Deployment Archetypes for Cloud Applications

ANNA BERENBERG and BRAD CALDER, Google, Inc.

This is a survey article that explores six Cloud-based deployment archetypes for Cloud applications and the tradeoffs between them to achieve high availability, low end-user latency, and acceptable costs. These are (1) Zonal, (2) Regional, (3) Multi-regional, (4) Global, (5) Hybrid, and (6) Multi-cloud deployment archetypes. The goal is to classify cloud applications into a set of deployment archetypes and deployment models that tradeoff their needs around availability, latency, and geographical constraints with a focus on serving applications. This enables application owners to better examine the tradeoffs of each deployment model and what is needed for achieving the availability and latency goals for their application.

CCS Concepts: • **Computer systems organization** → **Cloud computing**; **Availability**; • **Networks** → **Cloud computing**;

Additional Key Words and Phrases: Cloud deployments, cloud archetypes, cloud architecture, cloud availability

ACM Reference format:
Anna Berenberg and Brad Calder. 2022. Deployment Archetypes for Cloud Applications. *ACM Comput. Surv.* 55, 3, Article 61 (February 2022), 48 pages.
https://doi.org/10.1145/3498336

1 INTRODUCTION

In looking at how applications have changed over the past 20 years, we have evolved from a world where planned maintenance downtime was standard and business applications were typically available only 99% of the year [117] to today where applications are expected to be up and running 24/7. Similarly for latency, online transactions over the internet took on the order of seconds 20 years ago [123], where users today expect transactions to complete in milliseconds.

The drive toward higher availability and lower end-user latency is pushing application developers and operators to evolve and deploy applications with the best availability and latency possible. Even applications built around deployment options that were only available 20+ years ago need to be supported in this 24/7 available and low-latency world. With Cloud as the preferred platform for deploying and running applications, this means Cloud needs to help achieve these goals for (a) applications that have been around since before Cloud existed (Enterprise applications) and (b) greenfield applications born in the Cloud (Cloud-native applications).

As businesses move to the Cloud, some applications may need to continue to run as they did on-premises and potentially benefit from Cloud-managed services (e.g., storage, relational databases, data analytics, SAP). In comparison, some businesses may want to evolve applications within

Authors' address: A. Berenberg and B. Calder, Google, Inc, 1600 Amphitheatre Parkway Mountain View, CA 94043; emails: aberenberg@google.com, bcalder@google.com.

0360-0300/2022/02-ART61 $15.00
https://doi.org/10.1145/3498336

existing boundaries or go for partial or complete rewrites to achieve higher availability, better end-user latency, and increased operational efficiency and agility. In addition, the underlying technologies used for existing applications influences the deployment options that are suitable when migrating applications to the Cloud. The choices each business will make for their applications will be different depending on the needs of each business.

When running an application in the Cloud, there are many important aspects an application owner needs to address including security, identity, data recovery, data and traffic management, cost optimization, and much more. We touch on some of these, but this article is primarily focused on exploring different deployment models for the application serving stack.

We explore six Cloud-based deployment archetypes for Cloud applications and the tradeoffs between them to achieve high availability and low end-user latency. These are (1) Zonal, (2) Regional, (3) Multi-regional, (4) Global, (5) Hybrid, and (6) Multi-cloud deployment archetypes. Cloud applications consist of multiple services and microservices, and the application may mix services of different archetypes based upon its needs. We look at multiple categories of application deployments from Enterprise to Cloud-native applications, their impact on availability and latency, and how they can leverage these six deployment archetypes.

1.1 Principles of Availability

The level of availability each part of the application is targeting depends on its business purpose [33]. Some applications only need three nines (99.9%) availability, which means the service can be unavailable for at most 43 min a month. Other applications need four nines (99.99%) availability, which means the application can only be unavailable for at most 52 min a year. Then there are those mission-critical applications that need five nines (99.999%) availability, where they can only be unavailable at most 5 min a year. To achieve these levels of availability, it is important to understand what is needed for each part of the application and invest in closing the gap between current and desired availability for each part.

The investment in availability comes at a cost, but it is often crucial to the long-term success of the business, since availability directly influences the reputation of the business and the satisfaction of the application's users. For the purpose of this article, we group together into the **overall availability of the application** the following: (a) the time to access the application, (b) the time to get a response with valid results, (c) the application's access to its data, (d) the assurance that data is stored and maintained with integrity, and (e) the application's ability to scale and handle peak traffic demands.

For an application, availability is best designed from the start. Adding availability as a feature later can require re-architecting the application and potentially a full rewrite. A key part of the design is how the application reasons about fault domains and how it provides redundancy and scales across those fault domains to maximize availability. A fault domain is a set of infrastructure parts that together represent a single point of failure. To increase availability, applications need to run and store their data across multiple fault domains (zones and regions) and have the ability to balance load or failover in case of failure. Data needs to be replicated and backed up so that it is never lost, and checks must be in place to make sure data is never corrupted. In addition, applications need to be able to quickly load balance across multiple instances of the application to scale to the largest traffic the application can have. This includes minimizing time for startup and shutdown, so applications can be restarted, and scaled up and out quickly.

Two additional important concepts for minimizing the impact of an outage are (a) sharding the application and (b) making sure all application updates are done incrementally and can be rolled back. Applications may apply sharding across their users or data so that they are served across different fault domains [2]. In this way, an issue with one fault domain will only impact

a subset of the users/data, thus containing the failure radius (often called the blast radius) [59]. Similarly, code and configuration changes should be rolled out incrementally across the different fault domains to gradually introduce a change into production, with the ability to quickly roll back if any production issues are discovered to return the application to a healthy state. This allows code and configuration production issues to be discovered early on and reduces the impact to only those parts of the application running in the fault domains being updated. In addition to being able to quickly roll back recent application changes, having the ability to drain or shed load from the affected fault domains is often used to quickly mitigate issues.

These techniques collectively determine how large of an impact there is to the application and its users when there is an outage. Ideally there is no impact on users when an issue occurs, but if the best design and deployment practices are followed, when there is an issue then only a small set of users of the application are affected in one or a few fault domains.

Finally, applications need to understand their dependencies, the availability and failure modes of those dependencies, and to evaluate the multiplicative implications across these dependencies on the application's design and availability. Typically, the fewer dependencies a service has the better, and it is better to avoid linking in code, calling out to other services and APIs that bring in unknown dependencies. As part of the overall manageability, separating out parts of the application into its vital and non-vital services, identifying the availability targets of each, and continuously improving the vital parts, are important. If a service is vital, then for its vital components, all of their dependencies (recursively down the call chain) should be either highly available or the component should be able to function in the absence of the dependencies. Examine availability for each service in the application independently and for the application as a whole.

In this article, as we examine the different deployment archetypes, we examine the availability applications can achieve with each archetype with the focus on overall availability as described in this section.

1.2 Types of Applications

A business relies on multiple types of applications, each having different availability and latency requirements.

- **Business-critical applications**—these represent the critical applications for a business. If these applications are unavailable, then the business is down. Highest availability and lowest latency are desired for these applications. These applications could be user-facing or not, and the classification of a business-critical application depends on each business.
- **Line-of-business applications**—these are applications that support running the business. While these applications do not serve customer-facing traffic, they are typically instrumental for supplying data for the business. They often have requirements to finish work by a particular time. **Continuous Integration and Continuous Deployment (CI/CD)** pipelines fall into this category, as well as data processing and analytics. High availability is desirable for these applications, but they can sustain short-lived outages without having an immediate business impact.
- **Internal applications**—these are applications that are for internal consumption for a business (e.g., recruiting, time-off tracking). Best effort availability is required. Employees want these to be always available, but if they are not, then the impact on the business is lower.

This article is primarily focused on business-critical applications, though the deployment archetypes can also apply to the other types of applications as well.

1.3 Data Durability, Availability, and Backup

For an application to be available, its data has to be available, and for the application deployments, we examine there are several concepts that are related to how the data is stored and managed. These include:

- **Data durability**—long-term data protection, where the stored data does not suffer from corruption and is not lost or compromised. To achieve this, the underlying data storage system often replicates the data and performs error-correcting checks and scrubbing of the data to prevent data decay.
- **Data availability**—access to data upon request. High data availability is achieved by placing and replicating the data across more than one failure domain, keeping the data durable, ensuring the service provides access to the data, and making sure the data is appropriately resource-provisioned to serve requests. The type of replication, asynchronous (eventually consistent) or synchronous (strongly consistent), along with data failover capabilities are important building blocks for achieving data availability.
- **Data backup**—point-in-time snapshots of data. Backups are important for protecting against application and human errors, and they can also be used as a means for disaster recovery. All applications should use backup services to protect against accidental loss or corruption of data due to application-level issues.

Similar to defining the desired level of availability, it is as important to define objectives for disaster recovery [64] for each application (e.g., **Recovery Point Objective (RPO)** and **Recovery Time Objective (RTO)**), and to choose deployment models and infrastructure that can achieve the desired RPO and RTO. RPO captures how old a copy (backup or replica) of the data is compared to the current state in production. It is important to understand the RPO, since this copy of the data would become the active state of production in case the current state fails, and any data changes more recent than the recovery point would be lost. If an application has more than one source of data, then independent recovery points of each one need to be reconciled. RTO captures how long it takes to restore an application during recovery to bring it back online and available in production, which includes access to the data needed for running the application. A highly available application wants both RPO and RTO to be as close to zero as possible.

In this article, we examine deployment archetypes and deployment models within each archetype that focus on the availability of data, while maintaining durability, and Google Cloud [25, 55], Microsoft Azure [23, 83], and AWS [22, 29] have each developed storage and database products that meet the requirements needed for each deployment archetype examined. In addition, data backup services should be used in conjunction with these deployment options.

1.4 Six Deployment Archetypes for Cloud Applications

Figure 1 shows all deployment archetypes and models discussed in this article. In addition, we examine Hybrid and Multi-cloud archetypes that are composed of the models shown in Figure 1.

(1) **Zonal**—All components of an application run within a single zone. A zone provides a set of clusters with the infrastructure needed to run services (compute, storage, networking, data, etc.) within that zone. Should a zone go down, what is running within the zone is either restarted in another zone from the last checkpointed state, or a failover occurs to a standby instance of the application in another zone.
(2) **Regional**—All components of an application are deployed and run out of one Cloud region. A region consists of 3 or more zones, where each zone is treated as a separate fault domain. High availability can be achieved by replicating the application across zones within

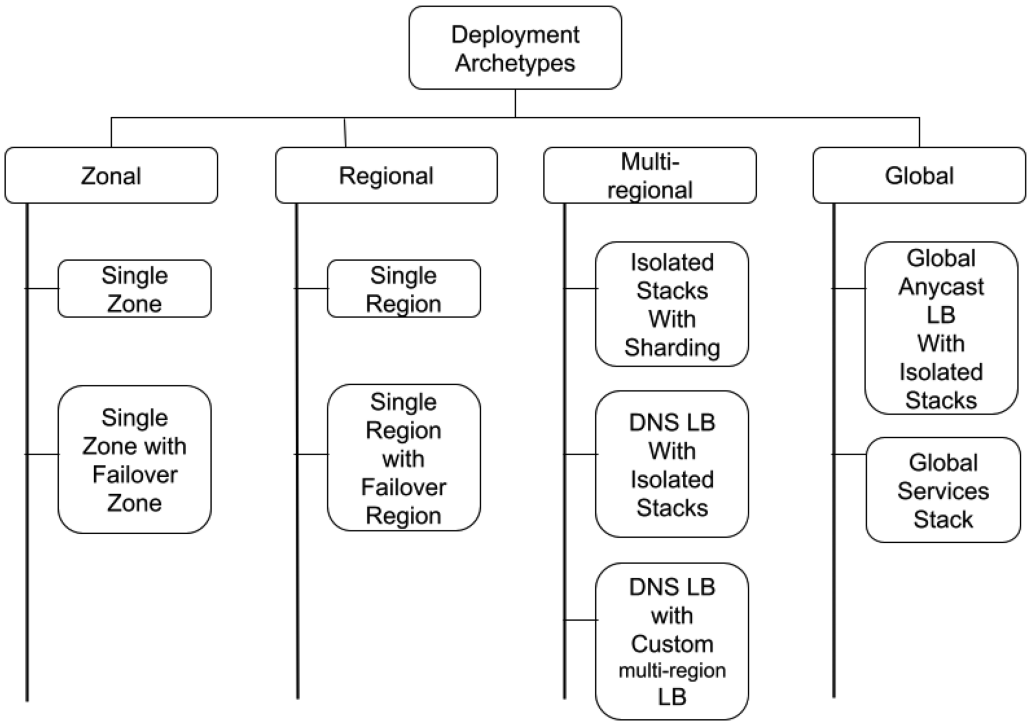

Fig. 1. Classification of deployment archetypes.

the region. These applications are typically designed to run with a data store that shares data and makes it accessible across that region. To serve application traffic, the requests are load-balanced across compute instances in multiple zones. To further increase availability and reliability, some applications may have a secondary standby region with an asynchronous copy of the data, where the application can failover to the secondary region in case the primary region is not available.

(3) **Multi-regional**—The application serving stack runs and is stitched together across multiple regions to achieve higher availability and low end-user latency through geographic distribution. In this deployment archetype, data is typically replicated and shared across regions. This archetype is commonly used for applications that want to achieve high availability, such as user-facing applications.

(4) **Global**—The application stack is spread and replicated across Cloud regions around the globe and data is available worldwide via global databases and storage. Applications consisting of a large number of services and microservices benefit from this deployment archetype. This is the five-nines deployment model used by retail, social media and other businesses requiring always-on availability, while running large services economically.

(5) **Hybrid**—Applications that have deployments combining on-premises and public cloud(s) are becoming increasingly common. On-premise software stacks will continue to evolve and be connected with the Cloud, to the point where on-premises will be considered to be another form of connected zone or region. Hybrid application availability and resilience is often achieved by (a) creating deployment archetypes that leverage failover between on-premise and Cloud, and (b) coordinating the execution of parts of the application that run in the Cloud versus run on-premises.

(6) **Multi-cloud**—Applications can potentially gain the highest availability by using two or more public Cloud platforms at the same time, to protect against one Cloud's unavailability. In each cloud, one of the deployment archetypes listed above is used, and then combined across clouds to create a multi-cloud deployment. This deployment archetype is in its infancy, but applications that require the highest availability are prime targets for multi-cloud deployments as this model evolves.

There are many reasons why one archetype will be used for an application over another. For applications that are required to have data reside in a particular region or jurisdiction, the choice of geographical distribution may be limited to an individual country or to a union of countries, and therefore the choice of deployment options will be limited. Other globally ubiquitous applications may have latency budgets, where, if latency is too high, then users may interpret it as an availability issue and abandon their requests.

Within each archetype there are multiple models that represent the deployment scenarios applications may use. We will now examine each of these deployment archetypes and models, and their tradeoffs in detail, and conclude with a summary comparing the tradeoffs.

2 ZONAL

In the Cloud, a zone represents a fault domain in which to deploy and run services and infrastructure. Running an application within a single zone typically means running the application within a compute cluster potentially spread across multiple racks near each other in the same datacenter. Should a zone go down, what is running within the zone is either restarted in another zone from the last checkpointed state, or a failover occurs to a standby instance of the application in another zone. We now go through these two types of zonal deployment models.

2.1 Single Zone

Running an application only within a single zone is not targeted toward high availability, since a zone is considered as a single failure domain from both software issues as well as other types of disasters (e.g., fire). Even so, applications that need supercomputer-like connectivity, as well as applications that do not need high availability, leverage single-zone deployments.

High Performance Computing [67] and Tensor Processing Unit Pods [56] are examples of Cloud applications that are deployed and run in a single zone. These applications typically require very low latency and high bandwidth usage, achievable within a single zone. They do not serve live traffic and can work with three-nines availability, and they can restart from the last checkpointed state. The data for these applications can be kept in a regional data store, with the primary or one of the copies of the data stored within the zone where the data is being read and processed. In addition, an advantage of keeping applications that have a lot of communication across VMs within the same zone is that Cloud providers typically have an additional charge for egress between VMs across zones.

Another important application type that works well with single zone deployments is developer testing workloads. This enables developers to continuously build and test their applications in the Cloud. It also may be suitable for use cases where downtime is acceptable or the application can be restarted elsewhere.

A single-zone application should be considered sufficient for these use cases, but not for most production applications.

2.2 Primary Zone with Failover Zone

As companies bring their on-premises applications to the cloud, a first step often taken is to choose a deployment model to run the application in the Cloud with minimal changes. Some of these may

be **commercial off the shelf (COTS)** applications that application owners acquired and may not be able to change. In addition, sometimes these applications come with per-instance licenses that can be prohibitively expensive to deploy for redundant extra copies. As a result, single-zone deployments continue to be a valid deployment option for these applications.

Single-zone applications still need as much redundancy and availability as possible. The deployment model typically used is to run the application in a primary zone, and to use a failover zone in the same region as a recovery zone. If the primary zone has issues, then the recovery zone is used to start the application up again. Many enterprise applications are built to run in a form of primary/failover configuration, also known as **Highly Available (HA)** topologies, and this is an established pattern used in enterprise and on-premises deployments over the years.

Let us look at the example of a single-license application running in the Cloud that wants to have failover support. Assume there are two VMs in two different zones (A and B), where one is the primary and the other is used as the failover. In this example the application owner has to pay for every instance running, so the application is only running in the primary zone and not the failover zone to control costs. In this case there are generally three options for how to connect to the application VM for license renewal:

- **Static IP address** (also referred to as floating IP address)—This static IP is used for the license renewal and can be either private RFC 1918 [105], RFC 6598 [127], or public IP. The static IP address initially points to the primary VM (for this example assume it is in zone A), which runs the single application. When zone A goes down, either manual or script-based reconfiguration occurs, and the application is started in zone B, with the same static IP address. In this case clients can continue to connect to the same IP address, whether they use DNS resolution or connect directly to the IP.
- **List of Static IP addresses**—List of IP addresses are used in a round-robin fashion in case connection is lost. The exact logic to pick one address from the list depends on application client-side behavior.
- **Dynamic Addresses with DNS**—If the IP for license renewal is not static, then DNS is used for resolution. In this case, DNS is configured to point to the primary VM in zone A. When zone A goes down, the DNS configuration is updated to point to the VM in zone B. The tradeoffs around DNS and how it relates to failover deployments are discussed in Section 4.2.

Now let us look at another example in Figure 2, which is a basic application deployed in a primary zone with a replica for failover purposes in a secondary zone. In this example, we have a **Load Balancer (LB)**, which denotes not just one instance, but a highly available replicated setup. The setup has a replicated compute workload named "Front-End," and the Cloud-managed database that holds the application data replicated across zones. Most databases will work in this configuration, and for this example, we assume it is a SQL database.

Let us consider zone A of region AA to be a primary zone, and zone B of region AA a failover zone. The primary instance of the SQL database is placed in zone A and all read and writes happen to this instance. In addition, the SQL database is configured with a standby in zone B, and the data is replicated from zone A to zone B by the cloud provider managed database. The Front-End in zone A and Front-End in zone B are configured identically with the same virtual IP address (10.3.2.1) to access the SQL database. This means the Front-End service does not need to change the IP address of the SQL instance when failover occurs. In addition, the load balancer is configured to have a primary set of compute instances (VMs or containers) in zone A and failover instances in B.

