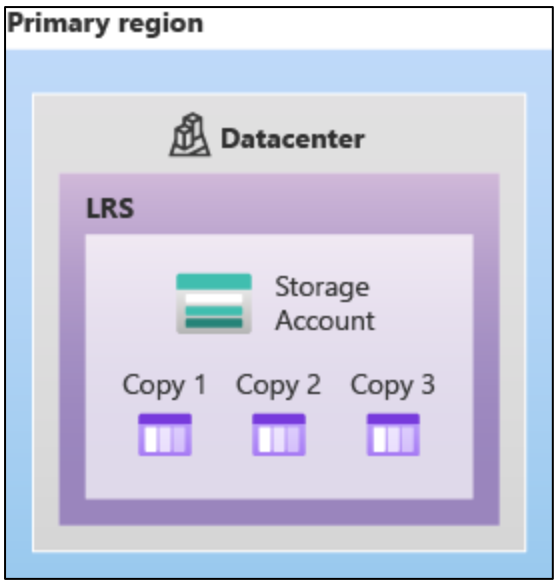
Azure Storage always stores multiple copies of your data so that it's protected from planned and unplanned events, including transient hardware failures, network or power outages, and massive natural disasters. Redundancy ensures that your storage account meets its availability and durability targets even in the face of failures. The services that comprise Azure Storage are managed through a common Azure resource called a *storage account*. The storage account represents a shared pool of storage that can be used to deploy storage resources such as blob containers (Blob Storage), file shares (Azure Files), tables (Table Storage), or queues (Queue Storage).

Azure Storage offers two options for how your data is replicated in the primary region:

-LRS or Locally Redundant Storage replicates your storage account three times within a single data centre in the primary region. LRS provides at least 99.999999999% (11 nines) durability of objects over a given year.

LRS is the lowest-cost redundancy option and offers the least durability compared to other options. LRS protects your data against server rack and drive failures. However, if a disaster such as fire or flooding occurs within the data centre, all replicas of a storage account using LRS may be lost or unrecoverable. To mitigate this risk, Microsoft recommends using zone-redundant storage (ZRS), geo-redundant storage (GRS), or geo-zone-redundant storage (GZRS). A write request to a storage account that is using LRS happens synchronously. The write operation returns successfully only after the data is written to all three replicas.

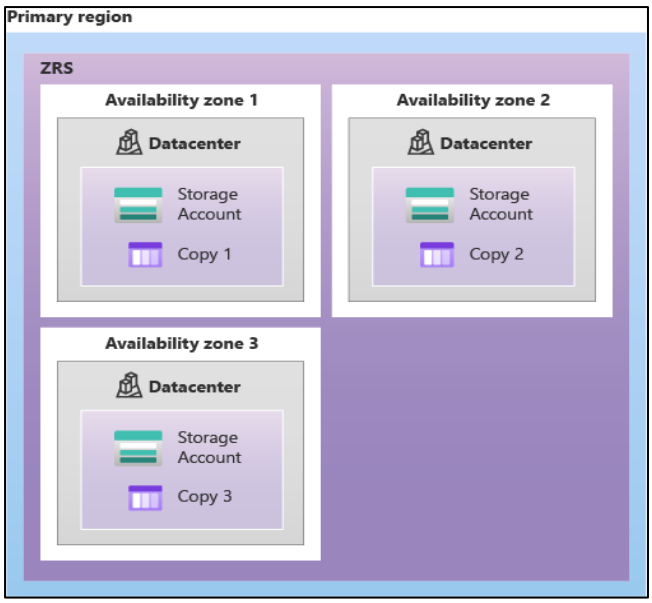
The following diagram shows how your data is replicated within a single data centre with LRS:



Source: https://docs.microsoft.com/en-us/azure/

-ZRS or Zone-redundant storage replicates your storage account synchronously across three Azure availability zones in the primary region. Each availability zone is a separate physical location with independent power, cooling, and networking. ZRS offers durability for storage resources of at least 99.9999999999% (12 9's) over a given year. With ZRS, your data is still accessible for both read and write operations even if a zone becomes unavailable. If a zone becomes unavailable, Azure undertakes networking updates, such as DNS repointing. A write request to a storage account that is using ZRS happens synchronously. The write operation returns successfully only after the data is written to all replicas across the three availability zones. Microsoft recommends using ZRS in the primary region for scenarios that require high availability. Microsoft recommends using ZRS for Azure Files workloads. However, ZRS by itself may not protect your data against a regional disaster where multiple zones are permanently affected. For protection against regional disasters, Microsoft recommends using geo-zone-redundant storage (GZRS), which uses ZRS in the primary region and also geo-replicates your data to a secondary region.