Now let us assume that zone A fails. Every second, the primary Front-End and SQL instance in zone A responds to a heartbeat signal from the monitoring system. If multiple heartbeats are not

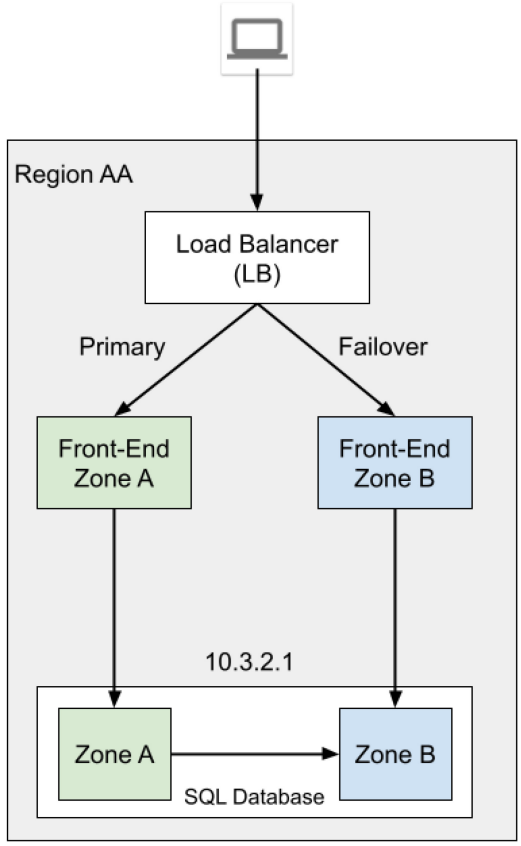

Fig. 2. Primary zone with failover zone deployment model.

detected by the monitoring system, then an alarm is sent and failover is initiated by the application owner or a script that has been automated. With failover initiated, the Front-End in zone B now serves user traffic, and the standby SQL instance in zone B is configured to now act as the primary SQL instance using the same virtual IP address (10.3.2.1) [50]. The load balancer will react to the failure in zone A by moving traffic to zone B, because it was configured to failover to the Front-End in the other zone based on health check status. Once the traffic is being served from zone B and the primary SQL instance is now in zone B, the availability is re-established for a single zone application on failover.

Health checking is an essential part of the failover process. As part of the health check status, the application's services need to decide if they are healthy or not, and this greatly depends on the service. Each instance of the service needs to determine its health based on error rates, exhaustion of resources, such as CPU and memory, or other custom signals, and declare itself unhealthy as part of a health checking response.

When zone A comes back, the traffic is not sent back to zone A by the load balancer unless the application owner decides to fail back. The deployment will now be in a steady state with zone B as primary and zone A as failover, until a failover is performed to make zone A the primary again. A best practice here is to reserve the capacity needed for failover in the failover zone and ready to go in case of a failure, and to routinely failover the application between zones to ensure failover works when it is needed. Note, there are additional scenarios to cover for an application using

failover (e.g., restarting up cross-zone data replication after failover, whether to allow failing over only part of the application stack, and more [58]), which we did not have time to go over in this article, and the application owner needs to make sure they are covered for this type of deployment model.

3 REGIONAL

In Cloud, a region is a specific geographical location in which to deploy and run application services and infrastructure that consists of multiple zones. A region consists of three or more zones, where each zone is treated as a separate fault domain. High availability can be achieved by replicating the application and its data across zones within the region.

We distinguish between zonal and regional archetypes with the following definition. The regional archetype has the application replicated across multiple zones within the region and actively serving traffic across the multiple zones at the same time. In comparison, the zonal archetype has an application serving traffic from a single zone and then failing over to another zone when there is an issue.

Single-region applications typically focus on users in one geography (e.g., country). This is used to (a) optimize for latency, where users are served from the same region they reside in, and/or (b) provide data sovereignty or location requirements, where user data is kept and served from a single country or region. To further increase availability and reliability, some applications may have a secondary standby region with an asynchronous copy of the data, where the application can failover to the secondary region in case the primary region is not available. We now describe these two deployment models (single region and a single primary region with failover).

3.1 Single Region

In the context of this article, running an application in a region means running an application spread across multiple zones within that region, where each zone is treated as an independent failure domain. A best practice here is to replicate the application across all of the zones within the region and keep the size of each deployment approximately the same across zones. This ensures the application always has capacity available in other zones when there is a zonal failure.

To demonstrate this, we will use a more complex application architecture shown in Figure 3. In this example, we have a service named "Front-End" that contains the interface the end-users interact through, a service "Back-End" that contains the business logic of the application, and a Cloud-managed database (e.g., SQL) that holds the application's data. In addition, there are load balancers in front of each service to load-balance requests across them. A request flow across the services in the diagram can be described as *user→Front-End→Back-End→SQL* and the response in the opposite order. In reality, applications consist of a large number of services and microservices, ranging from tens to hundreds in the same application.

As shown in the example, the single-region application should try to achieve higher availability by replicating data as well as compute workloads across multiple zones within the region. To achieve this for data, most Clouds support replicating SQL synchronously across the zones within a region, where writes and reads go to the primary zone for the SQL instance even though the data is replicated across zones for durability [52]. In addition, there can also be read replicas configured across all zones, where read requests are served from the closest zone.

When a request comes into Load Balancer 1, the request may be forwarded to any healthy replica of the Front-End service in any of the zones in the region, and then the request path is latency-optimized to keep the request flow within the same zone as the request traverses multiple services. For example, a request arriving at Front-End(A) will be sent to Back-End(A) to optimize latency by keeping the request within the same zone. Then the request to the SQL database may

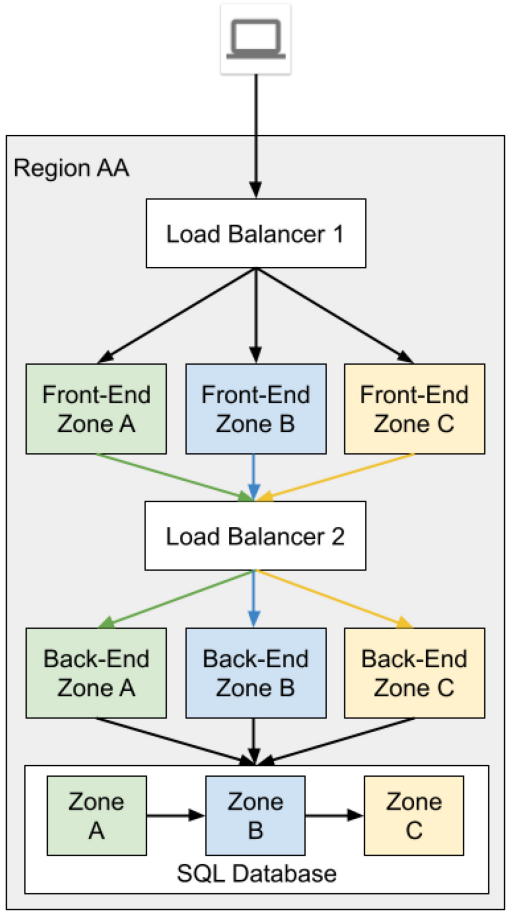

Fig. 3. Single region deployment model.

be across zones. The SQL(primary) is running in one of the zones, and communication with the SQL(primary) may happen across the zones, as writes are always sent to the primary.

Should one zone become unhealthy due to software failure, such as bad binary rollout for the Front-End or Back-End service, or due to infrastructure failure such as a power outage event, the requests destined to that zone will be steered to another zone. If the zone where the primary instance of SQL fails, then automatic failover is initiated [51] for the SQL(primary) and a standby replica now becomes the new primary. To achieve availability during failover, applications should retry idempotent requests or re-establish a new connection to the database [118].

As an example, assume that the SQL(primary) is in zone B and assume the Back-End microservice in zone B becomes unhealthy. For a request that comes into Front-End(B), the Load Balancer 2 will know that Back-End(B) is unhealthy and choose a different back-end to route the traffic to. In this case, the traffic flow coming into zone B is now *Front-End(B)→Back-End(C)→SQL(B)* or *Front-End(B)→Back-End(A)→SQL(B).* With this approach the load balancers can route the traffic around failures within a specific zone at a specific service layer.

For a single-region deployment, a question that comes up is how many zones to run the application across. The standard topology in Cloud is to have three zones for each region, where an

application is run across all three zones and it uses the Cloud provider managed regional storage and database services to manage the data. The reason for using three zones instead of two zones is because the loss of one zone means losing one-third of the serving capacity instead of half of the capacity. Not having enough serving capacity left after the loss of the zone can impact availability as it takes time for autoscaling to kick in. Another reason is to survive the unlikely but possible event of two simultaneous failures across two zones—one caused by the application and the other by the Cloud provider. This can potentially occur if (a) an issue in the application causes a single zone outage (e.g., as the application is incrementally updated one zone at a time and an outage is potentially found after the first zone is updated) and (b) the Cloud provider has an issue that results in one zone having an outage. With three zones, an application can still be available even if two zones have issues (one due to the application update and one due to the Cloud provider).

3.2 Primary Region with Failover Region

While single-region applications with multi-zone replication provide a highly available region, some business applications may have continuity requirements over large distances (e.g., having a primary and secondary separated by hundreds of miles). The desire for business continuity for a single-region application is fulfilled by maintaining a second region that is used for failover events. To satisfy compliance requirements, primary and standby regions may need to be located in the same country or union of countries. If there is no compliance requirement, then the failover region may be located anywhere where the latency increase on failover for serving response time is satisfactory.

In this deployment model as depicted in Figure 4, application data is synchronously replicated within a primary region, providing RPO = 0 for in-region failures. It is also asynchronously replicated to a standby region [49] that is sufficiently distant from the primary region. While this means a non-zero Recovery Point Objective and therefore potential data loss of recent updates on failover, the approach is used by Enterprise applications with availability or regulatory needs that require a replica in another region. Live traffic is always served from the primary region, and if the primary region becomes unhealthy either due to infrastructure or software component problems, then the standby region is used.

Some application owners prefer manual failover to the standby region. In this scenario, the DNS entry for the primary region is manually substituted with the VIP or IPs of the standby region when failover occurs. Otherwise, DNS Load Balancing (DNS LB) is used for automatic failover, which we describe in Section 4.2. If DNS is not used at all, then clients have the list of IPs for both the primary and standby region, and they are configured to use the current primary region.

For this model, the deployment is regional aside from the DNS LB. The DNS LB assigns traffic to the primary region, but if there is an issue with the primary region and a failover needs to occur, then DNS LB assigns traffic to the standby region. Health checking is done by the DNS LB sending probes to load balancers that represent a region (the Load Balancer 1 in Figure 3). If the health checks fail, then the application can failover to the standby for availability. Note, the DNS LB can also be used on an application owner's **Virtual Private Cloud (VPC)** for service-to-service communication as a service discovery mechanism.

For an application with more than one region to be operational, there needs to be an understanding of the full health of the service stack within a region. If there is a regional issue for just a single layer (e.g., Load Balancer 1, all Front-Ends, Load Balancer 2, all Back-Ends, or regional SQL), then the region would be unhealthy and a failover would need to occur to keep the service up and running. This means the application owner needs to build up an understanding of the health of

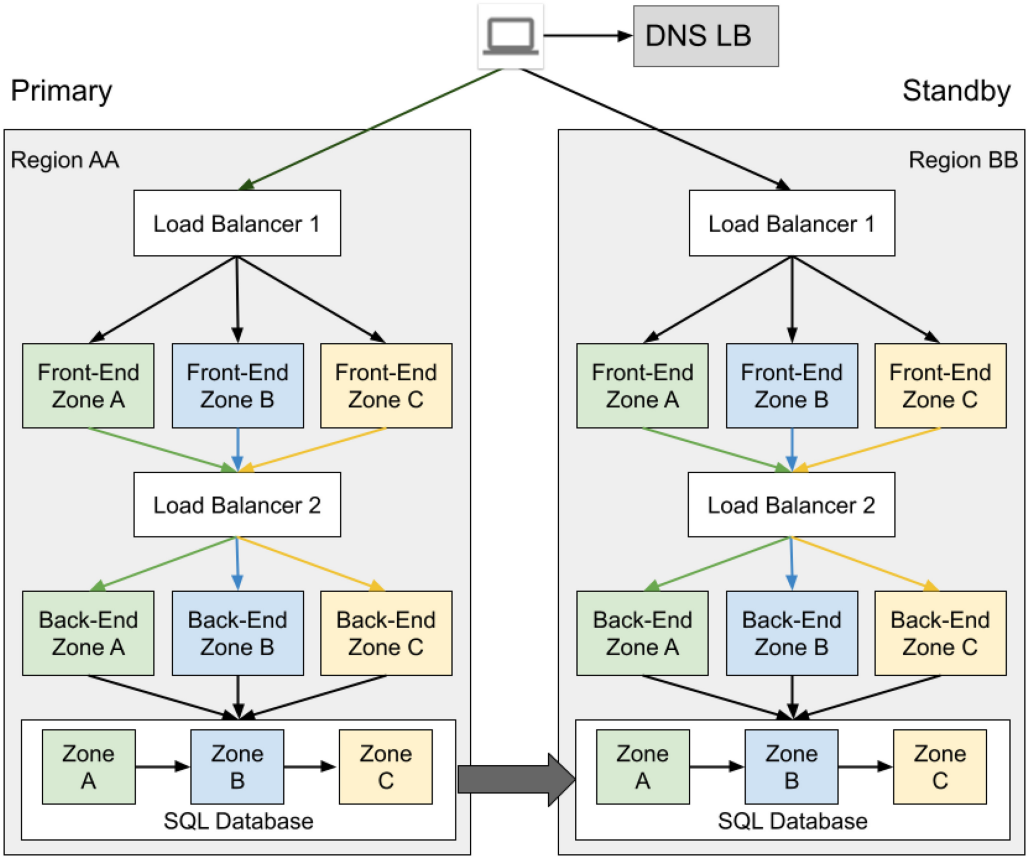

Fig. 4. Primary region with failover region deployment model.

all layers and be able to trigger failover if a given layer is having a regional issue. We will go into more details on this in Section 4.3.

When a failover is triggered, the primary for the database is switched to the new primary region (what was the standby database will be promoted to be the new primary [53]). If this is an unplanned failover, then recent updates to the database could be lost. For a planned failover, the failover can be coordinated to ensure all of the latest changes from the primary are made to the standby database before switching over.

With this deployment model, there is a need to ensure that the standby region functions well even though most of the time the standby region is idle. The best practice is to perform planned failovers on a timeline that makes sense for the business. In addition, health probers should be used to not only continuously check the health of the primary region but also the standby region. AWS provides Application Recovery Controller [11] to offer this deployment archetype as a solution to their application owners.

For some applications, this deployment can be simplified to have a primary region with a single zone and failover region with a single zone. This targets applications that are limited by number of licenses or limitations in architecture and is an improvement over the Primary Zone with Failover Zone deployment described in Section 2.2 for applications that need cross-regional business continuity.

4 MULTI-REGIONAL

The world has become more and more interconnected, with the users of the applications becoming more and more geographically dispersed. With that, the application deployment that was traditionally optimized for availability needs to evolve, because user-perceived latency has become a differentiating factor between competing applications. In addition, as users moved from desktop to mobile they have gotten accustomed to having access to their application and data anywhere with quick response times. This has pushed application deployment to being able to serve requests near to where users reside and means a single-region application is becoming less competitive from a user latency perspective.

Running an application across multiple regions gives lower end-user latency, drives higher availability, and meets some business continuity requirements by having the application and its data running and available in multiple regions separated by hundreds of miles. There are different deployment models for multi-regional archetypes, and we will go through two of them.

4.1 Fully Isolated Stacks with Data Sharding

If the application data is partitionable into separate databases, then one deployment option some applications have considered is to partition or shard the application and data across multiple regions into separate isolated stacks. In this approach, each stack would use the Primary Region with Failover Region approach described in Section 3.2, and a given user's data is confined to a single regional deployment based on the sharding.

This is shown in Figure 5, where users are routed to the region where their data resides, and their requests are fully processed within that region. A given client of the application knows the region that it is accessing and requests an application name resolution by regional hostname—our example application in region AA has a hostname aa.example.com and in region BB it has a hostname bb.example.com. We do not imply geographical proximity between the regions AA and BB. In practice two regions could be close to each other (e.g., 10–20 ms Round Trip Time) or far away across continents (e.g., 100–200 ms Round Trip Time).

For applications that can only be run within a single region, this gives potentially higher availability by sharding user data across multiple regions and better end-user latency for the users close to the regions hosting their data. Sharding the user data across regions can reduce the number of users impacted due to an application change that affects just a single region. In addition, this deployment model can be used to meet jurisdictional requirements for keeping data within a given region. This approach has the disadvantage of (a) having to deal with failover and loss of availability when a single region has issues, (b) having issues absorbing **distributed denial of service (DDoS)** attacks and large traffic spikes being directed at a specific region, since user requests go to a specific region and cannot be load-balanced across multiple regions, and (c) rendering user experience as a function of where their data is located (i.e., if a user's data resides in region AA, then it is always routed to region AA no matter where the user is in the world).

4.2 DNS Load Balancing

The next step in the evolution of multi-regional applications is the addition of a DNS Load Balancer to connect regional application stacks and load balance traffic across the regions. This is for applications that can run across multiple regions and be run with a data store that shares data and makes it accessible across those regions.