The following diagram shows how your data is replicated across availability zones in the primary region with ZRS:



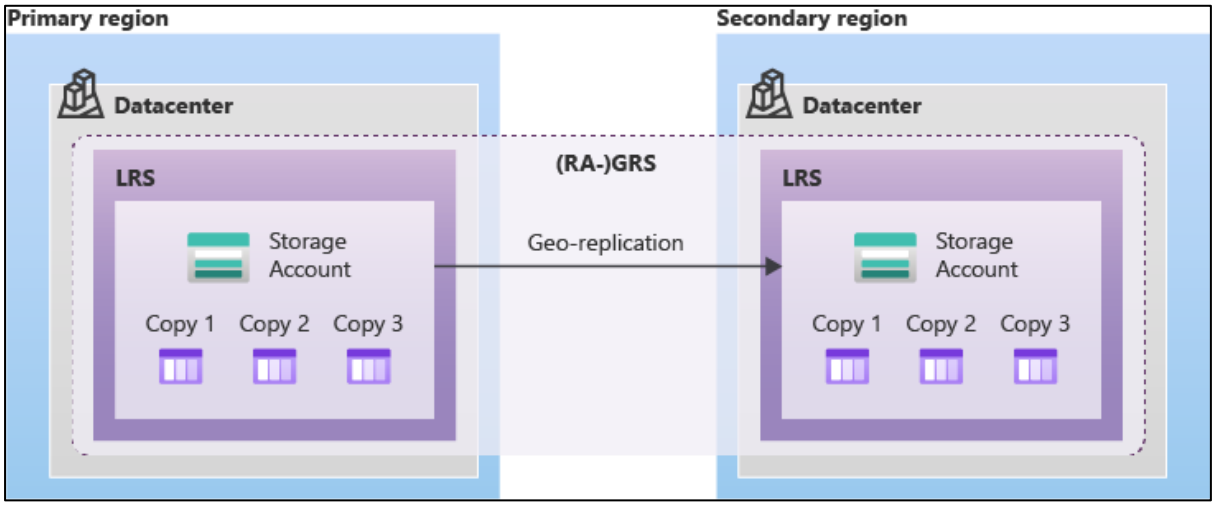
Source: https://docs.microsoft.com/en-us/azure/

Azure Storage offers two options for copying your data to a secondary region:

-GRS or Geo-redundant storage copies your data synchronously three times within a single physical location in the primary region using LRS. It then copies your data asynchronously to a single physical location in a secondary region that is hundreds of miles away from the primary region. GRS offers durability for storage resources of at least 99.99999999999999% (16 9's) over a given year.

A write operation is first committed to the primary location and replicated using LRS. The update is then updated asynchronously to the secondary region. When data is written to the secondary location, it's also replicated within that location using LRS.

The following diagram shows how your data is replicated with GRS or RA-GRS:

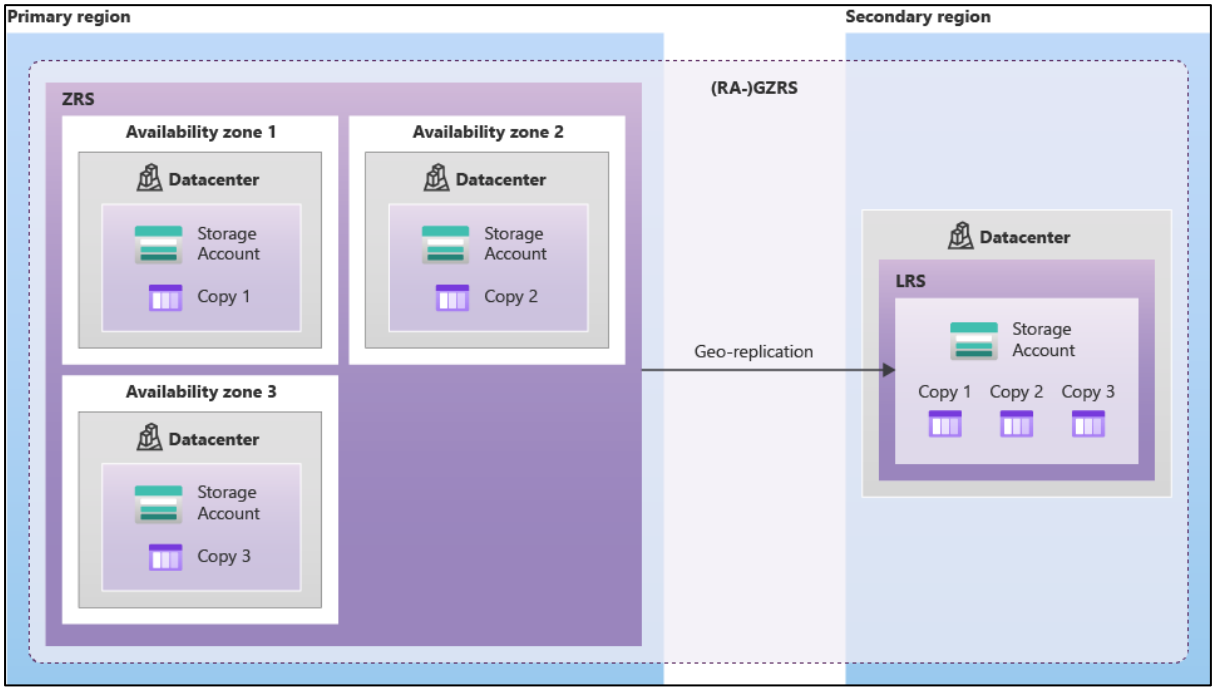


Source: https://docs.microsoft.com/en-us/azure/

-GZRS or Geo-zone-redundant storage combines the high availability provided by redundancy across availability zones with protection from regional outages provided by geo-replication. Data in a GZRS storage account is copied across three Azure availability zones in the primary region and is also replicated to a secondary geographic region for protection from regional disasters. Microsoft recommends using GZRS for applications requiring maximum consistency, durability, and availability, excellent performance, and resilience for disaster recovery.

With a GZRS storage account, you can continue to read and write data if an availability zone becomes unavailable or is unrecoverable. Additionally, your data is also durable in the case of a complete regional outage or a disaster in which the primary region isn't recoverable. GZRS is designed to provide at least 99.99999999999999% (16 9's) durability of objects over a given year.

The following diagram shows how your data is replicated with GZRS or RA-GZRS:



Source: https://docs.microsoft.com/en-us/azure/

Availability sets

What is an availability set?

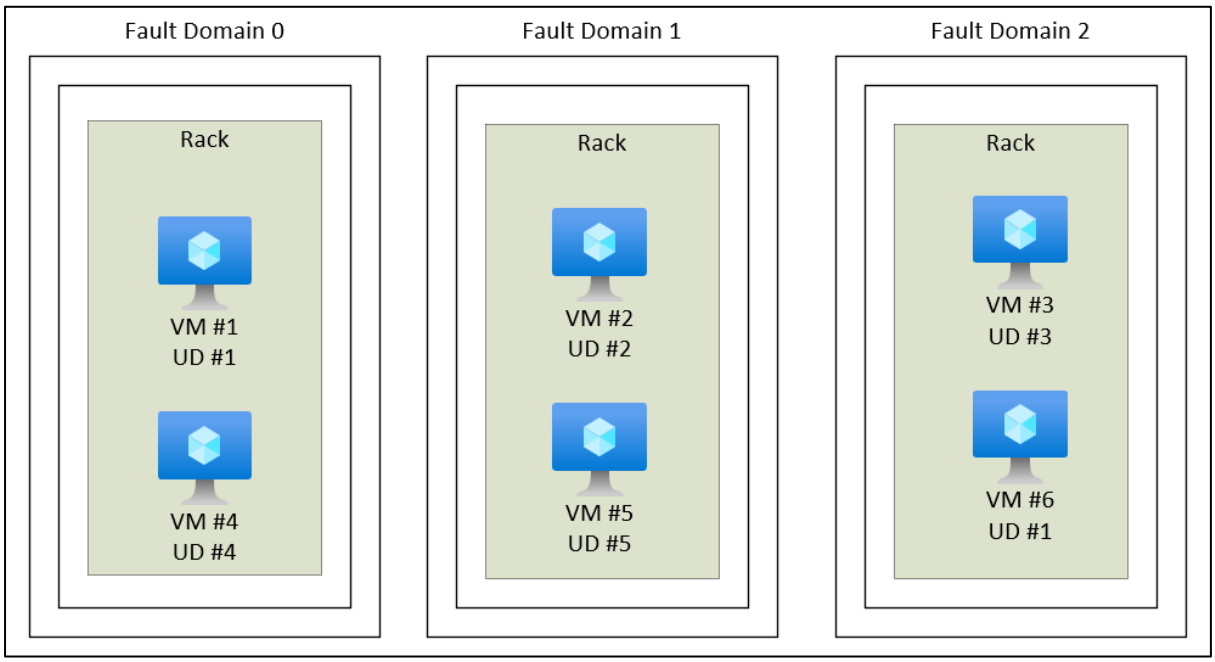
Availability sets are logical groupings of VMs that reduce the chance of correlated failures bringing down related VMs at the same time. Availability sets place VMs in different fault domains for better reliability, especially beneficial if a region doesn't support availability zones. When using availability sets, create two or more VMs within an availability set. Using two or more VMs in an availability set helps highly available applications and meets the 99.95% Azure SLA. There's no extra cost for using availability sets, you only pay for each VM instance you create.