DNS-based Load Balancing is considered to be the standard way for clients to resolve the domain name of the service to get the IP address to use when accessing the service [17, 73]. A domain can be configured with one or more IP addresses, usually **Virtual IPs (VIPs)**, and these VIP addresses

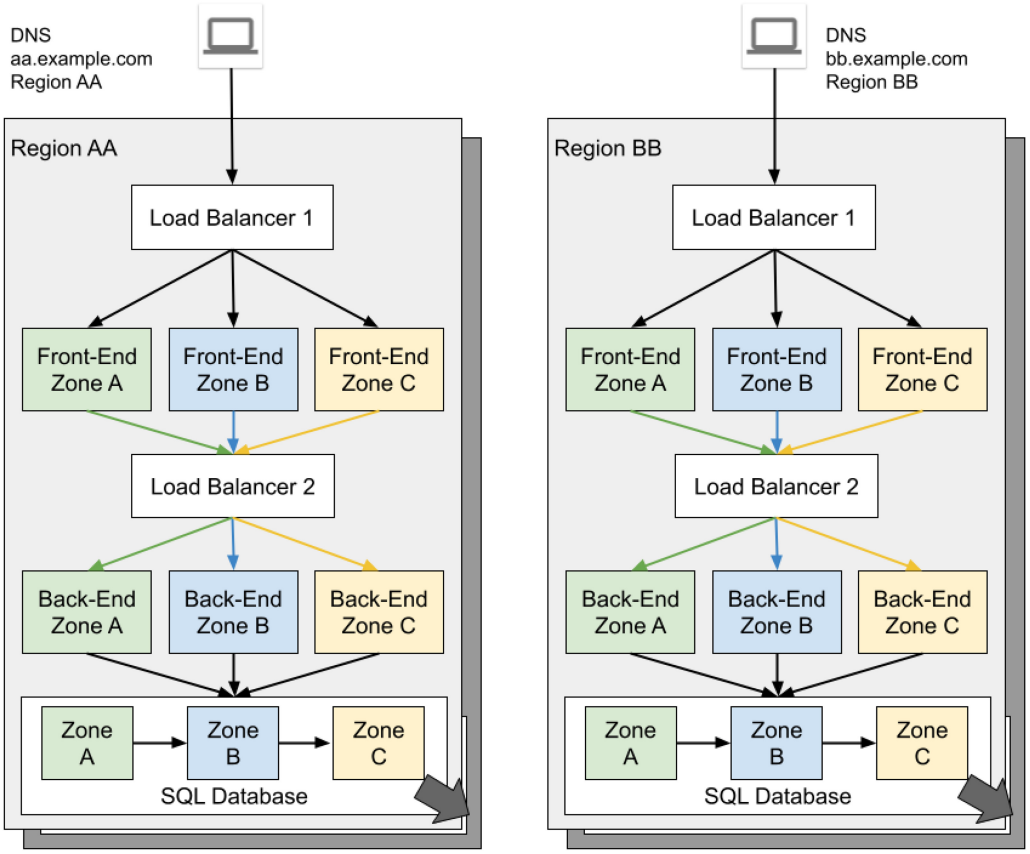

Fig. 5. Fully isolated stacks with data sharding in multi-regional deployment model.

target load balancers that front the application stacks. Then the DNS is configured by the application owner with one of the following routing policies that determines how the VIP addresses are given out for client requests:

- **Round Robin (RR)**—DNS requests are rotated and shared evenly across multiple IPs/VIPs that serve a domain.
- **Weighted Round Robin (WRR)**—DNS requests are assigned to different VIPs based on service owner-configured weight.
- **Geo-Mapping of Clients**—Another option is to create geo-mapping of clients to an edge region and DNS requests will be assigned to the closest IP/VIP. For this approach, DNS LB providers have their own knowledge of IP prefixes mapped to known geographies, as well as latency associated with reaching these geographies. This mapping is used to decide which region client IPs belong to, as well as which region destination VIPs belong to when processing a request.

When DNS is integrated with load balancing, the load balancer health-checks the application VIPs by sending requests to the VIP as if it were with real traffic. Typically, such health checking is done from several regions to make sure there is a reasonable level of confidence that the VIP is indeed healthy. After the health status is collected, unhealthy VIPs are removed from participating

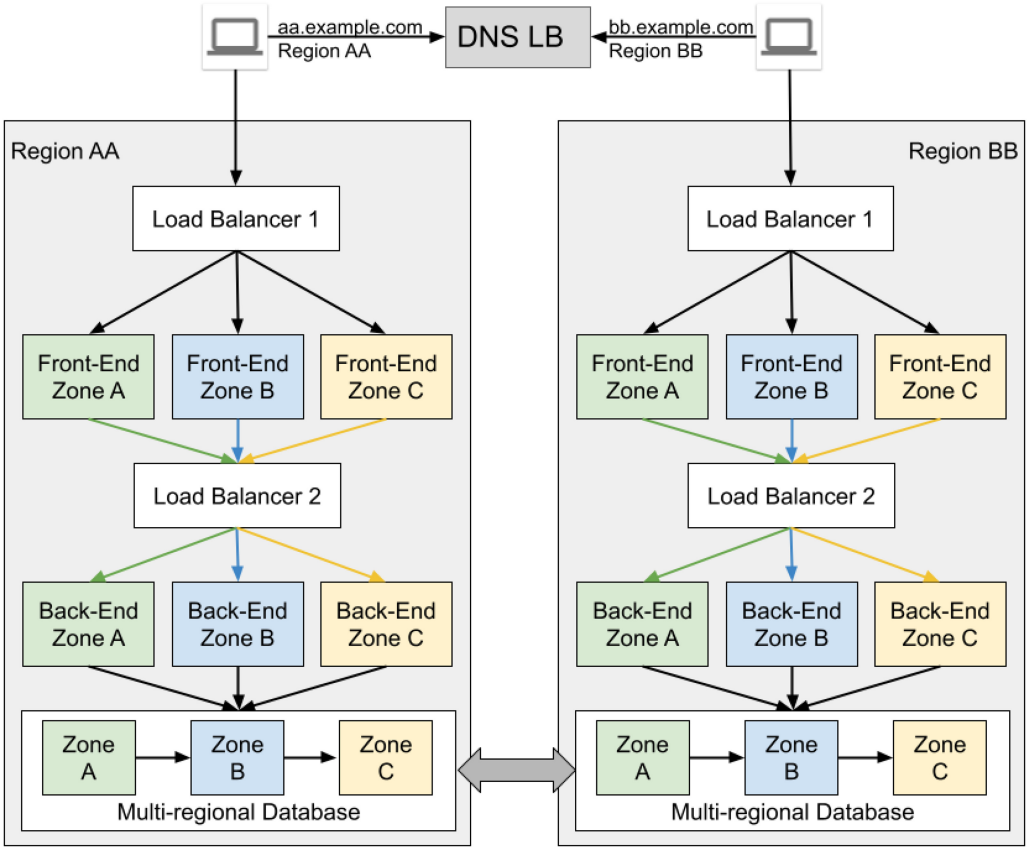

Fig. 6. DNS with multi-regional isolated stacks deployment model.

in the DNS routing assignments. In addition, DNS load balancing can be set up so that DNS requests can be redirected from a primary VIP to a failover VIP based on health checking.

With DNS, a client initiates a DNS request every time when the DNS' **time-to-live (TTL)** expires. This resolved address is the address the client will send the request to. The client will continue to use that resolved address until the TTL expires and continue to send requests to the assigned region and its services behind the domain. The TTL affects the failover time and availability of the application, and ultimately the user's experience.

We now walk through an example using Figure 6 with region AA and region BB. When a request to resolve example.com is answered by the DNS LB it is mapped to a VIP based on which one of the three routing approaches described earlier is used. If region AA becomes unhealthy and the DNS LB knows this, then it will update the DNS routing to not use AA. For clients already using AA, they will need to wait for their TTL to expire before they contact DNS again; then they will be routed to BB. The TTL will affect the availability of those clients during this resolution.

We now look at the tradeoff of using DNS LB with geo-mapping versus weighted round robin. Let us assume 100 DNS requests come from clients near the geographical area around region AA and 50 DNS requests come from clients near region BB.

- With DNS geo-mapping, region AA will receive all traffic generated by clients from region AA, and region BB will receive all traffic generated by clients from region BB. This produces

the best latency, but an unequal load on the two regions, and can lead to resource contention in some regions over others. In this case, it can be beneficial to use autoscaling to scale each region to the desired load.

- With DNS weighted round robin, that says to send 50% of requests to region AA and 50% to region BB, this will cause 75 DNS requests to go to region AA and BB each. This provides the best load distribution, but latency will suffer, because clients will be mapped across regions randomly. In addition, even for distributing load this is an approximation, since DNS does not understand the actual load received by the services in each region and is instead assuming the load will even out over time. But this may not hold true, since a given client may generate significantly more requests for example.com than other clients. Therefore, the number of requests sent to a region do not directly correlate with the number of DNS requests.

4.3 DNS Load Balancing with Isolated Stacks

With DNS load balancing, user data should no longer be contained to one region for primary operations, since requests can end up being sent to any of the regions participating in the DNS load balancing. For this deployment model, multi-regional or global storage and database solutions should be used, since the same data needs to be accessible at the same time across multiple regions. Asynchronous or synchronous cross-region replication is employed depending on capabilities of storage and databases as well as the need of the application [25].

For DNS load balancing to truly work and provide the highest availability, there needs to be an understanding of the end-to-end health of the service stack within each region, and it is **up to the application, or the monitoring infrastructure, to combine the end-to-end health of the application together and provide this to the DNS Load Balancer**. In this section, we will look at two strategies for providing full-stack health information to the DNS LB.

For multi-regional topology, the DNS Load Balancer usually is configured for health checking, combined with load balancing or routing options. For each regional VIP there are one or more VIPs dedicated as backup pointing to the other regions. If a health check of the regional VIP fails, then backup VIPs are used. If there is more than one backup VIP, then configured load balancing policies are used (e.g., RR, WRR or geo-mapping) over VIPs in the backup pool.

An alternative approach to having backup VIPs is to have DNS with the set of regional VIPs and weights used to guide the traffic across those VIPs. Health checks can be attached to load balancing policies and an unhealthy VIP is removed from the VIP pool, and in the WRR case the dynamic weights are recalculated to send traffic in proportion to the remaining VIPs using the configured weights. Let us say the intended configuration is VIP-AA:0.5, VIP-BB:0.25, VIP-CC:0.25, which means 50% of DNS responses contain VIP-AA, 25% of responses contain VIP-BB and 25% of responses VIP-CC. If VIP-AA becomes unhealthy, then DNS LB recalculates weights as VIP-BB:0.5 and VIP-CC:0.5, which means 50% of DNS responses contain VIP-BB and 50% responses contain VIP-CC.

Let us look again at Figure 5, where we are using DNS with multi-regional isolated stacks. If there is an issue for just a single layer (e.g., Load Balancer 1, all Front-Ends, Load Balancer 2, all Back-Ends, or regional SQL database) in a single region, then the region would be unhealthy and traffic should be directed to a different region. This means the monitoring infrastructure needs to build up an understanding of the health of all layers for each region and provide this to the DNS LB. There are two main approaches for achieving this:

- **Propagate layer failure up the stack to Load Balancer 1**—This approach assumes that the health of a layer is continuously propagated up the stack to each prior layer. In case of

a failure in the region all Front-Ends will eventually know that they are unhealthy, which tells Load Balancer 1 there are no Front-Ends to send the traffic to in the region. This will cause DNS LB health checks to Load Balancer 1 to fail and the region to stop being used. For this approach to work, every service in the application stack needs to implement such logic.

- **Aggregate health of all layers and report health to Load Balancer 1**—The stack of services within a region is declaratively defined and a region is considered healthy only when all services are healthy via aggregated status that is collected by an independent health observer service for the application. The observer aggregates the health status across all services in the stack and sends the combined health status to Load Balancer 1. If the aggregated status is unhealthy, then the DNS LB will fail a health check and take the region out of service.

For both of these approaches, Load Balancer 1 can only front one domain name at a time (e.g., example.com), and there are usually multiple services deployed behind one domain name (e.g., example.com/videoshare, example.com/news, example.com/shopping), separated by the path or other routing attributes. The issue is example.com has a set of VIPs shared across all of these services, so there is not a way to distinguish between the different services (paths) at the DNS load balancer, which would be required to understand the health of each service and act upon it. This means to understand and give the health status to Load Balancer 1, the health of all services behind that domain name need to be combined and will have the same shared fate if there is a failure. For example, if example.com/videoshare has an issue in a region and is down, then the aggregated health check will fail for example.com telling Load Balancer 1 to not use that region for any of the services under example.com, and all requests will be sent to other healthy regions. This aggregated health check creates a shared fate for services under a single domain name when using DNS load balancing for a region failure.

To summarize, the advantages of DNS load balancing with separated stacks are**:**

- Any region can serve a user request. This allows (a) a potential DDoS to be mitigated by a larger pool of resources across multiple regions and (b) traffic to be shifted to the remaining available regions if there is a failure with a specific region.
- The service owner has manual controls over per-region traffic distribution via the DNS LB configuration.
- Separate VIPs behind DNS can easily point to completely different deployments, different Clouds or on-premises data centers, providing mix-and-match options for increased failure isolation.

The disadvantages are:

- The need to implement propagation or aggregation of health across services and zones in regional stacks introduces considerable complexity.
- DNS TTL delays actuating failover to a healthy region and the time to failover is not deterministic. 75% of the TTLs are at 5 min and the remaining 25% are longer (can be hours to days) [28]. In addition, DNS TTLs can be ignored by some nameservers and some clients, which means the TTL can be much longer for clients than what the service provider specifies.
- DNS load balancing is based on DNS requests that do not represent volume of actual traffic and therefore cannot anticipate how much regional capacity will be needed to serve traffic represented by these DNS requests.

- Application capacity can be stranded within a region, especially for applications that are not autoscaled and applications that have diurnal traffic patterns. Capacity is not available to other regions for use, since the DNS LB may not have the means to take into account the region's capacity when directing traffic.

This deployment architecture has been a standard for user-facing applications for many years.

4.4 DNS LB with Custom Multi-regional Load Balancing

Large companies, such as Netflix [36, 64], who choose a multi-regional deployment but would like to build globally ubiquitous applications supplement the Cloud provider's multi-regional setup described above with their own multi-regional/global load balancing. In this approach, an application owner builds their own multi-regional load balancer using Cloud compute resources. In addition, this multi-regional load balancer must have higher availability than the target application. Such a load balancer can be placed to serve internet traffic and can also be used for service-to-service communication as well. A multi-regional load balancer needs to know the health of the stack across all of the regions and the capacity available in each region so as not to overload other regions with too much traffic. Building a multi-regional highly available Load Balancer requires specialized engineering skills and specialized processes to provide what is essentially described in the Global Services Stack in Section 5.3 (built and managed by Cloud providers).

This deployment model may be customized further as having only regional databases that have data for a subset of users. Such a model requires a custom load balancer to know which region each user mapped to and proxy the request cross-regions [86].

This deployment model depicted in Figure 7 increases the availability of a multi-regional deployment, but it puts the responsibility on the application owners for solving the hard problem of connecting regions in a way to optimize for lower latency, resource utilization and health. While sophisticated application owners build additional layers on top of Cloud providers, it is not expected that all application owners, who want global ubiquitous applications, will build their own traffic management, but rather use the global deployment archetypes described next.

5 GLOBAL

Consumer applications may evolve into global applications due to the global nature of the business, and/or the need to optimize for end-user latency and experience. This means businesses want to serve their cached and dynamic content as close to the users as possible no matter where users are located (both where they live as well as where they travel). In addition, as part of running a global business, application owners need to contend with global events that produce traffic spikes, as well as defend against massive DDoS attacks from around the globe. The difference between multi-regional and global deployments is that while multi-regional creates a deployment from regional building blocks where the application is aware of what region it is running in, a global deployment builds on a globally available fabric of network, data storage and databases that allow the application code to be location unaware.

There are several deployment models for global applications. Below, we discuss a few popular ones, but more variations are possible. These models are typically present in the Cloud and not on-premises as they require large global investments in network infrastructure and infrastructure systems that power these applications.

The global model also assumes the data is globally replicated and available in all regions where services run. This is because requests will be load balanced across regions, so multiple regions will need to have access to and read/write the same data. An example of such systems are Google's Spanner [25] and CockroachDB [113]. With global databases like Spanner and CockroachDB, the

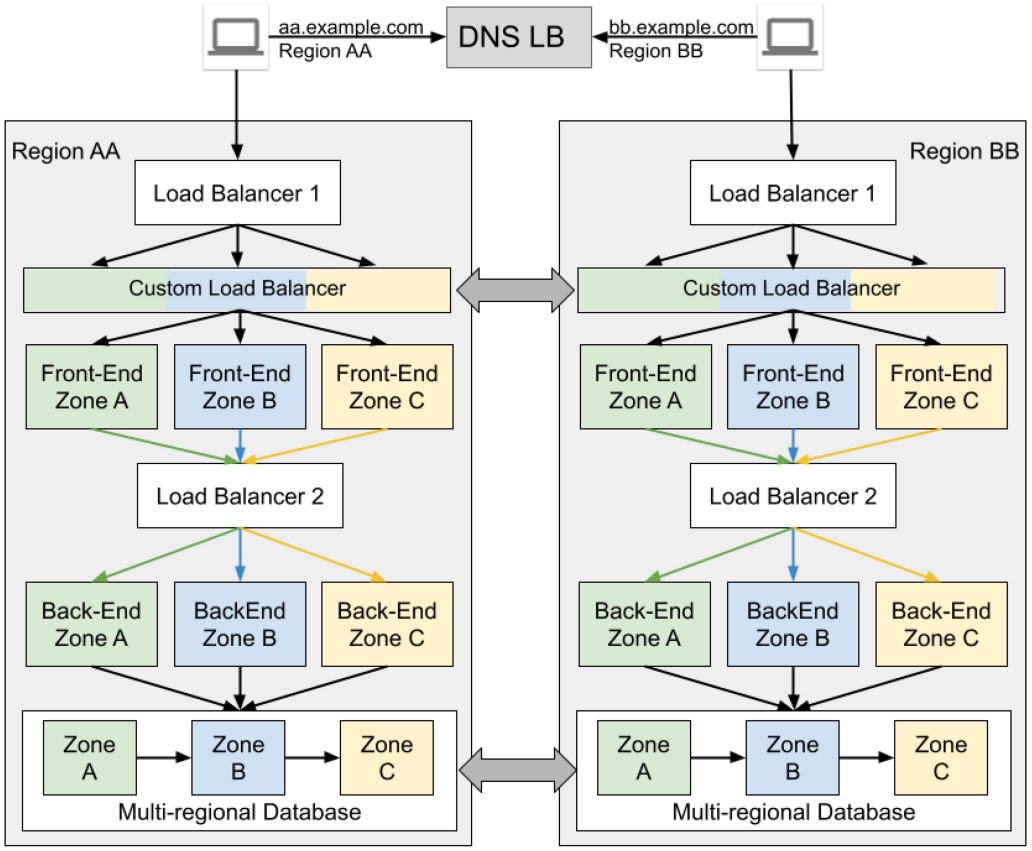

Fig. 7. DNS with custom multi-regional load balancing deployment model.

application can be accessed from any region around the world to perform SQL transactions, with strong consistency and five-nines availability. Some applications that do not require strong consistency due to their business requirements may achieve five nines with asynchronously replicated and eventually consistent data systems.

5.1 Global Anycast

The next step in the evolution of application traffic serving is using Global Anycast as an alternative to DNS Load Balancing to create a deployment capable of instantaneous failover of internet traffic should a multi-regional application become unavailable in one of its regions.

Global Anycast uses a single IP for the application to route traffic from a sender to the topologically closest destination IP address for a group of potential receivers, which for Cloud providers means edge load balancers. Google announces IPs via the **Border Gateway Protocol (BGP)** from multiple points across its global network [104]. A deployment model that uses Global Anycast eliminates the need for DNS Load balancing with multiple domain VIPs, since the application only needs a single Global Anycast VIP, but it still needs to address the following two issues:

- Too many close-by users can overwhelm an edge site where the traffic is being sent to.
- BGP route calculation might reset connections because of "route flap" [124], which happens when there is a pattern of repeated route withdrawal and re-announcement. This can

happen because of frequent problems on a particular link or misconfiguration or mismanagement of routers.

To address these issues, Google developed stabilized anycast using the Maglev [32] network load balancer. This solved the problem of route flap by redirecting a flapped request to a peer Maglev that is responsible for the connection [21]. Maglevs are deployed in each edge location and if an edge location goes down, BGP will reroute to Maglevs in the next closest edge location. In addition, Google uses global load balancing for the Global Anycast LB itself to distribute traffic to edge sites to ensure an edge site is not overloaded. The algorithm considers incoming **Requests Per Second (RPS)** load and the capacity of edge proxies and assigns new connections to each edge to ensure the best utilization of edge proxies, while at the same time optimizing for user latency.

Google Cloud uses stabilized Anycast technology as a front to the Cloud HTTP(S) Load Balancer [57]. This load balancer provides application owners with the ability to have a single global VIP that represents their global application deployed anywhere in the world.

5.2 Global Anycast LB with Isolated Regional Stacks

In this deployment model, a Global anycast LB ingests traffic and then sends traffic to the regional LB in the region containing the application owner's compute resources, depending on geo-mapping, health and weights. Other Cloud providers have also developed products using Anycast – Azure Front Door [84] and AWS Global Accelerator [10].

This approach, depicted in Figure 8, uses a regional stack like the prior one, and replaces the DNS LB with a Global Anycast LB. This means the application still has to build up an understanding of the health of all layers within a given region and provide this to the Global Anycast LB. This is the same as described in the DNS LB Section 4.3, where the Global Anycast LB only understands the health being propagated up to it via the Load Balancer 1 layer from each region.

The advantages of a Global Anycast LB with Isolated Regional Stacks are:

- Using DNS with Global Anycast means DNS always resolves a domain to the same single VIP. Lack of reliance on specific DNS resolution means that load balancing between the regions will be done instantaneously by Maglevs and not subject to DNS TTL.
- A single global VIP simplifies an application owner's setup as there is no need to use DNS LB and manage multiple IPs. Using Global Anycast with DNS will resolve a domain to the single global VIP anywhere in the world and traffic will reach the closest healthy destination with available capacity.

The disadvantages of this approach are:

- Since this is still a regional-based stack, the application still needs to implement propagation or aggregation of health across services in regional stacks.
- The application owner does not have control over traffic distribution from clients to the edge location where the LB service resides. In comparison, with DNS LB the application owner could redirect traffic administratively if needed. For example, if an application itself has a partial issue in a given region, but the health checks are passing (i.e., gray failure), then with DNS it is easy to tell it to not send any traffic to the VIP having the issue for that region.

This deployment architecture is for global user-facing applications.

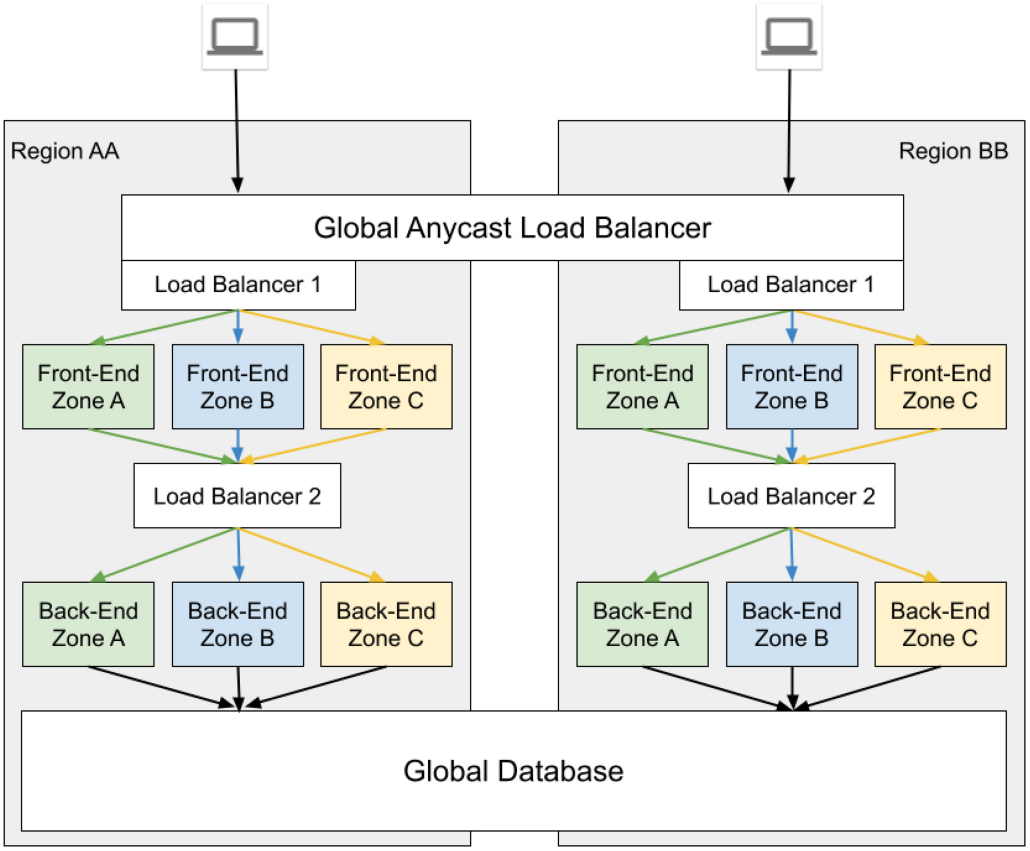

Fig. 8. Global Anycast with regional isolated stacks and global database deployment model.

5.3 Global Services Stack

In this deployment, services are global. The data is also global and synchronously or asynchronously replicated and available in all regions where services run (e.g., using Google Spanner [25]). In addition, having a global network is important to making a Global Services Stack possible.

This application deployment is targeted toward applications with a worldwide audience, that receive traffic spikes, serve a large volume of traffic, must run economically and need five nines availability. These are typically large-scale global applications deployed over three or more regions and a large number of microservices (a hundred or more) all communicating with each other, with global load balancers between them. At a high level, this approach puts a Global Load Balancer in front of each microservice. Given the large number of microservices, there is typically distributed ownership with each team owning one or a few microservices [69]. Having each microservice (or set of them) having their own Global LB provides the ability to manage traffic and reason about each microservice independently, which fits well with the distributed ownership of the microservices that make up the overall application.

This global load balancer provides the global service-to-service communication for each microservice in the stack. This functionality is provided by either middle proxies or by using a global service mesh with sidecar proxies or even a proxyless gRPC service mesh [103]. Using a managed service mesh has an advantage in that it aids in managing tens to hundreds of microservices with

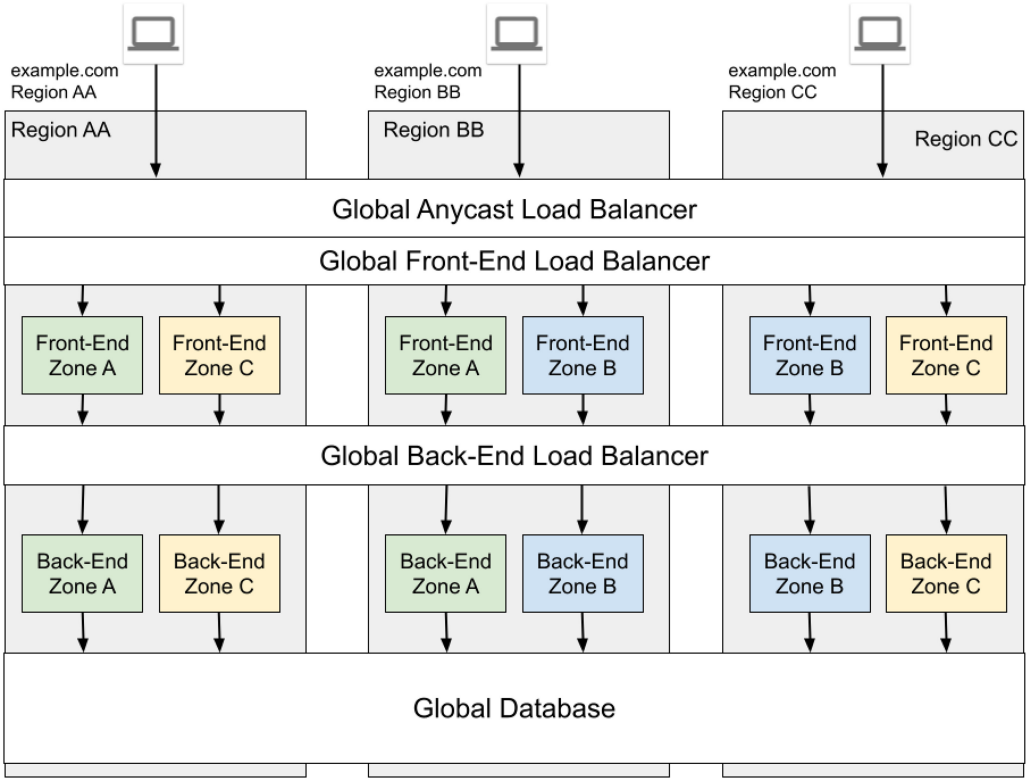

Fig. 9. Global services stack deployment model.

integrated load balancing, health checking and autoscaling [39] without needing to take care of proxy resilience and availability.

With a service mesh, the global service-to-service communication supports HTTP(S)/gRPC and TCP/UDP traffic. Global service communication (called east-west or service-to-service load balancing and routing) optimizes traffic globally for each microservice in the stack, so communication at each source-to-destination pair of services exhibits the lowest latency. This makes sure the destination service is not overloaded and redirects traffic to the closest available region in case of failure or administrative maintenance. The placement of services in a Global Services Stack is economical as only one zone is needed in each region where the application wants to run.

Let us consider the example.com application in a Global Services Stack deployment as depicted in Figure 9. A request to example.com is resolved to the Global Anycast VIP and the request is subsequently sent to the Global Front-End Load Balancer to decide where to serve this request. In addition, each path in example.com (assume example.com/video and example.com/photo in this example) is registered with the Global Front-End Load Balancer so they can be routed to different services. Depending on the path chosen (example.com/video or example.com/photo) as well as other routing options, the request is mapped to a different service, in this case a video service or photo service. This allows each path to have separate health checking, which was not achievable when using DNS LB described in Section 4.3. Each service may be deployed independently as a microservice with its own Front-End service or combined into a shared Front-End service. Health checking for each path is done independently and a shared Front-End service may reply as healthy for video requests and unhealthy for photo requests allowing global load balancing to direct traffic

separately for each service. Depending on the health, geographical proximity, and capacity of the Front-End service, requests will be forwarded to the most appropriate instances.

The same approach happens at the Global Back-End Load Balancer when the Front-End service wants to send requests to the Back-End service. For example, assume the Back-End service in region AA is unhealthy. Traffic from Front-End in region AA will be rerouted to the Back-End in region BB or region CC by the Global Back-End Load Balancer, without the need to propagate health up the stack (as described in the DNS Load Balancing with Isolated Stacks deployment model in Section 4.3). With the Back-End in region AA unhealthy, the Global Back-End Load Balancer will assess the health, geographic proximity, and capacity of each individual microservice/service in all zones of regions BB and CC, and requests will flow to the most appropriate Back-End instances. From there on, the closest, and most likely local, database will be used to read or write the data.

The following are benefits of the Global Stack Services deployment approach:

- Allows every layer and microservice in the stack to understand the health of and load to that layer to load balance for each layer. In comparison, Multi-region deployments (Section 4), DNS LB with Isolated Stacks (Section 4.3), and Global Anycast LB with Isolated Stacks (Section 5.2) approaches require applications to stitch together stacks and health signals to deal with failures. In addition, all of these approaches lack sufficient understanding of service capacity for load balancing down the stack.
- As shown in the example.com example, a whole region does not become unhealthy for an application if one microservice in the stack becomes unhealthy in that region, compared to the DNS LB approaches.
- Traffic spikes are automatically in real time load-balanced around the world (regions) as needed to keep the application available. Traffic spikes could be caused by users (e.g., triggered by inorganic events) or by services (e.g., an accidentally created DoS attack from lack of exponential backoff).
- This deployment is the cheapest option to run a worldwide application with the highest availability, since capacity can always be efficiently used. In Google Cloud, we integrate global load balancing with autoscaling to allow an application to scale up and down as the load changes [57].

The following is the disadvantage with this approach:

- Global service-to-service communication means the potential for a global outage if the Global LB has an issue. This requires the utmost care in operation of such services. Here are some of the practices used by Cloud providers to manage risk:
  - Rollouts and configuration changes are done incrementally and applied zone by zone to minimize blast radius for a bad software update or configuration change. Even within a zone, techniques such as blue-green deployment or rolling updates are employed. In addition, a new version of a service workload is canaried for some time in the first zone to understand the impact of a new binary on the availability and latency of the application before resuming rollout to other zones and regions.
  - To mitigate global outages, cloud providers use techniques such as sharding to reduce the blast radius across tenants (customers) in multi-tenant systems, as well as provide multiple paths to a service either via dual VIPs, or DNS failover from a global VIP to regional VIPs.

When using a global stack there is no requirement for services to be global, and they can be downscoped to regional when it makes sense. It is up to the particular application architecture, and a mix of regional and global services in one application deployment is possible.

6 HYBRID

Hybrid applications that consist of running across a combination of on-premises and cloud are becoming increasingly common. While on-premises applications have grown organically, they still fit into the archetypes and models described in this article. On-premises software stacks will continue to evolve and be connected with the Cloud [119], to the point where on-premises can be considered to be another form of a zone or region of a public cloud, thus creating InterCloud or Interconnected Cloud as described in Reference [19]. Hybrid application availability and resilience can then be improved by creating deployment models that leverage failover between on-premises and Cloud and coordinating the execution of the parts of the application and its services [99]. Deployment models discussed in Sections 2–6 are used as a building blocks for composed hybrid applications, and the following are a few examples:

- Cloud frontend serving for on-premises applications.
  - Use case: Better network latency and security for applications.
  - In this scenario, data resides on-premises. Incoming traffic (typically from the internet) is ingested into the Cloud using one of the front-end serving architectures for different archetypes. As traffic is ingested, Cloud-managed services such as content distribution networks, DDoS protection, or access policies are applied and enforced. Then the traffic is sent to the on-premises deployment for further processing. This example could be achieved by using multi-regional deployment archetype and all its models described in Section 4, and global deployment archetypes and all its models described in Section 5.
- Cloud disaster recovery [8] for on-premises applications.
  - Use case: Backup of important data for redundancy.
  - Some application owners prefer to use on-premises deployment and sync their data to the Cloud for recovery in case something happens with their on-premises data. In this scenario, the data placement archetype (Zonal in Section 2, Regional in Section 3, Multi-regional in Section 4, or Global in Section 5) can be employed, depending on the Cloud provider's availability and the application owner's needs.
- Replicated application between on-premises and Cloud [3, 85].
  - Use case: During migration from on-premises to Cloud or when traffic demand can grow inorganically.
  - Incoming traffic (typically from the internet) is ingested into the cloud using one of the front-end serving architectures for different archetypes. The application is replicated between cloud region(s) and on-premises datacenter(s). Data is also replicated between cloud and on-premises as if it were cross-region replication between cloud regions. Traffic management capabilities allow for scenarios such as:
    - Traffic can be directed to the closest regional application stack, independently of whether the stacks are in the Cloud or on-premises.
    - Allow the balancing of application capacity between the Cloud and on-premises parts of the application.
    - Burst into Cloud when on-premises application capacity is exhausted [100].
    - Failover to boost overall application availability [30] when on-premises application is unhealthy [1].
  - Data replication is also being supported between Cloud and on-premises. For example, a database can be replicated across the Cloud and on-premises, as with Cloud SQL [54].
- Using first-party Cloud services with an on-premises application [5].
  - Use case: The application needs specialized services that are easily available in the Cloud and hard to get on-premises.

- In this scenario, the application is mixed, with some services residing on-premises [7, 87, 113] and some on public Cloud. Services residing in the Cloud are potentially managed first-party services, such as Google Cloud BigQuery and Machine Learning.

If the on-premises application can be treated as running in a separate region, then the multi-regional and global archetypes described, respectively, in Sections 4 and 5, may be applicable.

Besides the enterprise on-premises, hybrid deployments may apply to edge or fog computing environments [15, 16, 111] where parts of applications may reside on the edge of the network to further improve latency or extend application deployment to mobile devices [112] and Internet of Things [26].

7 MULTI-CLOUD

To improve availability even further, the application may be deployed across multiple public clouds, but there are many areas of investment that need to be made to make this an accessible option for businesses.

A multi-cloud application has the potential for the highest availability, as it removes reliance on a single Cloud provider to be always up and provides cross-cloud load balancing and autoscaling for traffic spikes. In addition, using multiple clouds provides more vendor optionality and choices for the application owner. Also, a multi-cloud application may have lower latencies as it gives more options to distribute the load in a given geography across cloud providers.

The current state of this deployment is in its infancy and some of its challenges [20] include:

- **APIs**—a multi-cloud application needs portable APIs [96] to effectively run its application across clouds [19, 119]. There are many areas where the APIs are similar enough across clouds (e.g., Object/Blob storage), where others are fairly different. Increasingly there are a number of open source API standards [31, 96] that are becoming common across clouds (e.g., Kubernetes APIs [71], Envoy Proxy APIs [34], and Istio [63]), which will aid multi-cloud deployments. Another approach some applications have taken is to leverage the base semantics that are common across the clouds and abstract the cloud-specific differences away into a client library layer the application uses. The library layer can translate the requests to the appropriate APIs depending on the cloud being used.
- **Operational Complexity**—to run multi-cloud applications, multi-cloud operational tools and orchestrators for binary configuration, upgrade, rollout, monitoring and debugging must be developed [6]. The complexity of building such tools is based on the challenges of not having common cross-cloud APIs as well as the operational differences in platform. To help address this, application owners can use a cross-cloud application management platform such as Anthos [75] to provide consistent development and operational experiences across clouds. In addition, application owners can use cross-cloud configuration tools such as Terraform to simplify configuration to provision resources in multiple clouds [27].
- **Load Balancing**—there are multiple ways of load balancing between public and private clouds [3, 85, 92]. The majority of them are only in the beginning of a multi-cloud journey, such as:
  (i) DNS LB—similar to the DNS LB with Isolated Stacks deployment described in Section 4.3, a multi-cloud application could use DNS LB to route traffic across clouds.
  (ii) Client-driven Traffic Routing—each client receives IPs/VIPs for each of the clouds and distributes traffic based on either routing or load balancing configuration. Configuration is delivered via a control plane. An example of this approach is Envoy Mobile [35].
  (iii) Global Anycast LB with Isolated Stacks Across Clouds—in this deployment, one cloud has the primary Global LB [3] that distributes traffic between primary and secondary

clouds, based on algorithms chosen by the application owner. This approach assumes network connectivity and that the global load balancer has the endpoints from other clouds registered with it. The distribution algorithms may be (a) geographically optimized (e.g., send to the closest cloud), (b) burst to the other cloud due to lack of capacity [3], which is typically from private to public cloud, and (c) failover to the other cloud based on the health of the service behind the Global LB.