Availability sets offer improved VM-to-VM latencies compared to availability zones, since VMs in an availability set are allocated in closer proximity. Availability sets have fault isolation for many possible failures, minimizing single points of failure, and offering high availability.

How do availability sets work?

Each virtual machine in your availability set is assigned an *update domain* and a *fault domain* by the underlying Azure platform. Each availability set can be configured with up to 3 fault domains and 20 update domains. These configurations can't be changed once the availability set has been created. Update domains indicate groups of virtual machines and underlying physical hardware that can be rebooted at the same time. A rebooted update domain is given 30 minutes to recover before maintenance is initiated on a different update domain.

Fault domains define the group of virtual machines that share a common power source and network switch. By default, the virtual machines configured within your availability set are separated across up to three fault domains. While placing your virtual machines into an availability set doesn't protect your application from operating system or application-specific failures, it does limit the impact of potential physical hardware failures, network outages, or power interruptions.



Source: https://docs.microsoft.com/en-us/azure/

VMs are also aligned with disk fault domains. This alignment ensures that all the managed disks attached to a VM are within the same fault domains.

Only VMs with managed disks can be created in a managed availability set. The number of managed disk fault domains varies by region - either two or three managed disk fault domains per region. The following command retrieves a list of fault domains per region:

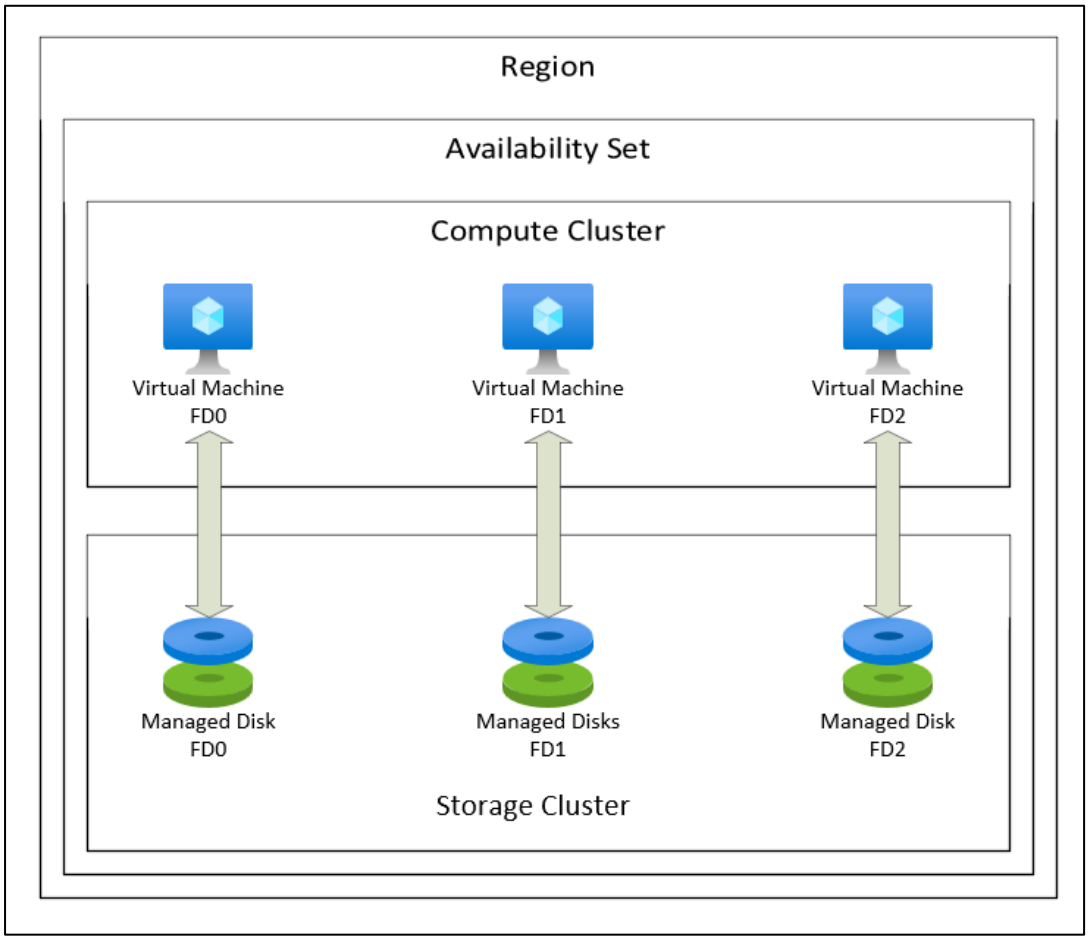
```
Azure CLI                                                    Copy   Open Cloud Shell
z vm list-skus --resource-type availabilitySets --query '[?name==`Aligned`].{Location:locati
```