(iv) Global Services Stack—this is similar to a single-cloud Global Services Stack deployment described in Section 5.3, where each layer either operates across clouds or the serving stack is distributed across clouds. In this case, a given service is replicated across multiple clouds and traffic is distributed based on geography or other constraints. This deployment model is hard to achieve today, and for this approach, having common cross-cloud APIs is important.

- **Networking and Security**—A multi-cloud application can have its services in different clouds connected to one another via public (Internet) or private (interconnects and VPN tunnels) connections. Connectivity is subject to QoS and bandwidth management, and network level encryption via IPSEC or SSL/TLS. For private connectivity, underlying pipes are shared between applications, and cloud providers must manage resources shared between different services and their customers [20]. Multiple networking topologies [48] are supported for public and private connectivity based on the needs of the application:

  (i) Flat Network—In this model a flat network spans multiple clouds and all services can communicate with one another. Firewalls are enforced on both sides. In addition, zero trust or end-to-end application security via mTLS authentication and authorization between services in different clouds is used. This requires either homogeneity of compute environments, such as Kubernetes and hence the same credentials, or have federated identities for heterogeneous compute environments.

  (ii) Gateway Model (ingress and egress)—In this model, there are gateways on both sides. Depending on the direction of the traffic, it is either ingress or egress gateways. Ingress gateway protects access to the services by providing a security perimeter to enforce access to the limited number of services by allowed or denied roles or IP addresses. The access is enforced for "workforce" (people) and "workloads" (services). Egress gateway protects source workloads and data, from events such as exfiltration attempts, and also enforces policies for destination services in other clouds. Zero trust model is also possible here by Gateway terminating mTLS from the client and creating another mTLS connection to the server.

  (iii) Handover Model—In this model, there is a shared environment between two clouds and data from one cloud is uploaded to this shared environment and picked up by the workload from another cloud (e.g., using Pub/Sub or worker queues). There is no private network connectivity between parts of the application. This model is not used by serving applications, but rather by processing applications and data analytics.
- **Data Management**—Multi-cloud database [90] and storage solutions that replicate across clouds [68] would need to be used. Approaches can be considered where the primary for the database is in one cloud and read replicas in another cloud [91]. If the primary cloud has issues, then first try to failover within that cloud, otherwise failover to another cloud.
- **Cost**—A multi-cloud deployment may entail higher costs [18, 19], with a tradeoff for higher availability and cross-cloud optionality [109]. A few examples are:

  (i) Data duplicated across clouds will typically be stored also redundantly within each cloud, which costs more than just keeping the data replicated within a single cloud.

Table 1. Zonal Deployment Archetype Comparison of Risks

Deployment Model	Scope of Failure	What Failed	Application Down	Mitigation	Instantaneous Recovery
Single Zone	zone	Zonal infra or managed services	Yes	Wait until zone is back or rebuild app in new zone	No
Single Zone with Failover (Figure 2)	zone	Zonal infra or managed services	Yes (during failover) No (after failover)	Continue operation via failover to standby zone	No
	region	Regional infra or managed services	Yes	Wait until region is back or rebuild app in new region	No

(ii) The egress costs for replicating and duplicating data across clouds may be higher than just storing the data replicated and durable within a single cloud.
(iii) There can be resource inefficiency unless the cross-cloud load balancer fully understands the capacity utilization in each cloud. This includes understanding the actual capacity and load to the microservice layers in each cloud.

The multi-cloud approach is promising [20] for increasing availability and optionality, but there are many areas of investment that need to be made to make this an accessible option for businesses.

8 COMPARING AND SELECTING DEPLOYMENT MODELS

Throughout this article, we examined how to span zones, regions, and geographical reach to achieve availability for serving applications. To achieve higher availability, an application deployment has copies of its serving stack, using either (a) an additional failover copy or (b) additional active serving stacks to load balance across. As the application increases its geographical spread from zonal, to regional, to multi-regional, to global, a higher number of nines of reliability can be achieved. The deployments that achieve the highest availability are multi-regional, global, and multi-cloud.

Multi-regional and global deployment archetypes need databases, object stores, and data caches that provide access to a shared state. Depending on the type of datastore needed, there is a spectrum of multi-regional and global data store deployments, using either synchronously or asynchronously replicated data, to choose from. The type of datastore used depends on the application's access patterns and type of data that it is serving, whether it serves read-only cached data (e.g., songs, images, or directions), can function well with eventual consistency (e.g., putting items in an online shopping cart), or requires strong consistency. In addition, the application owner needs to determine how much redundancy and freshness of data is needed for business continuity.

Tables 1, 2, 3, 4, and 5 provide a comparison between the various deployment archetypes and models. Some deployment archetypes (a table for each archetype) have more than one deployment model (first column). We compare each archetype and deployment model pair based on characteristics that are used to judge the risks of application failures. The second column is the potential scope of failure, which we separate into four types: zone, region, global, and cloud (for multi-cloud deployment). The third column describes, at an abstract level, the type of failure that could cause the corresponding scope of failure to occur.

Therefore, each row in the table describes the application impact and recovery for a given deployment archetype and model assuming the specified scope and type of failure has occurred. Then, for application impact and recovery, the last three columns describe: the impact to the

Table 2. Regional Deployment Archetype Comparison of Risks

Deployment Model	Scope of Failure	What Failed	Application Down	Mitigation	Instantaneous Recovery
Single Region (Figure 3)	zone	Zonal infra or managed services	No	Continue operation from remaining zones in region	Yes
	region	Regional infra or managed services	Yes	Wait until region is back or rebuild app in new region	No
Single Region with Failover (Figure 4)	zone	Zonal infra or managed services	No	Continue operation from remaining zones in the primary region	Yes
	region	Regional infra or managed services	Yes (during failover) No (after failover)	Continue operation via failover to standby region	No
	global	DNS LB (if DNS is used for failover)	Yes for new or expired TTL clients	Wait until DNS LB is back	No

application's serving stack's availability (fourth column), the type of mitigation needed to restore availability because of the failure (fifth column), and whether the mitigation provides instantaneous recovery or not (sixth column). For the Application Down column, "Yes" means the application is down with an outage until the specified Mitigation happens, and "No" means the application continues to serve traffic (typically due to load balancing across the scope of failure or after failover completes).

A regional application may experience zonal and regional failures, whereas a multi-regional application may experience zonal, regional, and global failures. In addition, the failures could be cascading from smaller scopes to a larger one. For example, cascading zonal failures limited within a region would be considered a regional failure and spreading beyond the region would be considered a multi-regional or global failure.

When we examine "What Failed," it could be the cloud service provider, or could be a service in the application, which may include dependencies on third party services. It is important to capture and understand all of the dependencies, their criticality to availability and their risks when considering what can fail and how.

8.1 Failover-to-Standby Versus Load Balancing

When comparing the failover-to-standby deployment models to those that use load balancing across failure domains, the failover-to-standby model has the following concerns an application owner needs to address:

- Failover-to-standby models do not provide instantaneous recovery due to the time to failover delays and therefore display unavailability even if for a short period of time. The startup time of the standby stack and the ability for it to go from zero load to full throttle is important, as traffic does not trickle in slowly on failover, but rather moves over all at once. For example, when the standby stack takes on primary traffic, its service caches may be cold and there is going to be observed latency impact, as well as additional compute capacity may be needed to process requests as the caches warm up. These failover startup delays can impact availability and can expose startup issues by putting high load on

Table 3. Multi-regional Deployment Archetype Comparison of Risks

Deployment Model	Scope of Failure	What Failed	Application Down	Mitigation	Instantaneous Recovery
Multi-regional Fully Isolated with Data Sharding (No Failover to Standby) (Figure 5)	zone	Zonal infra or managed services	No	Continue operation from remaining zones in the region	Yes
	region	Regional infra or managed services for a single region	Yes for users in region affected, since users are sharded across regions	Wait until shard is back	No
	global	DNS	Yes for new or expired TTL clients	Wait until DNS is back	No
Multi-regional Fully Isolated with Data Sharding with Failover to Standby (Figure 5)	region	Regional infra or managed services	Yes (during failover) No (after failover)	Continue operation via failover to standby for region affected	No
Multi-regional with DNS LB (Figure 6)	zone	Zonal infra or managed services	No	Continue operation from remaining healthy zones across the regions	Yes
	region	Regional infra or managed services	No	Continue operation from remaining regions	No (DNS TTL)
	region	Single app service down	No	Load balance traffic away from the affected regional stack to another region	No (DNS TTL)
	global	DNS LB	Yes for new or expired TTL clients	Wait until DNS LB is back	No (DNS TTL)
		Multi-regional Database	Yes	Wait until database is back	No
Multi-regional with DNS LB & Custom Multi-regional LB (Figure 7)	global	DNS LB	Yes for new or expired TTL clients	Wait until DNS LB is back	No (DNS TTL)
		Custom LB	Yes	Mitigate custom LB failures	No
		Multi-regional Database	Yes	Wait until database is back	No

Table 4. Global Deployment Archetype Comparison of Risks

Deployment Model	Scope of Failure	What Failed	Application Down	Mitigation	Instantaneous Recovery
Global Anycast with Isolated Stacks and Global Database (Figure 8)	Zone	Zonal infra or managed services	No	Continue operation from remaining healthy zones across the regions	Yes
	region	Regional infra or managed services down	No	Continue operation from remaining regions	Yes
	region	One instance of application microservice in region down	No	Continue operation of this microservice from remaining regions	Yes
	global	Global Anycast	Yes	Wait until Global Anycast recovered, Unless backup VIPs can be used for failover	No (if no backup VIP) Yes (with backup VIPs)
		Global Database	Yes	Wait until database is back	No
Global Services Stack (Figure 9)	zone	Zonal infra or managed services	No	Continue operation from remaining healthy zones across the regions	Yes
	region	Regional infra or managed services	No	Continue operation from remaining regions	Yes
	region	One instance of application microservice in region down	No	Region(s) not considered down. Load balance individual microservice to another region	Yes
	global	Global Anycast	Yes	Wait until global Anycast recovered, Unless backup VIPs can be used for failover	No (if no backup VIP) Yes (with backup VIPs)
		Global Database	Yes	Wait until database is back	No
		Global service mesh (proxyless or sidecar proxy)	No	Continue in degraded using old endpoints and health	Yes

infrequently stressed code paths that themselves may cause application outages or delays during failover.

- A standby stack must be maintained even though it is unused most of the time. In addition, there is always a risk that the standby stack will not be ready to take on primary responsibility during an outage event. To avoid this risk, failover systems should be regularly tested by

Table 5. Multi-cloud Deployment Archetype Comparison of Risks

Deployment Model	Scope of Failure	What Failed	Application Down	Mitigation	Instantaneous Recovery
Client Side Load Balancing	one cloud	Global or multi-regional issue	No	Continue operation from remaining cloud	Yes (if using client-side load balancing)
DNS Load Balancing	one cloud	Global or multi-regional issue	No	Continue operation from remaining cloud	No (DNS TTL)

promoting the standby stack to the primary stack. Special functionality testing, load testing and fault injection tools and monitoring must be used to have high confidence in a standby stack.

- Failover-to-standby models are typically more expensive, as the extra standby serving stack must be up and running, instead of reusing active additional setups in other zones or regions via load balancing.

In comparison, an application that is using a load balancing-based deployment model steers traffic away from the failure domain having an issue. This means the application does not have to deal with the above issues, because every zone and region is always taking traffic. However, it does require the application to either provision for N + 1, where 1 is a failure domain and N is needed to serve traffic for the application, or use auto-scaling with a large enough buffer.

The failover in load balancing is triggered by the change in the health checking results or by detection of outlier instances that appeared unhealthy. Typically, there is a threshold of unhealthy endpoints in a failure domain configured, so once a number of unhealthy endpoints crosses the threshold, traffic starts to be load-balanced away from the failure domain.

Some cloud provider load-balancing products implement the notion of gentle failover, where instead of moving traffic over in one step, the traffic trickles over to the other zones and regions to warm up the caches. Even though all locations have warmed up caches for their standard traffic patterns, when traffic is load-balanced to a new region, there may still be warming up that needs to occur if this type of traffic is new to that region (e.g., if traffic is language-specific).

8.2 Regional Versus Global Application Stacks

It is also interesting to compare regional application stacks to global application stacks. With a regional application stack, the whole region has to be pronounced unhealthy if there is an issue with any part of the application in a region, and that region is removed from serving traffic. This occurs even if it is only one service in the application stack that is unhealthy for that region.

In comparison, a global application stack is not considered unhealthy if an individual service in a region has an issue, because each individual service in the application can be independently load-balanced to another region as needed. Independence of individual service or microservice failover makes the Global Services Stack deployment model to be the most cost-optimized and have lower complexity as there is no need to aggregate failures together or propagate them up the stack. This model also matches the management of the running application where various microservices are owned and operated by different teams.

In addition, with the Global Service Stack, it is more nimble to expand the service into additional zones and regions, should the business expand into new geographies. For example, some applications may need part of the stack to be as close to the consumer as possible and therefore

want to serve only part of the stack from edge locations, where the amount of compute resources and network bandwidth is limited. Because each service and microservice is individually deployed in a Global Services Stack, this provides the ability to deploy only latency-critical services at the edge and leave the rest of the stack in cloud provider regions.

Ownership and management of infrastructure, of the outage when infrastructure fails, and the subsequent recovery is an important dimension to consider. When an application owner needs a global deployment, but only has access to or decides to use regional cloud infrastructure, this requires the application owner to build custom infrastructure that stitches regions together to create a seamless global deployment. This custom global infrastructure is complex to build and operate to achieve high availability, and the ownership falls on the application owner. Using a Global Services Stack deployment provided by a cloud provider moves the complexity and responsibility of custom LBs from application owner to the cloud provider, along with the responsibility for the availability of all the infrastructure needed to provide the global serving stack.

8.3 Instantaneous Recovery

The final column in Tables 1, 2, 3, 4, and 5 examines which cases support instantaneous recovery, which is the ability to switch away from the failed application stack in a zone or region without incurring a delay in sending traffic to an available zone or region. Automatic instantaneous recovery via load balancing is needed for applications requiring five-nines availability, since they cannot have manual mitigation with a budget of less than 5.25 min of unavailability per year. To achieve instantaneous recovery, cloud-native Global Anycast Load Balancing products are preferable to DNS load balancing to avoid delays that the DNS protocol introduces via TTL configuration and clients (e.g., web browsers) disobeying TTLs, as well as due to Global Anycast Load Balancing having integrated health checking. Having said that, when using Global Anycast, the availability of the application is highly dependent on the availability of Global Anycast and its single VIP. This is why there is significant investment into reliability by Cloud providers for Global Anycast load balancing. In addition to the ongoing investment in reliability, cloud providers are looking at providing a backup VIP to the Global Anycast VIP.

8.4 Cost of Availability

When comparing all of these models, every application defines an acceptable probability of failure that makes sense for the business. Based on the tolerance for failure, application and data needs, a deployment archetype is chosen. Each archetype has a cost to achieve the desired number of nines. The higher the desired number of nines, the higher the cost to achieve it. There are multiple dimensions of cost, ranging from the number of instances per zone, number of zones to use, network bandwidth cost for cross-zonal or cross-regional failover traffic, the cost of synchronous and asynchronous data replication, the cost of redundant storage, the cost of the software complexity required to assemble an application within each archetype, the cost of training and professional skills, and the cost of managing the application.

Zonal and regional archetypes are similar in cost, while deploying an application beyond one region can add cross-regional network bandwidth costs. Multi-regional and Global archetypes have even higher cost due to data replication around the world. Hybrid and Multi-cloud deployments can be even more costly due to egress cost, and data storage and replication costs, as well as the cost of redundant compute instances that are not exactly the same in each cloud and on-premises environment, which can result in unutilized resources. For example, the hardware configurations, memory sizes, networking bandwidths, storage latencies, and more, can be different across clouds and on-premises, and need to be optimized differently to arrive at efficient deployments across them. How far the application owner pushes toward using higher availability deployment models

comes at a cost, and is a tradeoff based on what the application needs and the impact on application users and the business when there is an outage.

When comparing the economics of redundancy in regional and global application stacks, we can observe that global deployments are cheaper, while providing higher reliability. Let us look at an example. Multi-regional deployments typically replicate within a region and run three zones for each region for redundancy with the cost of 3*N, where N is the number of regions. Three zones should be used, because these are regional vertical stacks (Section 4.3), and you need redundancy within a region to maintain regional availability when a zone fails, versus having to failover the regional vertical stack. This model is typically used across two regions. As you increase the number of regions, the cost goes up dramatically, and it makes sense to move to a global deployment from a cost perspective. A global deployment requires only one fault domain (zone) within a region, so the cost is N instead of 3*N for the application serving stack. As more regions are added, the costs of additional regions are incremental for a global deployment.

8.5 Using Deployment Models

In this section, we compare archetypes and deployment models on geographic redundancy, failover to standby versus load balancing, instantaneous recovery, ownership of the infrastructure, and cost. There is no perfect deployment model, and every application owner needs to choose a deployment model(s) that is best for their application, and business. We now summarize the choice of deployment models and use cases fitting for each deployment model discussed in this article.

- **Single Zone**—Single-zone model (Section 2.1) is used for development and testing, as high availability is not required and single zone configurations reduce costs. It also can be used for high performance processing, or applications that are not user-facing or business-critical where availability is not a priority. Legacy license-based workloads may also benefit from this deployment due to license costs.
- **Single Zone with Failover Zone**—Targets for this deployment model (Section 2.2) are non-critical services or services that can have a downtime or maintenance window. This includes off-the-shelf software workloads that cannot be changed or single license workloads. This deployment model increases availability for a single zone application. It doubles the cost if configured for instantaneous failover (active-active or hot-standby), but cost could be kept down if instantaneous failover is not needed and the failover zone is only used when the primary zone is down. Depending on the choice of standby type [9], the time to recover changes. In cloud deployments, the time to recovery can be mitigated by modern tooling [27] (e.g., instance templating, configuration automation and infrastructure as a code), so the service operator can recreate resources in another zone in a matter of minutes to reduce recovery time, though it continues to be non-instantaneous.
- **Single Region**—The Single Region model (Section 3.1) is used for serving localized applications with their local users, which keeps latency low, data locally replicated within a region and are naturally single homed. Availability is achieved by replication across three zones. It also could be part of a larger setup where application owners configure N single regions independently to serve a worldwide audience in separated regions by issuing different application endpoints for the audience in various geographies. If high availability is not an overriding priority, then two zones within a region could be used for replication to allow for one zone to be down or one zone to be in maintenance, but not both at the same time. This balances availability with cost.
- **Single Region with Failover Region**—When enterprise legacy applications that are designed for a single on-premises region move to the cloud, they want to increase their

availability by having a **Disaster Recovery (DR)** option [131]. There are a range of disasters (e.g., natural disasters, regional network failure, regional system failure, human errors) where having failover across regions provides protection, as described in Reference [115]. Single Region with Failover Region (Section 3.2) improves availability of a single region deployment model by keeping an extra region, should the primary region go down. Data is replicated, usually asynchronously, between two regions and is used for DR. Public cloud providers (e.g., AWS [12] and GCP [47]) advocate for using multiple regions with DR as best practice. When using multi-region DR, an important recommendation is to observe the health of each layer of the application and fail the primary region should any layer (service) go down across all zones within a region. This recommendation applies to all deployment models with isolated regional stacks. If instantaneous failover and hence high availability is not required, then the failover region may be kept dormant (via scaling down) to keep costs down. The choices of cold versus warm standby described in "Single Zone with Failover" in this Section apply here as well.

- **Multi-regional: Isolated Regional Stacks with Data Sharding**—This deployment model is used by application owners who in case of a region down event are willing to have service unavailable for users mapped to the affected region, in exchange for keeping the data resident in the region. An example of an application with this deployment model (Section 4.1) is banking ("www.us.hsbc.com" for U.S. customers and "www.hsbc.co.uk" for UK customers), where U.S. customers will only be served from U.S. cloud region due to data locality and local regulations. Users use this model via regional API endpoints. For higher availability, the variation of this deployment model calls for each isolated region to have a failover region, which increases availability, but it also increases the cost. Data replication across primary and failover regions is required. This variation can be used when data residency requirements are applied to a jurisdiction, where jurisdiction encompasses at least two regions.
- **Multi-regional: DNS LB with Isolated Regional Stacks**—This deployment model (Section 4.3) is commonly used to produce a serving application for worldwide users [86]. While it does have regional isolation at each layer, it costs three times of global services stack (Section 5.3) with a lower potential availability due to the DNS TTL problem in Section 4.3. In addition to global DNS LB, there are other global services needed such as authentication. Choice of this deployment is often dictated by the architecture of the cloud provider platform (e.g., AWS encourages designs for isolated regional stacks), so applications on the top of the platform generally use cloud providers endorsed architecture [12, 44].
- **Multi-regional: DNS LB with Custom Multi-region LB**—This deployment model, described in Section 4.4, is used by application owners [36, 64], who want to use regional deployments as building blocks to produce a global application with custom logic to stitch layers of a multi-layered (micro-service) application together. This model is expensive, because there is an additional cost to maintaining your own custom load balancer across regional/global layers responsible for routing and authentication. Applications that are built on the top of cloud platforms geared toward regional isolation, should use this architecture if they want to provide seamless global service.
- **Global Anycast LB with Isolated Stacks**—Global Anycast LB with Isolated Stacks targets global highly available applications as described in Section 5.2. Similar to "Multi-regional DNS LB with Isolated Regional Stacks" (Section 4.3), the benefit of this deployment model is one VIP that is mapped to a DNS name worldwide. As an example, this model enables application use cases described in Reference [4] where for a global application, the data would be dynamically placed to optimize for user latency. Even as more vertical regional compute stacks are added in new regions, both front-end serving and dynamic data placement do

not need to change as Global Anycast abstracts regional VIPs, and dynamic data placement abstracts the region where users of the application are mapped to. Combination of Global Anycast, global dynamic data placement and regional compute is an incremental step from multi-regional to global architecture. GCP, Azure, and AWS support this deployment model.

- **Global Services Stack**—The Global Services Stack (Section 5.3) is used for latency sensitive serving applications with a worldwide audience or even with local users that need to be served out of more than one region. This model is the cheapest of all multi-region and global deployment models as it only requires one zone per region, while the rest of deployments require three zones per region to keep regional availability high. This deployment model has a risk of a global aspect of the service having an outage, since those global aspects do not have regional isolation, as we discussed in Section 5.3.
- **Hybrid**—The Hybrid model, described in Section 6, is used for connecting on-premises services with public cloud services and as such is used either for transition of application from on-premises to public cloud, or when part of application always stays on-premises. If this deployment consists only of one public cloud, then an application developer has the option of using the cloud's APIs while calling from on-premises. Should the hybrid deployment also contain services from more than one public clouds [19, 119], then the application developer should consider an abstraction layer to isolate each cloud's APIs or use portable solutions [96] across both on-premises and public cloud.
- **Multi-cloud**—The Multi-cloud model, described in Section 7, relies on composing deployment out of the above deployment models [5]. The choice of deployment within each cloud depends on the purpose of multi-cloud deployment. If the purpose is replication with instantaneous failover, then the same deployment models in each cloud are recommended and developers of such applications should consider feature parity across clouds and as such lean toward portable products and solutions. If the purpose of deployment is to connect services in multiple clouds (e.g., due to acquisitions), then the mix and match of deployments work, and an application owner may choose the deployment model depending on the product portfolio cloud providers have (e.g., multi-regional on one cloud and global on another).

8.6 Best Practices

Deployment archetype questions come up often during all phases of the application owner's journey to cloud: from the design of their first deployment, to scaling the application as traffic grows, to enforcing consistency of technologies across the teams, and more. While questions are the same, the answers are different for digital-native versus enterprise application owners. Even within each category of application owners (e.g., digital natives versus enterprise) the desirable deployment model depends not only on the type of application but also on philosophy of how teams operate within the company, what are the principles of availability the application owner subscribes to and which of them are more important. Some examples of challenges [20] application owners face are:

- **Move from Services to Microservices**—If an application owner decides to move their deployment from monolith or set of services to microservices [7, 45, 110, 61], then it changes deployment needs and creates a massively distributed system that increases layering and with it creates requirements in the area of observability [80], service dependency tracking and reporting [77], and applying security policies across the regional or global stack of microservices [98].
- **Lift and Shift**—To simplify the transition from on-premises to public cloud [5], as a first step an application owner may decide to replicate their on-premises deployment. While the

deployment in the on-premises datacenter worked for them, a cloud deployment that mirrors on-premises deployment may not be satisfactory, either due to non-optimized costs or change in operations model, as now the public provider operates the platform and may generate unanticipated events. The modernization of deployment, including cloud native technologies such as autoscaling, replication and investing in fault tolerance may be needed. Authors in Reference [76] examine the pluses and minuses of lift and shift and how to optimize a strategy for a particular deployment. Migration planning to understand enterprise's infrastructure readiness, public cloud costs, performance implications, and security are discussed in Reference [102].

- **Organic Growth**—as a company grows, it may require changes to the deployment model of applications [94] either due to (a) scalability challenges a current deployment creates, (b) a business going global and hence has new geographies where users are accessing this application from, (c) usage of different storage and database products and solutions, or (d) the desire to integrate deployments of acquired companies. There are typically two paths: (i) evolution in place by gradually moving from one deployment model to another, or (ii) creating brand new deployment strategy and moving services one by one to the new framework.

  These paths are often accompanied by application modernization, and here are three examples.
  (1) Transition from VMs to Containers leads to a transition from services to microservices described in "Move from services to microservices" in this Section. It will overtime trigger a change in the existing deployment archetype from one monolithic deployment model to a purpose-focused collection of deployment models.
  (2) Transition from proprietary service to service communication protocols to gRPC results in a fine-grained, method-based service to service communication paradigm that encourages purpose-focused deployment models.
  (3) Transition from self-implemented service discovery to cloud provider's native solution modernizes networking toward adoption of service mesh with sidecars or even proxyless solutions, both of which introduce flexibility in choice of deployment models and an ease of evolution from one deployment model to another.
- **Increased Regulations**—Regulatory requirements such as GDPR, impose data residency and data transfer topology constraints [109]. Application owners need to choose deployment archetypes to fit into these geographical constraints [13]. As a best practice, users are mapped to jurisdictions where they live and their data at rest and data in transfer are contained within that same jurisdiction. Application deployment to address regulation requirements may use Single Region with Failover within the same jurisdiction (e.g., GCP has Sydney and Melbourne regions in Australia), or any of the Multi-regional archetypes (Section 4) that are constrained to VIPs and compute resources within the jurisdiction.
- **Deployment Archetype Change**—Time to market or some other constraints may cause application owners to choose a deployment archetype that serves them well to get them to market, but not the desired archetype for the long term. Cloud providers should facilitate migration of deployments from one archetype to another [94]. This is better done when products and solutions are built as a set of building blocks instead of monolithic products, so parts may be exchanged. Every cloud application could be thought of as a combination of data plane, control plane and management plane. Each of these planes may have different availability, disaster recovery or regulation requirements. Each plane may have a different deployment model to satisfy its requirements. As requirements change, each plane can be evolved from one deployment model to the next one independently from other planes.

- **Service Layering Considerations**—An application is a collection of services and microservices built on a mix of **Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), Network-as-a-Service (NaaS)** service, and cloud delivery models [120], and also usually includes multiple **Software-as-a-Service (SaaS)**, provided by the cloud provider and by **Independent Software Vendors (ISV)**. The bottom layer, consisting of IaaS and NaaS, becomes a foundational deployment model and each layer on the top will inherit properties of layers below. These concepts are orthogonal to the deployment models described in our article. Let us examine how to combine the deployment models in our article and the layers from Reference [120]. One option could be to assemble Multi-regional with Isolated Stack and Data Sharding deployment (Section 4.1) from the bottom up, starting with regional IaaS, and then adding regional Kubernetes cluster as SaaS and regional Serverless events as PaaS. Another option is to build on the global stack (any deployment models from Section 5). The option to avoid is the one where layering violates desired properties of isolation, availability, determinism, and RTO/RPO, such using a regional foundation, then adding global SaaS on the top and still expecting regional isolation and geographical determinism in request serving. For applications focused on high availability, the weakest link amongst all included layers and components will define application availability.
- **Hybrid and Multi-cloud**—This is a deployment composed across multiple clouds or on-premises and cloud. This presents the question of how to architect each cloud's deployment and compose the cross-cloud deployment overall. This usually depends on the purpose of the application as described in Sections 6 and 7. The typical choices are (a) replication across clouds for failover [1], (b) ability to run on any cloud to maintain the optionally to exit or move to any other cloud, (c) connecting clouds due to integration of acquisitions [120], and (d) using best of breed products on each cloud [97]. Depending on the choice, the deployment models will be different.
- **Organizational Culture**—The choice of deployment models often depends on organizational culture. Usually, the choice is made by the team that either provides (a) blueprints for application developers on how their applications should be deployed or (b) allows teams to pick a model that fits their application ((a) assures consistency of deployments across the company and enforces best practices, while (b) enables flexibility of how teams operate services, while advising on best practices).

8.7 Deployment Efficiency

Aside from availability, latency and cost, other factors such as sustainability [18], energy efficiency, and compute and storage resource rightsizing, also contribute to the deployment model decision process. Below, we look at several factors that contribute to efficiency of application deployment and how it utilizes resources:

- **Preemptable Resources in Non-serving Applications**—Data pipelines that process and analyze the data can achieve greater efficiency and cost by using preemptible VMs or spot instances [78] either directly or running containers over preemptible VMs. While preemptible VM and spot instances are cost-efficient, they can create uncertainty about completion of work items. References [82, 107] discuss the tradeoffs for when to choose between using spot instances versus regular instances. To solve the uncertainty about spot instances going away, data pipelines can be designed to checkpoint and restart or to have their task complete in short time periods. Reference [95] describes efficiency gains by running a music ingestion and analysis data pipeline consisting of multiple microservices, each of which performs a single task in a time shorter than preemption notice [139–141], and the

preemption notice typically gives 30 seconds to a couple of minutes for the task to shutdown. Microservices can also be designed to be resumed from the last known state to better utilize preemptible computing. The efficiency gain in cost savings of using preemptible VMs over regular VMs can be substantial but does require re-architecting applications [95].

- **Cost Optimized VMs**—GCP's E2 VMs [74, 134] employ dynamic resource management by mapping virtual CPU and memory to physical CPU and memory, thus packing more VMs on fewer but larger physical servers, creating potential savings up to 55% [14]. Cloud applications that are not latency sensitive and want to run efficiently, should use this type of VMs.
- **Scale Down Unused Resources**—There are types of stateless and stateful applications that can be scaled down to minimum or zero compute consumption when unused [93, 122]. These applications typically perform scheduled tasks and spin up when they need to run, or the application can tolerate the time to spin up compute when a request arrives. Also, scaling down of unused resources can be used for the warm standby stack in failover archetypes described in Sections 2.2 and 3.2.
- **Overall Efficient Application Deployment**—Applications can get efficiency gains by optimizing the choice of the computing platform (VMs, containers, serverless, functions), the choice of storage and database, and the choice of RPC or work-queue-based service communication [106, 128]. Reference [24] describes the modeling of applications based on the choice of suitable VM type to meet the deployment constraints for each service in the application. The goal is to choose the VMs that consume the minimum number of resources that provide the performance needed and minimize execution costs. To achieve overall efficient application deployment [106, 128], developers design their application for the tasks to be accomplished, from serving real time traffic to backend pipeline processing, from mission critical 24/7 applications to ones that can tolerate some downtime, from daemons or work-item-based application that run for a short time, and more.
- **Sustainable Application Model**—[43] suggests that serving applications use thread and tasks to parallelize execution for efficiency. Similar to applications moving from single threaded to multi-threaded microservices, the transition from VMs to containers orchestrates better bin packing. The bin packing increases utilization of underlying VMs by placing applications that bottleneck on different resources (some applications are memory intensive, some CPU intensive, and some I/O intensive) on the same VM. Similarly, bursty applications can utilize compute capacity in close-by zones by reusing existing compute capacity, while not increasing network costs and still maintaining data locality.
- **Minimize Presence in Each Cloud Region**—As described in Sections 5.3 and 8.4, the Global Services Stack minimizes total resource usage, because only one zone within each region is needed, which in turn can provide up to three times resource savings compared to Multi-regional deployments.
- **Capacity-aware Load Balancing**—Reference [41] enables traffic distribution where only enough traffic is sent to the closest zone or region until target server utilization is reached and the rest is sent further away. This algorithm improves server utilization as in steady state it keeps servers highly utilized and reduces the size of the autoscaling buffer needed and minimizes network costs, because traffic is assigned to the closest zone. Note, while this algorithm provides savings on compute and network resources during steady state, during overflows there is a temporary rise in network costs and latency. These algorithms can be present in multi-regional deployment models for load balancing within a region, but it is more efficient in the Global Services Stack deployment model. In on-premises environments capacity can be constrained and capacity aware-load balancing combined with hybrid

deployments can overflow traffic to the cloud. In GCP, capacity aware load balancing can be configured as described in Reference [72].

- **Data Locality**—It is important to note that co-location of data and compute resources minimizes data transfer [109] and as such contributes to efficiency of application. In all deployment models, we assumed that compute resources and data are co-located. The choice of the location of compute resources is driven either by where the majority of the application users are located or by regulatory requirements. Once compute resource locations are chosen, data is co-located with compute by specifying the same locations in multi-regional storage and database configuration, or alternatively data locality is decided algorithmically based on what is best for the user [108]. It is possible to have compute resources placed algorithmically based on minimum RTT to the users as well and have data always co-located with compute resources. To further increase efficiency of data access, caching is used at multiple layers of the application. Global and Multi-regional deployment archetypes use a global CDN to bring data as close as possible to the user. Subsequently, each layer caches to optimize for serving latency as well as for minimizing capacity of compute resources to serve data from storage and databases.

9 ADDITIONAL RELATED WORK

In this section, we will cover additional related work in areas of building deployment archetypes, building applications for availability, application delivery using load balancing, autoscaling and health-checking, and for hybrid and multi-cloud applications.

Reference [20] provides an extensive view of the foundation for cloud computing technologies, investigates their state-of-the-art solutions, and identifies their limitations. It offers comprehensive directions, identifies trends, challenges, and solutions needed to produce cloud applications of different types—from cost sensitive, to scalable global enterprise applications, and to pervasive and ubiquitous applications that span globe, hybrid, multi-cloud, further spreading to edge to enable **Internet of Things (IoT)** and fog computing. Our article adds to this work, by describing in detail the deployments archetypes spanning zonal, regional, multi-regional, global, hybrid and multi-cloud and their tradeoffs and implications for application architecture, availability, load balancing and autoscaling, geographic redundancy and isolation, cost, and best practices.

Architecting for Availability—Reference [137] examines in detail cloud application survivability concepts such as fault-tolerance, reliability, and availability. They consider failures of software components, hardware infrastructure, and a failure of a zone or one or more regions, as events contributing to cloud application failure, and they describe deployment archetypes to sustain these failures. Reference [9] proposes models to calculate availability of standby application stacks that depend on standby modes (cold, warm, hot) and application internal architecture, such as shareable components. Reference [129] survey classifies layers of resiliency in centralized clouds and decentralized clouds, such as fog, mist, mobile and edge mobile computing. In comparison, our article examines the deployment archetypes focused on availability, latency, and geographical redundancy and isolation across zonal, regional, and global deployment models. This is just as important to the understanding of the overall architecture as choosing the type of infrastructure and services (e.g., VMs, containers, serverless, microservices) to use to build your application, which the prior work focuses on.

Pervasive and ubiquitous applications are built on common architecture for application owners all over the globe. For e-Commerce sites, streaming video services, news sites, gaming platforms, and IoT services [26] that require 24/7 uptime, a multi-regional architecture is preferred [37] due to minimized network latency, DR strategy (active-active or active-passive) and meeting regulatory compliance (build multi-regional architecture in different geographic regions, but store application

user related data in the same geographic region as the application user). Estrin surveys 3 public cloud providers, Amazon Web Services, Microsoft Azure, and Google Cloud, in the areas of DNS, CDN, DDoS, storage, and databases, while also considering cost and operational aspects [37]. We build on this work to examine deployment archetypes beyond multi-regional.

References [65] and [133] examine models for estimating availability within a region for different failure types, functionality constraints, redundancy, and interdependency models between various components. Reference [65] models availability for a multi-tiered application with an example of three-tiers with (1) frontend HTTP servers, (2) business logic application as the middle tier, and (3) a database storing the system state at the back-end. Reference [133] looks at the spectrum of deployment stack for stateless versus stateful applications within a zone, within a region and sustaining regional failures with a failover option to obtain the best possible RPO and RTO.

Automated Data Placement—Reference [4] proposes dynamic data placement for geo distributed cloud applications using Volley—a system that solves for constraints such as WAN bandwidth costs and datacenter capacity limits, while also minimizing user-perceived latency. In addition, issues of shared data, and data inter-dependencies complicate placement especially for regulated industries. The Volley system analyzes collected logs to drive data migration to geographical location closest to each user based on a set of constraints. This approach relates to the data sharded model in Section 4.1, applying intelligence for moving the data closer to the user.

Traffic Management, Load Balancing and Autoscaling for Cloud Applications—Each of the deployment archetypes described in this article requires fault tolerance techniques to meet their needs. The approach needs to make sure geographically distributed services and microservices are not overloaded and the application as a whole is optimized for latency and can sustain faults. The higher the expected availability level [40, 42], the more traffic management support needed [89], which includes load balancing algorithms discussed in References [85, 92], health checking for resiliency as described in Reference [46], autoscaling for stateless workloads [39, 100], health-driven failover, cross-regional overflows, rate limiting [101] and load shedding [112] to protect service instances from overload. The approaches described in our article are standard ones existing in the cloud computing industry [41], and we have focused on which ones to use for the various archetype models described. The following is a set of related work that provides the details for how these approaches work for DNS, Anycast and health checking.