Source: https://docs.microsoft.com/en-us/azure/

Under certain circumstances, two VMs in the same availability set might share a fault domain. You can confirm a shared fault domain by going to your availability set and checking the Fault Domain column. A shared fault domain might be caused by the completing following sequence when you deployed the VMs:

- Deploy the first VM.
- Stop/deallocate the first VM.
- Deploy the second VM.

Under these circumstances, the OS disk of the second VM might be created on the same fault domain as the first VM, so the two VMs will be on the same fault domain. To avoid this issue, we recommend that you don't stop/deallocate VMs between deployments.



Source: https://docs.microsoft.com/en-us/azure/

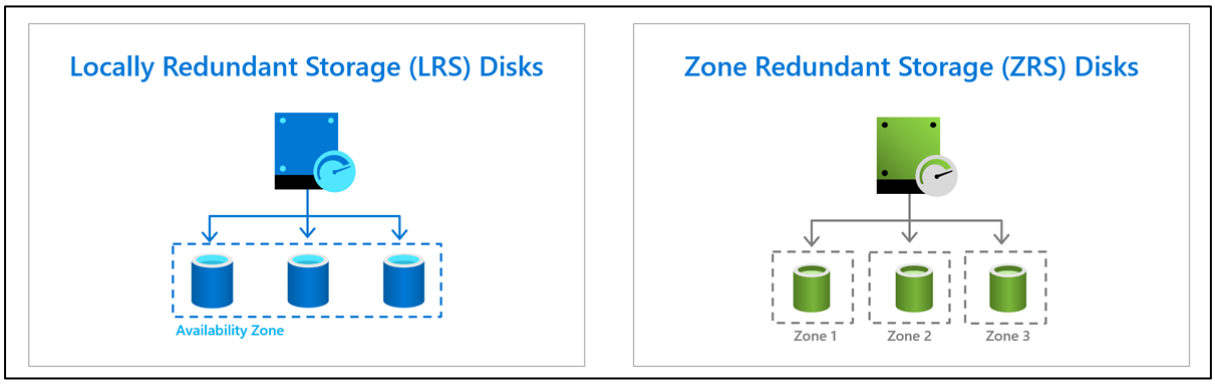
Azure managed disks

Azure managed disks are block-level storage volumes that are managed by Azure and used with Azure Virtual Machines. Managed disks are like a physical disk in an on-premises server but, virtualized. With managed disks, all you have to do is specify the disk size, and the disk type, and provision the disk. Once you provision the disk, Azure handles the rest. The available types of disks are ultra disks, premium solid-state drives (SSD), standard SSDs, and standard hard disk drives (HDD).

Benefits of managed disks:

- Highly durable and available
- Simple and scalable VM deployment
- Integration with availability sets
- Integration with Availability Zones: Managed disks support Availability Zones, which is a high-availability offering that protects your applications from data centre failures.
- Azure Backup support
- Azure Disk Backup
- Granular access control (Azure role-based access control or Azure RBAC)
- Upload your VHD (Virtual Hard Disk)

Managed disks have two redundancy models, locally redundant storage (LRS) disks, and zone-redundant storage (ZRS) disks. The following diagram depicts how data is replicated with either model.



Source: https://docs.microsoft.com/en-us/azure/

Managed disks offer two different kinds of encryption. The first is Server Side Encryption (SSE), which is performed by the storage service. The second one is Azure Disk Encryption (ADE), which you can enable on the OS and data disks for your VMs.

- **Server-side encryption** is enabled by default for all managed disks, snapshots, and images, in all the regions where managed disks are available.
- **Azure Disk Encryption** allows you to encrypt the OS and Data disks used by an IaaS Virtual Machine. This encryption includes managed disks.

Managed disk snapshots: A managed disk snapshot is a read-only crash-consistent full copy of a managed disk that is stored as a standard managed disk by default. With snapshots, you can back up

your managed disks at any point in time. These snapshots exist independent of the source disk and can be used to create new managed disks.

Images: Managed disks also support creating a managed custom image. This image contains all managed disks associated with a VM, including both the OS and data disks. This managed custom image enables the creation of hundreds of VMs using your custom image without the need to copy or manage any storage accounts.

Images versus snapshots: It's important to understand the difference between images and snapshots. With managed disks, you can take an image of a generalized VM that has been deallocated. This image includes all of the disks attached to the VM. You can use this image to create a VM, and it includes all of the disks. A snapshot is a copy of a disk at the point in time the snapshot is taken. It applies only to one disk. If you have a VM that has one disk (the OS disk), you can take a snapshot or an image of it and create a VM from either the snapshot or the image.

Availability Zones

Availability Zones in Azure are unique physical locations within an Azure region, designed to ensure high availability and resilience for your applications and data. They are designed so that if one zone experiences an outage, then regional services, capacity and high availability are supported by the remaining zones. Each Availability Zone is an isolated location, with independent power, cooling, and networking. Using Availability Zones helps protect your applications and data from data centre failures and provides higher availability for your critical resources. We can achieve high availability and business continuity in all available Azure regions without compromising data residency. Access your data even if your primary data centre fails while supporting high availability needs and backup.

Key features of Availability Zones:

- Fault isolation
- High Availability
- Low Latency
- Resiliency

Using Availability Zones:

Zone-Redundant Services: Azure offers several zone-redundant services that automatically distribute instances across Availability Zones for high availability:

Azure Virtual Machines:

- **Zone Redundant VM Scale Sets**: Automatically distribute VM instances across zones.
- **Availability Sets**: Distribute VMs across multiple zones for fault tolerance.

Azure Storage:

- **Zone-Redundant Storage (ZRS)**: Replicates data synchronously across three zones within a region.

Azure SQL Database:

- **Zone-Redundant Configuration**: Deploy instances across zones for high availability.

Azure Kubernetes Service (AKS):

- **Node Pool Distribution**: Distributes node pools across Availability Zones.

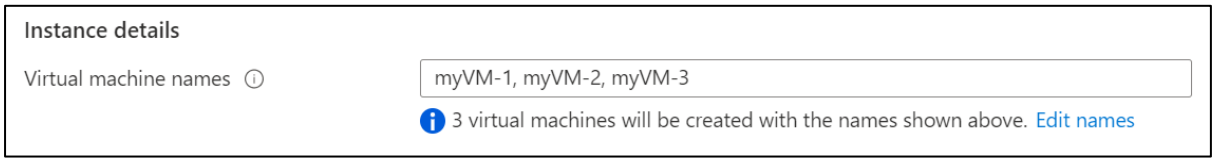
Manually Distributing Resources:

You can also manually distribute resources across zones to achieve higher availability:

- **Virtual Machines**: When creating a VM, select the desired Availability Zone.
- **Managed Disks**: Ensure that VMs and their corresponding disks are in the same zone to avoid cross-zone charges and latency.
- **Load Balancers**: Use zone-redundant load balancers to distribute traffic across VMs in different zones.

A best practice is to use Zone-Redundant Azure Services that offer built-in zone redundancy for higher availability.

Example Configuration: For the **Availability zone**, the drop-down defaults to *Zone 1*. If you choose multiple zones, a new VM is created in each zone. For example, if you select all three zones, then three VMs are created. The VM names are the original names you entered, with **-1**, **-2**, and **-3** appended to the name based on the number of zones selected. If you want, you can edit each of the default VM names.



Source: https://docs.microsoft.com/en-us/azure/

Here's an example of how to create VMs across multiple Availability Zones using the Azure CLI.

Creating VMs in Different Availability Zones:

Create a VM in zone 1

az vm create --resource-group myResourceGroup --name myVM1 --image UbuntuLTS --zone 1 --admin-username azureuser --generate-ssh-keys

Create a VM in zone 2

az vm create --resource-group myResourceGroup --name myVM2 --image UbuntuLTS --zone 2 --admin-username azureuser --generate-ssh-keys

Create a VM in zone 3

az vm create --resource-group myResourceGroup --name myVM3 --image UbuntuLTS --zone 3 --admin-username azureuser --generate-ssh-keys