Authors in Reference [17] described using DNS Load Balancing to distribute internet traffic to the closest geographically and healthy regional stacks. Such distribution optimizes dynamic application capacity and instant health status of geographically distributed instances. Authors in Reference [79] describe how DNS Load Balancing evolved as a traffic management tool for cross regional traffic management. While there are more dynamic load balancing algorithms for cloud applications surveyed in Reference [3], it is basic DNS load balancing that is the best fit for vertical stack failover and an acceptable option for routing between vertical geographically distributed application stacks.

An important part of load balancing-as-a-solution is health checking. Authors in Reference [108] describe how Fastly built a distributed health checking system for making a decision about optimal traffic distribution and failover should geo distributed servers become unhealthy. Anycast Load Balancing [136] described communication paradigm at the application layer as a way to find latency-optimized replicated servers by having an anycast resolver to map an anycast domain name and a selection criteria into an IP address.

Tenereillo [116] showed that while DNS is often used for global server load balancing [62], it is not suitable for global cloud applications that require five nines of availability because of inherent delays built into DNS protocol. Globally ubiquitous applications that often use CDN [38] cannot afford DNS slow reaction during failover. Instead they use anycast for geographic routing

of internet traffic to their application stacks that are spread out across regions [66]. Google Cloud [57], Microsoft Azure [84] and AWS [10] built global anycast products for global applications to address this.

Defining QoS or SLA for application stack would translate to the need of load balancing to support QoS and SLA targeted load balancing [88, 121, 135] as intent-based constraints that internally would need to be translated into signals such as target utilization, number of errors per type, adjustment to changing health of servers. Load balancing for long (streaming) and short (request/response) flows discussed in Reference [138] further influence the choice of deployment archetype for a specific type of traffic an application receives.

Autoscaling is as important as load balancing for an application delivery. In fact, the integration of load balancing and autoscaling helps multi-regional and global deployments to sustain inorganic traffic spikes. Authors in Reference [39] describe such integration depending on the performance and capacity of servers and desired level of QoS to sustain events such as the death of celebrities [81, 130], which can send internet serving infrastructure and services into overload across many companies. Authors in Reference [100] define autoscaling taxonomy and survey autoscaling algorithms based on architecture of application (single tier versus multi-tier), reaction type to event (proactive or reactive), scaling method and other factors that contribute to multi-tier cloud application. In our article, we build on prior work by discussing how Global Services Stack archetype (Section 5.3) combined with autoscaling based on load balancing signals, creates a deployment that can withstand such inorganic traffic spikes by trading latency for availability.

Service-oriented Architecture and Hybrid Applications—The architecture used for cloud applications has evolved from monolith into **service-oriented architecture (SOA)**. The authors in Reference [70] present a review of cloud application architectures and its evolution that results in a decentralized, distributed system [114] with large numbers of moving parts of services and microservices, each of which has a deployment mode. An application that is split between on-premise data centers and a public cloud faces the pressure of how to evolve the application parts that reside on-premises (whether legacy applications or newly written) to work with cloud as described in References [7, 87, 113]. An approach often taken is to evolve the application to a service-oriented architecture through partial, in-place re-architecture and rewrite, or building the bridge to service-based application deployment [60].

We view on-premises datacenters evolving into a private region in a multi-regional deployment of inter-connected cloud as authors in Reference [119] propose. The hybrid application is then built using the desired regional with failover deployment archetype, using cross cloud autoscaling [19], multi-regional or global deployment archetypes. As enterprises bring critical workloads to public clouds, multi-tenancy of public clouds and single tenancy of on-premises can create SLA challenges for overall hybrid deployment as discussed in Reference [30].

Multi-cloud Applications—A multi-cloud application can potentially gain the benefit of improved availability by using independent cloud stacks, that do not share failures due to software bugs, and wider geographical reach and presence. This cannot be achieved without application services using cross-cloud APIs and portable open source solutions as discussed in References [31, 96] to assure that the parts of application can be either replicated or interconnected across the clouds. Multi-cloud orchestration via automation and declarative configuration [6] simplifies topology and configuration of multi-cloud applications. Authors in Reference [125] describe an approach of cloud federation using service layers (Infrastructure, Middleware, and Application) that would make an application cloud independent, but it requires all cloud providers to collaborate on federation APIs and protocols at each service layer. Multi-cloud applications can attain optimized latency [132] via extending deployments that have mobile and IoT components to edge and fog environments as discussed in References [15, 16, 111]. While edge, fog and mobile deployments

minimize user-perceived latency, they also come with limited capacity, storage, and available energy as discussed in Reference [126]. These limitations create new deployment archetypes where computations are offloaded from mobile to edge cloud, where a cloud application stack goes all the way from cloud to edge to the mobile devices.

10 SUMMARY

In this article, we examined several deployment archetypes for Cloud applications. The deployment archetypes are evolutionary, from zonal, to regional, to multi-regional, to global, to hybrid, and to multi-cloud, where each step progressively provides increasing higher availability and better end-user latency.

Each archetype has its place and importance according to unique application requirements and the tradeoffs it provides. Applications will want to push as far up the deployment archetypes as possible. For user facing-applications, DNS Load Balancing with separate stacks is the standard and familiar approach for multi-regional and cross-cloud deployments.

For applications that want to achieve the highest level of availability, the Global Services Stack deployment model is preferable to stitching isolated regional service stacks together with DNS Load Balancing, due to its ability to load-balance with fully integrated health monitoring and understanding capacity management. Multi-cloud with load balancing is the deployment model to watch as it evolves, with related technologies also rapidly evolving. This model drives the importance of open APIs and client-side traffic management.

ACKNOWLEDGMENTS

We thank Andi Gutmans, Geoff Voelker, Amit Ganesh, Kara Moscoe, Sachin Gupta, Ben Treynor, John Laham, Sam Greenfield, Chris Taylor, Dave Nettleton, Nirav Mehta, Olaf Schnapauff, Ines Evid, Davis Hart, Michael Abd-El-Malek, Sameet Agarwal, Zach Seils, Jai Haridas, James Duncan, Barbara Stanley, Uday Naik, Mike Dahlin, Amin Vahdat, Philippe Poutonnet, and the anonymous reviewers for providing valuable feedback on this article, and we thank everyone working on Cloud whose work informed and inspired this survey.

REFERENCES

[1] I. D. Addo, S. I. Ahamed, and W. C. Chu. 2014. A reference architecture for high-availability automatic failover between PaaS cloud providers. In *Proceedings of the International Conference on Trustworthy Systems and Their Applications*. 14–21. DOI: 10.1109/TSA.2014.12

[2] A. Adya, D. Myers, J. Howell, J. Elson, C. Meek, V. Khemani, S. Fulger, P. Gu, L. Bhuvanagiri, J. Hunter, R. Peon, L. Kai, A. Shraer, A. Merchant, and K. Lev-Ari. 2016. Slicer: Auto-sharding for datacenter applications. In *Proceedings of the USENIX Conference on Operating Systems Design and Implementation*.

[3] S. Afzal and G. Kavitha. 2019. Load balancing in cloud computing–A hierarchical taxonomical classification. *J. Cloud Comp.* 8, 22 (2019). https://doi.org/10.1186/s13677-019-0146-7

[4] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan. 2010. Volley: Automated data placement for geo-distributed cloud services. In *Proceedings of the Conference on Network System Design and Implementation (NSDI'10)*.

[5] N. Ahmad, Q. N. Naveed, and N. Hoda. 2018. Strategy and procedures for migration to the cloud computing. In *Proceedings of the IEEE 5th International Conference on Engineering Technologies and Applied Sciences (ICETAS'18)*. 1–5. DOI: 10.1109/ICETAS.2018.8629101

[6] K. Alexander, C. Lee, E. Kim, and S. Helal. 2017. Enabling end-to-end orchestration of multi-cloud applications. *IEEE Access* 5 (2017), 18862–18875. DOI: 10.1109/ACCESS.2017.2738658

[7] A. A. Almonaies, J. R. Cordy, and T. R. Dean. 2010. Legacy system evolution towards service-oriented architecture. In *Proceedings of the International in Workshop SOA Migration and Evolution*.

[8] M. M. Alshammari, A. A. Alwan, A. Nordin, and I. F. Al-Shaikhli. 2017. Disaster recovery in single-cloud and multi-cloud environments: Issues and challenges. *Proceedings of the 4th IEEE International Conference on Engineering Technologies and Applied Sciences (ICETAS'17)*. 1–7. DOI: 10.1109/ICETAS.2017.8277868

[9] S. V. Amari and G. Dill. 2009. A new method for reliability analysis of standby systems. In *Proceedings of the Annual Reliability and Maintainability Symposium*. 417–422. DOI: 10.1109/RAMS.2009.4914713

[10] Amazon Web Services. 2018. Global Accelerator. Retrieved from https://aws.amazon.com/global-accelerator/.

[11] Amazon Web Services. 2021. Amazon Route 53 Application Recovery Controller. Retrieved from https://aws.amazon.com/route53/application-recovery-controller/.

[12] Amazon Web Services. 2021. Multi-Region Application Architecture. Retrieved from https://aws.amazon.com/solutions/implementations/multi-region-application-architecture/.

[13] C. R. Baudoin. 2018. The impact of data residency on cloud computing. In *Proceedings of the 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA'18)*. 430–435. DOI: 10.1109/WAINA.2018.00124

[14] N. Bavis. 2021. Google cloud machine types comparison. Retrieved from https://www.parkmycloud.com/blog/google-cloud-machine-types/.

[15] L. F. Bittencourt, J. Diaz-Montes, R. Buyya, O. F. Rana, and M. Parashar. 2017. Mobility-aware application scheduling in fog computing. *IEEE Cloud Comput.* 4, 2 (Mar./Apr. 2017), 26–35. DOI: 10.1109/MCC.2017.27

[16] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. 2012. Fog computing and its role in the internet of things. In *Proceedings of the 1st Edition of the MCC Workshop on Mobile Cloud Computing (MCC'12)*. ACM, New York, NY, 13–16. DOI: https://doi.org/10.1145/2342509.2342513

[17] T. Brisco. 1995. RFC 1794. DNS Support for Load Balancing. IETF.

[18] R. Buyya and S. S. Gill. 2018. Sustainable cloud computing: Foundations and future directions. Business Technology & Digital Transformation Strategies. *Cutter Consort.* 21, 6 (2018), 1–10.

[19] R. Buyya, R. Ranjan, and R. N. Calheiros. 2010. InterCloud: Utility-oriented federation of cloud computing environments for scaling of application services. *Algor. Architect. Parallel Process.* (2010), 6081.

[20] R. Buyya, S. N. Srirama, G. Casale, R. Calheiros, Y. Simmhan, B. Varghese, E. Gelenbe, B. Javadi, L. M. Vaquero, M. A. S. Netto, A. N. Toosi, M. A. Rodriguez, I. M. Llorente, S. De Capitani Di Vimercati, P. Samarati, D. Milojicic, C. Varela, R. Bahsoon, M. Dias De Assuncao, O. Rana, W. Zhou, H. Jin, W. Gentzsch, A. Y. Zomaya, and H. Shen. 2018. A manifesto for future generation cloud computing: Research directions for the next decade. *ACM Comput. Surv.* 51, 5, Article 105 (Jan. 2019), 38 pages. DOI: https://doi.org/10.1145/3241737

[21] C. Bethea, G. Sheerin, J. Mace, R. King, G. Luo, and G. O'Connor. 2018. The site reliability workbook. Retrieved from https://sre.google/workbook/managing-load/.

[22] M. Brantner, D. Florescu, D. Graf, D. Kossmann, and T. Kraska. 2008. Building a database on S3. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 251–264.

[23] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, Y. Xu, S. Srivastav, J. Wu, H. Simitci, J. Haridas, C. Uddaraju, H. Khatri, A. Edwards, V. Bedekar, S. Mainali, R. Abbasi, A. Agarwal, M. F. ul Haq, M. I. ul Haq, D. Bhardwaj, S. Dayanand, A. Adusumilli, M. McNett, S. Sankaran, K. Manivannan, and L. Rigas. 2011. Windows azure storage: A highly available cloud storage service with strong consistency. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP'11)*.

[24] M. Ciavotta, G. P. Gibilisco, D. Ardagna, E. Di Nitto, M. Lattuada, and M. A. Almeida da Silva. Architectural design of cloud applications: A performance-aware cost minimization approach. *IEEE Trans. Cloud Comput.* 10.1109/TCC.2020.3015703

[25] J. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford. Spanner: Google's globally distributed database. In *Proc. USENIX Conference on Operating Systems Design and Implementation (2012)*. USENIX, 251–264.

[26] L. Chen et al. 2021. IoT microservice deployment in edge-cloud hybrid environment using reinforcement learning. *IEEE Internet Things J.* 8, 16 (15 Aug. 2021), 12610–12622. DOI: 10.1109/JIOT.2020.3014970

[27] DarylsCorner. 2016. Using terraform across multiple cloud providers. http://darylscorner.com/2016/11/using-terraform-across-multiple-cloud-providers/.

[28] F. Denis. 2019. Stop using ridiculously low DNS TTLs. APNIC Blog (Nov. 2019). https://blog.apnic.net/2019/11/12/stop-using-ridiculously-low-dns-ttls/.

[29] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. 2007. Dynamo: Amazon's highly available key-value store. In *Proceedings of the ACM Symposium on Operating Systems Principles*.

[30] L. Dhirani and T. Newe. 2020. Hybrid cloud SLAs for industry 4.0: Bridging the gap. *Ann. Emerg. Technol. Comput.* 4 (2020), 41–60.

[31] B. Di Martino, G. Cretella, and A. Esposito. 2015. Advances in applications portability and services interoperability among multiple clouds. *IEEE Cloud Comput.* 2, 2 (Mar./Apr. 2015), 22–28.

[32] D. E. Eisenbud, C. Yi, C. Contavalli, C. Smith, R. Kononov, E. Mann-Hielscher, A. Cilingiroglu, B. Cheyney, W. Shang, and J. D. Hosein. 2016. Maglev: A fast and reliable software network load balancer. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI'16)*.
[33] C. Engelmann and S. L. Scott. 2005. Concepts for high availability in scientific high-end computing. In *Proceedings of High Availability and Performance Workshop (HAPCW'05)*.
[34] Envoy Proxy APIs. Retrieved from https://www.envoyproxy.io/docs/envoy/latest/api/api#.
[35] Envoy Proxy Mobile. Retrieved from https://envoy-mobile.github.io/.
[36] J. Evans. 2016. #NetflixEverywhere global architecture. QCon London (Mar. 2016). https://www.infoq.com/presentations/netflix-failure-multiple-regions/.
[37] E. Estrin. 2020. Using the cloud to build multi-region architecture (May 2020). https://www.europeclouds.com/blog/using-the-cloud-to-build-multi-region-architecture.
[38] A. Flavel, P. Mani, D. Maltz N. Holt, J. Liu, Y. Chen, and O. Surmachev. 2015. FastRoute: A scalable load-aware anycast routing architecture for modern CDNs. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI'15)*.
[39] H. Fernandez, G. Pierre, and T. Kielmann. 2014. Autoscaling web applications in heterogeneous cloud infrastructures. In *Proceedings of the IEEE International Conference on Cloud Engineering*. 195–204. DOI: 10.1109/IC2E.2014.25
[40] P. Garraghan et al. 2018. Emergent failures: Rethinking cloud reliability at scale. *IEEE Cloud Comput.* 5, 5 (Sep./Oct. 2018), 12–21. DOI: 10.1109/MCC.2018.053711662
[41] P. Geetha and C. R. R. Robin. 2017. A comparative-study of load-cloud balancing algorithms in cloud environments. In *Proceedings of the International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS'17)*. 806–810. DOI: 10.1109/ICECDS.2017.8389549
[42] R. Gensh, A. Rafiev, A. Romanovsky, A. Garcia, F. Xia, and A. Yakovlev. 2017. Architecting holistic fault tolerance. In *Proceedings of the IEEE 18th International Symposium on High Assurance Systems Engineering (HASE'17)*. 5–8. DOI: 10.1109/HASE.2017.13
[43] S. S. Gill and R. Buyya. 2018. A taxonomy and future directions for sustainable cloud computing: 360 degree view. *ACM Comput. Surv. CSUR.* 51 (2018), 1–33. https://arxiv.org/pdf/1712.02899.pdf.
[44] T. Golding. 2018. Architecting multi-region SaaS solutions on AWS. AWS Partner Network (APN) Blog. Mar. 2018. Retrieved from https- aws.amazon.com/blogs/apn/architecting-multi-region-saas-solutions-on-aws/.
[45] B. Götz, D. Schel, D. Bauer, C. Henkel, P. Einberger, and T. Bauernhansl. 2018. Challenges of production microservices. *Procedia CIRP.* 67 (2018), 167–172. https://doi.org/10.1016/j.procir.2017.12.194.
[46] M. K. Gokhroo, M. C. Govil, and E. S. Pilli. 2017. Detecting and mitigating faults in cloud computing environment. In *Proceedings of the 3rd International Conference on Computational Intelligence & Communication Technology (CICT'17)*. 1–9. DOI: 10.1109/CIACT.2017.7977362
[47] Google Cloud. Architecting disaster recovery for cloud infrastructure outages. Retrieved from https://cloud.google.com/architecture/disaster-recovery.
[48] Google Cloud Solutions. Hybrid and multi-cloud network topologies. Retrieved from https://cloud.google.com/solutions/hybrid-and-multi-cloud-network-topologies.
[49] Google Cloud SQL. Cross Region Replicas. Retrieved from https://cloud.google.com/sql/docs/mysql/replication#cross-region-read-replicas.
[50] Google Cloud SQL. Failover. Retrieved from https://cloud.google.com/sql/docs/mysql/high-availability#failover.
[51] Google Cloud SQL. Failover Process. Retrieved from https://cloud.google.com/sql/docs/mysql/high-availability#failover-process.
[52] Google Cloud SQL. High Availability. Retrieved from https://cloud.google.com/sql/docs/mysql/high-availability.
[53] Google Cloud SQL. Promoting replicas for regional migration or disaster recovery. Retrieved from https://cloud.google.com/sql/docs/mysql/replication#cross-region-read-replicas.
[54] Google Cloud SQL. Replicating from an external server to Cloud SQL (v1.1). Retrieved from https://cloud.google.com/sql/docs/mysql/replication/replication-from-external.
[55] Google Cloud Storage. Retrieved from https://cloud.google.com/storage.
[56] Google Cloud TPU Pods. Retrieved from https://cloud.google.com/tpu/docs/training-on-tpu-pods.
[57] Google Cloud Global load balancing with single anycast IP. Retrieved from https://cloud.google.com/load-balancing.
[58] Google Vitess. A database clustering system for horizontal scaling of MySQL. https://vitess.io.
[59] R. Govindan et al. 2016. Evolve or Die: High-availability design principles drawn from googles network infrastructure. In *Proceedings of the ACM SIGCOMM* 2016.
[60] A. Gunka, S. Seycek, and H. Kuhn. 2013. Moving an application to the cloud—An evolutionary approach. In *Proceedings of the International Workshop on Multi-cloud Applications and Federated Clouds*. ACM. 35–42
[61] M. Gysel, L. Kölbener, W. Giersche, and O. Zimmermann. 2016. Service cutter: A systematic approach to service decomposition. In *Proceedings of the European Conference on Service-oriented and Cloud Computing (ESOCC'16)*, M.