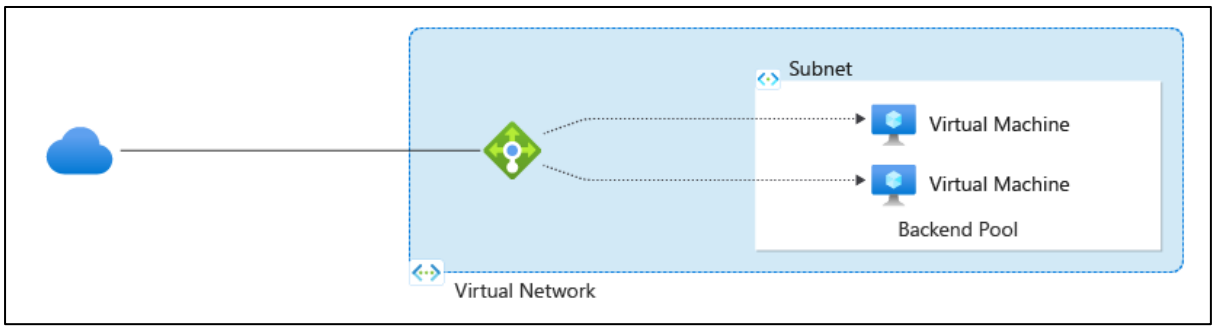
Azure Load Balancer

Load balancing refers to efficiently distributing incoming network traffic across a group of backend servers or resources.

Azure Load Balancer operates at layer 4 of the Open Systems Interconnection (OSI) model. It's the single point of contact for clients. The load balancer distributes inbound flows that arrive at the load balancer's front end to backend pool instances. These flows are according to configured load-balancing rules and health probes. The backend pool instances can be Azure Virtual Machines or instances in a Virtual Machine Scale Set.

A **public load balancer** can provide outbound connections for virtual machines (VMs) inside your virtual network. These connections are accomplished by translating their private IP addresses to public IP addresses. Public Load Balancers are used to load balance internet traffic to your VMs.

An **internal (or private) load balancer** is used in scenarios where private IPs are needed at the frontend only. Internal load balancers are used to load balance traffic inside a virtual network. A load balancer frontend can be accessed from an on-premises network in a hybrid scenario.



Source: https://docs.microsoft.com/en-us/azure/

The above figure represents balancing multi-tier applications by using both public and internal Load Balancer.

Why use Azure Load Balancer?

With Azure Load Balancer, you can scale your applications and create highly available services. Load balancer supports both inbound and outbound scenarios. Load balancer provides low latency and high throughput, and scales up to millions of flows for all TCP and UDP applications.

Key scenarios that you can accomplish using Azure Standard Load Balancer include:

- Load balance **internal** and **external** traffic to Azure virtual machines.
- Pass-through load balancing which results in ultra-low latency.

- Increase availability by distributing resources **within** and **across** zones.
- Configure **outbound connectivity** for Azure virtual machines.
- Use **health probes** to monitor load-balanced resources.
- Employ **port forwarding** to access virtual machines in a virtual network by public IP address and port.
- Enable support for **load-balancing** of **IPv6**.
- Standard load balancer provides multi-dimensional metrics through Azure Monitor. These metrics can be filtered, grouped, and broken out for a given dimension. They provide current and historical insights into the performance and health of your service. Insights for Azure Load Balancer offers a preconfigured dashboard with useful visualizations for these metrics. Resource Health is also supported. Review **Standard load balancer diagnostics** for more details.
- Load balance services on **multiple ports, multiple IP addresses, or both**.
- Move **internal** and **external** load balancer resources across Azure regions.
- Load balance TCP and UDP flow on all ports simultaneously using **HA ports**.
- Chain Standard Load Balancer and Gateway Load Balancer.

Secure by default:

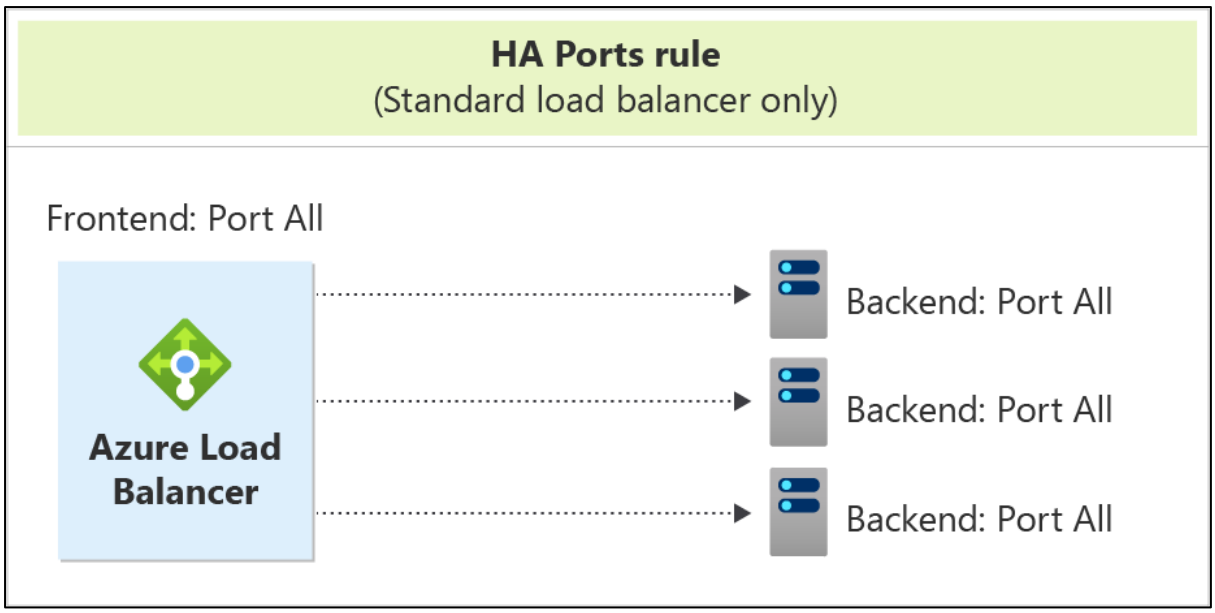
- Standard load balancer is built on the zero trust network security model.
- Standard Load Balancer is secure by default and part of your virtual network. The virtual network is private and isolated.
- Standard load balancers and standard public IP addresses are closed to inbound connections unless opened by Network Security Groups. NSGs are used to explicitly permit allowed traffic. If you don't have an NSG on a subnet or NIC of your virtual machine resource, traffic isn't allowed to reach this resource. To learn about NSGs and how to apply them to your scenario, see Network Security Groups.
- Basic load balancer is open to the internet by default and is offered at no charge.
- Load balancer doesn't store customer data.

High Availability Ports: A load balancer rule configured with **'protocol - all and port - 0'** is known as a High Availability (HA) port rule. This rule enables a single rule to load-balance all TCP and UDP flows that arrive on all ports of an internal Standard Load Balancer.

The load-balancing decision is made per flow. This action is based on the following five-tuple connection:

- source IP address
- source port
- destination IP address
- destination port
- protocol

The HA ports load-balancing rules help you with critical scenarios, such as high availability and scale for network virtual appliances (NVAs) inside virtual networks. The feature can help when a large number of ports must be load-balanced.



Source: https://docs.microsoft.com/en-us/azure/

Conclusion

The internship project titled **"Study of High Availability Approaches in Public Cloud Environment "** has provided valuable insights into the implementation and management of high-availability solutions within cloud environments, specifically using Google Cloud Platform. This project has demonstrated the critical importance of high availability in public cloud environments, particularly within the telecommunications sector.

By leveraging Managed Instance Groups, load balancing, and failover strategies, the project underscored the importance of designing a system that can effectively handle disruptions and maintain continuous service availability. The performance evaluation of different failover mechanisms—load balancer configuration, alias IP failover, and route modification—revealed crucial data regarding latency and response times, contributing to a deeper understanding of their efficiency in real-world scenarios.

Additionally, including theoretical concepts from Microsoft Azure enriched the project's scope, allowing for a comparative perspective on cloud services. This comprehensive approach highlighted best practices in GCP and illuminated the similarities and differences in high-availability strategies across major cloud platforms.

Overall, the findings provide a framework for future implementations and underscore the need for continuous improvement and adaptation in response to evolving technological challenges. As organizations increasingly rely on cloud infrastructure, the insights gained from this project will be instrumental in guiding strategies for optimizing high availability and ensuring the reliability of critical applications.

References

Google Cloud SDK (gcloud CLI) Documentation:

- Google Cloud SDK: https://cloud.google.com/sdk/docs
- gcloud CLI Overview: https://cloud.google.com/sdk/gcloud/reference

Google Compute Engine Documentation:

- Overview of Google Compute Engine: https://cloud.google.com/compute/docs
- Managed Instance Groups: https://cloud.google.com/compute/docs/instance-groups/manage-instance-groups
- Load Balancing Documentation: https://cloud.google.com/load-balancing/docs

Google Cloud Networking Documentation:

- VPC Documentation: https://cloud.google.com/vpc/docs
- Firewall Rules: https://cloud.google.com/vpc/docs/firewalls

Google Cloud Monitoring Documentation:

- Monitoring and Logging: https://cloud.google.com/monitoring/docs

Microsoft Azure Documentation:

- Azure Overview: https://docs.microsoft.com/en-us/azure/
- Azure Virtual Machines: https://docs.microsoft.com/en-us/azure/virtual-machines/
- Azure Load Balancer: https://docs.microsoft.com/en-us/azure/load-balancer/
- Azure Resource Manager: https://docs.microsoft.com/en-us/azure/azure-resource-manager/

Cloud Architecture and Best Practices:

- Google Cloud Best Practices: https://cloud.google.com/docs/architecture
- Azure Architecture Centre: https://docs.microsoft.com/en-us/azure/architecture/