Aiello, E. B. Johnsen, S. Dustdar, I. Georgievski. (eds.). LNCS. Springer, Cham, 185–200. https://doi.org/10.1007/978-3-319-44482-6_12

[62] Y. S. Hong, J. H. No, and S. Y. Kim. 2006. DNS-based load balancing in distributed Web-server systems. In *Proceedings of the 4th IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems and the 2nd International Workshop on Collaborative Computing, Integration, and Assurance (SEUS-WCCIA'06)*. 4. DOI: 10.1109/SEUS-WCCIA.2006.23

[63] Istio APIs. Retrieved from https://istio.io/latest/docs/concepts/what-is-istio/.

[64] Y. Izrailevsky and C. Bell. 2018. Cloud reliability. *IEEE Cloud Comput.* 5, 3 (May/Jun. 2018), 39–44.

[65] M. Jammal, A. Kanso, P. Heidari, and A. Shami. 2016. A formal model for the availability analysis of cloud deployed multi-tiered applications. In *Proceedings of the IEEE International Conference on Cloud Engineering Workshop (IC2EW'16)*. 82–87. DOI: 10.1109/IC2EW.2016.21

[66] H. Jamous. A reference architecture for building highly available and scalable cloud application. https://scholar.google.com/scholar?hl=en&as_sdt=0%2C5&q=H.+Jamous.+A+reference+architecture+for+building+highly+available+and+scalable+cloud+application.&btnG=.

[67] M. Jayadevan and D. Wetherall. 2020. Getting higher MPI performance for HPC applications on Google Cloud. Dec. 2020. https://cloud.google.com/blog/products/compute/how-to-reduce-mpi-latency-for-hpc-workloads-on-google-cloud.

[68] L. Johansson. 2018. Database multi-cloud strategy. Retrieved from https://www.elephantsql.com/blog/2018-11-19-multi-cloud-postgresql.html.

[69] M. Jung, S. Mallering, P. Dalbhanjan, P. Chapman, and C. Kassen. 2016. Microservices on AWS. https://d1.awsstatic.com/whitepapers/microservices-on-aws.pdf.

[70] N. Kratzke. 2018. A brief history of cloud application architectures. *Appl. Sci.* 8 (2018), 1368.

[71] Kubernetes APIs. Retrieved from https://kubernetes.io/docs/concepts/overview/kubernetes-api/.

[72] Google Cloud Backend Services Traffic Distribution. Retrieved from https://cloud.google.com/load-balancing/docs/backend-service?hl=bg#traffic_distribution.

[73] P. Lewandowski. 2016. Load balancing at the frontend. Retrieved from https://sre.google/sre-book/load-balancing-frontend/.

[74] A. Liberman and T. Sanderson. 2019. Performance-driven dynamic resource management in E2 VMs. *Google Cloud Blog*. Retrieved from https://cloud.google.com/blog/products/compute/understanding-dynamic-resource-management-in-e2-vms.

[75] J. Lin and E. Brewer. 2019. Application modernization and the decoupling of infrastructure services and teams. Retrieved from https://services.google. com/fh/files/blogs/anthos_white_paper.pdf.

[76] D. S. Linthicum. 2017. Cloud-native applications and cloud migration: The good, the bad, and the points between. *IEEE Cloud Comput.* 4, 5 (Sept./Oct. 2017), 12–14. DOI: 10.1109/MCC.2017.4250932

[77] S. Ma, C. Fan, Y. Chuang, W. Lee, S. Lee, and N. Hsueh. 2018. Using service dependency graph to analyze and test microservices. In *Proceedings of the IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC'18)*. 81–86. DOI: 10.1109/COMPSAC.2018.10207

[78] G. D. Maayan. 2021. The complete guide to spot instances on AWS, *Azure and GCP*. Retrieved from https://www.datacenterdynamics.com/en/opinions/complete-guide-spot-instances-aws-azure-and-gcp/.

[79] L. MacVittie. 2015. Cloud balancing: The evolution of global server load balancing. *F5 Networks*.

[80] N. Marie-Magdelaine, T. Ahmed, and G. Astruc-Amato. 2019. Demonstration of an observability framework for cloud native microservices. In *Proceedings of the IFIP/IEEE Symposium on Integrated Network and Service Management (IM'19)*. 722–724.

[81] R. Marshall. 2009. Michael Jackson's death overloads Google, Twitter. Retrieved from http://www.mtv.com/news/1614812/michael-jacksons-death-overloads-google-twitter/.

[82] R. G. Martinez, A. Lopes, and L. Rodrigues. 2019. Planning workflow executions when using spot instances in the cloud. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing (SAC'19)*. ACM, New York, NY. 310–317. DOI: https://doi.org/10.1145/3297280.3297313

[83] Microsoft Azure Cosmos DB. Retrieved from https://azure.microsoft.com/en-us/services/cosmos-db/.

[84] Microsoft Azure Front Door. Retrieved from https://docs.microsoft.com/en-us/azure/frontdoor/front-door-overview.

[85] S. K. Mishra, B. Sahoo, and P. P. Parida. 2020. Load balancing in cloud computing: A big picture. *J. King Saud Univ. Comput. Info. Sci.* 32, 2 (2020), 149–158. https://doi.org/10.1016/j.jksuci.2018.01.003.

[86] Daniel Mittelman. 2021. Retrieved from https://engineering.monday.com/monday-coms-multi-regional-architecture-a-deep-dive/.

[87] P. Mohagheghi and T. Sæther. 2011. Software engineering challenges for migration to the service cloud paradigm: Ongoing work in the REMICS project. In *Proceedings of the IEEE World Congress on Services*.

[88] S. M. Moghaddam, M. O'Sullivan, C. P. Unsworth, S. F. Piraghaj, and C. Walker. 2021. Metrics for improving the management of Cloud environments—Load balancing using measures of Quality of Service, Service Level Agreement Violations and energy consumption, *Future Gen. Comput. Syst.* 123 (2021), 142–155, ISSN 0167–739X. https://doi.org/10.1016/j.future.2021.04.010.

[89] R. Moreno-Vozmediano, R. S. Montero, E. Huedo, et al. 2018. Orchestrating the deployment of high availability services on multi-zone and multi-cloud scenarios. *J. Grid Comput.* 16 (2018), 39–53. https://doi.org/10.1007/s10723-017-9417-z

[90] Multi-cloud MongoDB. Retrieved from https://www.mongodb.com/multicloud.

[91] P. Namuag. 2019. Deploying secure multicloud MySQL replication on AWS and GCP with VPN. https://severalnines.com/database-blog/deploying-secure-multicloud-mysql-replication-aws-and-gcp-vpn.

[92] A. A. Neghabi, N. Jafari Navimipour, M. Hosseinzadeh, and A. Rezaee. 2018. Load balancing mechanisms in the software defined networks: A systematic and comprehensive review of the literature. *IEEE Access.* 6 (2018), 14159–14178. DOI: 10.1109/ACCESS.2018.2805842

[93] S. M. R. Nouri, H. Li, S. Venugopal, W. Guo, M. He, and W. Tian. 2019. Autonomic decentralized elasticity based on a reinforcement learning controller for cloud applications. *Future Gen. Comput. Syst.* 94 (2019), 765–780. https://doi.org/10.1016/j.future.2018.11.049.

[94] C. Pahl, P. Jamshidi, and D. Weyns. 2017. Cloud architecture continuity: Change models and change rules for sustainable cloud software architectures. *J. Softw. Evol. Proc.* 29 (2017), e1849. https://doi.org/10.1002/smr.1849

[95] A. Pettersson. 2019. Music to their ears: Microservices on GKE, Preemptible VMs improved Musiio's efficiency by 7000%. Retrieved from https://cloud.google.com/blog/products/containers-kubernetes/microservices-on-gke-preemptible-vms-improved-musiios-efficiency-by-7000.

[96] D. Petcu, G. Macariu, S. Panica, and C. Crăciun. 2013. Portable cloud applications—From theory to practice. *Future Gen. Comput. Syst.* 29, 6 (2013).

[97] B. Power. 2018. Digital transformation through SaaS multiclouds. *IEEE Cloud Comput.* 5, 3 (May/June 2018), 27–30. DOI: 10.1109/MCC.2018.032591613

[98] D. Preuveneers and W. Joosen. 2019. Towards multi-party policy-based access control in federations of cloud and edge microservices. In *Proceedings of the IEEE European Symposium on Security and Privacy Workshops (EuroS&PW'19).* 29–38. DOI: 10.1109/EuroSPW.2019.00010

[99] W. Qiu, Z. Zheng, X. Wang, X. Yang, and M. R. Lyu. 2014. Reliability-based design optimization for cloud migration. *IEEE Trans. Serv. Comput.* 7, 2 (April–June 2014), 223–236. DOI: 10.1109/TSC.2013.38

[100] C. Qu, R. N. Calheiros, and R. Buyya. 2018. Auto-scaling web applications in clouds: A taxonomy and survey. *ACM Comput. Surv.* 51, 4, Article 73 (Sept. 2018), 33 pages. DOI: https://doi.org/10.1145/3148149

[101] B. Raghavan, K. Vishwanath, S. Ramabhadran, K. Yocum, and A. C. Snoeren. 2007. Cloud control with distributed rate limiting. *SIGCOMM Comput. Commun. Rev.* 37, 4 (Oct. 2007), 337–348. DOI: https://doi.org/10.1145/1282427.1282419

[102] K. Ramchand, M. Baruwal Chhetri, and R. Kowalczyk. 2021. Enterprise adoption of cloud computing with application portfolio profiling and application portfolio assessment. *J. Cloud Comp.* 10, 1 (2021). https://doi.org/10.1186/s13677-020-00210-w

[103] S. Reichling and S. Polavarapu. 2020. Traffic director and gRPC-proxyless services for your service mesh. Retrieved from https://cloud.google.com/blog/products/networking/traffic-director-supports-proxyless-grpc.

[104] Y. Rekhter, T. Li, and S. Hares. 2006. A border gateway protocol 4 (BGP-4). *RFC* 4271 (Jan. 2006).

[105] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. 1996. RFC 1918 address allocation for private internets. *Best Curr. Pract.* (Feb. 1996), 1–9.

[106] D. Riane and A. Ettalbi. 2018. A graph-based approach for composite infrastructure service deployment in multi-cloud environment. In *Proceedings of the International Conference on Advanced Communication Technologies and Networking (CommNet'18).* 1–7. DOI: 10.1109/COMMNET.2018.8360254

[107] M. Ribas, C. G. Furtado, J. Neuman de Souza, G. Cordeiro Barroso, A. Moura, A. S. Lima, and F. R.C. Sousa. 2015. A Petri net-based decision-making framework for assessing cloud services adoption: The use of spot instances for cost reduction. *J. Netw. Comput. Appl.* 57 (2015), 102–118. https://doi.org/10.1016/j.jnca.2015.07.002.

[108] L. Saino. 2018. Stable and accurate health-checking of horizontally scaled services. https://www.usenix.org/conference/srecon18americas/presentation/saino.

[109] M. Abu Sharkh, M. Jammal, A. Shami, and A. Ouda. 2013. Resource allocation in a network-based cloud computing environment: Design challenges. *IEEE Commun. Mag.* 51, 11 (Nov. 2013), 46–52. DOI: 10.1109/MCOM.2013.6658651

[110] D. Shadija, M. Rezai, and R. Hill. 2017. Towards an understanding of microservices. In *Proceedings of the 23rd International Conference on Automation and Computing (ICAC'17).* 1–6. DOI: 10.23919/IConAC.2017.8082018

[111] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. 2016. Edge computing: vision and challenges. *IEEE Internet Things J.* 3, 5 (Oct. 2016), 637–646. DOI: 10.1109/JIOT.2016.2579198

[112] R. Stanojevic and R. Shorten. 2009. Load balancing vs. distributed rate limiting: An unifying framework for cloud control. In *Proceedings of the IEEE International Conference on Communications*. 1–6. DOI: 10.1109/ICC.2009.5199141
[113] R. Taft, I. Sharif, A. Matei, N. VanBenschoten, J. Lewis, et al. 2020. CockroachDB: The resilient geo-distributed SQL database. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD'20)*. ACM, 1493–1509.
[114] A. S. Tanenbaum and M. Van Steen. 2002. *Distributed Systems*. Prentice Hall.
[115] A. A. Tamimi, R. Dawood, and L. Sadaqa. 2019. Disaster recovery techniques in cloud computing. In *Proceedings of the IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT'19)*. 845–850. DOI: 10.1109/JEEIT.2019.8717450
[116] P. Tenereillo. 2004. Why DNS based global server load balancing (GSLB) doesn't Work. Retrieved from http://tenereillo.com/GSLBPageOfShame.htm.
[117] CoreSite. The Importance of Uptime and All Those Nines. Retrieved from, 2021. https://www.coresite.com/blog/the-importance-of-uptime-and-all-those-nines.
[118] Microsoft. 2021. Troubleshoot transient connection errors in SQL Database and SQL Managed Instance. Retrieved from https://docs.microsoft.com/en-us/azure/azure-sql/database/troubleshoot-common-connectivity-issues.
[119] A. N. Toosi, R. N. Calheiros, and R. Buyya. 2014. Interconnected cloud computing environments: Challenges, taxonomy, and survey. *ACM Comput. Surv.* 47, 1, Article 7 (July 2014), 47 pages. DOI: https://doi.org/10.1145/2593512
[120] S. Totman. 2019. Merging companies, merging clouds. Retrieved from https://www.darkreading.com/cloud/merging-companies-merging-clouds.
[121] A. Tsagkaropoulos, Y. Verginadis, N. Papageorgiou, et al. 2021. Severity: A QoS-aware approach to cloud application elasticity. *J. Cloud Comp.* 10, 45 (2021). https://doi.org/10.1186/s13677-021-00255-5
[122] L. A. Vayghan, M. A. Saied, M. Toeroe, and F. Khendek. 2021. A Kubernetes controller for managing the availability of elastic microservice based stateful applications. *J. Syst. Softw.* 175 (2021), 0164–1212. https://doi.org/10.1016/j.jss.2021.110924
[123] C. Viles and J. French. 1994. Availability and latency of world-wide web information servers. *University of Virginia Department of Computer Science Technical Report* CS-94-36.
[124] C. Villamizar, R. Chandra, and R. Govindan. 1998. BGP route flap damping. RFC 2439, IETF https://datatracker.ietf.org/doc/rfc2439/.
[125] D. Villegas, N. Bobroff, I. Rodero, J. Delgado, Y. Liu, A. Devarakonda, L. Fong, S. Masoud Sadjadi, and M. Parashar. 2012. Cloud federation in a layered service model. *J. Comput. Syst. Sci.* 78, 5 (2012), 1330–1344. https://doi.org/10.1016/j.jcss.2011.12.017
[126] J. Wang, J. Pan, F. Esposito, P. Calyam, Z. Yang, and P. Mohapatra. 2019. Edge cloud offloading algorithms: Issues, methods, and perspectives. *ACM Comput. Surv.* 52, 1, Article 2 (Feb. 2019), 23 pages. DOI: https://doi.org/10.1145/3284387
[127] J. Weil, V. Kuarsingh, C. Donley, C. Liljenstolpe, and M. Azinger. 2012. RFC 6598 IANA-Reserved IPv4 Prefix for Shared Address Space, Best Current Practice (Apr. 2012). 1–11.
[128] E. Weintraub and Y. Cohen. 2017. Multi objective optimization of cloud computing services for consumers. *Int. J. Adv. Comput. Sci. Appl.* 8, 2 (2017), 139–147.
[129] T. Welsh and E. Benkhelifa. 2020. On resilience in cloud computing: A survey of techniques across the cloud domain. *ACM Comput. Surv.* 53, 3, Article 59 (June 2020), 36 pages. DOI: https://doi.org/10.1145/3388922
[130] West.andrew.g, Milowen. Examining the popularity of Wikipedia articles: Catalysts, trends, and applications. Retrieved from https://en.wikipedia.org/wiki/Wikipedia:Wikipedia_Signpost/2013-02-04/Special_report.
[131] T. Wood, E. Cecchet, K. K. Ramakrishnan, P. Shenoy, J. V. D. Merwe, and A. Venkataramani. 2010. Disaster recovery as a cloud service: Economic benefits & deployment challenges. In *Proceedings of the 2nd USENIX Conference on Hot topics in cloud computing (HotCloud'10)*.
[132] Z. Wu and H. V. Madhyastha. 2013. Understanding the latency benefits of multi-cloud webservice deployments. *SIGCOMM Comput. Commun. Rev.* 43, 2 (2013), 13–20
[133] X. Xu, Q. Lu, L. Zhu, Z. Li, S. Sakr, H. Wada, and I. Webber. 2013. Availability analysis for deployment of in-cloud applications. In *Proceedings of the 4th International ACM SIGSOFT Symposium on Architecting Critical Systems (ISARCS'13)*. ACM, 11–16. DOI: https://doi.org/10.1145/2465470.2465472
[134] J. Yang. 2019. Introducing E2, new cost-optimized general purpose VMs for Google Compute Engine. Google Cloud. https://cloud.google.com/blog/products/compute/google-compute-engine-gets-new-e2-vm-machine-types.
[135] K. Yoshida, K. Fujiwara, A. Sato, and S. Sannomiya. 2019. Spread of anycast and GSLB. In *Proceedings of the IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC'19)*. 30–35. DOI: 10.1109/COMPSAC.2019.10179
[136] E. W. Zegura, M. H. Ammar, Zongming Fei, and S. Bhattacharjee. 2000. Application-layer anycasting: A server selection architecture and use in a replicated Web service. *IEEE/ACM Trans. Network.* 8, 4 (Aug. 2000).
[137] Mohamed Faten Zhani and R. Boutaba 2015. *Survivability and Fault Tolerance in the Cloud*. Wiley Online Library.

[138] T. Zhang, Y. Lei, Q. Zhang, et al. 2021. Fine-grained load balancing with traffic-aware rerouting in datacenter networks. *J. Cloud Comp.* 10, 37 (2021). https://doi.org/10.1186/s13677-021-00252-8

[139] Google Cloud Platform Blog. 2015. Introducing Preemptible VMs, a new class of compute available at 70% off standard pricing. Retrieved from https://cloudplatform.googleblog.com/2015/05/Introducing-Preemptible-VMs-a-new-class-of-compute-available-at-70-off-standard-pricing.html.

[140] AWS Blog. 2015. Amazon Web Services EC2 Spot Instance Termination Notices. Retrieved from https://aws.amazon.com/blogs/aws/new-ec2-spot-instance-termination-notices.

[141] Azure Blog. 2020. Announcing the general availability of Azure Spot Virtual Machines. Retrieved from https://azure.microsoft.com/en-us/blog/announcing-the-general-availability-of-azure-spot-virtual-machines.

Received April 2021; revised September 2021; accepted November 2021